NTNU
Norwegian University of
Science and Technology

Lecture Quiz 3.0
A Gaming Platform for Lectures

Kristian Døvik
John Andre Hestad

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Alf Inge Wang, IDI
Co-supervisor: Bian Wu, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Lecture Quiz is a game similar to Screenlife's Scene It? and Sony's Buzz! games, used in lecture halls for rehearsing theory and making learning more entertaining for students. The system consists of a server with quizzes and statistics, a player client running on any device (both mobile and stationary) with a graphical web browser, and a presentation client running the game shown on a video projector. The software should at least support Microsoft Windows and Mac OS X.

The goal of this master thesis is to create a fully functional lecture quiz game, with flexible software architecture. The architecture should support easy extension of game logic, and easy modification of visual and input interfaces. The quality focus of the system is on usability, modifiability and availability; while making it entertaining for the end-user.

The thesis should include a user test, where the game is tried out in a real lecture environment. This is to evaluate the usefulness of the system, its user-friendliness and if the system can be used to increase student learning.

Assignment given: 26. January 2011
Supervisor: Alf Inge Wang, IDI

Preface

This thesis is the result of the subject *TDT4900 - Computer and Information Science, Master Thesis* at the Department of Computer and Information Science, under Faculty of Information Technology, Mathematics and Electrical Engineering, at the Norwegian University of Technology and Science.

We would like to thank our supervisor professor Alf Inge Wang for guidance, and support throughout the implementation and writing of this thesis, giving us the opportunity to work on the *Lecture Quiz*-project from fall 2010 to spring 2011, and the freedom to explore our areas of interest. It has been an exciting year for us.

We owe our deepest gratitude to Ole Goethe for creating the graphics of our Presentation and Player clients.

We would also like to thank Kristin Robberstad Næss for thorough proofreading and commenting on the thesis.

Trondheim, June 27th, 2011.

Kristian Døvik John André Hestad

Abstract

This thesis is the continuation of our specialization project, Lecture Quiz 2.5. This platform is a game-like system where lecturers can hold quizzes in lectures to increase student participation and interactivity. The current version is a finished lecture quiz system that can be used in lecture environments. Lecture Quiz 3.0 has moved away from earlier implementations, by centralizing and minimizing the effort to start and run quizzes. One focus was multi-platform and we developed the system to support Microsoft Windows, Mac OS X and Linux.

This system can be used in lecture environments to promote more student participation, and enable variation in teaching methods. To run quiz games, the lecturer can use a PC, connect it to the projector, and run the Presentation Client. Students access the Player Client via a mobile device such as a smart phone or notebook, the address to the Player Client web page is presented on the Presentation Client. Once connected, they choose a username, and answer multiple choice questions, which are presented on the projector screen.

To keep things interesting for the students, we focused on the visual expression of the Presentation Client and Player Client. This is to give the players the experience of playing a game, rather than answering a questionnaire. We developed the system with usability in mind. This is to ensure that the system feel easy to use, for both students and lecturers. One of the main goals is to make lecturers see the system as an alternative to a regular presentation, and not as extra work.

A lecturer might be interested in collecting statistics about the students' overall progress in the course. This way they might be able to give a larger focus to the parts of the syllabus where the students lack performance. Another factor is that creating quizzes is time consuming, and needs to be done in advance of a lecture. We developed a separate quiz manager and statistics tool that can be used by lecturers, named Quiz Server. It is a Web based application, utilizing Java EE to enable multi-platform support.

We performed an experiment in a lecture to get feedback from students on how they perceived the Lecture Quiz game. This experiment was performed by running a quiz in the lecture hall and then the students were asked to fill in an evaluation form. The students who participated thought that Lecture Quiz had a positive effect on the lecture.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

Educational games have always been concerned with having a larger focus on education than on gaming [44]. This is a balance that is quite hard to strike, and games often end up being playable but not educational or just plain work/learning. Lecture Quiz takes a shot at combining the fun of party games like Scene It? [62] and Buzz! [27] with educational content.

Today, lectures at university level are generally traditional, and critics point out that lecturing is mainly a one-way method of communication, that does not involve significant audience participation. Usually the lecturer presents information with the help of slides or presentations, and gets feedback from the students by allowing them to raise their hands, and ask questions. Critics, such as Edward Tufte, contend that this style of lecturing bombards the audience with unnecessary and possibly distracting or confusing graphics [67]. The aforementioned type of feedback usually limits the number of involved students, as not everyone likes to talk in front of a crowd. Using technology to overcome this obstacle is one of the main purposes of this thesis, as smartphones and notebooks are common equipment for students these days.

1.1 Motivation

Party games have become the new form of home entertainment for people of all ages. Quiz games are popular today, where a leading example of a quiz video game is Buzz!. In 2006, the second game in the Buzz series, Buzz!: The Big Quiz, won the British Academy of Film and Television Arts (BAFTA) award for Best Casual and Social game [53].

The goal is to create a system which will be frequently used by lecturers and students. We want the system to help enhance participation and learning while keeping it entertaining for students. In addition to this we want to give lecturers a tool to present information in an alternative way, and the ability to track students overall progression by storing quantitative data. The main focus and goal are to be able to bring gaming into lectures, so that students' interest in learning might increase.

Our motivation is to create a solid playable solution that will be used and not archived. We want to get experience with working on larger projects and exploring technologies. Team work was an important factor in our specialization project, and has motivated us to continue with the master thesis.

1.2 Project Context

Norwegian University of Science and Technology (NTNU), with Alf Inge Wang in the lead, has carried out a project on involvement in lectures for several years. The project is based on quizzing students in the syllabus of a subject, to give lecturers an impression of the knowledge level of their students. The project started out as a prototype, then it was created as a framework, and now as Lecture Quiz 3.0, a game with flexible architecture. The goal of this master thesis is to create a finished system, a solution that involves students, focusing on lecture content while making it diverse enough to keep them active.

Lecture Quiz is created with education and teaching in mind. It can therefore be used in real lecture halls, where both lecturers and students can benefit. With some creativity, we assume it can be used for many other purposes, such as in specialist courses, general training, ceremonies, hospitals and more. In addition to this, it can also be used in informal gatherings, such as parties, weddings, small gatherings and more. Because of this, there are stakeholders connected to this project beyond the authors and Alf Inge Wang. These include both the students and teachers of NTNU and hopefully other teaching institutions in both Norway, and around the world. However, if NTNU with Alf Inge Wang and the authors of Lecture Quiz decides to go commercial; NTNU Technology Transfer AS (TTO) will also be represented as a stakeholder.

1.3 Problem Definition

In the previous semester, during our specialization project, we were handed the Lecture Quiz 2.0 framework. Our supervisor Alf Inge Wang, wanted us to look at ways to improve it. He was concerned about the playability of the framework, and wanted us to focus on creating a visual experience. We reused the Lecture Quiz 2.0 service orientated communication structure in Lecture Quiz 2.5, but created the rest of the components from scratch. In the specialization project, we concluded that using a web service for communication is not suitable for this type of solution.

Finding a better solution, that handles two-way communication is crucial for the success of the project. The new solution should be easy to extend, to include support for future components.

To create a user-friendly system, one focus should be to minimize the effort to get it up and running. This includes limiting the number of servers needed to be configured for running games.

It is important that games can be played on a large variation of client types, and that the presentation can be run on different platforms. To solve this, we have to consider the type of client and what language to use when creating the system.

As stated in the problem description, we want the game to be entertaining for students. We have to look at what makes a game entertaining and how we can apply it to Lecture Quiz. This should be verified by performing a user test. The test should also measure the usability of our system, as this is one of our main quality goals.

Lecturers are interested in collecting statistics from quiz games. We have to find a stable, yet simple way to store and display game statistics.

Creating quizzes is time consuming, and should be possible to do in advance of a lecture. It is important that this process feels uncomplicated for the lecturer and will require focus on usability. We have to find a way to allow lecturers to create, store and distribute quizzes. This should be done within a comfortable environment, to make the process as easy as possible for lecturers.

These problem definitions require us to reevaluate the technologies that are currently used.

1.4 Thesis Structure and Readers' Guide

The document is organized into seven parts, with fourteen chapters, and five appendices. Depending on your level of knowledge and your goal for reading the report, you can approach it from different angles.

A reader seeking knowledge about how to use the Lecture Quiz system should focus on the user guide, found in Appendix E. System administrators should also look into the deployment guide, found in Appendix C.

Lecture Quiz developers unfamiliar with the system should focus on Part IV and the development guide, found in Appendix D. The first and last chapter of Part IV can be omitted if you are just extending Lecture Quiz with a client or game mode.

Researchers can be interested in Part II and Part III, which contain the research design and prestudy of this project. They should read Chapter 11, which presents our lecture experiment, and Part V, which evaluate both the experiment and the overall project. Finally, we suggest reading our concluding remarks about the development of Lecture Quiz, as seen in Part VI.

The thesis is divided in the following way:

- Part I serves as an introduction to this thesis. It describes our motivation, the project context, the problem definition, and the structure of this thesis.
- Part II contains the definition of the research questions and the research method we have used. It also denotes the experiment method we have used.
- Part III is the result of the prestudy performed during the project. The first chapter outlines technological choices regarding Lecture Quiz 3.0. The second chapter gives an introduction to related work in the field, such as characteristics of educational games, research of games in lecture environments, state of the art, and the previous Lecture Quiz frameworks.
- Part IV focuses on our contribution, and describes the development phases of Lecture Quiz. In the first chapter, we define a set of requirements to define the functionality and the overall quality of the system. In the second chapter, we present the software architecture, and how it achieves the quality goals. The third chapter gives an overview of the most important design choices regarding Lecture Quiz. The fourth chapter provides insight into the implementation of Lecture Quiz. The last chapter contains information regarding the lecture experiment and how it was executed.
- Part V contains discussion of the results we obtained in Part IV and provides an evaluation of these results.
- Part VI includes information that brings the thesis to a close with concluding remarks, recommendations, and directions for further work.
- Part VII contains our appendices, and includes the acronym list, the experiment questionnaire, and the developer and users' guides.

Part II

Research Design

Chapter 2

Research Questions

The purpose of this chapter is to introduce the reader to our research questions and give a short description of them. Based on an analysis of our problem definition in Section 1.3, we pursue the following research questions:

RQ1 Which systems related to Lecture Quiz exist today, and what differentiate them from ours?

We will, in our prestudy, look at the previous work done in the field of educational games in lecture environments. These systems should enable increased student participation in lectures, and should preferably include a quiz element.

We will discuss what differentiate these systems from ours, and how we can use this information to improve the Lecture Quiz framework.

RQ2 How can we make Lecture Quiz easier for the end-user to install and use?

Our solution in the specialization project had many components, which needed to be installed and configured separately. We want the system to be available for a larger range of users, without having to install and configure several servers. This will also apply to the end-users, as we will try to make the client work without any custom software installed.

RQ3 How can using a quiz game in teaching environments affect the students?

We will perform a system test during a lecture held at NTNU, with at least 50 persons, to receive feedback. The data will be obtained through observation, video analysis and an evaluation form. The main focus will be:

a) Their ability to learn.

b) Their interest in learning.

c) Whether they have fun playing.

RQ4 How can we make Lecture Quiz 3.0 look, act and feel like a game?

One of our main goals is to make it feel like one is playing a game and not just answering a questionnaire. There are certain elements that need consideration when creating games. Which elements can we use to make Lecture Quiz a fun and interesting experience for the end-user.

Chapter 3

Research Method

There is a fair amount of research being conducted in software engineering, i.e. people are building technologies, methods, tools, etc. However, unlike in other disciplines, there has been very little research in the development of models for the various components in the discipline. Victor R. Basili has done much research in this field and has produced a set of experimental methodologies, which categorize research methods that can be used in software engineering [7]. He lists several paradigms and how to choose the most suitable research method for a software engineering project. We look at these paradigms from a software engineering perspective, as shown below:

1. **The scientific method:** In this approach, the software process is investigated by using analysis, observation and evaluation. The procedure is as follows; observe the world, propose a model or a theory of behavior, measure and analyze, validate hypotheses of the model or theory, and if possible repeat [7].

   This inductive paradigm might best be used when trying to understand the software process, system, people or environment. It attempts to extract a model from the world which tries to explain the underlying phenomena, and evaluate whether the model is truly representative of the phenomenon being observed. An example of this can be to understand the way software are developed. If it is possible to abstract this process model, or if there can be built a tool to automate the process. Basili lists two variations of this inductive approach:

   1.1. **The engineering method:** In this approach, the domain of the problem is examined to observe existing solutions, for then to propose better solutions. The procedure is as follows; observe existing solutions, propose better solutions, build or develop, measure and analyze, and repeat the process until no more improvements seem possible [7].

   This version of the paradigm can be described as an evolutionary improvement oriented approach. It assumes that one already has models to compare with, in order to improve the thing being studied. An example is the study of improvements to methods being used in the software development process. It can also be a demonstration to show that some tool performs better than its predecessors, relative to certain characteristics. A crucial part of this method is the need for careful analysis and measurement.

1.2. **The empirical method:** This approach is based on a proposed model of the problem domain, statistical/qualitative methods are developed. The procedure is as follows; propose a model, develop statistical/qualitative methods, apply to case studies, measure and analyze, validate the model and repeat the procedure [7].

   This version of the paradigm can be described as a revolutionary improvement oriented approach. It begins by proposing a new model, not necessarily based upon an existing model. Further it attempts to study the effects of the process or system suggested by the new model. An example could be the proposal of a new method or tool used to perform software development in a new way. As with the engineering method, measurement and analysis are crucial to the success of this method. Just proposing a method or a tool is not enough, as there must be some way of validating that this is an advance over current models or tools.

   Experimentation must be guided and there must be a rational for collecting data. Experiments must be designed to get information useful for the building of a suitable model of the systems under study. Issues related to these methods include; the types of experimental design appropriate for different environments, whether the experiment is exploratory or confirmatory, the validation of the data, the cost of the experiment, the problems of reproducibility, etc.

Basili also describes a more deductive approach, the mathematical method:

2. **The mathematical method:** In this approach a theory is proposed (a set of axioms) and further developed. The procedure is as follows; propose a formal theory or set of axioms, develop a theory, derive results and if possible compare with empirical observations [7].

   This paradigm is a deductive analytical model, which does not require an experimental design in the statistical sense. It provides an analytic framework for developing models and understanding their boundaries based upon manipulation of the model itself. An example could be treatment of programs as mathematical objects and their analysis of the mathematical object or its relationship to the program. This would satisfy the paradigm.

In our specialization project, we chose the engineering method as our research approach. The reasoning behind this was that we wanted to develop a flexible framework as the basis for our master thesis. We used most of our time developing it, and did not have time to focus on testing a system that did not have the quality it has now.

In this master thesis, we continue the development of the Lecture Quiz 2.5 framework. We will examine and observe the current solution, purpose better solutions, and then analyze and measure these new results. Based on the propositions, we continue to follow the engineering method during this thesis.

This time, we choose to put aside time to test the system in an experiment. In this experiment, we will try to get information on the usability of the system, and if the system can be used to increase student learning. Based on this, in addition to the engineering method, we will also follow the empirical method.

The chosen methods, are both categorized as scientific methods, which will be the used paradigm when we develop Lecture Quiz 3.0. The empirical method can be seen more as an approach than an experimental method. Because of this, we have to choose an experimental method that suits our needs in developing Lecture Quiz 3.0. In the following section, we will describe our experimental method and how we will acquire the necessary empirical data.

3.1 Experiment Method

Collecting data about the application of Lecture Quiz is considered necessary to evaluate its usability, both technical and educational. We wanted to gather information about:

- Technical challenges, like:
  - How the Player Client works on different client device types.
  - How the underlying network infrastructure handles Lecture Quiz' network traffic.
- How Lecture Quiz can affect student learning.
- How user-friendly students find Lecture Quiz.

Testing Lecture Quiz in a real lecture could give such information. The testing itself would give information to us about potential technical drawbacks with the system. A survey could also give information on such drawbacks, in addition to the students' evaluation of e.g. user-friendliness, and experienced learning effect.

The questionnaire used in the survey should therefore include questions regarding what type of both wired and wireless network types works with our system, the player's device, operating system and web browser. This gives us some numbers on how many device types that work with our solution. All this can help us improve the overall compatibility of the system.

It is also important for us that students find Lecture Quiz educational. Without the educational aspect of Lecture Quiz, it would just be a regular quiz game. Learning can be the product of not only the quiz content, but also increased student attention level. Videotaping the lecture could give some data on the likely level of attention.

The typical survey contains closed-ended questions [6]. We do not have enough time to do thorough, in depth interviews with the participants, and do a following analysis, but could use some statements formulated by the students themselves. Thus we should have some open-ended questions with a couple of lines for the students to write what they want, and not only"what we ask them to". Closed-ended questions are statements that the students rank by their subjective value.

One of our main concerns is how user-friendly both students and lecturers find the system. It is important for the continued use of Lecture Quiz, that the students do not find it too cumbersome. This will decrease the participation and would work against the purpose of Lecture Quiz.

The usability of a system, as defined by the standard International Organization for Standardization (ISO) 9241 Part 11, can be measured only by taking into account the context of use of the system [54]. This can further be explained as; who is using the system, what are they using it for, and the environment in which they are using it [54]. This standard measure usability through three key aspects, namely:

- effectiveness (can users successfully achieve their objectives?)
- efficiency (how much effort and resource are spent in achieving those objectives?)
- satisfaction (was the experience satisfactory?)

The System Usability Scale (SUS) endorses this model by providing a high-level subjective view of usability. To measure the usability of our system, we decided to use the System Usability Scale.

We arrange for testing Lecture Quiz during the summary lecture in the Software Architecture course, which our supervisor Alf Inge Wang is holding. Number of participants is of relevance, and at least 50 students should attend this lecture.

3.1.1 System Usability Scale

SUS is a scale used to measure the usability of software based on a questionnaire of 10 questions [36]. The scale goes from 0 to 100, where 100 is the best score. All ten questions are publicly available, and SUS has proven to give similar results as other more complex and time consuming usability tests.

Each question in a SUS test may be rated by the user from 1 to 5, where 1 is "strongly disagree", and 5 is "strongly agree". To calculate the final SUS score, every even question gets a score of 5 minus the average rating for that question. While the odd questions score is calculated by subtracting 1 from the average rating of that question. The overall SUS score will then be the sum of all the scores of the individual questions multiplied by 2.5. The full equation can be seen in Equation 3.1.

$$P_{SUS} = 2.5 \times \sum_{i=1}^{N/2} (S_{2i-1} - 1) + (5 - S_{2i}) \tag{3.1}$$

In this equation, N means the number of questions, which in our case is 10. S means the average score a question has gotten. The result and calculation of our SUS score is presented in Section 11.1.

In research, an experiment, as a scientific term, can only be called an experiment under certain conditions [66]. All in all, our experiment cannot be called a real experiment as such, yet we choose to call it "experiment". What we will actually be doing, is mainly testing our system, and asking participants how they experience playing Lecture Quiz, and how user-friendly they find it. The experiment is described in Chapter 10.

Part III

Prestudy

Chapter 4

Previous Work

In this chapter, we look at some of the previous work done in the field of educational gaming, and gaming in lecture environments. In the state of the art section, we introduce the reader to related quiz/response systems, and give a feature summary of these. The last three sections give a short summary of the previous Lecture Quiz versions 1.0 and 2.0, and our specialization project, Lecture Quiz 2.5.

4.1 Characteristics of Educational Games

In Lecture Quiz 1.0, eight important characteristics of a good educational game are presented [50]. The following list of characteristics is meant as a reference for people designing educational games. The architects of Lecture Quiz 1.0 argue that excluding one of the characteristics may not mean that the game will be unpopular or unsuccessful, but including them in the game concept may make it better.

- **Variable instructional control** - How the level of difficulty is adjustable, or adjusts to the skills of the player.
- **Presence of instructional support** - The possibility to give the player hints when he or she is incapable of solving a task.
- **Necessary external support** - The need for use of external support.
- **Inviting screen design** - The feeling of playing a game, and not operating a program.
- **Practice strategy** - The possibility to practice the game without affecting the user's score or status.
- **Sound instructional principles** - How well the user is taught how to use and play the game.

- **Concept credibility** - Abstracting the theory or skills to maintain integrity of the instruction.
- **Inspiring game concept** - Making the game inspiring and fun.

The architects of Lecture Quiz 1.0 also present a taxonomy of educational games [50]. Using three criteria, listed below, they group educational games in a set of game genres.

- **Player interaction** - Is it possible for several players to interact with the system in some way?
- **Fantasy and skills interaction** - Is the fantasy of the game extrinsic[1] or intrinsic[2]?
- **Game concept type** - In what genre of computer games does the game concept belong?

Based on this taxonomy they present a model, shown in Figure 4.1, that shows how each game genre provides theoretical knowledge. We may classify the Lecture Quiz game as a group/multiplayer quiz, and according to this model it is an effective and simple genre. We will try to achieve as many of the presented characteristics as possible during the design and implementation of our system, and thus hopefully make a good system for educational purposes.

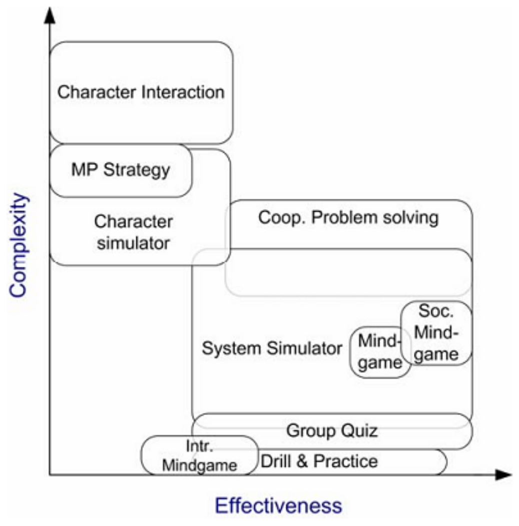


Figure 4.1: Genre relevance for theoretical knowledge [50].

[1]Extrinsic fantasies can be seen as independent of the application of skills in the game [50].
[2]Intrinsic fantasies are inherent and essential to the game concept [50].

4.2 Previous Research

In this section, we present and discuss research from the field of serious gaming, focusing on video game-based learning. Lecture Quiz can be seen as an educational game and a learning tool which tries to engage students and increase their participation and interaction. The field is still in its embryonic stages, but it is upcoming, and we will present the research we find most relevant to the Lecture Quiz game. Per spring 2011, we have found little or no research directly related to Lecture Quiz.

Educational video games are a promising medium. Merrilea Mayo has argued that the video game format has many advantages over the old-fashioned school lecture [48]. She links five characteristics with better learning outcomes:

- Games can break down complex tasks, guiding players through a series of small steps.
- Learning can control their navigation of the games.
- Games can give learners immediate and continuous feedback.
- Games can be adapted to the individual pace of the learner.
- Game-based tasks may require students to formulate hypotheses and experiment.

These characteristics are based on single player games, but some of them are applicable in multiplayer games.

Blakely et al. investigates the use of games to support classroom learning in the health sciences, through a systematic review of educational gaming. [10]. The aim of the review is to investigate the use of games to support classroom learning in healthcare sciences. They address three questions:

- How effective is educational gaming as a teaching tool for health science students in comparison to the traditional didactic style?
- Does educational gaming enhance long-term retention of knowledge or skills by the student?
- Is the method of educational gaming a more enjoyable teaching strategy from the student perspective?

These are relevant questions for our project as well. The first and third questions are related to our research questions RQ3 and RQ4, respectively, as seen in Chapter 2. Several of our success criteria in Section 10.3 and questionnaire statements in Appendix B are also related to these questions.

Blakely et al. conducted a search based on several key words such as “educational games” and “teaching strategies”. After using inclusion and exclusion, sixteen papers reporting empirical studies or reviews that involved comparison of gaming with didactic methods were included. [10]. They found that the limited research available at that time indicated that, while both traditional didactic methods and gaming have been successful in increasing student knowledge, neither of the methods are clearly more helpful to students. The use of games generally enhances student enjoyment and *may* improve long-term retention of information. Blakely et al. conclude that, while the use of games can be viewed as a viable teaching strategy, care should be exercised in the use of specific games that have not been assessed objectively. They suggest that further research on the use of gaming is needed to enable educators to make gaming techniques appropriate for the benefit of students, and ultimately patients.

Akl et al. call an educational game an instructional method requiring the learner to participate in a competitive activity with preset rules [2]. In their review, they investigate the effect of educational games on medical students’ satisfaction, knowledge, skills, attitude, and behavior. They find that the research available at that time suggested but did not confirm a positive effect of the games on medical students’ knowledge. They conclude that the available evidence to date neither confirms nor refute the utility of educational games as an effective teaching strategy for medical students. Akl et al. suggest that there is a need for additional and better-designed studies to assess the effectiveness of the relevant games.

Schuh et al. tries to find evidence for generalizability of a game and team oriented educational intervention in clinical neurophysiology in a neurology residency program [61]. They study prospective educational intervention in a single neurology residency program and compare with a historical control. The effectiveness of a team-oriented educational intervention in clinical neurophysiology, with gaming and oral quizzing, gives evidence for generalizability in improving subset Residency In-service Training Exam (RITE) performance compared with faculty prepared didactics. Their intervention consisted of weekly presentations, followed by a game show-type oral quiz, which was team-based. Examination score is 63.6 ś 4.12 for the intervention group and 49.4 ś 2.35 for the control (P = 0.002).

Thus, it seems like the field lacks a satisfactory amount of well-designed studies to conclude the effectiveness of gaming to learn in lectures, while some well-designed studies show generalizable evidence. On the other hand, gaming does not seem to *reduce* learning in lectures.

Knowing that many students expect immediate results, prefer active learning as opposed to passive learning [14], and in addition enjoy working both independently or collaboratively in groups [52], we hope that some aspects of multiplayer lecture gaming could at least increase enjoyment of, and thus initial engagement in learning.

Our hope is that software for lecture gaming, such as Lecture Quiz, can increase participation, cooperation, and eventually learning in students. Future research can reveal evidence for generalizability of the effectiveness of Lecture Quiz.

4.3 State of the Art

In our specialization project, we did a thorough investigation of state of the art regarding related solutions. In this section, we use this information and extend it with the highlights from the last six months. We present existing solutions of educational software that increase classroom interaction, with special emphasis on quiz services. These solutions have been of great motivation to us when developing both Lecture Quiz 2.5, and the final Lecture Quiz game.

4.3.1 Related Quiz/Response Systems

In this section, we present the reader to quiz/response systems related to the Lecture Quiz game. These systems should be usable in lectures and have multi-student support. Preferably, the systems should use standard student hardware, like smartphones and notebooks, as well as the use of already existing network infrastructure. In addition to this, we will look at systems that use 2D/3D graphics and sound.

Buzz!: The Schools Quiz is an educational game based on the Buzz series for Playstation 2, released in early 2008. It is a commitment by the Key Stage 2[3] teachers, founded by the United Kingdom Government, and developed by Relentless Software [11]. The game is featuring over 5000 questions from the National Curriculum, and supports up to 8 players. Its gameplay follows the standard format of the Buzz series, but it is made easier with respect to the younger players. Both the video game cover, and controllers can be seen in Figure 4.2.

ClassInHand is software developed by the Research and Development team in Information Systems at Wake Forest University [59]. It turns a Pocket PC equipped with a wireless card into a Web server, a presentation controller, and a quizzing and feedback device for a classroom instructor. The last version, shown in Figure 4.3 was made in 2003, and is only compatible with Pocket PC 2003 and Windows Mobile 5.

They claim the following software features:

- An instant-on, easy-to-manage, portable Web server, that is completely under your control.

[3]Key Stage 2 is the legal term for the four years of schooling in the United Kingdom when the pupils are between 7 and 11 of age.

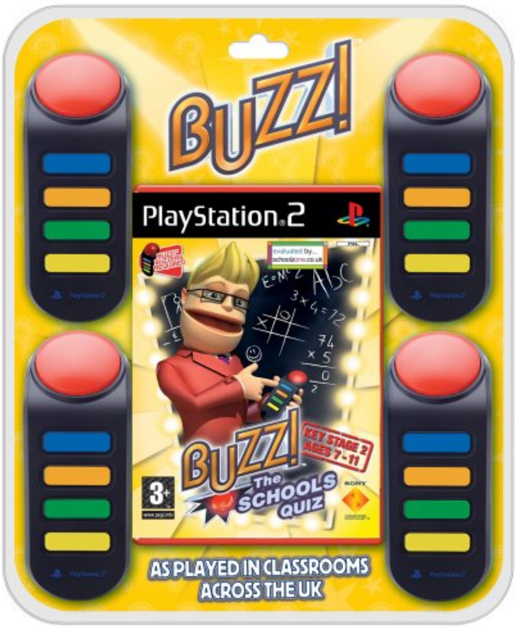
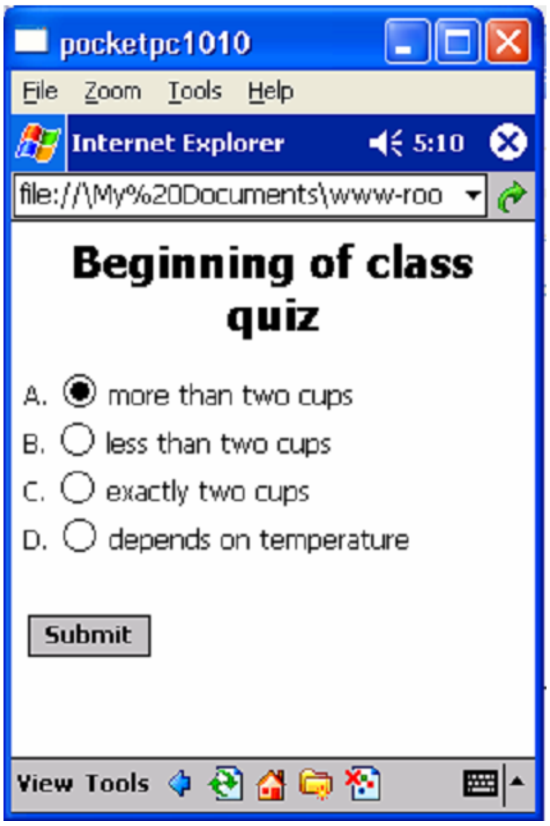


Figure 4.2: Buzz!: The Schools Quiz cover and controllers [11].



Beginning of class quiz

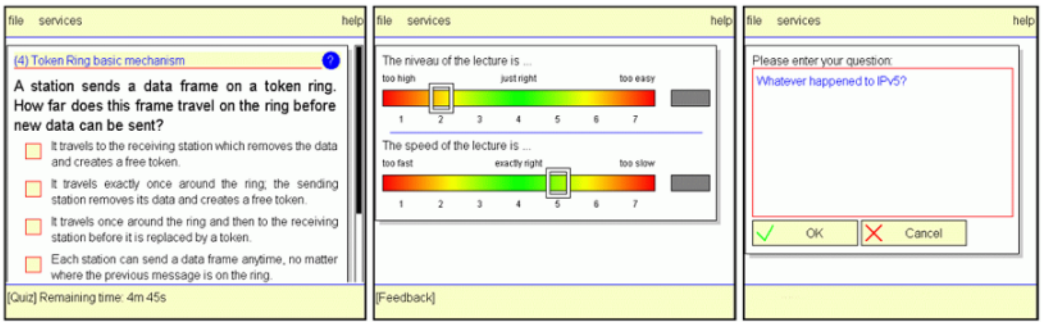
A. ● more than two cups
B. ○ less than two cups
C. ○ exactly two cups
D. ○ depends on temperature

Submit

Figure 4.3: Typical ClassInHand quiz, shown on a Pocket PC [59].

- Presentation capabilities, that enable you to use the Pocket PC as a remote control to navigate to a PowerPoint presentation on your desktop or laptop computer, start and manage the presentation, and see slide text and speaker notes on your Pocket PC.
- A quizzing feature, that enables you to present a question with up to four answers, and see the distribution of student responses immediately on your Pocket PC. You can also choose to display the results to the class.
- A text feedback mechanism, that enables your students or audience to submit questions or comments that appear immediately on your Pocket PC.
- A feedback meter, that enables students to submit numeric responses (range: -10 to 10) according to your directions. These submissions appear as a continuous curve on your Pocket PC, and are useful for quick assessments.

Wireless Interactive Learning - Mannheim (WIL-MA) is software developed at the University of Mannheim, Germany. It enables bi-directional synchronous communication between the students' and the lecturer's mobile devices, via an access point [24]. The client, shown in Figure 4.4, is implemented with Java Micro Edition (Java ME) version 1.1.3 (last update was in 2004), and because of this, most mobile devices are supported. The deployment view of their client-server architecture can be seen in Figure 4.5.



A station sends a data frame on a token ring. How far does this frame travel on the ring before new data can be sent?

- It travels to the receiving station which removes the data and creates a free token.
- It travels exactly once around the ring; the sending station removes its data and creates a free token.
- It travels once around the ring and then to the receiving station before it is replaced by a token.
- Each station can send a data frame anytime, no matter where the previous message is on the ring.

[Quiz] Remaining time: 4m 45s

Whatever happened to IPv5?

Figure 4.4: Students' client showing a quiz, feedback and call-in [38].

The current available services are:

- **Call-In**: students can provide questions to the teacher or moderator at any time.
- **Quiz**: short quizzes can be displayed on a presentation unit, and each student can answer anonymously from their device.
- **Feedback** can be sent to the lecturer during a lecture.
- short **Messages** can be sent between individual users.

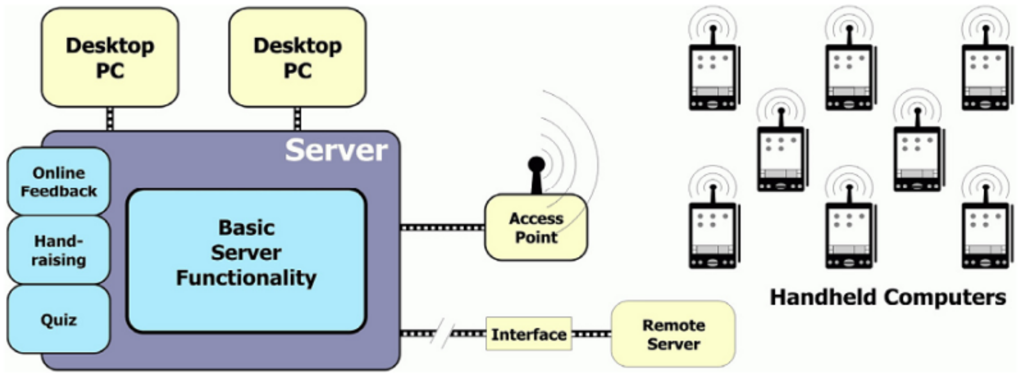


Figure 4.5: Overview of the WIL/MA system [38].

- a **Chat-room** can be used by many users, e.g., to solve bigger problems.

Classroom Presenter is a Tablet PC based interaction system that supports the sharing of digital ink on slides between instructors and students [23]. It enables the lecturer and students to collaborate by writing and editing on the same slide set, as shown in Figure 4.6. The slides can be created via an internal program called DeckBuilder, or they can be loaded from a Microsoft PowerPoint file directly. The source code is written in .Net, and is made publicly available. This is the only project which has no quiz service, but it has been continuously updated until 2008 and is well documented. The last version, 3.1, was released in the fall of 2008.

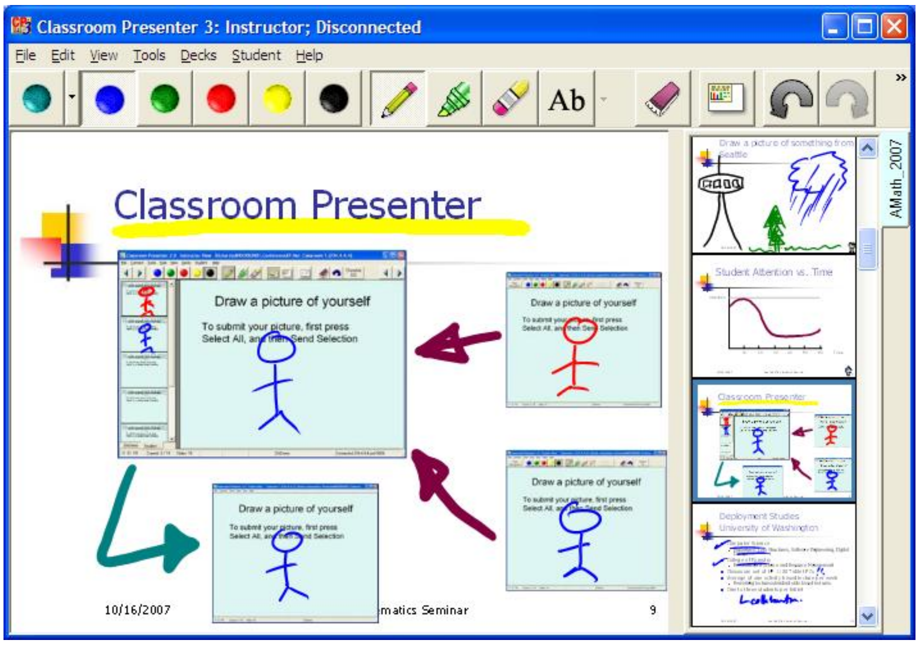

Classroom Presenter 3: Instructor; Disconnected
File Edit View Tools Decks Student Help
Ab
Classroom Presenter
Draw a picture of yourself
To submit your picture, first press Select All, and then Send Selection
10/16/2007
matics Seminar
9
AMath_2007
Draw a picture of something from Seattle
Student Attention vs. Time
Deployment Studies University of Washington

Figure 4.6: Screenshot from Classroom Presenter 3 [5].

TVRemote Framework is a student interaction tool which supports a set of user devices, shown in Figure 4.7, for feedback and interaction [5]. It was developed at the Darmstadt University of Technology, Germany, and was meant to make student-teacher interaction in lectures more easy, and as fast as possible. The supported services is feedback, polling of student opinion, and question submission. It is also possible for the teacher to broadcast notes, links, and multiple choice questions. The network infrastructure supported by the server is wireless local area network (WLAN), Ethernet, general packet radio service (GPRS)/high-speed circuit-switched data (HSCSD), and Bluetooth. The project seems to have been externally discontinued since 2005, and their official web site does no longer exist.

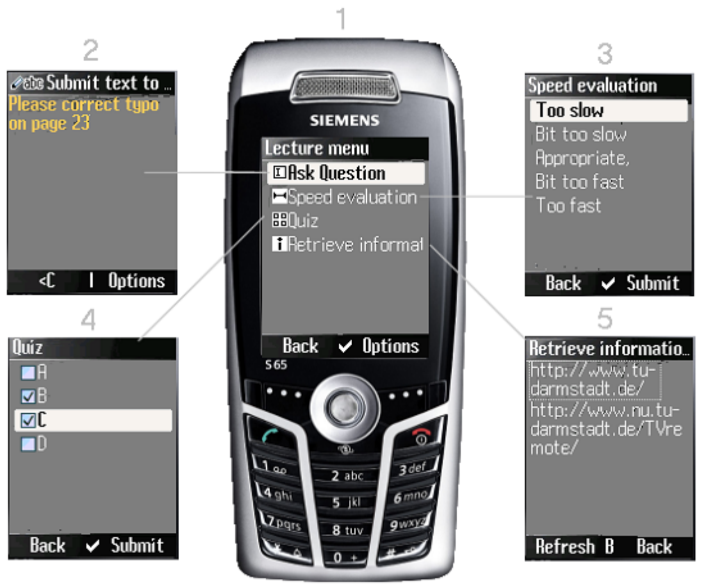


Figure 4.7: TVRemote prototype running on a Siemens S65 mobile phone [5].

JustVote (former EzClickPro) is a simple, cost effective, interactive, handheld voting system [45]. It is a commercial application developed for teaching in primary school, secondary school and higher education. Its developer company is Avrio Ideas Ltd. It utilizes the 2.4 GHz, industrial, scientific and medical (ISM) radio band, and supports 50 meter range for data transmission to the custom handheld controls, as shown in Figure 4.8. The teacher uses a personal computer (PC) to interact with the presentation client, as shown in Figure 4.9, which is connected to a projector or similar devices. They have an extensive feature list [46]. One of their drawbacks is multi-platform support, as they only support the Microsoft Windows platform. An excerpt of the features can be seen below:

- Supports up to 1000 connected handheld controls
- Add images, sounds and video
- Extensive range of statistical data

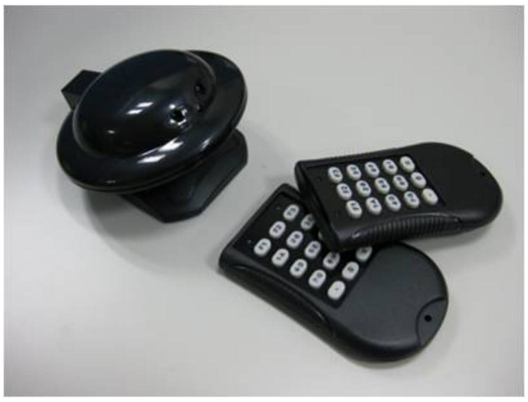
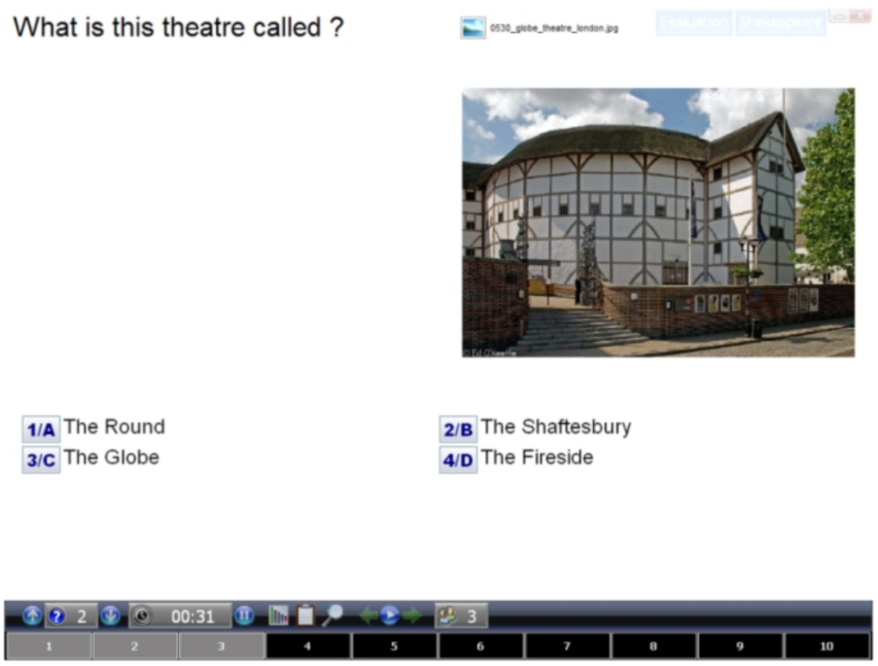


Figure 4.8: JustVote sensor and remote controls [46].



What is this theatre called ?

1/A The Round
2/B The Shaftesbury
3/C The Globe
4/D The Fireside

Figure 4.9: JustVote question presentation from a quiz [46].

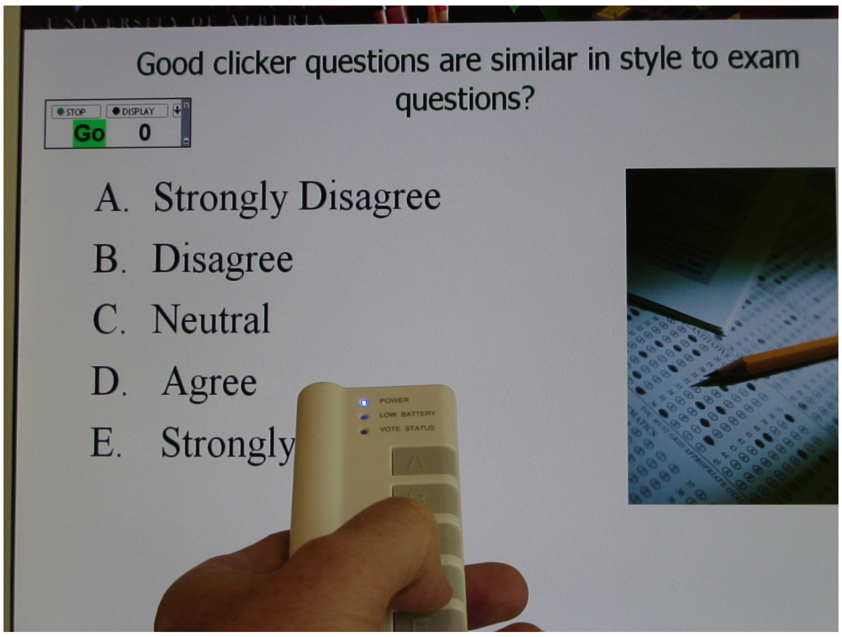
- Multiple question layout templates
- Monitors the progression of the whole class, groups and individuals
- Reviews past sessions
- Easy loading of pre-prepared material
- Simple drag and drop between questions
- Ability to print and export data
- Over 1,500 questions provided, and over 15,000 more can be purchased

They support the following game modes [46]:

- Standard (questions and response)
- Fastest Response (fastest fingers first)
- Buzz-in (verbal response)
- Elimination (knock out)
- Verbal Prompt (leading to handset response)
- Pick Out (group or individual to verbally answer)
- Ask Question (pupil to intervene)
- PowerPoint mode

i>clicker is an audience response system which allows students to get instantaneous feedback, and answer questions posted by their instructors [35]. Students can use a portable clicker, as shown in Figure 4.10, to vote by "clicking" on the appropriate button for his/her choice. An instructor uses a receiver, that collects votes sent by students' clickers, which works over WLAN. The receiver is connected via Universal Serial Bus (USB). Instructors can present questions, enable polling and access statistics both ad hoc and later. The instructor software has support for both Microsoft Windows and Mac OS X. The instructor's question screen can be seen in Figure 4.10.

The authors behind i>clicker, have also created web>clicker, a client for smartphones and other web browser enabled devices. Web>clicker is a browser-based voting tool that combine i>clicker's simplicity, with the flexibility of laptops and handheld devices [34]. It works on any standard web browser, including Internet Explorer, Firefox, Chrome, and Safari, including Safari for iPhone and iPod Touch. Web>clicker can be used alongside the already mentioned i>clicker. An answer screen on a smartphone, is shown in Figure 4.11.



Good clicker questions are similar in style to exam questions?

A. Strongly Disagree

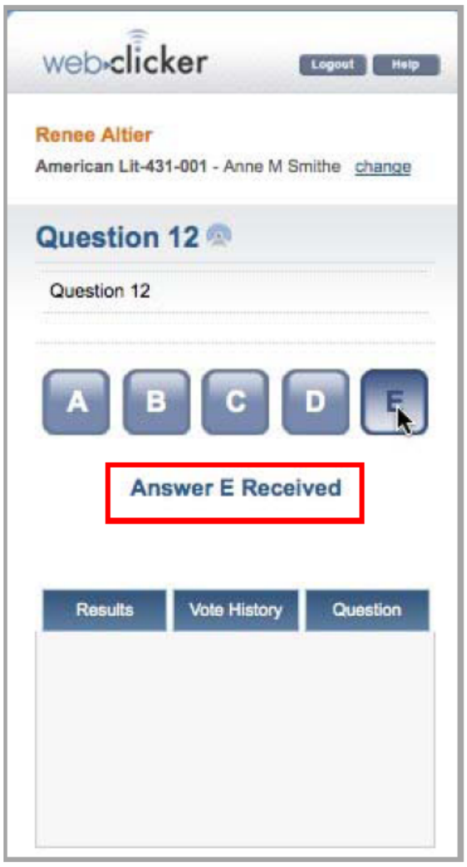
B. Disagree

C. Neutral

D. Agree

E. Strongly

Figure 4.10: i>clicker software.



Renee Altier

American Lit-431-001 - Anne M Smithe change

Question 12

Question 12

Answer E Received

Figure 4.11: i>clicker web>clicker [34].

4.3.2 Feature Summary

In this section, we summarize the features of the related quiz/response systems presented in Section 4.3.1. Table 4.1 summarizes these feature. The three most relevant features are: quiz service, animated graphics, and no custom hardware needs, as Lecture Quiz is built around these. We also list three features, which are relevant for interactive lectures on a general level, but not very relevant to our framework.

Feature	**Buzz**	**ClassInHand**	**WIL/MA**	**Cl.Pres**	**TVRemote**	**JustVote**	**i>clicker**
Quiz service	x	x	x	x		x	x
Animated graphics	x						
No custom hardware					x		x
Student comments		x	x		x		
Info broadcast		x	x	x	x		x
Public feedback	x			x	(x)	x	x

Table 4.1: Feature comparison table.

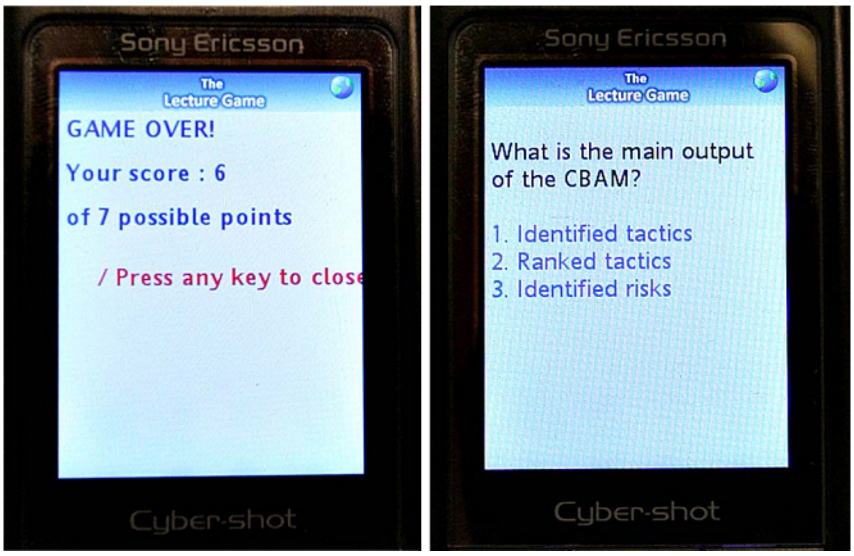
Based on the findings in Table 4.1, we can summarize the following. Neither TVRemote nor web>clicker, have custom hardware needs, and only Buzz has animated graphics. JustVote and the i>clicker software support plain graphical customization, but the user interface looks more like a plain presentation than a 2D or 3D game. The most common feature is the quiz service, but it is mainly used by the lecturer to monitor the knowledge of the students, and not for further learning or competition. Buzz focuses on gameplay and the graphical, and we can learn much about gaming experience from them. JustVote and i>clicker, have many features, and JustVote also introduced the term game modes, which is related to our game modes. We can learn from the web>clicker software and the TVRemote framework, as they require no custom hardware needed. This feature should also be a requirement for the Lecture Quiz framework.

4.4 Lecture Quiz 1.0

The first version of the Lecture Quiz (LQ) framework was created during the master project of Ole Kristian Mørch-Storstein and Terje Øfsdahl in 2007 [50]. They focused on the impact of using games in lectures, with the main purpose to develop a prototype quiz game, where the students can answer questions using their laptops or mobile phones.

The developed prototype consists of a main server, a teacher client, and student clients. The architects built a customized communication framework on top of Transmission Control Protocol (TCP), and performance was not an issue for them.

Their student client was developed using Java 2 Platform, Micro Edition (J2ME) (now Java Micro Edition (JME)), and was simple and user friendly. To begin a session, each student has to download the software to their mobile phone using WLAN, Bluetooth or a mobile network service, such as (GPRS/Enhanced Data rates for GSM Evolution (EDGE)/3rd generation mobile telecommunications (3G)). When the download completes, the software has to be installed before the students are ready to participate. Their conclusion is that this is a cumbersome process [50]. They suggested to improve it by using the World Wide Web to communicate with the clients in their further work section. Two screenshots of the Lecture Quiz 1.0 student client are shown in Figure 4.12.

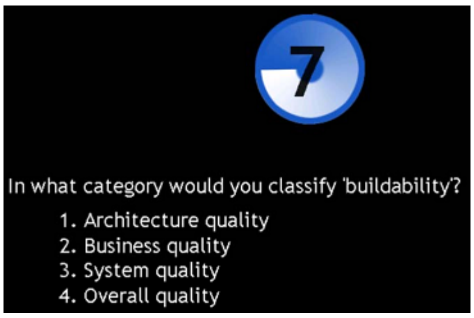


GAME OVER!
Your score : 6
of 7 possible points
/ Press any key to close

What is the main output of the CBAM?
1. Identified tactics
2. Ranked tactics
3. Identified risks

Figure 4.12: LQ 1.0 - Student Client [50].

Their teacher client was implemented in Java, and uses the Java OpenGL (JOGL) library to display graphics on a projector or similar device. Typical gameplay screenshots are shown in Figure 4.13, Figure 4.14 and Figure 4.15.



In what category would you classify 'buildability'?
1. Architecture quality
2. Business quality
3. System quality
4. Overall quality

Figure 4.13: LQ 1.0 - Teacher Client question screen [50].

Their prototype consists of two game modes:

1. **plain game mode** where all students answer all the questions in a quiz. Each question has a unique time limit, and the students have to answer

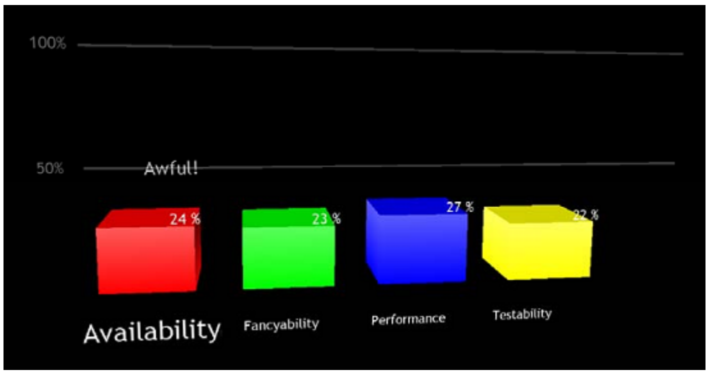


Figure 4.14: LQ 1.0 - Teacher Client measure up mode [50].

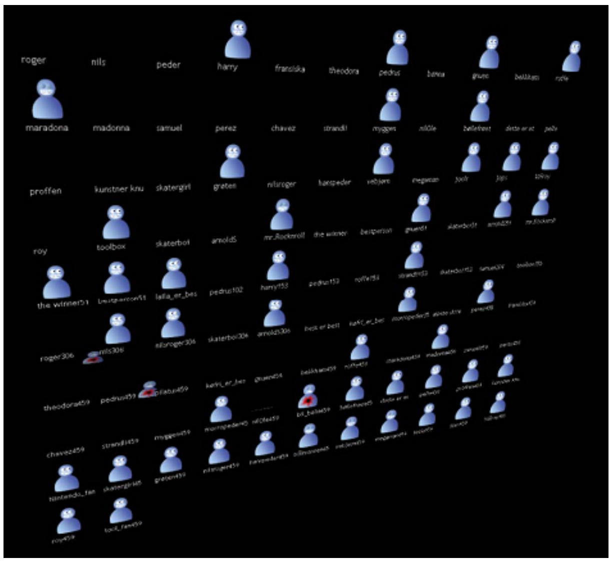


Figure 4.15: LQ 1.0 - Teacher Client elimination mode [50].

within that time to increase their score. After each question, a screen with statistics is displayed, providing the answer distribution. At the end of a quiz, their teacher client displays a list of the students having the most correct answers.

2. The **last man standing game mode** is similar to the plain game mode, but if a student answers incorrectly he or she is removed from the game. The game continues until only one student remains, and is the winner.

One of the main drawbacks of their prototype is that it is more or less hardcoded and lacks a good architecture. This makes it hard to extend, modify and maintain. It also lacks good documentation on how to add new quizzes and questions, and database entries have to be manually edited. The time spent on downloading and installing the software on the student devices also makes it less interesting for regular use in lectures [50].

4.5 Lecture Quiz 2.0

The second version of the Lecture Quiz (LQ) framework was created in the master thesis of Erling A. Børresen and Knut A. Tidemann in 2010 [12]. While the first version is more a proof of concept than a framework, the second version is more of a framework than an actual game.

In our specialization project, we tried to deploy this system on a GlassFish application server, but realized that the framework was incompatible with GlassFish. The framework is intended to run on any Java Enterprise Edition (Java EE) application server, but it was only tested on the Apache Tomcat application server.

The framework consists, like LQ 1.0, of a main server, a teacher client and student clients. In contrast to LQ 1.0, the communication framework is based on a web service (Web Services Description Language (WSDL) and SOAP), making performance an issue [12].

Implementing a web service as the only communication framework, was the main goal of the architects behind LQ 2.0 [12]. Using a web service for all the communication between the different components in the system leads to excessive one-way communication and polling to keep all components updated. This can lead to performance issues, but with a good implementation, it can work to some degree for a slow paced game like Lecture Quiz. The framework implements game modes on the server, which in part complicates the customization of appearance onto the teacher client. Database transactions are handled by the server, but the database has insufficient tables, and lacks optimal usage of foreign tables [25]. The server controls overall access, but lacks decent handling of user authentication.

The student client was developed using Google Web Toolkit (GWT). Its graphical user interface (GUI), as shown in Figure 4.16, is simple and feels not as user friendly as LQ 1.0 [25]. It uses GWT to display fancy error messages and feels responsive on an isolated level. Since it relies on a web service it can feel unresponsive related to the teacher client. The authentication via the Student Client, shown in Figure 4.17, requires a quiz code, which seems unnecessarily complicated.

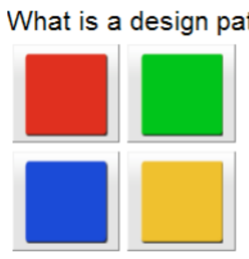
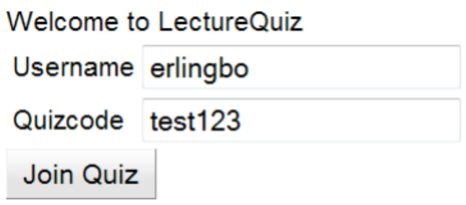


Figure 4.16: LQ 2.0 - Student Client answer question user interface [12].

The teacher client is simple in design, not very user friendly and lacks the impression of a game [25]. It was implemented in Java, and uses the JOGL library to

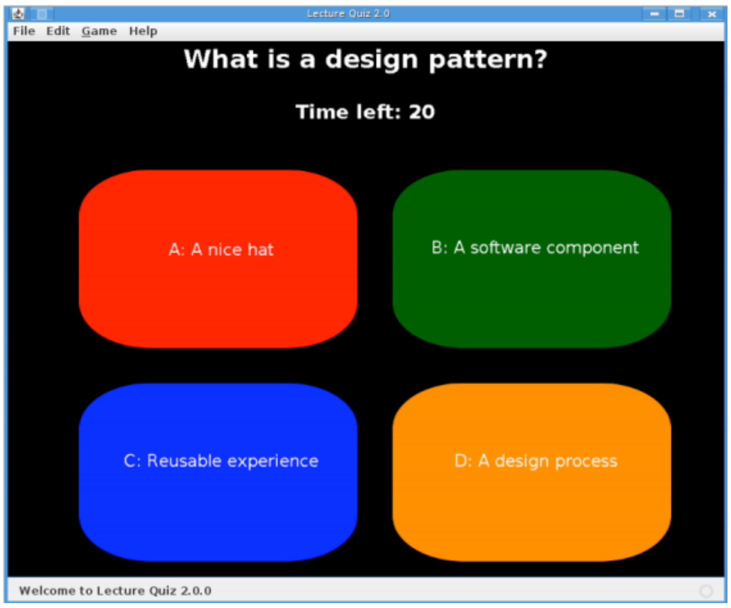

Welcome to LectureQuiz

Username: erlingbo

Quizcode: test123

Figure 4.17: LQ 2.0 - Student Client login user interface [12].

display graphics on a projector or similar device. The graphics are basic, as shown in Figure 4.18 and Figure 4.19. The creation of quizzes has been embedded into the teacher client using Java Swing.



What is a design pattern?

Time left: 20

A: A nice hat

B: A software component

C: Reusable experience

D: A design process

Welcome to Lecture Quiz 2.0.0

Figure 4.18: LQ 2.0 - Teacher Client question screen [12].

The authors behind Lecture Quiz 2.0 performed an experiment in a lecture to get feedback from students on how they perceived the system. The experiment was performed by running a quiz and the students participating answered a questionnaire afterwards. The students perceived the system as a good concept and enjoyed an alternative, more fun way of learning [12]. Based on the SUS score, they found it easier to use than LQ 1.0, and more than half would like to use this system as a recurring element in lectures.

4.6 Lecture Quiz 2.5

The Lecture Quiz (LQ) 2.5 framework was created in our specialization project, in the fall of 2010 [25]. The version 2.5 was chosen over 3.0 as the framework was supposed to extend the Lecture Quiz 2.0 framework. During the architectural and

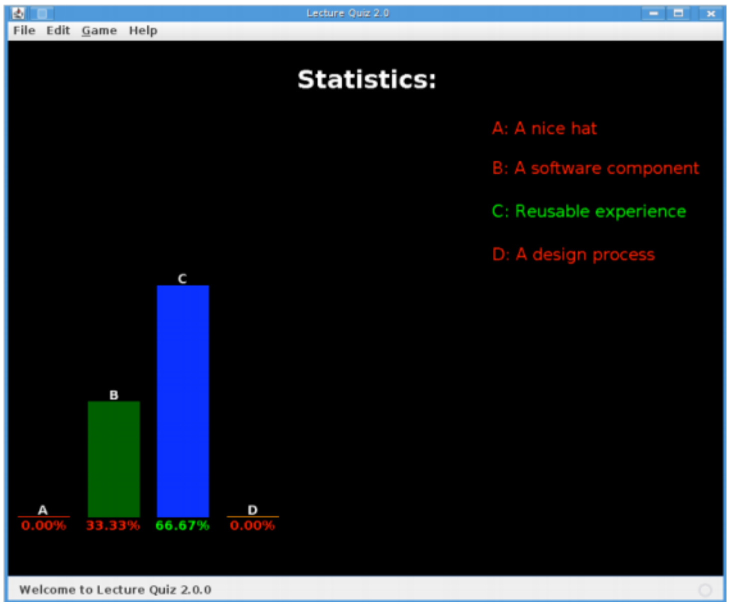


Figure 4.19: LQ 2.0 - Teacher Client question statistics screen [12].

implementation phase of Lecture Quiz 2.5, the original framework was gradually excluded. The reason for this was that the Lecture Quiz 2.0 framework seemed harder to extend than to create from scratch [25].

The prestudy phase of the specialization project was thorough, and several technologies, and related solutions were investigated. Based on these findings and the overall evaluation of LQ 2.0, we concluded that there was room for improvements [25]. New technologies and design choices were made, which led to the creation of functional and non-functional requirements.

These requirements led to the creation of a new and improved architecture which promoted usability and modifiability. The teacher client concept of LQ 2.0 was separated into two new projects; Quiz Editor and Presentation Client [25]. This choice was based on that the lecturer usually manage quizzes and view statistics in the comfort of his own office. Because of this, the Presentation Client in LQ 2.5 was created from scratch without any elements from LQ 2.0. It was implemented in Java, and used the JOGL library to display graphics on a projector or similar device, as shown in Figure 4.20. The Quiz Editor now served as a separate component which had the support of quiz and statistics management.

The game server of LQ 2.0 was reimplemented, with the exception of the Web service and Java DataBase Connectivity (JDBC) logic, which was re-used and extended [25]. It functions as a shared point of access, like in LQ 2.0. This server offers services through the use of WSDL and SOAP. The game mode logic was moved to the Presentation Client which was the only application depending on it. This, and similar improvements, was done to make the clients less dependent on the application server. The game server's logic was reduced to make it serve as

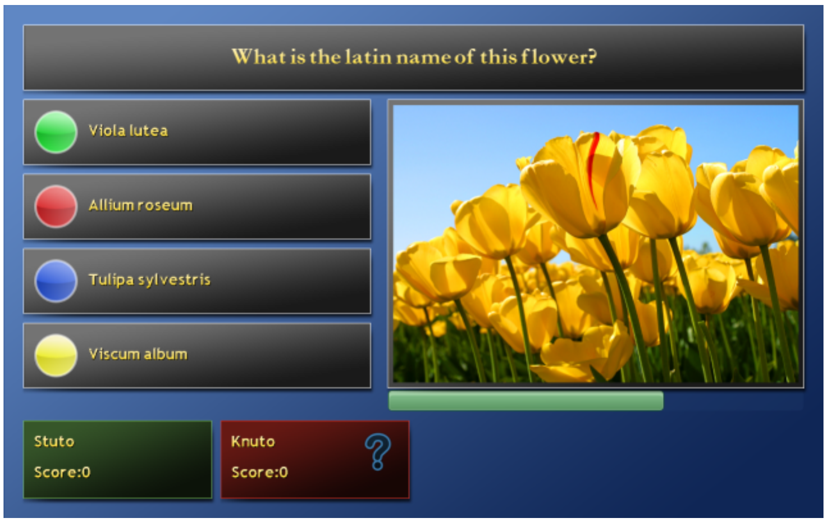


Figure 4.20: LQ 2.5 - Presentation Client answer question screen [25].

an intermediate for clients and the presentation, by running simple game sessions. Using a Web service for this communication seemed like the right choice when the development of LQ 2.5 was started [25]. During development and evaluation of LQ 2.5, it was concluded that it was not suitable to use request based communication through an intermediate (game server), for this type of solution.

The earlier named student client was renamed to Player Client [25]. The use of GWT for this client was disbanded and a new Player Client Web application project was created from scratch. It was created using plain Java with JavaServer Pages (JSP), Hypertext Markup Language (HTML) and JavaScript. The requirements of game codes to access a game was dropped and replaced with a game list, as shown in Figure 4.21. Instead each game session had a unique Uniform Resource Locator (URL), which routed the player clients' requests directly to the lecturer's Presentation Client. This reduces a login step for the Player Client and feels more user-friendly.

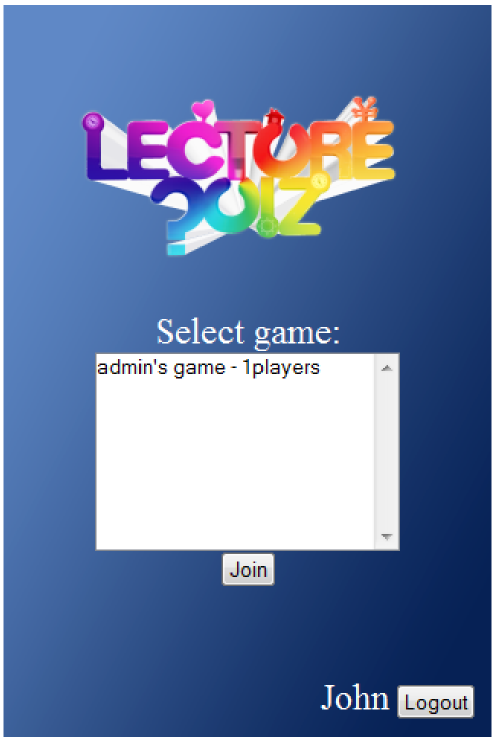


Figure 4.21: LQ 2.5 - Player Client select game screen [25].

Chapter 5

Chosen Technologies

In this chapter, we present the technologies we have chosen for developing Lecture Quiz 3.0. In our specialization project, we researched different technologies suitable for the Lecture Quiz game. Based on this research, the evaluation, and further work of that report, we have selected technologies that best fit the system. Several technologies used in the specialization project still applies in our thesis. We re-use this information in this chapter where applicable. We have created new sections for technologies that have been added in the master thesis, e.g., Sockets, Java Persistance API (JPA) and Java Server Faces (JSF).

5.1 Java Programming Language

In our problem definition, we defined the need for multi-platform support, as described in Section 1.3. In our specialization project, Java and C# .Net was pointed out as the two best programming language alternatives for Lecture Quiz 2.5 [25]. In both Lecture Quiz 1.0 and 2.0, Java was selected as the one and only programming language.

In the specialization project, we made a choice, together with our supervisor, that Java would be both suitable and probably the best choice for keeping the system platform-independent. The context of the project has not changed that much since the specialization project, and we have decided to stick to Java.

Java is a programming language originally developed by James Gosling at Sun Microsystems, released in 1995. Java application developers "write once, run anywhere" because of its Java Virtual Machine (JVM), which is built specifically for the computer architecture [18]. Sun relicensed most of its Java technologies under the GNU General Public License (GPL) in May 2007 [16]. The Java platform consists of a bundle of components, as shown in Figure 5.1, that allows developing and running applications written in the Java programming language. In the transition

2009-2010, Oracle Corporation acquired Sun Microsystems. Oracle has described itself as "the steward of Java technology with a relentless commitment to fostering a community of participation and transparency" [18].

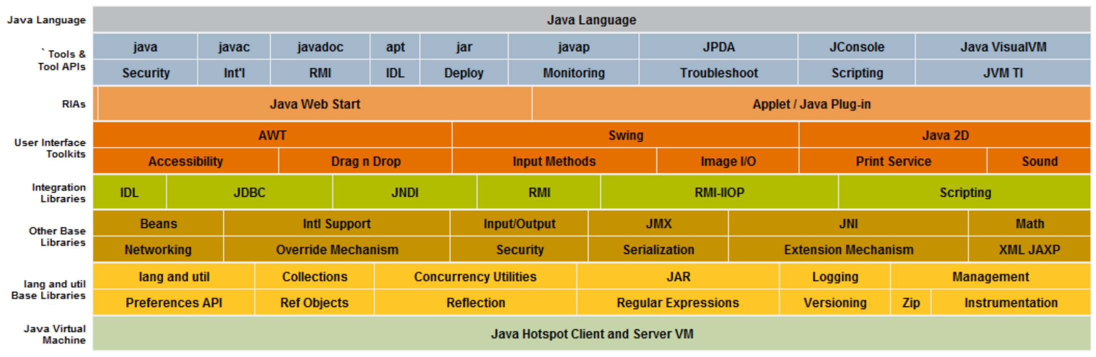


Figure 5.1: Java 6 Platform and Class libraries diagram [22].

5.2 Java Persistance API

In this section, we present technology related to the storage of information. The Quiz Server needs persistent storage to store information such as quizzes and statistics. In the specialization project, we utilized JDBC directly as the bridge between Java and a MySQL relational database. JDBC is an application programming interface (API) for the Java programming language that defines how a client may access a database [20]. The JDBC 3.0 API consists of two packages, "java.sql" and "javax.sql", which both are included in Java Standard Edition (Java SE) 6.

During the research phase of this project, we decided to use a more abstract way of managing our data storage. With this in mind, we chose to use the Java Persistance API (JPA) 2.0 framework. JPA is a Java programming language framework managing relational data in applications using Java SE or Java EE [17]. Several existing vendors support JPA 2.0. We chose to use EclipseLink (former Oracle TopLink). One of the reasons for this is that Sun Microsystems selected the EclipseLink project to be the reference implementation for JPA 2.0 [28]. The EclipseLink JPA provides developers with a standard Object-Relational persistence solution that has additional support for advanced features. EclipseLink JPA provides advanced support for leading relational databases and Java containers [29]. The EclipseLink architecture diagram can be seen in Figure 5.2.

EclipseLink supports any relational database that is compliant with Structured Query Language (SQL) and has a compliant JDBC driver [30]. EclipseLink has extended support for several database platforms. Among them are Oracle, MySQL, PostgreSQL, Derby, Microsoft SQL Server and many more. We continue to develop Lecture Quiz using MySQL, as we did in our specialization project.

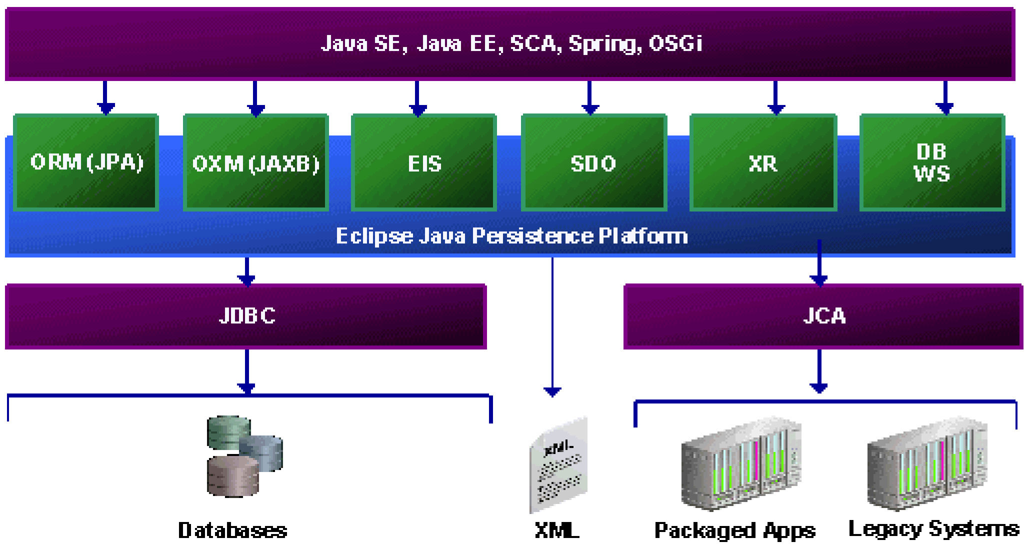


Figure 5.2: EclipseLink architecture diagram [29].

5.2.1 MySQL

MySQL is an open source relational database management system (RDBMS) initially developed by MySQL[1] AB[2]. It is the world's most popular open source database software, and has high speed, great reliability and is easy to use [65, Oracle]. It is dual-licensed with an open GNU GPL or a commercial proprietary end-user licensing agreement (EULA). The commercial license is required if you only distribute your application in binary-form to end-users. MySQL has a client API for many languages, including C, C++, Java and C# .Net. It runs as a server, and both the server and client applications run on many systems, including Windows, Linux and Mac OS X. We only use MySQL for development purposes, as mentioned in the previous section.

5.3 Java Server Faces

In the specialization project, the Quiz Server web interface was based on JavaServer Pages Standard Tag Library (JSTL). JSTL is a component of the Java EE Web application development platform. It is an extension of the JSP specification and adds a tag library for common functional tasks, such as Extensible Markup

[1]currently owned and developed by Oracle

[2]Aktiebolag (literally "share company" or "stock company") is the Swedish term for "limited company" or "corporation"

Language (XML) data processing, conditional execution, loops and internationalization [15]. The latest version, JSTL 2.0, was released in May, 2006.

In version 3.0 of the Lecture Quiz game we wanted to use a more updated Web application framework, which also has more direct support for the model-view-controller (MVC) architectural pattern. We looked into frameworks like Struts, Spring MVC, Play, GWT and more. We already tried the GWT framework in our specialization project and concluded that it contains several flaws, and has too much overhead [25]. The Play framework looks appealing, but it has not existed for long and few have adopted it. Struts, was one of the first MVC frameworks for the Web. It is becoming cumbersome compared to newer frameworks. Version 2.0 is not expected to be released, and our conclusion is that Struts becomes obsolete in the near future. Spring MVC is an upcoming framework, but few professionals use it, and it is not accepted as an official standard. This leaves us with JSF, which is an official Java specialization standard. Many professionals use JSF, it is well documented, and has continued releases. Based on this preliminary evaluation, we chose the JSF framework.

JSF is a Java based Web application framework intended to simplify development integration of web-based user interfaces [19]. It can be seen as a request-driven MVC Web framework, based on component-driven user interface (UI) design model, using XML files called view templates or Facelets views. JSF is the combination of APIs for representing UI components and managing their state, handling events and input validation, defining page navigation, and supporting internationalization and accessibility. Core features include Managed Beans; a template-based component system; built-in asynchronous JavaScript and XML (AJAX), bookmarking, and page-load actions support; Expression Language (EL) integration; a default set of HTML and web-application specific UI components; A server-side event model; State management; and two XML-based tag libraries (core and HTML). The life cycle of JSF is divided into six phases, as shown in Figure 5.3.

5.4 Lightweight Java Game Library

In the specialization project we used Java OpenGL (JOGL), which is a wrapper library for Open Graphics Library (OpenGL). It is designed specially to be used with the Java programming language. JOGL is not a multimedia library, which means it does not come with support for audio. Lightweight Java Game Library (LWJGL), on the other hand supports both OpenGL and Open Audio Library (OpenAL). This allows us to play sounds via OpenAL and render graphics via OpenGL. OpenGL is not platform independent, although it is supported on most platforms. Running OpenGL on a different operating systems (OSs), requires separate system libraries for each platform. This was handled poorly in JOGL, compared to LWJGL. Based on the support for audio and OS compatibility, we decided to change to LWJGL.

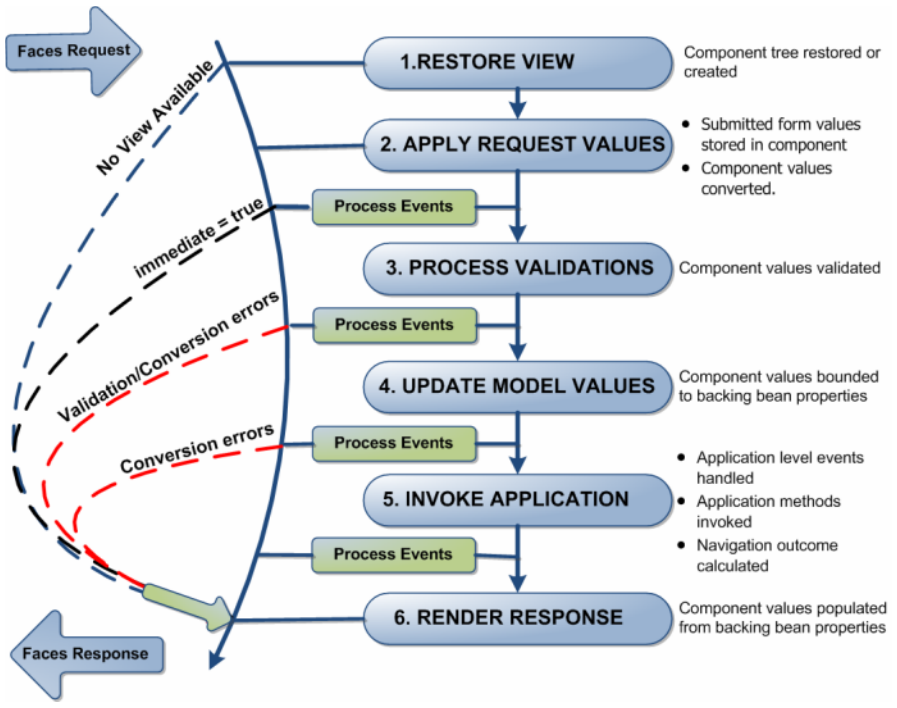

Faces Request
1.RESTORE VIEW
Component tree restored or created
2. APPLY REQUEST VALUES
• Submitted form values stored in component
• Component values converted.
Process Events
3. PROCESS VALIDATIONS
Component values validated
Process Events
4. UPDATE MODEL VALUES
Component values bounded to backing bean properties
Process Events
5. INVOKE APPLICATION
• Application level events handled
• Application methods invoked
• Navigation outcome calculated
Process Events
6. RENDER RESPONSE
Component values populated from backing bean properties
Faces Response
No View Available
immediate = true
Validation/Conversion errors
Conversion errors

Figure 5.3: JSF life cycle [4].

LWJGL is a solution aimed directly at professional and amateur Java programmers alike to enable commercial quality games to be written in Java [56]. It provides developers access to high performance cross-platform libraries such as OpenGL and OpenAL allowing for state of the art 3D games and 3D sound. Additionally, LWJGL provides access to controllers such as gamepads, steering wheel and joysticks; all in a simple and straight forward API [56]. We utilize the OpenGL and OpenAL libraries via LWJGL, which are explained in the following two sections.

5.4.1 Open Graphics Library

OpenGL is a cross-platform API for developing applications that utilize 2D and 3D computer graphics, developed by the OpenGL Working Group under the Khronos Group consortium [47]. It is an open standard, and has support for many platforms, such as Mac OS X, Microsoft Windows, Linux, Android and iOS. The graphics pipeline process of OpenGL can be viewed in Figure 5.4. OpenGL was originally developed for the C programming language, but has many standard language bindings.

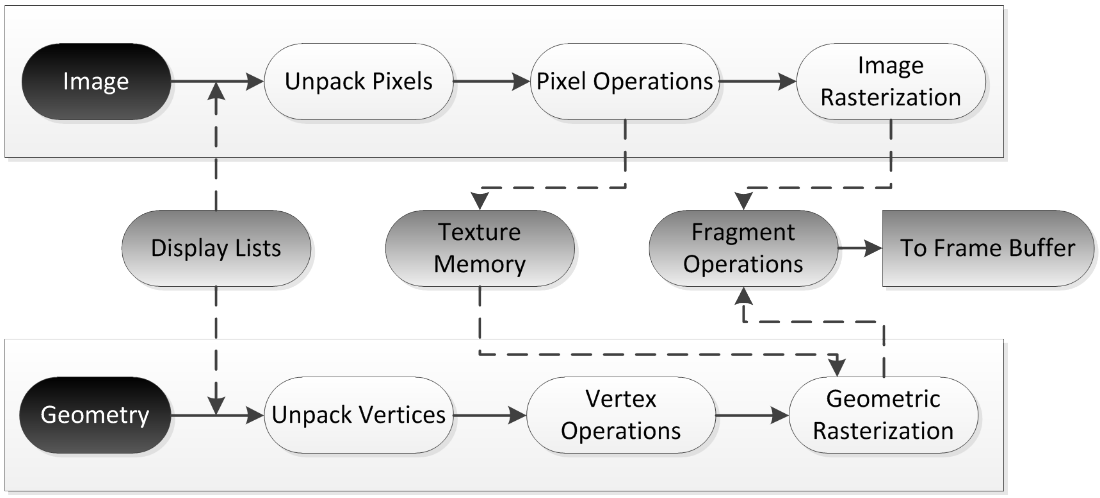


Figure 5.4: OpenGL graphics pipeline [47].

5.4.2 Open Audio Library

OpenAL is a cross-platform 3D audio API appropriate for use with gaming applications and many other types of audio applications [63]. It was originally created by Loki Software, but was given to the community as free software after its demise. LWJGL is endorsing the 1.0 version, which was released under the GNU Lesser General Public License (GNU LGPL) license in June 2000. The current version of OpenAL is 2.1, released February 2010, but is unfortunately proprietary. OpenAL has been proprietary since version 1.1, when Creative Technology started to host and develop it. Because of this, LWJGL has to endorse version 1.0, even though the specification has become legacy. We use OpenAL to play sound effects according to the pattern shown at the top of Figure 5.5.

5.5 TCP/IP Sockets

In our further work section in our specialization project, we suggested to change the communication logic from using web services to sockets [25]. Web services, does not match the communication flow of our framework. We chose to use sockets because they offer us a way of creating an easy and fast two-way communication.

Calvert and Donahoo describe TCP/Internet Protocol (IP) sockets as “an abstraction through which an application may send and receive data” [13]. Since sockets can be considered an abstraction of the protocols they utilize, their transfer speed and latency reflect their implementation and the network topology. Internet sockets are mechanisms for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote IP-addresses and port numbers. One of the benefits using sockets is the interoperability be-

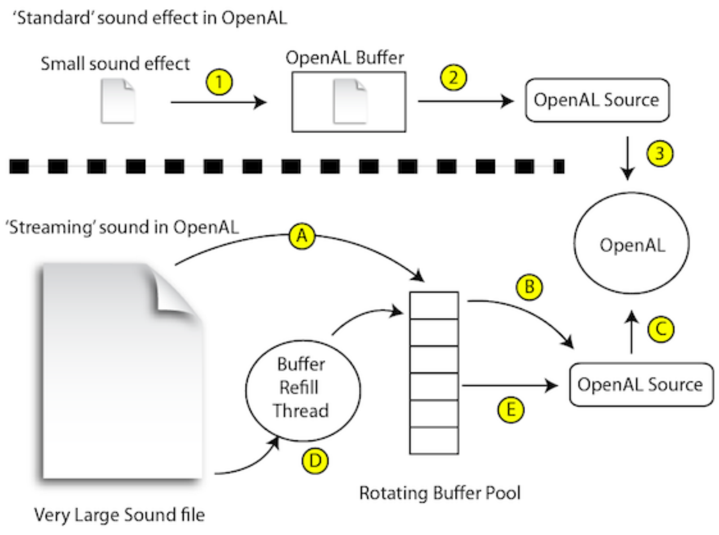


Figure 5.5: Playing sound effects with OpenAL [3].

tween platforms, which is crucial for mobile units with unique platforms. This is possible because TCP/IP operates independently of frameworks and platforms. E.g. applications using Java, .Net or Objective-C, can easily communicate with each other via sockets.

Most socket implementations are based on Berkeley-sockets, first introduced in 1983. We do not use this API directly, but will create our own implementation. The TCP socket flow of our implementation is based on the diagram shown in Figure 5.6.

5.6 GlassFish Server

To deploy a Java EE Web application, an application server is required. Apache is the most popular server (58%), as of November 2010, and Microsoft with Internet Information Services (IIS) (23%) comes second [51]. There exist several Java EE application servers, where the most popular ones are GlassFish, Apache Tomcat (web container), JBoss and WebSphere CE. Our goal is to implement our solution as independent as possible, with respect to application servers. We have the most experience with GlassFish and Apache Tomcat, and we will use these as target servers.

In the specialization project we used GlassFish 3.0 which is bundled with NetBeans. This worked well and we have chosen to continue using GlassFish.

GlassFish is an open source application server for the Java EE platform. It is developed by Sun Microsystems, and is dual-licensed under the Common Devel-

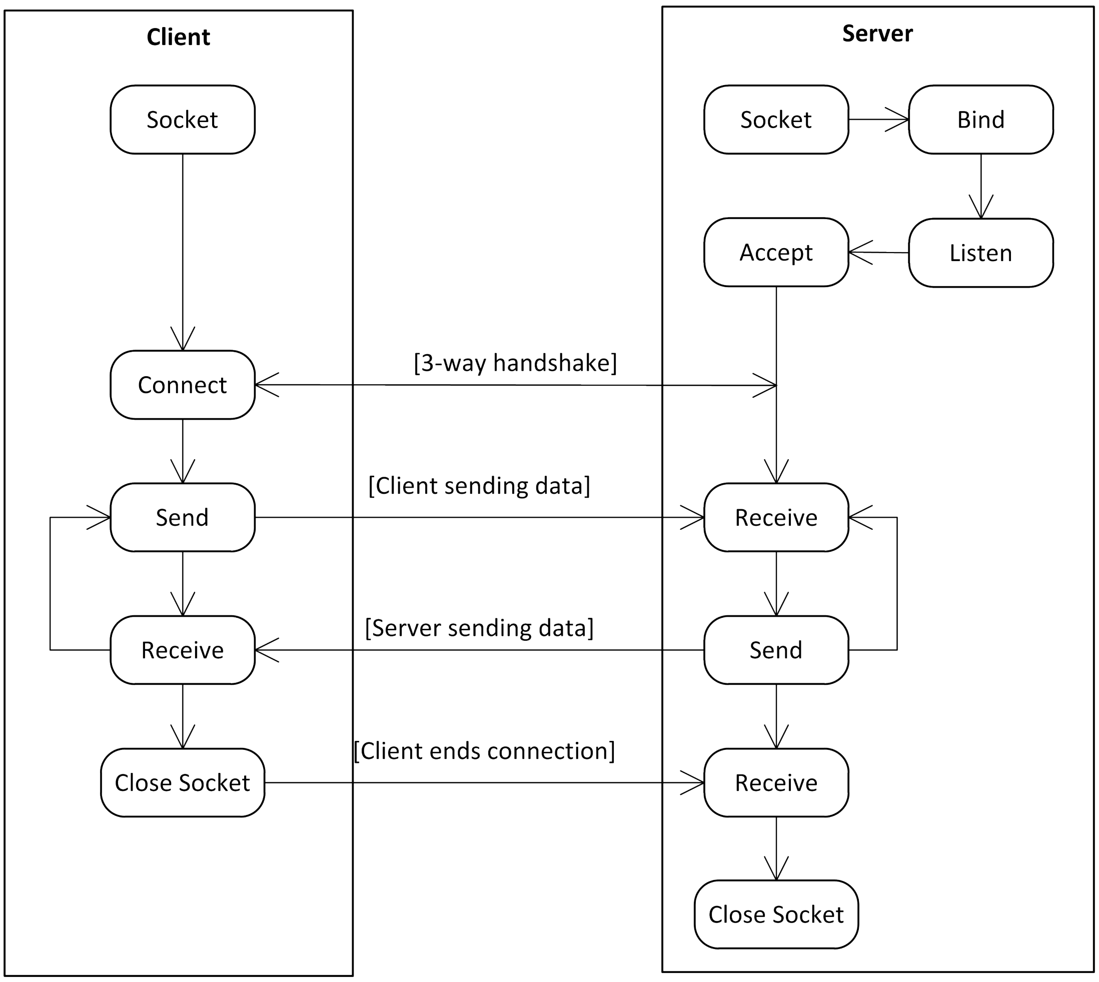


Figure 5.6: TCP socket flow diagram.

opment and Distribution License (CDDL), and the GNU GPL. Being the Java EE reference implementation, this was the first application server to completely implement Java EE 6 Java Specification Request (JSR) 316[3] [21]. GlassFish is written in Java, and is cross-platform.

5.7 Embedded HTTP Server

In Lecture Quiz 2.5, the Player Client was deployed and run as a Web application. The solution used JSP web pages with AJAX for communication. This was a separate component in the framework and did not directly communicate with the Presentation Client.

We want a shorter, more direct way of communicating with the Presentation Client.

[3]http://jcp.org/en/jsr/detail?id=316

This is possible with an embedded Hypertext Transport Protocol (HTTP) server running on the top of the Presentation Client.

The Java httpserver package provides a simple high-level HTTP server API, which can be used to build embedded HTTP servers [22]. We chose to use the Java HttpServer class, as it is native to our chosen programming language.

Part IV

Own Contribution

Chapter 6

Requirements

In this chapter, we outline our functional and non-functional/quality requirements. These requirements are the product of our overall evaluation of Lecture Quiz. We have used these requirements in our development of Lecture Quiz. We have used these requirements in our development of Lecture Quiz, and they have helped us keep the focus on the main functionality and the overall quality of the system.

6.1 Functional Requirements

In this section, we describe our functional requirements, as shown in Table 6.1, Table 6.2 and Table 6.3. These are based on how we intend to implement the functionality of Lecture Quiz. This functionality includes: playing games as a student; running games with game logic based on a quiz as a lecturer; and creating quizzes and view statistics.

The functional requirements are presented as user stories. Leffingwell et al., describes a user story as brief statements of intent that describe something the system needs to do for some user [42]. As commonly taught, the user story often takes a standard user-voice form. All our requirements have been created with the template: "As a <role>, I want <goal/desire>, so that <benefit>".

The functional requirements are divided into three parts and are represented as tables: Presentation Client, Player Client and Quiz Server. These are the three main applications of our system and are related to users having goals/desires and benefits. Each functional requirement row is divided into three columns: an identifier (FR1, FR2, ...), the user story, and the priority. The priority is divided into three discrete values: Low, Medium and High. The values are used to prioritize the order in which we develop features.

Presentation Client		
ID	Description	Priority
FR1	As a lecturer, I want to run quiz games, so that I can test the knowledge level of my students.	High
FR2	As a lecturer, I want to start team games, so that I can divide the students into groups.	Medium
FR3	As a lecturer, I want to start free for all games, so that I can put each student up against each other.	Medium
FR4	As a lecturer, I want to display my computer address, so that I can tell students where they can access the game.	Low
FR5	As a lecturer, I want to choose a quiz, so that I can decide what type of questions the students have to answer.	High
FR6	As a lecturer, I want to view a quiz, so that I can get a description of what the quiz contains.	Low
FR7	As a lecturer, I want to select one or more game modes, so that I can create variation in the game play.	Medium
FR8	As a lecturer, I want to see how many students are on each team, so that I can decide if the teams are even.	Low
FR9	As a lecturer, I want to see how many students are connected, so that I can check if everybody is playing.	Medium
FR10	As a lecturer, I want to move a student from one team to another, so that I can decide who plays on which team.	Low
FR11	As a lecturer, I want to automatically even teams, so that I don't have to move a student from one team to another.	Low
FR12	As a lecturer, I want to let the students select teams, so that I can tell students to pick their own team.	Medium

FR13	As a lecturer, I want to display information about the game mode, so that I don't have to tell the students how the game works.	Medium
FR14	As a lecturer, I want the game to display questions and alternatives, so that I don't have to tell students the question and what they can answer.	High
FR15	As a lecturer, I want to decide when the next question appears, so that I can speak about the current question.	High
FR16	As a lecturer, I want to export statistics after a quiz, so that I can review them later.	High
FR17	As a lecturer, I want the statistics to automatically be uploaded to server when connected, so that I don't have to export.	Medium
FR18	As a lecturer, I want to see student rankings, so that I can reward the best team or student.	Medium
FR19	As a developer, I want to add a new game mode, so that I can change the game logic.	High

Table 6.1: Functional requirements for the Presentation Client.

Player Client		
ID	Description	Priority
FR20	As a student, I want to log in, so that I can participate in a game.	High
FR21	As a student, I want to specify a username when logging in, so that I can identify my player.	Medium
FR22	As a student, I want to pick an alternative, so that I can answer questions.	High
FR23	As a student, I want to see my score, so that I can track my progress.	Low

FR24	As a student, I want to select a team, so that I can join the team I want to play together with.	Medium
FR25	As a student, I want to pick a vote selection, so that I can affect my team's choices.	Medium
FR26	As a student, I want to see my rank at the end of the game, so that I can determine how good I did in the quiz.	Low
FR27	As a student, I want to see what I answered on a question, so that I can compare it to the results on the screen.	Medium
FR28	As a student, I want to see my nickname associated with a team, so that I know which team I am playing on.	Medium

Table 6.2: Functional requirements for the Player Client.

Quiz Server		
ID	Description	Priority
FR29	As a lecturer, I want to create quizzes, so that I can use them in lectures.	High
FR30	As a lecturer, I want to edit a quiz, so that I can correct errors.	Medium
FR31	As a lecturer, I want to delete quizzes, so that I can remove unwanted quizzes.	Low
FR32	As a lecturer, I want to log in, so that I can have a private account.	High
FR33	As a lecturer, I want to see statistics for a quiz, so that I can evaluate student progression.	High
FR34	As a lecturer, I want to assign an icon to a quiz, so that I can create questions involving images.	Medium

FR35	As a lecturer, I want to download a quiz, so that I can run the quiz without having a connection to the server.	Medium
FR36	As a lecturer, I want to upload a quiz, so that I can get other people's quizzes.	Low
FR37	As a lecturer, I want to be able to upload statistics, so that I don't have to be connected to the server when generating statistics.	High

Table 6.3: Functional requirements for the Quiz Server.

6.2 Quality Requirements

This section contains the quality requirements defined for Lecture Quiz. Quality requirements are the non-functional requirements of a software system. We describe the quality requirements using quality attribute scenarios, as described by Len Bass et al. Figure 6.1 shows a visual representation of a quality attribute scenario. A quality attribute scenario is a quality-attribute-specific requirement which consists of six parts [8]:

- *Source of stimulus.* This is some entity (a human, a computer system, or any other actuator) that generated the stimulus.
- *Stimulus.* The stimulus is a condition that needs to be interpreted when it arrives at a system.
- *Environment.* The stimulus occurs within certain conditions. The system may be in an overload condition or may be running when the stimulus occurs, or some other condition may be true.
- *Artifact.* Some artifact is stimulated. This may be the whole system or some pieces of it.
- *Response.* The response is the activity undertaken after the arrival of the stimulus.
- *Response measure.* When the response occurs, it should be measurable in some fashion so that the requirement can be tested.

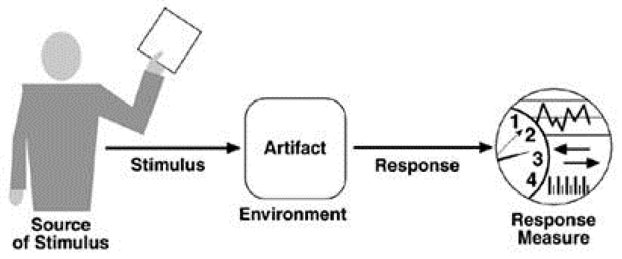


Figure 6.1: Quality attribute scenario [8].

Quality attribute scenarios are divided into sub-genres. In our system, we have focused on modifiability, usability and availability.

6.2.1 Modifiability

This section describes our scenarios regarding modifiability. Modifiability is the ability of the system to undergo changes with a degree of ease [8]. These changes could impact components, services, features, and interfaces when adding or changing the functionality, fixing errors, and meeting new business requirements.

We want the game to be extendable and modifiable, since it is a new system in this context. The aim is to be able to use the system in a lecture environment, and we have to be prepared for changes and requests during development. Since lecturers are used to prepare their own program for lectures, we do not want the application to feel limited when it comes to customization. We want the lecturer to be able to set up how the game plays, through their quiz.

M1 - Create a new game mode	
Source of stimulus	Game mode developer
Stimulus	The game mode developer wants to create a new game mode
Environment	Design Time
Artifact	Presentation Client
Response	A new game mode is created
Response measure	The game mode should be finished within two hours

Table 6.4: M1 - Create a new game mode.

M2 - Multiple game modes	
Source of stimulus	Lecturer
Stimulus	The lecturer wants multiple game modes run in one game
Environment	Runtime
Artifact	Presentation Client
Response	Game runs with multiple game modes
Response measure	The game is run with more than one game mode

Table 6.5: M2 - Multiple game modes.

M3 - Creating a new player client	
Source of stimulus	Client Developer
Stimulus	The client developer wants to create a new client for the Lecture Quiz game
Environment	Design time
Artifact	Framework
Response	A new Player Client
Response measure	The client should be finished withing two hours

Table 6.6: M3 - Creating a new player client.

M4 - Change client skin	
Source of stimulus	Graphic artist
Stimulus	Modify the graphic resources of the player and presentation client
Environment	Design time
Artifact	Presentation and Player clients
Response	Modified image resources
Response measure	Should be able to modify resources within one hour

Table 6.7: M4 - Change client skin.

M5 - Change the database back-end	
Source of stimulus	Developer
Stimulus	The developer wants change to another database back-end
Environment	Design time
Artifact	Quiz Server
Response	Server running on another database back-end
Response measure	Should be able to change to another database back-end within one hour

Table 6.8: M5 - Change the database back-end.

M6 - Change GUI composition	
Source of stimulus	Developer
Stimulus	The developer wants to change the arrangement of GUI components
Environment	Design time
Artifact	Presentation Client
Response	User interface is changed
Response measure	The developer should be able to change the composition of GUI elements within an hour

Table 6.9: M6 - Change GUI composition.

M7 - Add support for a new input device	
Source of stimulus	Developer
Stimulus	The developer wants to add support for a new input device
Environment	Design time
Artifact	Presentation Client
Response	User interface is navigable with the new device
Response measure	The developer should be able to add support for the new input device within an hour

Table 6.10: M7 - Add support for a new input device.

M8 - Add support for Java SE compatible OS	
Source of stimulus	Developer
Stimulus	The developer wants to add support for running the Presentation Client on another Java SE compatible OS
Environment	Design time
Artifact	Presentation Client
Response	The Presentation Client running on the target OS
Response measure	The developer should be able to add support for the os within three hours

Table 6.11: M8 - Add support for Java SE compatible OS.

6.2.2 Usability

This section describes our scenarios for the usability attribute. Usability defines how well the application meets the requirements of the user and consumer by being intuitive; easy to localize and globalize; and resulting in a good overall user experience [8]. Our usability scenarios are based on people's experience with computer interfaces. For example, the ability to recognize commonly used visual elements and know how to interact with them.

U1 - Support visual gaming interface	
Source of stimulus	Developer
Stimulus	The developer wants to create a recognizable gaming interface
Environment	Design time
Artifact	Presentation Client
Response	A recognizable gaming interface
Response measure	At least 90 percent of the user base recognize the interface as a common gaming interface

Table 6.12: U1 - Support visual gaming interface.

U2 - A common input interface	
Source of stimulus	Student
Stimulus	The student wants to use the Player Client
Environment	Runtime
Artifact	The client interface
Response	Easy to use input interface
Response measure	At least 90 percent of the students using the system should understand the interface

Table 6.13: U2 - A common input interface.

6.2.3 Availability

This section describes our scenarios for the availability attribute. Availability defines the proportion of time in which the system is functional and working [8]. It can be measured as a percentage of the total system downtime over a predefined period. Availability will be affected by system errors, infrastructure problems, malicious attacks, and system load.

When it comes to the quality of a game, availability is an important factor, specially for network games. Lecture Quiz uses communication between each player and the Presentation Client, and between the Presentation Client and the Quiz Server. Game statistics and Quizzes are transferred between the Presentation Client to the Quiz Server. It is important that a player is able to communicate to the Presentation Client during a game, or else they will lose interest. We want the game to be available on a wide range of clients, so that students can use their own devices to play the game during a lecture.

A1 - Reconnect to server	
Source of stimulus	Presentation Client
Stimulus	The Presentation Client should reconnect to the Quiz Server if connection is lost
Environment	Runtime
Artifact	Network connection
Response	Reestablish connection to server
Response measure	The Presentation Client should reconnect to the Quiz Server within 30 seconds if connection is possible

Table 6.14: A1 - Reconnect to server.

A2 - Simple web client	
Source of stimulus	Player Client
Stimulus	The Player Client should not crash during gameplay
Environment	Runtime
Artifact	Player Client
Response	Client not crashing during gameplay
Response measure	The Player Client should not crash during a game session with at least 20 clients

Table 6.15: A2 - Simple web client.

A3 - Run on different client types	
Source of stimulus	Player Client
Stimulus	The Player Client should be available on different client types
Environment	Runtime
Artifact	Player Client
Response	The client is able to play the game
Response measure	The Player Client should support at least 90 percent of todays web browsers

Table 6.16: A3 - Run on different client types.

Chapter 7

Architecture

In this chapter, we describe the main architectural structures of Lecture Quiz. The previous chapter outlines several requirements, both functional and non-functional. This chapter contains architectural choices used to fulfill the quality requirements of Lecture Quiz.

Our architecture follows the “4+1” view model designed by Philippe Kruchten. “4+1” presents a model for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views [39]. This use of views allows separation of the architecture to fit the concerns of the various stakeholders. Examples of stakeholders are end-users, developers, system engineers, project managers, etc. It also allows for separate handling of functional and non-functional requirements. The views are designed using an architecture-centered, scenario-driven, iterative development process. The 4 main views are the physical view, development view, logic view, and process view. The fifth view can be seen as a composition of the first four views, represented as use cases or scenarios. The “4+1” view model with stakeholders can be seen in Figure 7.1. In the following sections, we describe the architecture based on each of the four main views.

The notation we have chosen to describe our architectural views is Unified Modeling Language (UML). UML is a standardized general-purpose modeling language in the field of object-oriented software engineering [40]. The standard is managed, and was created by, the Object Management Group. It includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems [40].

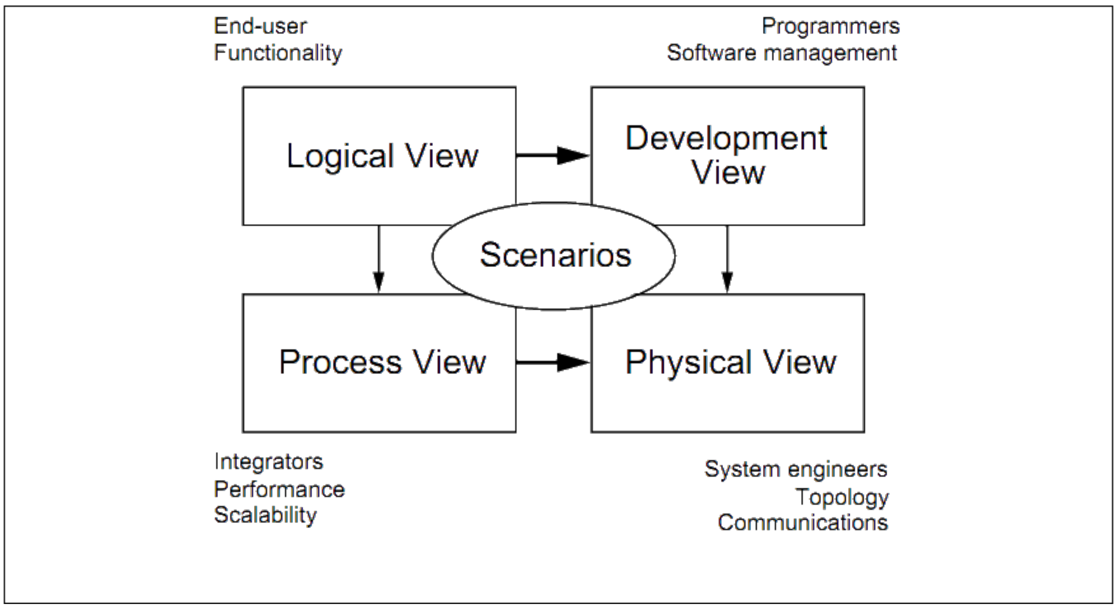


Figure 7.1: The “4+1” view model [39].

7.1 Physical View

The physical view describes the mapping of the software onto the hardware and reflects its distributed aspects [39]. It is meant to give an overview of the physical component structure, and how components communicate. This view is intended for system engineers and administrators.

The solution consists of two main software applications, the Presentation Client and the Quiz Server. The Presentation Client contains the game logic and visuals for displaying quiz games on a projector. In addition to this, the Presentation Client uses the Player Client library to run a local web server for communication with the clients. The Presentation Client is meant to be operated by the lecturer while the students participate in games through the Player Client. For communication with the Quiz Server, the Presentation Client uses the Distributed Message Queue library. This library uses sockets to connect to the server and abstracts the communication logic into messages. Quizzes are created and stored on the Quiz Server by the lecturer. Access to the Quiz Server is done through a web page created with the JSF framework.

To give an overview of where each physical component is intended to be used, we have separated the components into physical locations. These different locations are the lecture hall, the office and the server room, as shown in Figure 7.2.

Quiz games are run on the Presentation Client in the lecture hall, during lectures. The information is presented on a big-screen through a projector. Students can participate by looking at questions at the big-screen and answering on their tablet, mobile or laptop. Each question has multiple choices, and the alternatives are shown on the big-screen, while the students answer on their individual devices. Student's participating only requires a connection to the same network as the presentation computer, and a graphical web browser.

The Quiz Server is deployed on a GlassFish server and is created with Java EE. It requires a database for storing information about users, statistics, quizzes and questions. Both the Quiz Server and the database are meant to be run inside a server room. This is because multiple Presentation Clients can use one Quiz Server, and because installing a new Quiz Server for each game would be too cumbersome.

We expect lecturers to prepare a quiz for a lecture in the same manner as they would prepare a presentation. This is usually done in the lecturer's office or workspace, on a PC.

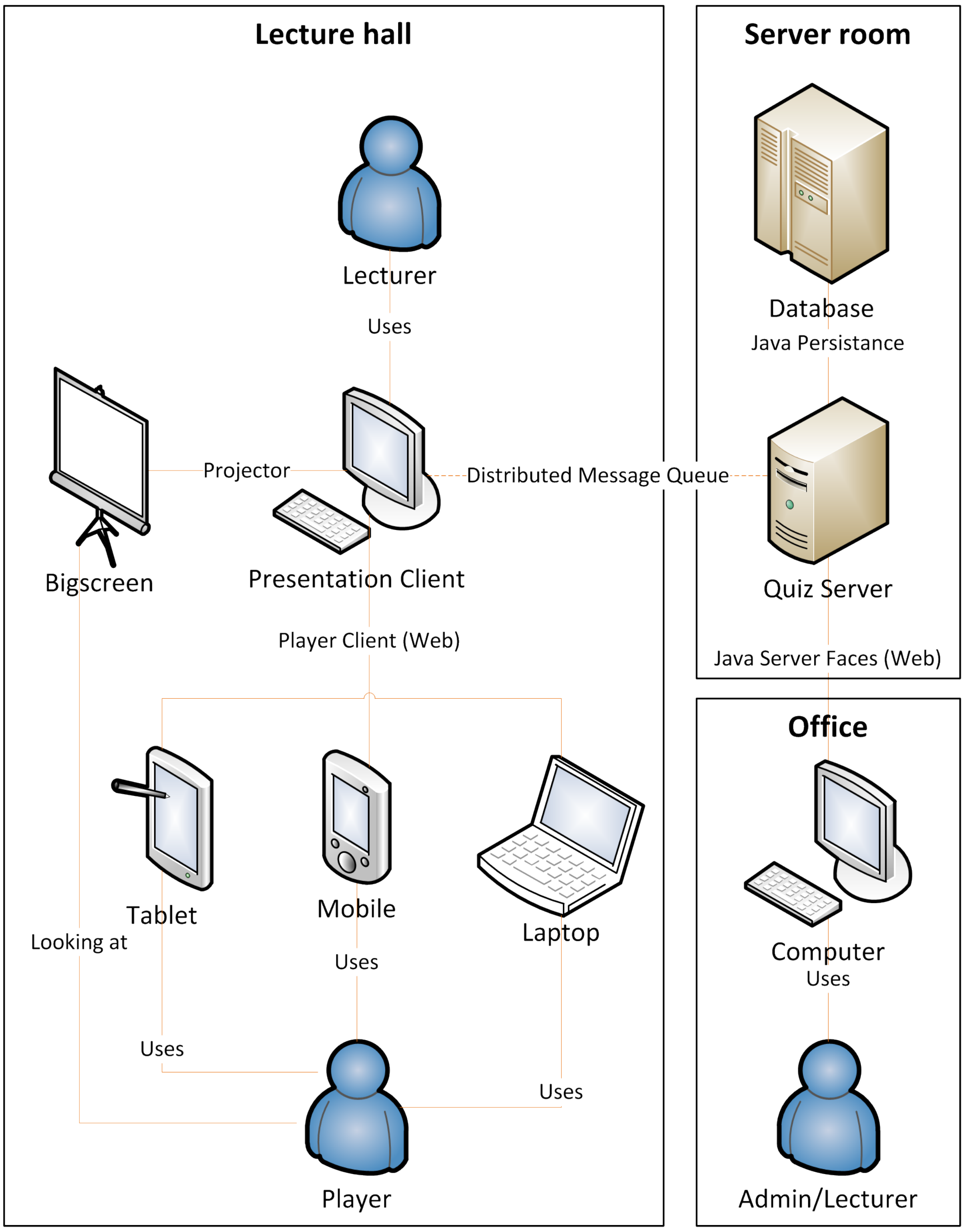


Figure 7.2: Physical deployment of the Lecture Quiz solution.

7.2 Development View

The development view describes the static organization of the software in its development environment [39]. It gives architectural insight to the software components of our system and how they communicate. Lecture Quiz uses libraries to handle communication, e.g. between the Presentation Client and the Quiz Server or between the Presentation Client and the players.

The communication between the Presentation Client and the Quiz Server is done through the message queue, as shown in Figure 7.3. Both the Quiz Server and the Presentation Client creates an instance of the Distributed Message Queue at initialization. Messages are added to the queue through the IMessageQueue interface. The receiver will get messages via the IMessageReceiver interface.

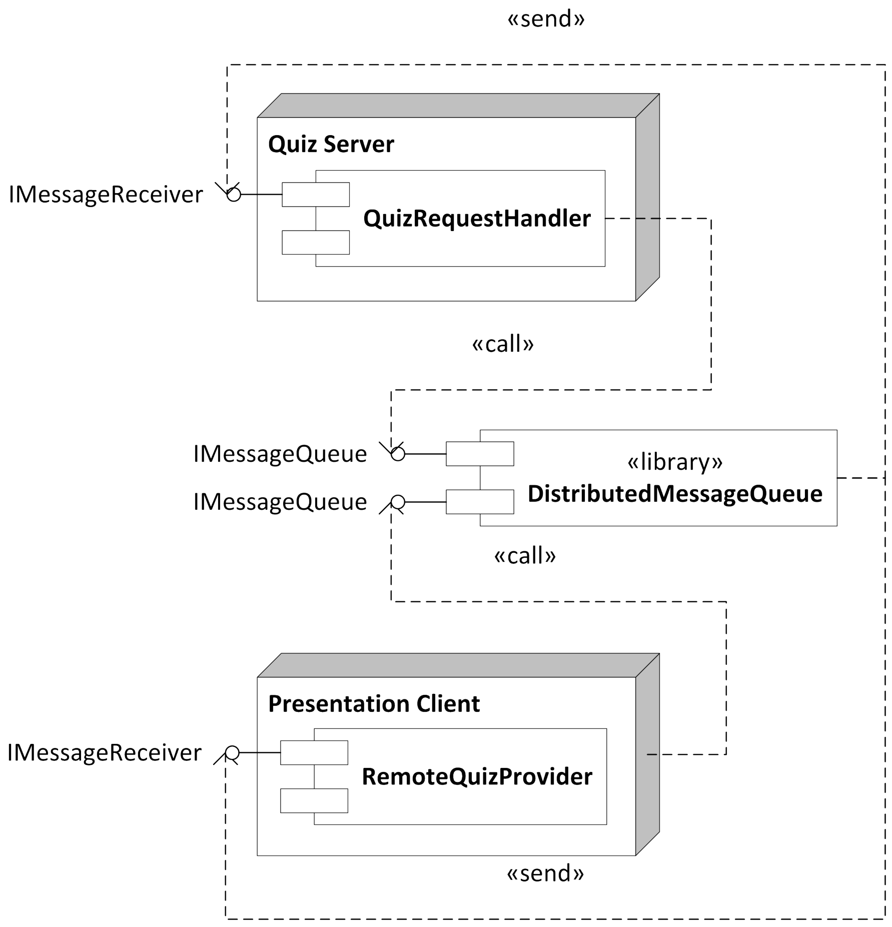


Figure 7.3: Communication between the Presentation Client and the Quiz Server for requesting quizzes.

Messages sent on the Distributed Message Queue are serialized into binary data. Serialization is done with Java object streams on objects implementing the Serializable interface. When reconstructing a message, the application needs to have the same class definition for the message as the sender [22]. This is where the Lecture Quiz Messages library comes into play. This library contains shared message classes that the Presentation Client and Quiz Server use for communication, as shown in Figure 7.4.

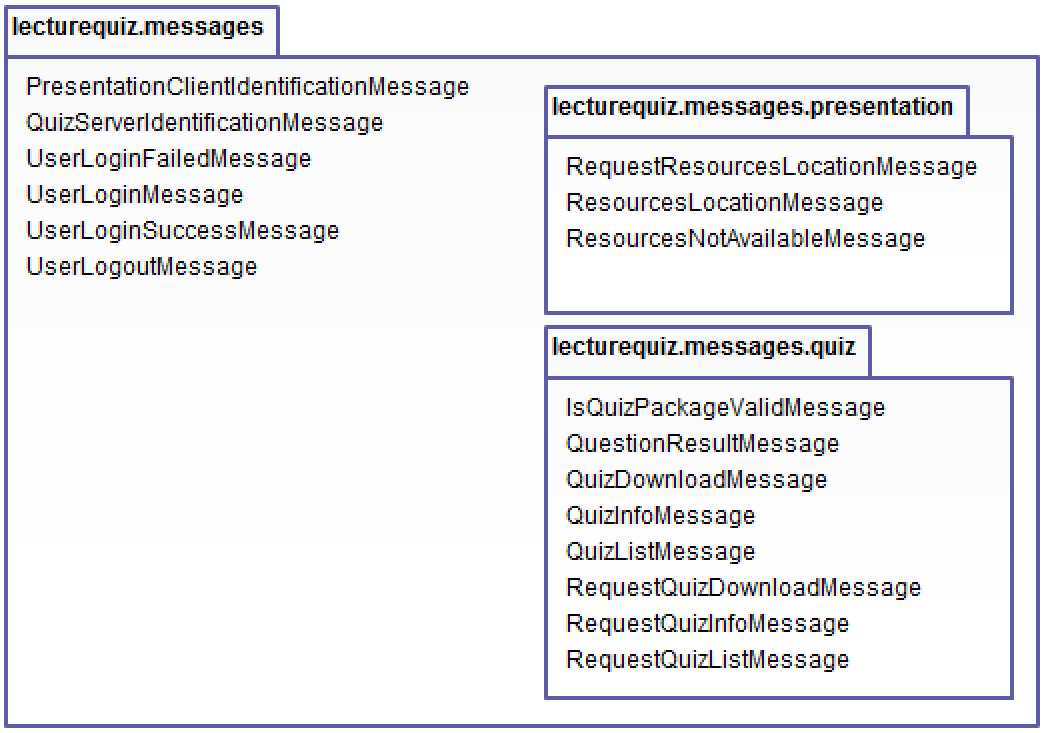

lecturequiz.messages
PresentationClientIdentificationMessage
QuizServerIdentificationMessage
UserLoginFailedMessage
UserLoginMessage
UserLoginSuccessMessage
UserLogoutMessage
lecturequiz.messages.presentation
RequestResourcesLocationMessage
ResourcesLocationMessage
ResourcesNotAvailableMessage
lecturequiz.messages.quiz
IsQuizPackageValidMessage
QuestionResultMessage
QuizDownloadMessage
QuizInfoMessage
QuizListMessage
RequestQuizDownloadMessage
RequestQuizInfoMessage
RequestQuizListMessage

Figure 7.4: Lecture Quiz Messages package diagram.

The game logic on the Presentation Client communicates with the connected players through the IPlayerClient interface, as shown in Figure 7.5. The Presentation Client uses the Player Client library to be able to send data over HTTP.

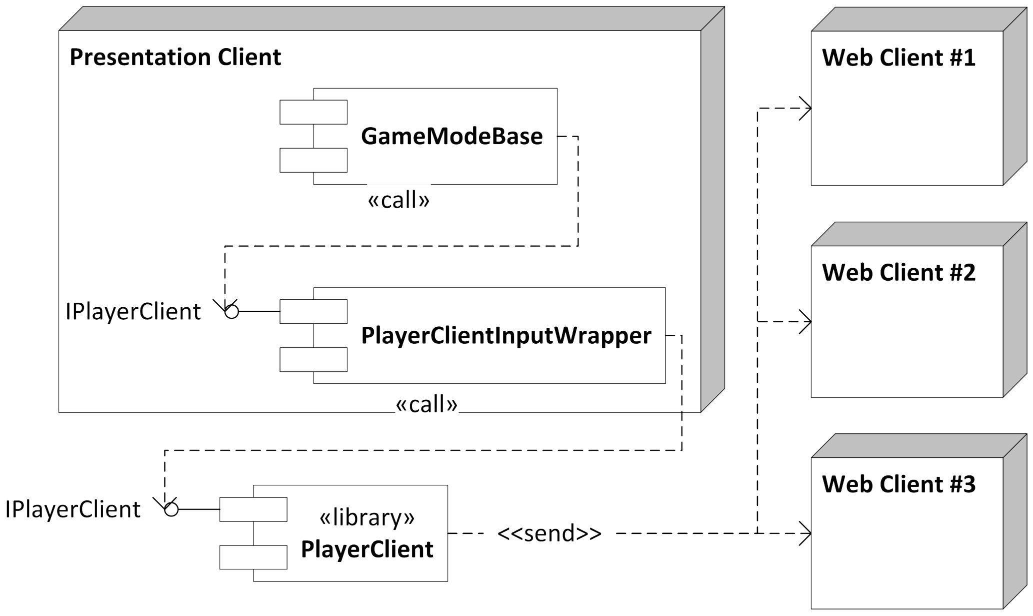


Figure 7.5: Communication between the game mode logic and the web clients through the Player Client library.

7.3 Logic View

The logical view can be seen as the object model of the design (when an object-oriented design method is used) [39]. The logical architecture primarily supports the functional requirements; what the system should provide in terms of services to its users.

7.3.1 Presentation Client

The Presentation Client is separated into six parts. Each part has its own manager class to handle their respective components. The managers implement the IManager interface, as shown in Figure 7.6. This interface allows for uniform initialization and disposing of the different managers. The managers follow the object lifetime manager pattern [43]. The separation of logic helps keep the code clean and structured.

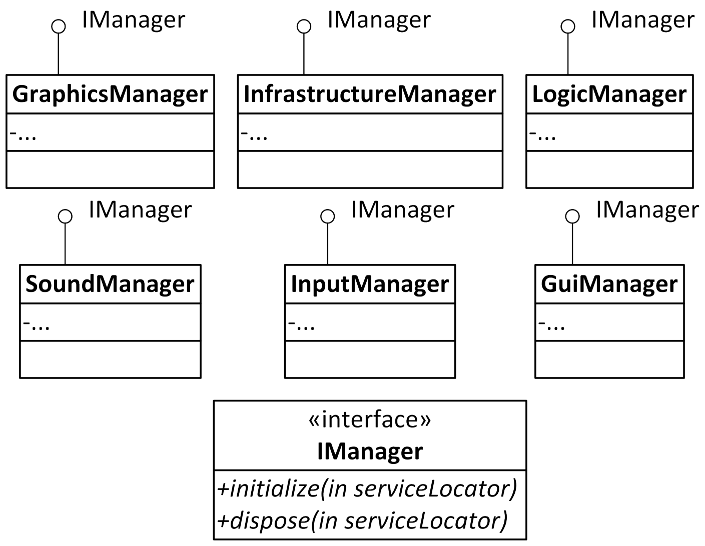


Figure 7.6: IManager interface class diagram.

To allow interaction between the different parts of the application we use a service locator, as shown in Figure 7.7. The service locator is a thread safe singleton that can be referenced from anywhere in the application. The singleton pattern is a design pattern used to implement the mathematical concept of a singleton [68, p. 37]. It restricts the instantiation of a class to one object, and is useful when exactly one object is needed to coordinate actions across the system [31]. The initialization managers are responsible for registering the services that their part of the application provides. These services will then be accessible via other parts of the application through the service locator. An example of a service registered with the service locator is the Distributed Message Queue. The message queue is managed by the InfrastructureManager and registered with the service locator through the IMessageQueue interface.

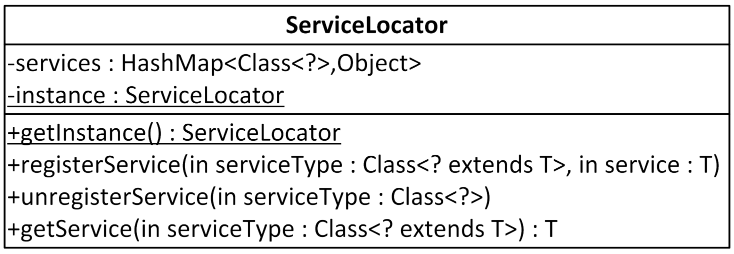

ServiceLocator
-services : HashMap<Class<?>,Object>
-instance : ServiceLocator
+getInstance() : ServiceLocator
+registerService(in serviceType : Class<? extends T>, in service : T)
+unregisterService(in serviceType : Class<?>)
+getService(in serviceType : Class<? extends T>) : T

Figure 7.7: Service locator class diagram.

Some parts of the application use settings to control their behavior. For example, the GraphicsDisplay needs to know the resolution it should run at. These settings are read from the Configuration class through the IConfiguration interface, as shown in Figure 7.8. The Configuration contains a set of properties that is the combination of the information found in "configuration.xml" and the command-line arguments. The "configuration.xml" file is XML formatted, using the Java properties Document Type Definition (DTD) scheme [22]. The last version of this XML schema was released January 19, 2005 [1]. At application startup an instance of the Configuration class is created and registered as a service on the service locator. This allows the rest of the application to read settings without defining where the settings are located. In addition to this, it allows easy implementation of other ways to read configurations, without having to reimplement large parts of the application. An example of a future use would be to add a configuration window to the application.

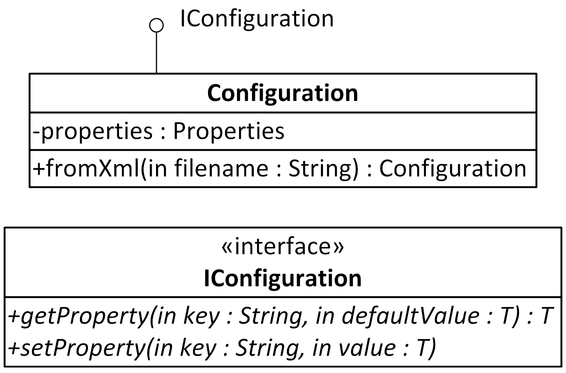

IConfiguration
Configuration
-properties : Properties
+fromXml(in filename : String) : Configuration
«interface»
IConfiguration
+getProperty(in key : String, in defaultValue : T) : T
+setProperty(in key : String, in value : T)

Figure 7.8: IConfiguration interface class diagram.

Rendering graphics are done with OpenGL in the Presentation Client. Since Java does not have a direct implementation of OpenGL, the application relies on a third-party library. This library handles the communication with OpenGL and the system files, and is dependent on the host OS. There are several different libraries available for accessing OpenGL from Java. The library we are currently using is named LWJGL. The rendering of the GUI is done by the GraphicsRenderer. The GraphicsRenderer implements the IRenderer interface, as shown in Figure 7.9. This allows the application to render graphics without the knowledge of how to interact with OpenGL.

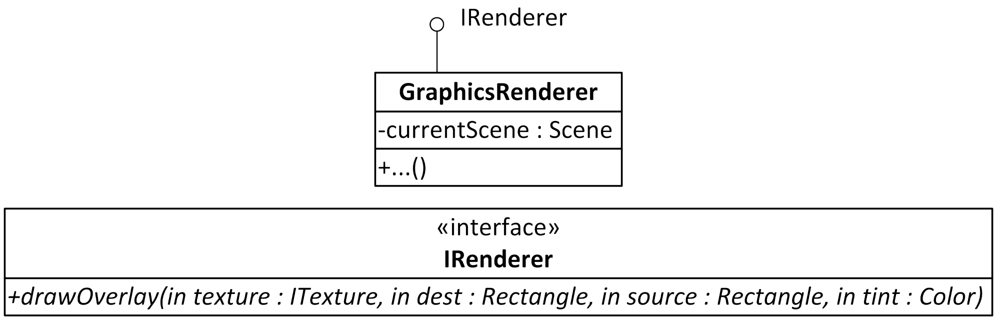

IRenderer
GraphicsRenderer
-currentScene : Scene
+...()
«interface»
IRenderer
+drawOverlay(in texture : ITexture, in dest : Rectangle, in source : Rectangle, in tint : Color)

Figure 7.9: IRenderer interface class diagram.

The application uses graphics loaded from the file system to render the GUI. To be able to render images, OpenGL needs to create a reference to the image data. This allows it to use hardware acceleration if available, by sending the image data to the graphics processing unit (GPU). Images loaded by OpenGL, are referred to as textures. To allow for easy change of the graphics library, the application uses an abstraction layer when loading graphics. The GraphicsManager creates a TextureLoader instance at initialization. The TextureLoader is used to load graphics into OpenGL, and is registered with the service locator through the ITextureLoader interface. Loading a texture with the TextureLoader will return an ITexture, as shown in Figure 7.10. This texture object is used when calling draw methods on the IRenderer interface to represent the graphics you want drawn to screen.

Usability is a quality goal for the Presentation Client. This affects the GUI, which is made up of components and scenes. A component can be an image or a textbox, while a scene is the composition of components that represent a screen in the game. The component structure implements the composite pattern, as shown in Figure 7.11. Implementing the composite pattern lets clients treat individual objects and compositions uniformly [32]. The Scene class inherits from the Component class. Classes extending Scene needs to implement the initializeComponents() method. By changing the implementation of this method, a developer can change the composition of the GUI. This supports modifiability quality requirement M6 "Change GUI composition", as seen in Table 6.9. This method is used to create the composition of GUI components for that scene. This enables treatment of scenes in the same way as components, and will affect the whole screen. For example, the fading transition between scenes is set as an effect on the scene, which affects every component on that scene.

To make the visuals more appealing to the end-user, the GUI components use render effects. A render effect is set on a component, and affects the default draw()

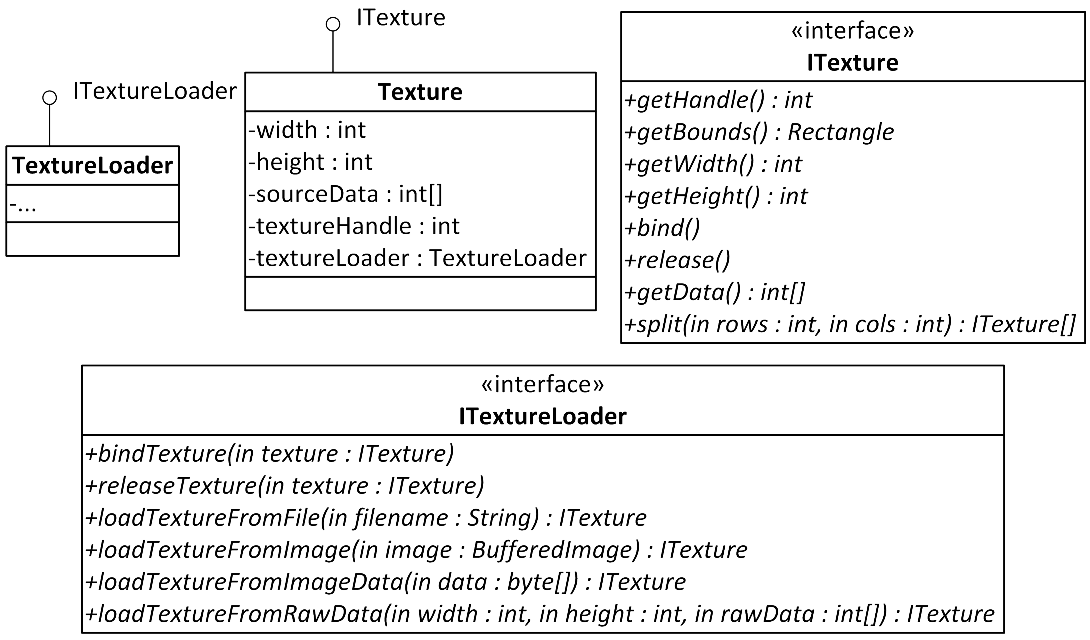

ITextureLoader
TextureLoader
-...
ITexture
Texture
-width : int
-height : int
-sourceData : int[]
-textureHandle : int
-textureLoader : TextureLoader
«interface»
ITexture
+getHandle() : int
+getBounds() : Rectangle
+getWidth() : int
+getHeight() : int
+bind()
+release()
+getData() : int[]
+split(in rows : int, in cols : int) : ITexture[]
«interface»
ITextureLoader
+bindTexture(in texture : ITexture)
+releaseTexture(in texture : ITexture)
+loadTextureFromFile(in filename : String) : ITexture
+loadTextureFromImage(in image : BufferedImage) : ITexture
+loadTextureFromImageData(in data : byte[]) : ITexture
+loadTextureFromRawData(in width : int, in height : int, in rawData : int[]) : ITexture

Figure 7.10: TextureLoader and Texture class diagram.

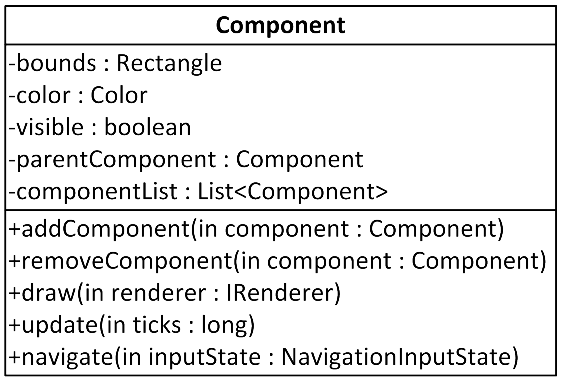

Component
-bounds : Rectangle
-color : Color
-visible : boolean
-parentComponent : Component
-componentList : List<Component>
+addComponent(in component : Component)
+removeComponent(in component : Component)
+draw(in renderer : IRenderer)
+update(in ticks : long)
+navigate(in inputState : NavigationInputState)

Figure 7.11: GUI component class diagram.

method. When the GraphicsRenderer calls the draw() method on the current scene it passes itself as an IRenderer. If a component has a render effect set, it will inject the IRenderer passed to the draw() method into the render effect, as shown in Figure 7.12. The render effect is then used for drawing this component, and its sub components. Since the RenderEffect implements the IRenderer interface it is possible to chain multiple effects. This structure works like a pipe-and-filter pattern, which allows for multiple render effects [8]. For example, chaining a ShadowEffect with a PulseEffect will give a pulsing component with a shadow. When changing scenes, we use the FadeInEffect and FadeOutEffect, to create a transition. These effects are timed so that the SceneHandler will know when it is finished and can change the scene.

GUI navigation is done by the SceneNavigator class. Components contain a navi-

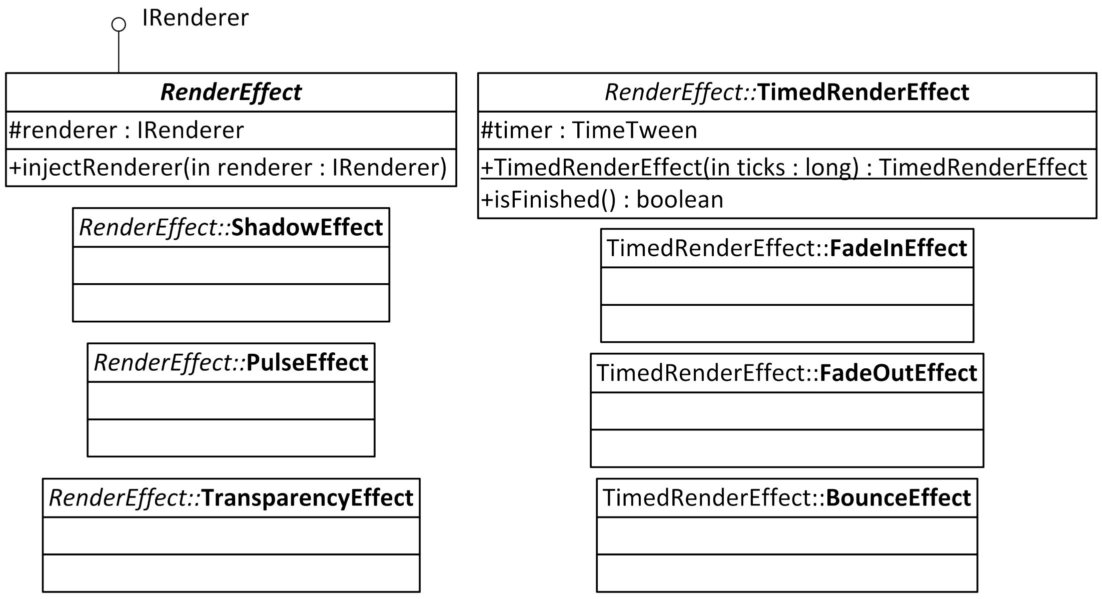

IRenderer
RenderEffect
#renderer : IRenderer
+injectRenderer(in renderer : IRenderer)
RenderEffect::TimedRenderEffect
#timer : TimeTween
+TimedRenderEffect(in ticks : long) : TimedRenderEffect
+isFinished() : boolean
RenderEffect::ShadowEffect
RenderEffect::PulseEffect
RenderEffect::TransparencyEffect
TimedRenderEffect::FadeInEffect
TimedRenderEffect::FadeOutEffect
TimedRenderEffect::BounceEffect

Figure 7.12: GUI render effects class diagram.

gation() method to allow them to run logic based on input device states, as shown in Figure 7.11. Extending the NavigableComponent instead of the Component class will give a component the default navigation methods, as shown in Figure 7.13. By using the template pattern, it is easy for a developer to add new components to the GUI. The template method pattern is a behavioral design pattern which defines a program skeleton of an algorithm [64]. The NavigableComponent implements the ISelectable interface. This interface allows the component to maintain focus on the screen. Selections are controlled by the SceneNavigator class, and it uses the ISelectable interface, as shown in Figure 7.13. By using an interface to control selection, it will be available for any type of object.

The InputManager handles the implementation of input devices. At application launch the InputManager will create instances of the LWJGLMouse and LWJGLKeyboard, as shown in Figure 7.14. The LWJGLMouse class is an implementation of the mouse functions of LWJGL, which can be utilized through the IPointerDevice interface. The IPointerDevice interface allows the Presentation Client to use different implementations of pointer devices, without changing the usage logic. The same logic applies for the LWJGLKeyboard, which is an implementation of the keyboard functions of LWJGL. The LWJGLKeyboard class uses the IInputDevice interface to allow the Presentation Client to access keyboard states. In the Presentation Client, input device states are polled instead of being event-driven. This means that the components that depend on input, needs to retrieve the input device state from the individual device. This is done through the IInputDevice interface's getState() method, for input devices, and through the IPointerDevice interface's getState method, for pointers. An input device will return an InputDeviceState object, which contains fields for: left, right, up, down,

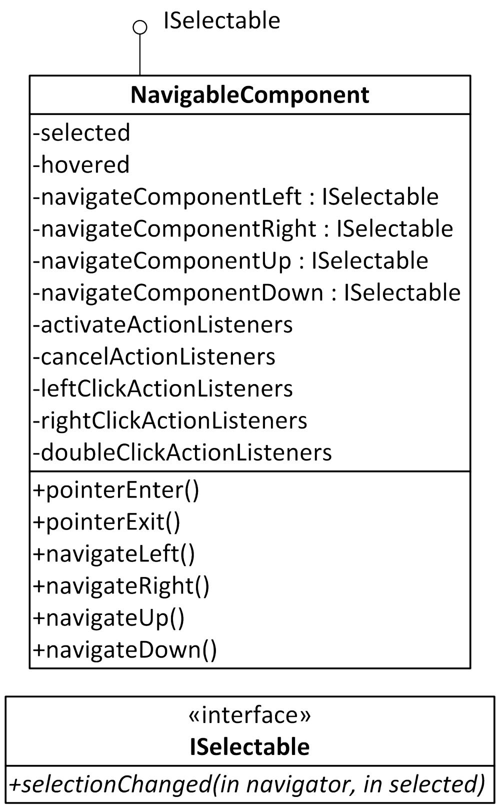

ISelectable
NavigableComponent
-selected
-hovered
-navigateComponentLeft : ISelectable
-navigateComponentRight : ISelectable
-navigateComponentUp : ISelectable
-navigateComponentDown : ISelectable
-activateActionListeners
-cancelActionListeners
-leftClickActionListeners
-rightClickActionListeners
-doubleClickActionListeners
+pointerEnter()
+pointerExit()
+navigateLeft()
+navigateRight()
+navigateUp()
+navigateDown()
«interface»
ISelectable
+selectionChanged(in navigator, in selected)

Figure 7.13: NavigableComponent class diagram with ISelectable interface.

accept and cancel, as seen in Figure 7.14. This state is used in the Presentation Client to move focus from one component to another and navigate back and forth between scenes. A pointer device will return an InputPointerState that contains values for the location of the pointer, which buttons have been pressed, and mouse wheel change. This is used to point and control the GUI in the Presentation Client and is created to keep things simple for common users. To add support for new input devices in the Presentation Client; it will require the implementation to implement either the IInputDevice interface, or the IPointerDevice interface. This will allow the developer to implement a new device without changing the usage logic. This fulfills the modifiability quality requirement M7, as seen in Table 6.10.

The InputDeviceContainer class, provides storage of input devices. This class is created and populated with input devices when the application starts, by the InputManager. The InputDeviceContainer implements the IInputDeviceContainer interface, as seen in Figure 7.15. The created instance of the InputDeviceContainer is registered on the service locator with the IInputDeviceContainer interface. This gives components access to input devices and input logic. By changing the devices in the InputDeviceContainer, one can control which devices that are currently controlling the Presentation Client.

The Presentation Client uses a Game object to represent the current game. An in-

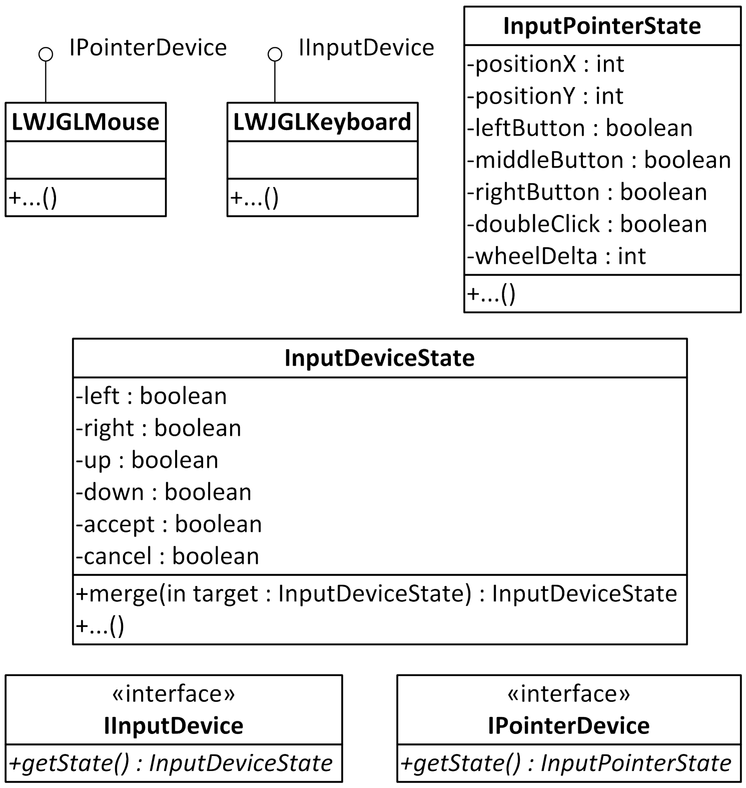


Figure 7.14: Input devices and device states class diagram.

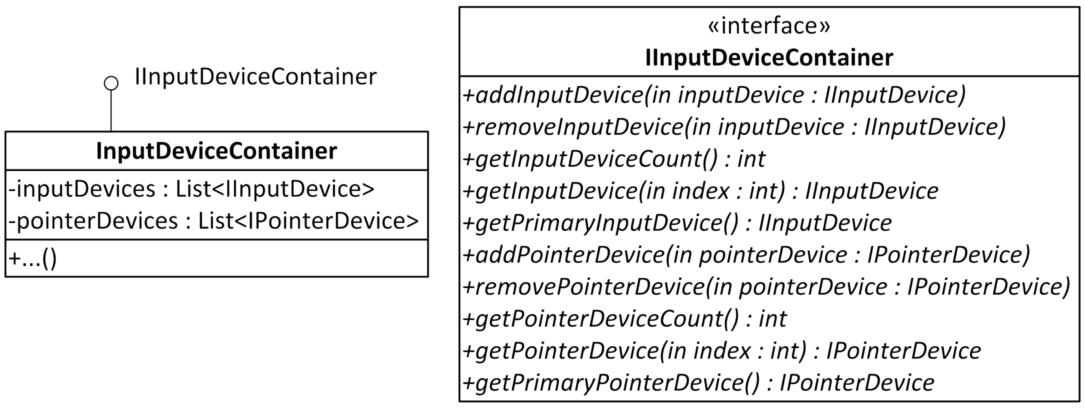

IInputDeviceContainer
InputDeviceContainer
-inputDevices : List<IInputDevice>
-pointerDevices : List<IPointerDevice>
+...()
«interface»
IInputDeviceContainer
+addInputDevice(in inputDevice : IInputDevice)
+removeInputDevice(in inputDevice : IInputDevice)
+getInputDeviceCount() : int
+getInputDevice(in index : int) : IInputDevice
+getPrimaryInputDevice() : IInputDevice
+addPointerDevice(in pointerDevice : IPointerDevice)
+removePointerDevice(in pointerDevice : IPointerDevice)
+getPointerDeviceCount() : int
+getPointerDevice(in index : int) : IPointerDevice
+getPrimaryPointerDevice() : IPointerDevice

Figure 7.15: InputDeviceContainer and IInputDeviceContainer interface class diagram.

stance of the Game class is created each time a new game starts. The game object will have a set of game modes assigned to it. It will also contain the selected quiz, with the questions. When the Game's start() method is called, the application creates a GameModeQuestionChain, as shown in Figure 7.16. A GameModeQuestionChain object contains a game mode and questions to be run on that game mode. The object also contains a reference to the next GameModeQuestionChain. This way of chaining game modes with questions allows the game to run while there are game modes left, and a game mode to run while it has questions left. When there are no more game modes, the game will end. By enabling chaining of game modes, the Presentation Client will be able to run multiple game modes for one game. This fulfills the modifiability quality requirement M2, as seen in Table 6.5.

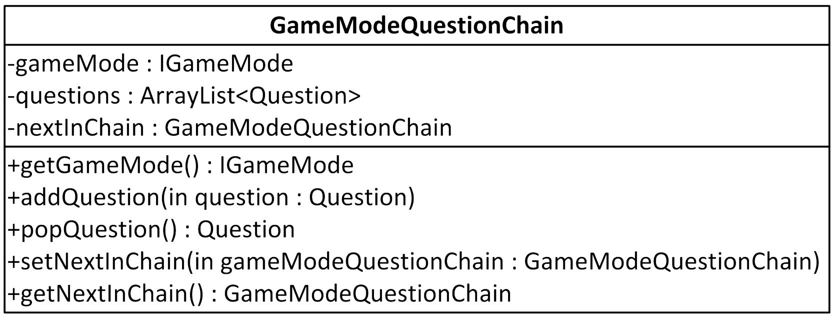

GameModeQuestionChain
-gameMode : IGameMode
-questions : ArrayList<Question>
-nextInChain : GameModeQuestionChain
+getGameMode() : IGameMode
+addQuestion(in question : Question)
+popQuestion() : Question
+setNextInChain(in gameModeQuestionChain : GameModeQuestionChain)
+getNextInChain() : GameModeQuestionChain

Figure 7.16: GameModeQuestionChain class diagram.

Game modes implement the IGameMode interface. This allows developers to create game modes to control game logic in Lecture Quiz. The current game modes extend the GameModeBase class, as shown in Figure 7.17. The GameModeBase class handles general logic to run a game mode. It takes care of creating rounds for each question and changing to the correct scene. The GameModeBase class also includes methods that can be overridden to change their behavior. E.g. overriding the changeTeamScore() method allows the game mode to control how points are awarded. Having this template makes it easier for developers to create game modes. The IGameMode interface is the foundation for creating new game modes. With the GameModeBase class, this should satisfy the modifiability quality requirement specified in M1, as seen in Table 6.4.

7.3.2 Player Client

The Player Client is a library used by the Presentation Client to allow communication with players. The communication is done through an embedded HTTP server using the Java HttpServer class. The HttpServer class provides a simple high-level HttpServer API, which can be used to build embedded HTTP servers [22]. For each request to the web server, a HttpExchange object is created. This object contains request information and methods for forming responses. Classes that im-

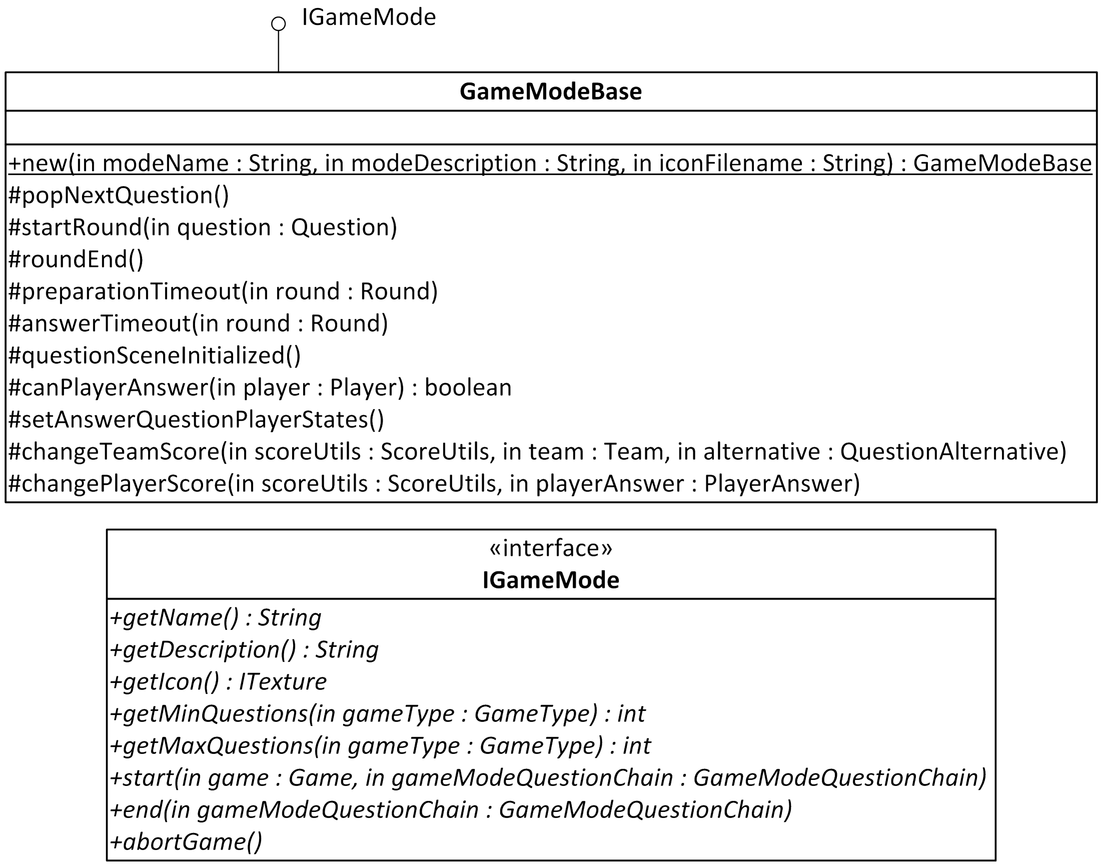

IGameMode
GameModeBase
+new(in modeName : String, in modeDescription : String, in iconFilename : String) : GameModeBase
#popNextQuestion()
#startRound(in question : Question)
#roundEnd()
#preparationTimeout(in round : Round)
#answerTimeout(in round : Round)
#questionSceneInitialized()
#canPlayerAnswer(in player : Player) : boolean
#setAnswerQuestionPlayerStates()
#changeTeamScore(in scoreUtils : ScoreUtils, in team : Team, in alternative : QuestionAlternative)
#changePlayerScore(in scoreUtils : ScoreUtils, in playerAnswer : PlayerAnswer)
«interface»
IGameMode
+getName() : String
+getDescription() : String
+getIcon() : ITexture
+getMinQuestions(in gameType : GameType) : int
+getMaxQuestions(in gameType : GameType) : int
+start(in game : Game, in gameModeQuestionChain : GameModeQuestionChain)
+end(in gameModeQuestionChain : GameModeQuestionChain)
+abortGame()

Figure 7.17: GameModeBase class diagram.

plement the HttpHandler interface can be assigned to the HttpServer, to receive these HttpExchange objects. The Player Client uses this architecture to handle communication with the players' client devices.

A PlayerClient is created with the static createPlayerClient() method, as shown in Figure 7.18. This will start the embedded web server, and allow player access. The PlayerClient implements the IPlayerClient interface, as seen in Figure 7.18. This interface makes it easy for other applications to create test versions of the Player Client. Being able to test user input has been an important factor in our development. The PlayerClient object uses a PlayerManager object to keep track of players. This logic is made as simple as possible, and it is up to the application running the PlayerClient to apply the game logic. By using the IPlayerClient interface, a developer can create a new player client. To make the Presentation Client use the new client, the developer only needs to register it on the service locator. This enables the modifiability quality requirement M3 to be fulfilled, as seen in Table 6.6.

The Player Client uses sessions to keep track of which clients have authenticated and to respond uniquely to each client. Since sessions and cookies are not directly supported by the Java HttpServer, the Player Client uses the SessionManager to

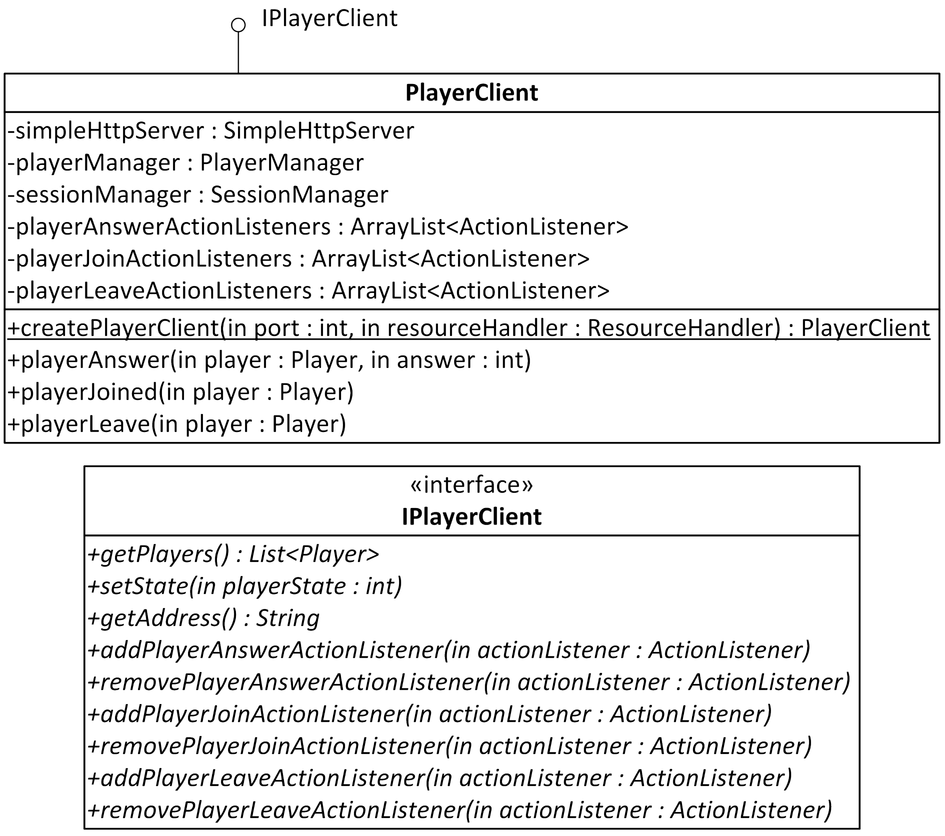

IPlayerClient
PlayerClient
-simpleHttpServer : SimpleHttpServer
-playerManager : PlayerManager
-sessionManager : SessionManager
-playerAnswerActionListeners : ArrayList<ActionListener>
-playerJoinActionListeners : ArrayList<ActionListener>
-playerLeaveActionListeners : ArrayList<ActionListener>
+createPlayerClient(in port : int, in resourceHandler : ResourceHandler) : PlayerClient
+playerAnswer(in player : Player, in answer : int)
+playerJoined(in player : Player)
+playerLeave(in player : Player)
«interface»
IPlayerClient
+getPlayers() : List<Player>
+setState(in playerState : int)
+getAddress() : String
+addPlayerAnswerActionListener(in actionListener : ActionListener)
+removePlayerAnswerActionListener(in actionListener : ActionListener)
+addPlayerJoinActionListener(in actionListener : ActionListener)
+removePlayerJoinActionListener(in actionListener : ActionListener)
+addPlayerLeaveActionListener(in actionListener : ActionListener)
+removePlayerLeaveActionListener(in actionListener : ActionListener)

Figure 7.18: PlayerClient class diagram with IPlayerClient interface.

handle these tasks. The SessionManager uses cookies to create a unique identifier that associates the client with a session. At the beginning of each HTTP request, the SessionManager creates the current session based on the HttpExchange object. It uses the getSession() method to get the session, and if a session does not exist it will create one, as shown in Figure 7.19. By using sessions in this way, the application can store information about the client.

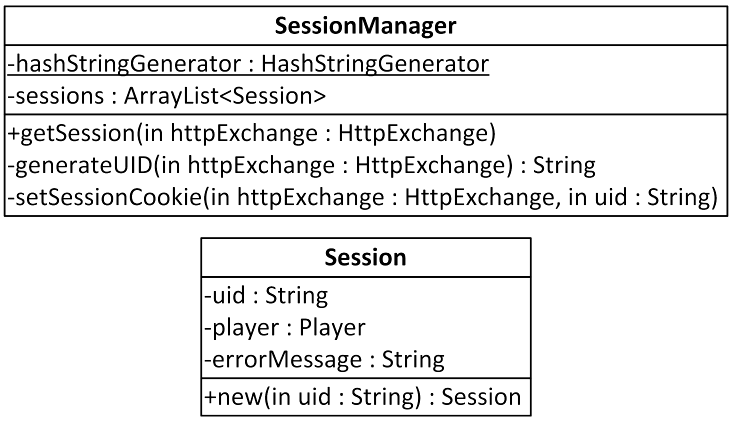

SessionManager
-hashStringGenerator : HashStringGenerator
-sessions : ArrayList<Session>
+getSession(in httpExchange : HttpExchange)
-generateUID(in httpExchange : HttpExchange) : String
-setSessionCookie(in httpExchange : HttpExchange, in uid : String)
Session
-uid : String
-player : Player
-errorMessage : String
+new(in uid : String) : Session

Figure 7.19: SessionManager class diagram.

The Player Client is used to present the client interface to end-users. The GUI needs to be loaded from somewhere, this is where the ResourceHandler comes into play. The Player Client has an abstract ResourceHandler class, as shown in Figure 7.20. Creating an instance of the PlayerClient class is done through the static createPlayerClient() method, as shown in Figure 7.18. This method requires a ResourceHandler object, which is the object that will allow access to client resources. By using an abstract class, the Player Client does not need to know from where the resources are loaded. To extend the ResourceHandler class, the target needs to create an implementation of the getMimeType() and the getData() methods, as these are abstract. The resource loading can just as easily be changed to load resources from, e.g., the Quiz Server. The current implementation uses a FolderResourceHandler to load resources, as shown in Figure 7.21. This class requires a parameter, a path to a folder on the local file system, from where it will load its resources. By using this class, the application can easily change the source of the files, e.g. by defining unique folders to different client skins. This fulfills the modifiability quality requirement M4, as seen in Table 6.7.

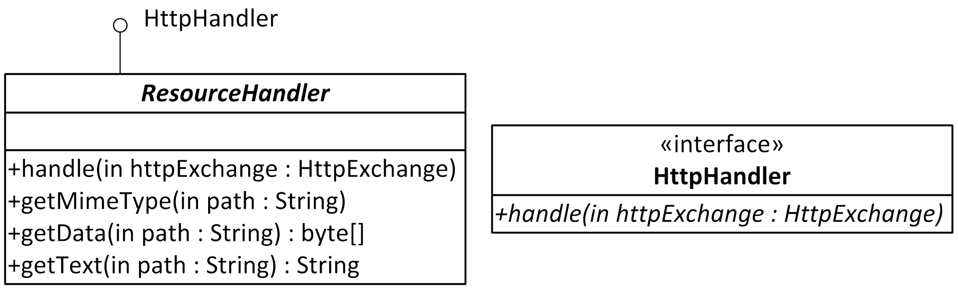

HttpHandler
ResourceHandler
+handle(in httpExchange : HttpExchange)
+getMimeType(in path : String)
+getData(in path : String) : byte[]
+getText(in path : String) : String
«interface»
HttpHandler
+handle(in httpExchange : HttpExchange)

Figure 7.20: ResourceHandler class diagram.

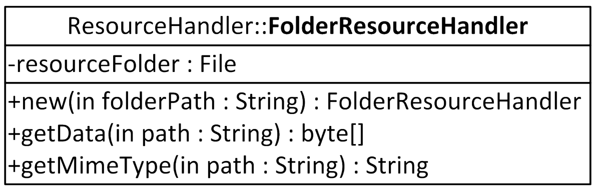

ResourceHandler::FolderResourceHandler
-resourceFolder : File
+new(in folderPath : String) : FolderResourceHandler
+getData(in path : String) : byte[]
+getMimeType(in path : String) : String

Figure 7.21: FolderResourceHandler class diagram.

7.3.3 Quiz Server

The architectural design of the Quiz Server is based on the model-view-controller (MVC) pattern. MVC is an architectural pattern which isolates “domain logic” from the user interface, permitting independent development, testing and maintenance of each separation of concerns [57]. This is implemented by using the Java Server Faces (JSF) framework. JSF is a request-driven MVC web framework, and was described in Chapter 5, Section 5.3. To achieve the MVC architecture; we use Entities as models, ManagedBeans as controllers, and .xhtml facelets to represent the view. An example of our MVC implementation, is the quiz presentation structure, as shown in Figure 7.22.

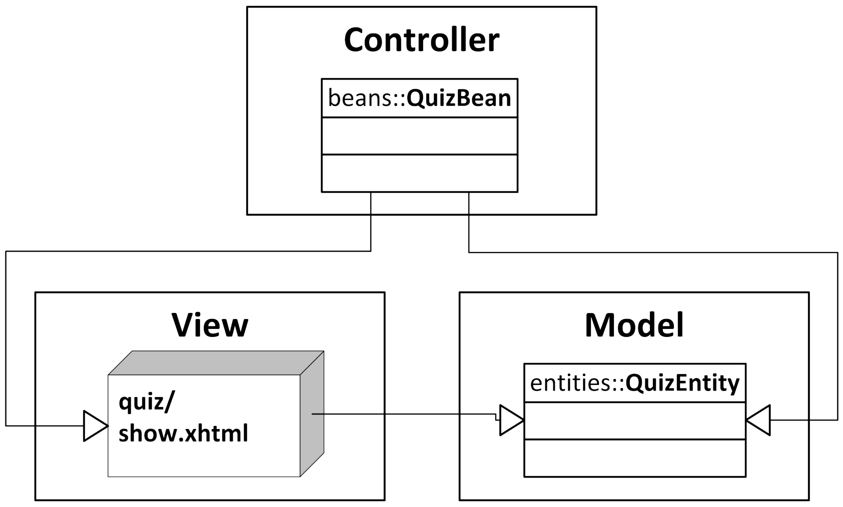


Figure 7.22: Quiz Server's MVC communication diagram.

The Quiz Server does not use a direct implementation of a database back-end. To manage information it uses Java Persistance API (JPA) with entities. Each Entity represents information needed to be stored by the Quiz Server, as shown in Figure 7.23. By using JPA and entities, the application does not need to handle the database back-end directly. A developer can configure JPA to use a different database without changing the implementation. This fulfils the modifiability quality requirement M5, as seen in Table 6.8. After an Entity has been created, it can be treated as a regular data model. When changing a value on an Entity, JPA will make sure the database reflects this change [17].

7.3.4 Distributed Message Queue

The Distributed Message Queue can either be run in client or server mode. Client/server is often a generic umbrella term for any application architecture that divides processing among two or more processes, often on two or more machines [58]. This means that the application using the Distributed Message queue has to choose either modes. To create an instance of the Distributed Message Queue, the application needs to use the DistributedMessageQueueFactory, as seen in Figure 7.24. The DistributedMessageQueueFactory has two methods, one for each mode. By using this factory, the Distributed Message Queue can isolate construction logic from the creator. The DistributedMessageQueue object uses several threads and socket injections, which needs to be disposed when the application closes. To dispose the DistributedMessageQueue, the application should use the recycle() method on the DistributedMessageQueueRecycler class, as seen in Figure 7.24. The Distribut-

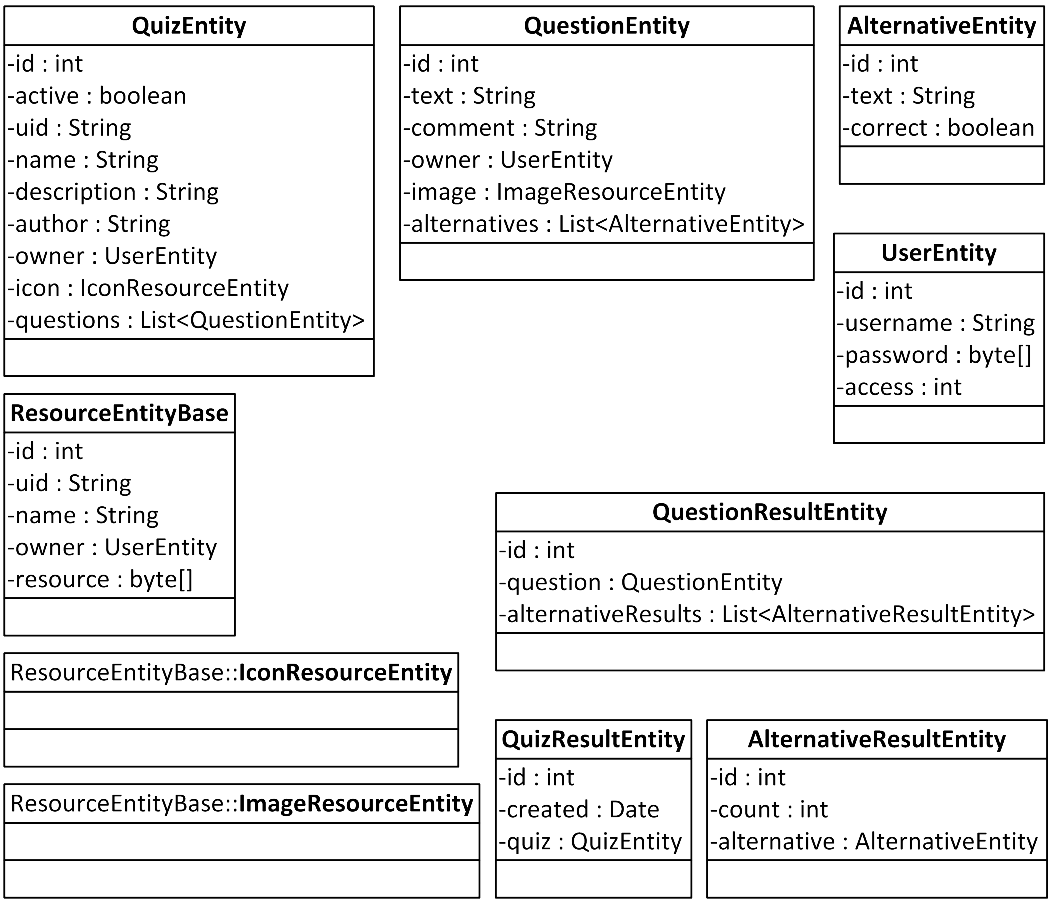


Figure 7.23: Entities class diagram.

edMessageQueueRecycler will inform of shutdown and close threads correctly. By using patterns for construction and destruction, the Distributed Message Queue can be used as a regular message queue through the IMessageQueue interface.

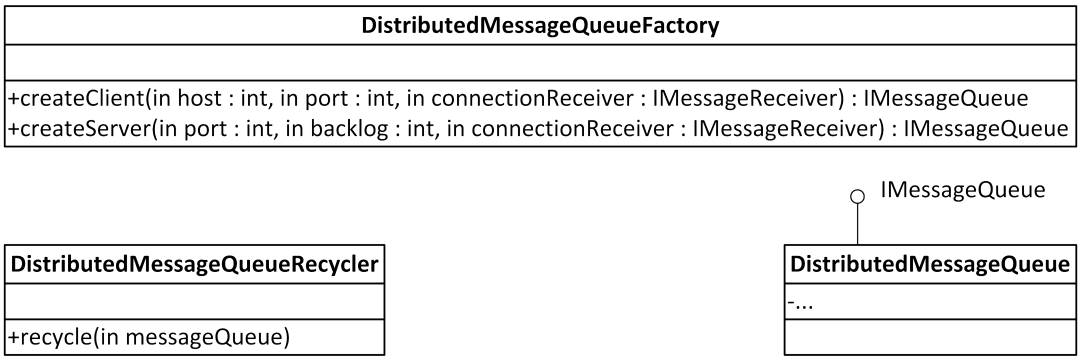

DistributedMessageQueueFactory
+createClient(in host : int, in port : int, in connectionReceiver : IMessageReceiver) : IMessageQueue
+createServer(in port : int, in backlog : int, in connectionReceiver : IMessageReceiver) : IMessageQueue
IMessageQueue
DistributedMessageQueueRecycler
+recycle(in messageQueue)
DistributedMessageQueue
-...

Figure 7.24: Distributed Message Queue Factory class diagram.

The Distributed Message Queue can be utilized through the IMessageQueue interface. The IMessageQueue interface contains methods for adding, removing and queuing messages, as shown in Figure 7.25. Classes used for receiving messages needs to implement the IMessageReceiver interface, as seen in Figure 7.25. If a class wants to receive a message, it has to register itself and the respective message type with the message queue. This will make the message queue call the incomingMessage() method on the receiver, if a message of that type is queued. Objects that are registered on the message queue are stored in a Java WeakMap, meaning that they will be garbage collected if they are not referenced somewhere else. This will make sure that the message queue does not contain unused objects, by keeping them alive. Lecture Quiz uses the message queue pattern for both local and distributed messages. The Lecture Quiz' message queue is based on the Java Message Service (JMS), and enables distributed communication that is loosely coupled, reliable, and asynchronous [33]. This means both local logic and distributed instances can be added or modified, with minimal change in the rest of the implementation.

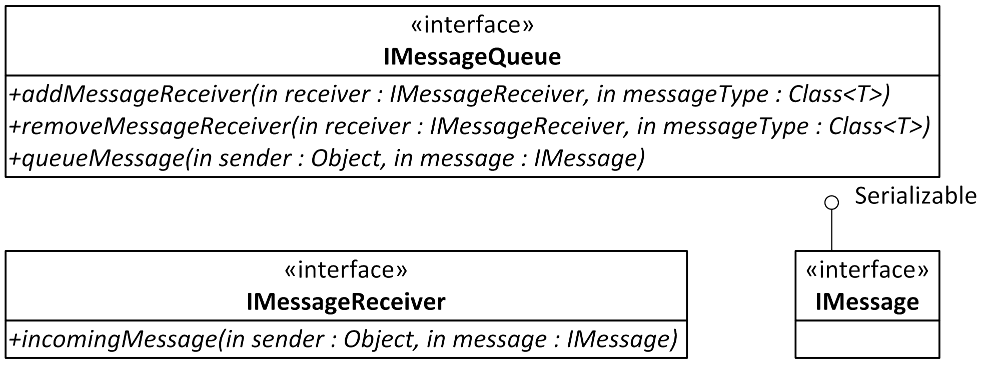

«interface»
IMessageQueue
+addMessageReceiver(in receiver : IMessageReceiver, in messageType : Class<T>)
+removeMessageReceiver(in receiver : IMessageReceiver, in messageType : Class<T>)
+queueMessage(in sender : Object, in message : IMessage)
Serializable
«interface»
IMessageReceiver
+incomingMessage(in sender : Object, in message : IMessage)
«interface»
IMessage

Figure 7.25: Message Queue interface class diagram.

7.4 Process View

The process view captures the concurrency and synchronization aspects of the design [39]. It shows how different parts of the system communicate and the process lifeline for important functions. This view is meant for integrators and developers.

7.4.1 Distributed Message Queue

A Distributed Message Queue can be seen as clients sending messages to a queue, where it will be picked up some time in the future by a remote system, and acted upon [9]. Instances of the Distributed Message Queue can be created with the DistributedMessageQueueFactory class, as shown in Figure 7.24. When creating the DistributedMessageQueue object, the factory will spawn a worker thread. This thread is used to process messages that are sent to the message queue, as shown in Figure 7.26. By using the worker thread, messages can be processed without blocking the rest of the application.

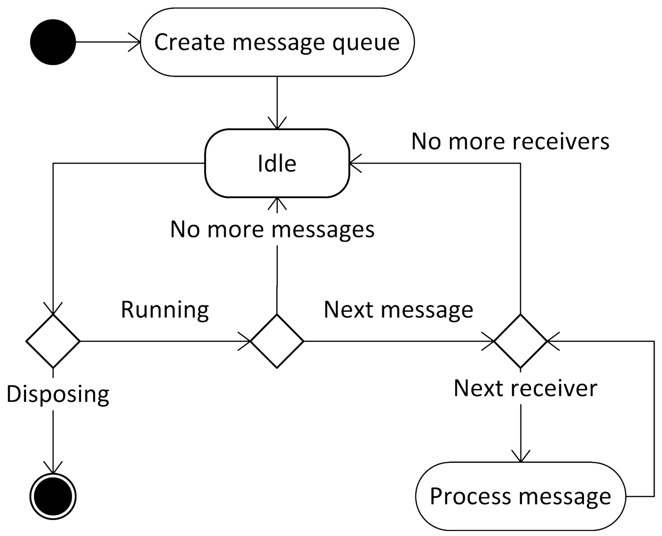


Figure 7.26: Distributed Message Queue worker thread activity diagram.

The Distributed Message Queue uses sockets to connect clients to a server. Message queues created in server mode will listen to a specified port for connections. When a new client connects to the server, the message queue sends a NewConnectionMessage to inform that a client has connected. The same is done for message queues that are running in client mode. The NewConnectionMessage is sent after it has successfully connected to the server. If the connection to the server is lost, the message queue will send a ConnectionLostMessage and try to reconnect, as shown in Figure 7.27. If the client is able to re-establish a connection to the server, the message queue will send another NewConnectionMessage. This makes it simple to implement logic involving server connection. The feature supporting automatic server reconnection, fulfills the availability quality requirement A1, as seen in Table 6.14.

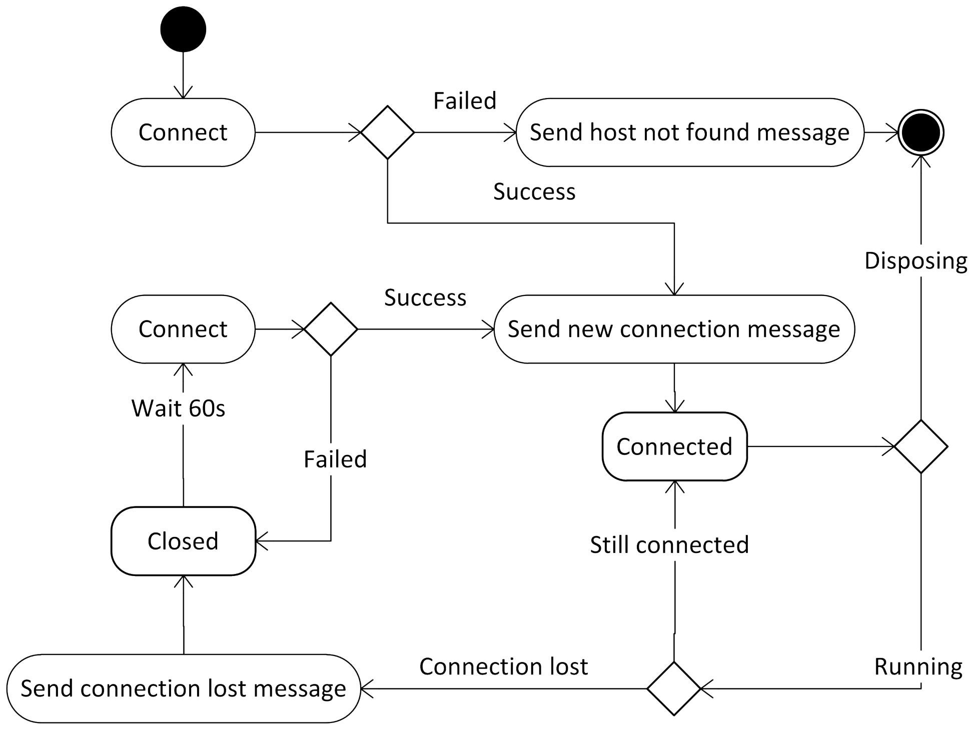


Figure 7.27: Distributed Message Queue socket reconnect activity diagram.

7.4.2 Player Client

The Player Client uses HTTP to communicate with players, which use a web browser to see the client interface. A player is represented by the Player model which has a state value. The state is used to specify what state of the game the player is in. Thus each state represents a different type of screen, on the Player Client. The client web page needs to notice changes in a player's state, to update the display. This is done by sending a request each second, to get the current state of the player, as shown in Figure 7.28. This request is done by a simple asynchronous JavaScript and XML (AJAX), and according to our experiment, should work on most web browsers. The fact that the web client is both simple and run on most standard web browsers, fulfills the availability quality requirement A3, shown in Table 6.16.

The Player Client uses threads to process web requests from players. To make sure the Player Client feels responsive to each player, the requests need to be processed by different threads. This is handled in the Player Client, by using a thread pool, as shown in Figure 7.29. The thread pool pattern is where a number of threads are

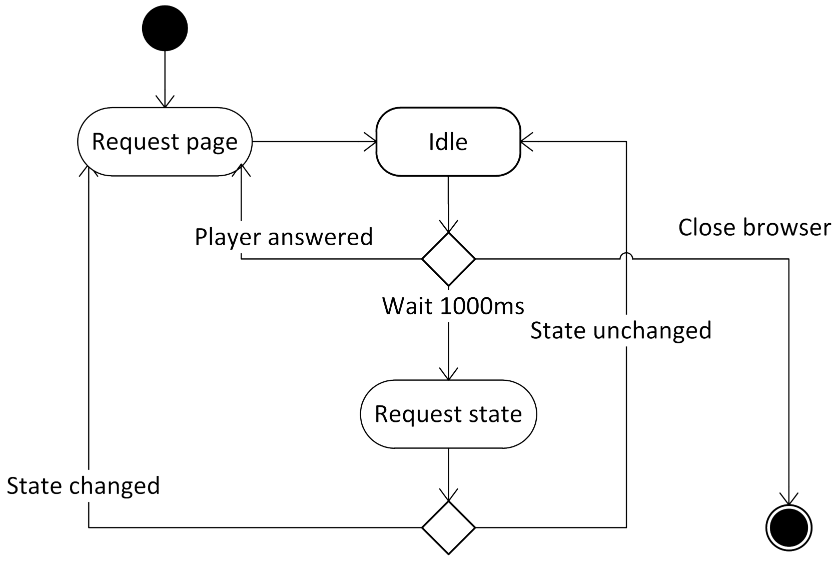


Figure 7.28: Player Client state request activity diagram.

created to perform a number of tasks, which are usually organized in a queue [41]. When a client request a page, the thread pool will check if there are any free threads. If available, it will use an existing thread, otherwise it spawns a new thread to process the request. By using the thread pool pattern, the thread count is decreased, compared to using one thread for each client. Threads that stay idle for too long will be disposed.

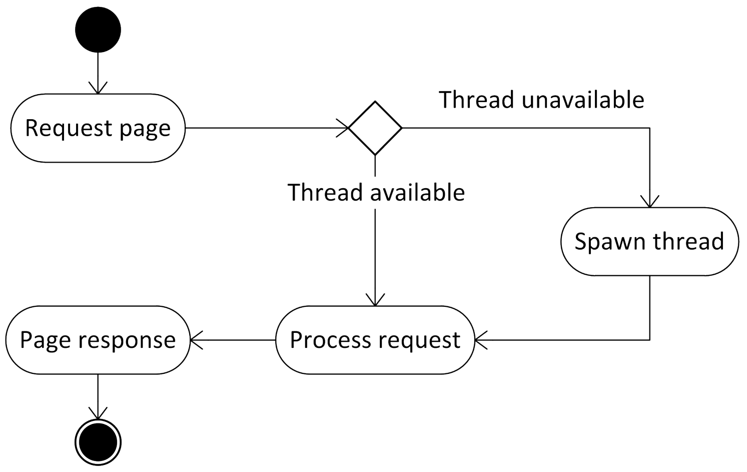


Figure 7.29: Player Client thread pool activity diagram.

7.4.3 Quiz Server

The Quiz Server stores images for quizzes and questions. These are presented as thumbnails on the Quiz Server web page, and is stored in a memory cache, as shown in Figure 7.30. This is to improve response time to clients. When clients request thumbnails, the Quiz Server looks up its memory cache. If the thumbnail exists in cache, the Quiz Server is relieved of getting it from the database. Otherwise the Quiz Server will create a thumbnail from the original image and store it to cache.

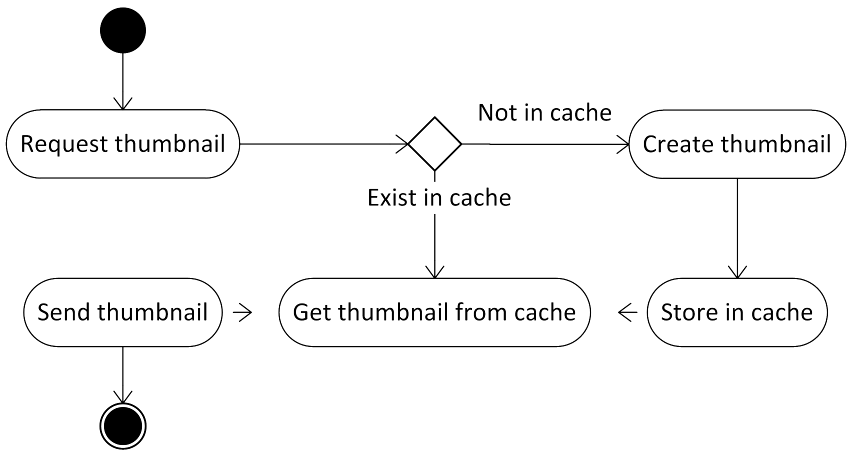


Figure 7.30: Quiz Server cache activity diagram.

Chapter 8

Design Choices

In this chapter, we describe our main design choices. The first section describes the choices made regarding the infrastructure of the Lecture Quiz solution. The second section presents visual effects and design choices.

8.1 Infrastructure

The Lecture Quiz infrastructure has been changed from the previous version, with focus on usability and modifiability. We want to make the system easy to use and flexible when it comes to extensions. This section includes the main changes in the Lecture Quiz infrastructure.

8.1.1 Enterprise Architecture

In the specialization project, we used a total of three application server deployables. This meant that the end-user had to install and configure at least one application server to be able to test the solution. In the quality requirements, we specify that we want our application to be easy to use. This should also apply to the setup and installation process of Lecture Quiz.

Our solution to this was to limit the number of applications that require an application server. The Quiz Server is the only application that depends on an application server in Lecture Quiz 3.0. The Presentation Client used the Game Server and the Player Web Client deployables in Lecture Quiz 2.5. We removed both for Lecture Quiz 3.0 and expanded the logic of the Presentation Client. This means that teachers are able to run quiz games without depending on any server.

8.1.2 Offline Support

The Presentation Client connects to the Quiz Server to retrieve quiz information. In Lecture Quiz 2.5, the quiz information was transferred over a web service. The web service was run on a separate server, meant that the communication was done in two steps. Lecture Quiz 3.0 supports direct communication between the Presentation Client and the Quiz Server. Unlike in Lecture Quiz 2.5, the Quiz Server will send quiz information as packages.

Lecture Quiz 3.0 uses two types of packages, quiz and statistics packages. A Quiz package is a package containing all the data necessary to run a quiz game. Instead of retrieving the quiz package at runtime, the lecturer can download quizzes as quiz packages on the Quiz Server. These packages can be played directly by the Presentation Client which makes the quiz available offline. One of the main purposes of running a lecture quiz game, is to collect statistical data. At the end of a game the lecturer can save statistics as a statistics package. This package can be uploaded to the Quiz Server for later review.

8.1.3 Communication Structure

In Lecture Quiz 2.5, different parts of the framework communicated through the Game Server. This was done as a service oriented architecture where the Game Server provided services to the other components. The service-oriented architecture (SOA) model is a distributed technology with distinct characteristics in support of realizing service-orientation and the strategic goals of service-oriented computing [26]. By using this architecture the Game Server needed to implement logic for each service. For the services used to communicate between two components, the logic was quite simple, but it still had to be defined by the Game Server. The communication to the Game Server was done as “one-way” request based communication. This made the communication between the Presentation Client and Player Web Client, and Presentation Client and Quiz Server cumbersome.

In Lecture Quiz 3.0 we changed the communication logic. The new communication logic utilizes a distributed message queue. We created the distributed message queue as a standalone library to make it available for each component. The distributed message queue is implemented as a regular message queue, where messages can be queued and received. One of the great advantages by using this logic is that the sender of a message does not need to know who will process the message. Compared to the Game Server in Lecture Quiz 2.5, we can add new components to Lecture Quiz without adjusting the behavior of any of the other components. For example, we can add a component that counts how many games have been played. The logic for such a component would be to listen for game messages being sent on the message queue. This would not require any change on the other components.

The most important feature that Lecture Quiz 3.0 gains from using the distributed message queue is the two-way communication. In Lecture Quiz 2.5 the clients could send requests to the Game Server, but the Game Server could not send information back, unless it was requested by the client. By having two way communication, the Presentation Client will now receive a message if a new quiz has been created on the server. This allows the Presentation Client to stay updated and the end-users can start using new quizzes instantly.

8.1.4 Data Storage

Lecture Quiz 2.5 utilize the JDBC API directly, to access a relation database. We have re-implemented all data storage logic to utilize JPA, which is described in Chapter 5. In addition to this we have changed the way we organize our resources, such as images, icons and ultimately sound and video, in later releases. The previous version managed this by storing resources to the file-system directly, but in this version we use JPA to handle resources as well. This removes the need to configure file-system access, and will make the setup process easier.

8.2 Visuals

One of our goals has been to make Lecture Quiz feel more like a game and not a statistics collection framework. By using an OpenGL rendered GUI environment, the presentation gets a more game like look and feel. A graphical artist helped us improve the graphics for Lecture Quiz. The two following sections include visual design choices, and how the graphics improved the usability of Lecture Quiz.

8.2.1 Presentation Client

The Presentation Client starts off with a splash screen, as shown in Figure 8.1. The splash screen contains the Lecture Quiz logo, and information about the developers. It is common in games, to present short information about the developers at startup.

The Presentation Client background includes some subtle texture elements. These give the impression of depth in the background, making it more interesting to look at. We added an ambient effect to the background to create movement and change in the application. To create the ambient effect, a timed color tint change is run on the background.

An important factor in making the Presentation Client look more game like, was the removal of frames. The initial version used framing of GUI components to create grouping of related elements. By removing the frames, the background and ambient effects are more visible to the users.

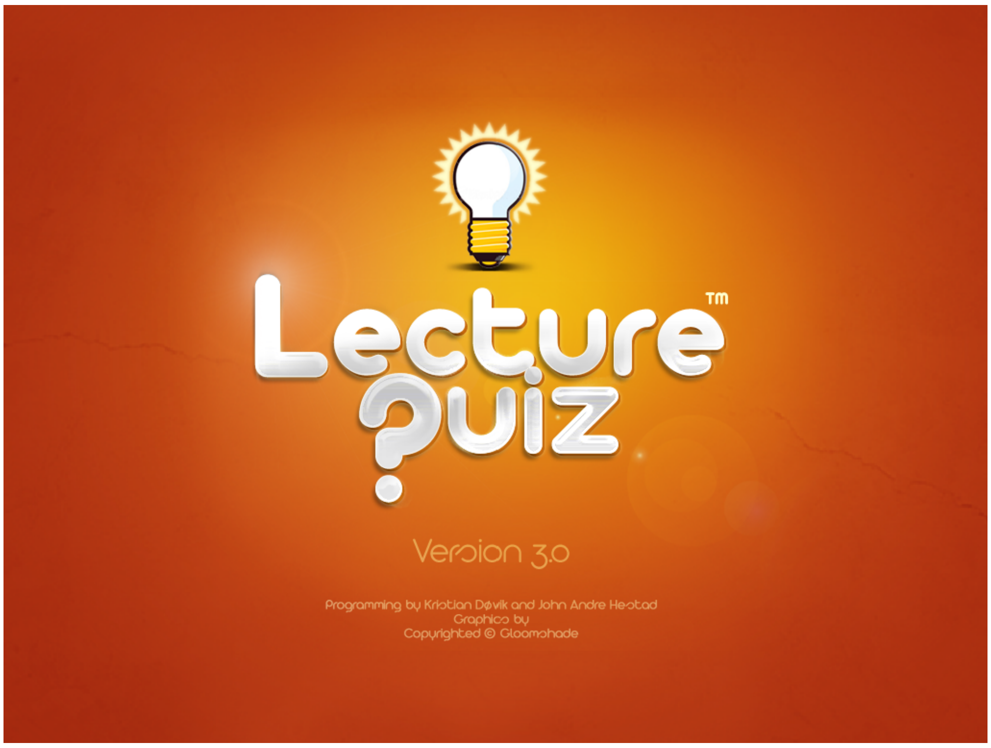


Figure 8.1: Presentation Client splash screen.

Instead of using radio buttons to select the game type, we have added graphical representations of what the user is selecting, as shown in Figure 8.2. This makes the selection more intuitive and gives a more professional game look. To indicate the selected game type, we have created an animated arrow pointing at the chosen element.

Setting up a game with the Presentation Client is done in four steps. These steps have been created to resemble an application wizard. To navigate between the different configuration screens, the Presentation Client uses navigation buttons. There are two buttons, one to take the user to the next step in the configuration, and one to take the user back to the previous step. These are located in the lower right part of each screen, and maintain the same position for easy access, as shown in Figure 8.2. The Presentation Client can display the "Next button" as disabled, which indicates that a configuration step is not complete.

Each screen in the Presentation Client has an underlying composition. This composition consists of a header, content and a footer. The header is at the upper section of the screen, and contains information about the screen you are currently viewing. The content section is at the center, and contains different elements depending on the current screen. The footer is at the bottom of the screen, and contains the navigation buttons and player login information. By using this composition, users can easily find the elements that are interesting.

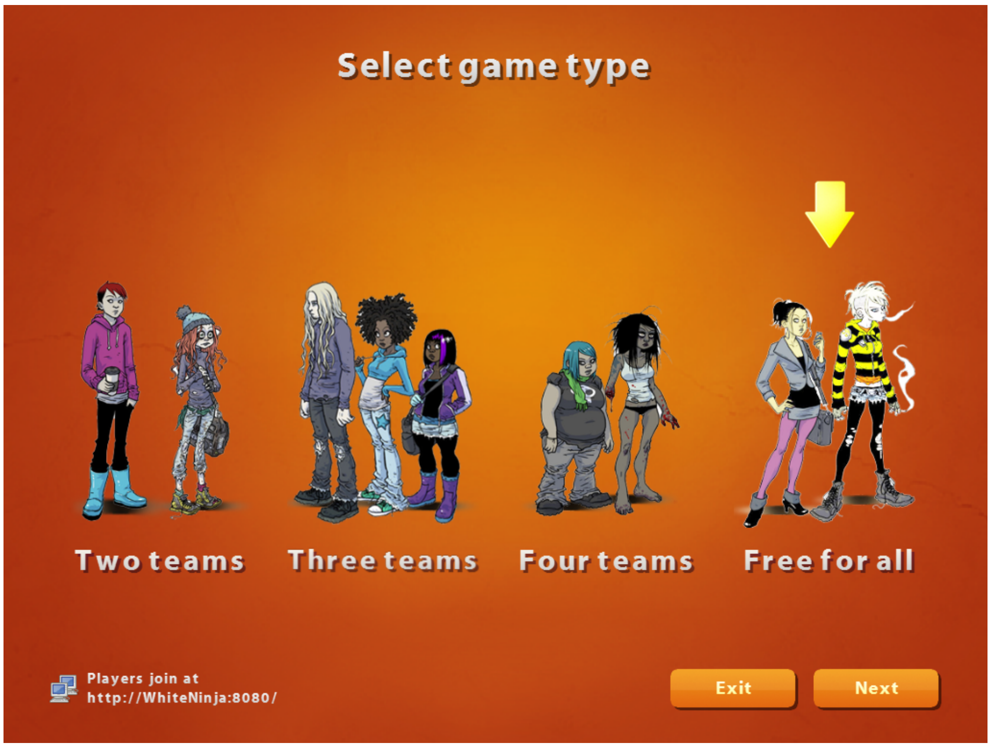


Figure 8.2: Presentation Client game type select screen.

Before starting a Lecture Quiz game, the Presentation Client displays the team selection screen. The screen contains a list of players, one for each team, as shown in Figure 8.3. Each team has an associated mascot, that is presented over the team's list. Lecture Quiz uses these mascots to represent the team on both the Presentation Client and the Player Client. Players can be allowed to select the team they want to play with. They do this by selecting the associated mascot on the Player Client, for more information see Section 8.2.2.

In the question screen, we have added component movement. At first, the question is presented at the top of the screen, with the question alternatives listed vertically below, as shown in Figure 8.4. When a question timer starts, the Presentation Client moves the question alternatives down to their respective locations, as shown in Figure 8.5. The reason why we chose to have the Presentation Client display the alternatives in a vertical list, is that it matches the display shown on the players' devices. Since there is no text on the Player Client buttons, we want players to associate the buttons with the alternatives presented on the Presentation Client. When the question timer runs out, the wrong alternatives turn grey and are scaled down eight pixels. We have created a pulsing effect, that is used to highlight the correct alternative.

An alarm clock will be displayed after the alternative buttons have been moved. We added the alarm clock to put some pressure on the players. The alarm clock is created by using combinations of six different images. One of the images is the

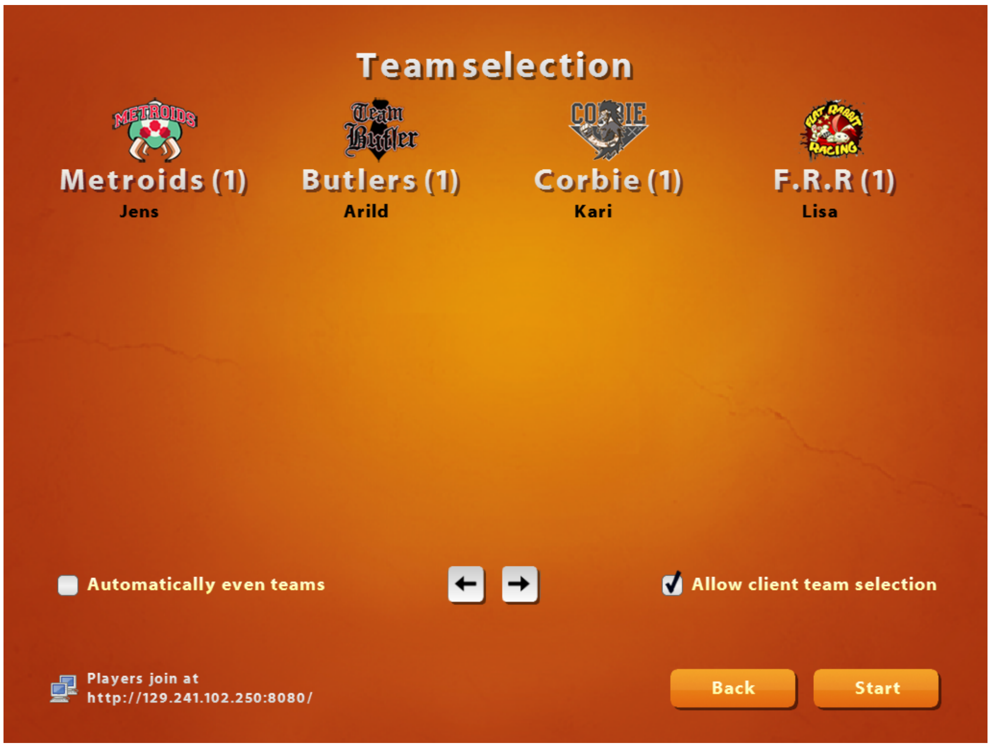


Figure 8.3: Presentation Client team selection screen.

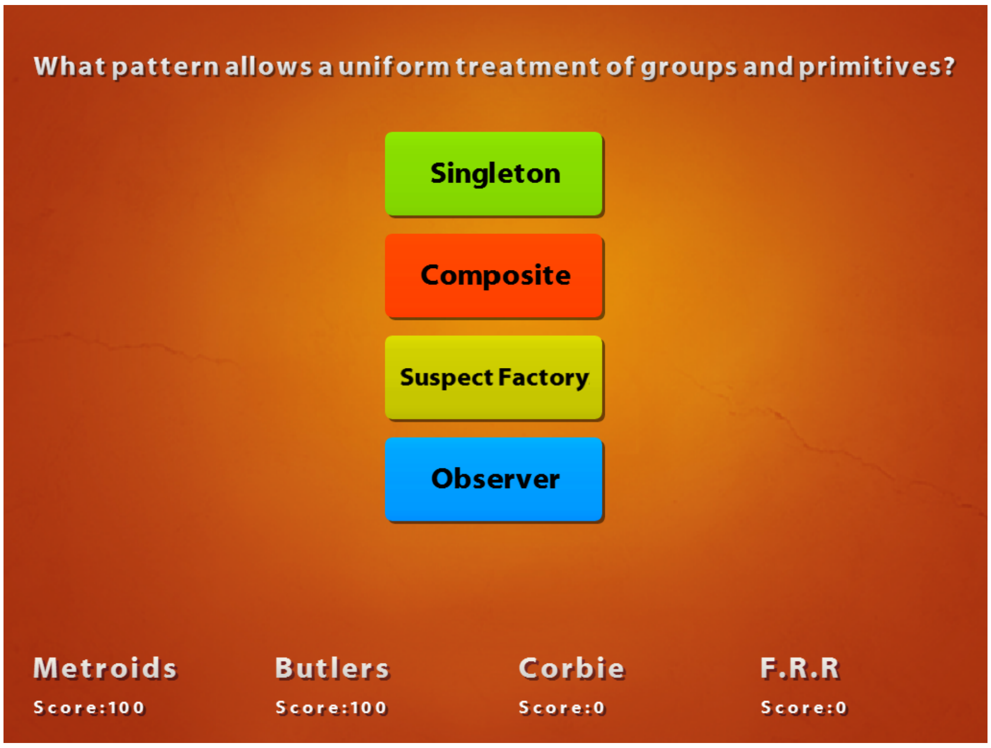


Figure 8.4: Presentation Client question screen before timer starts.

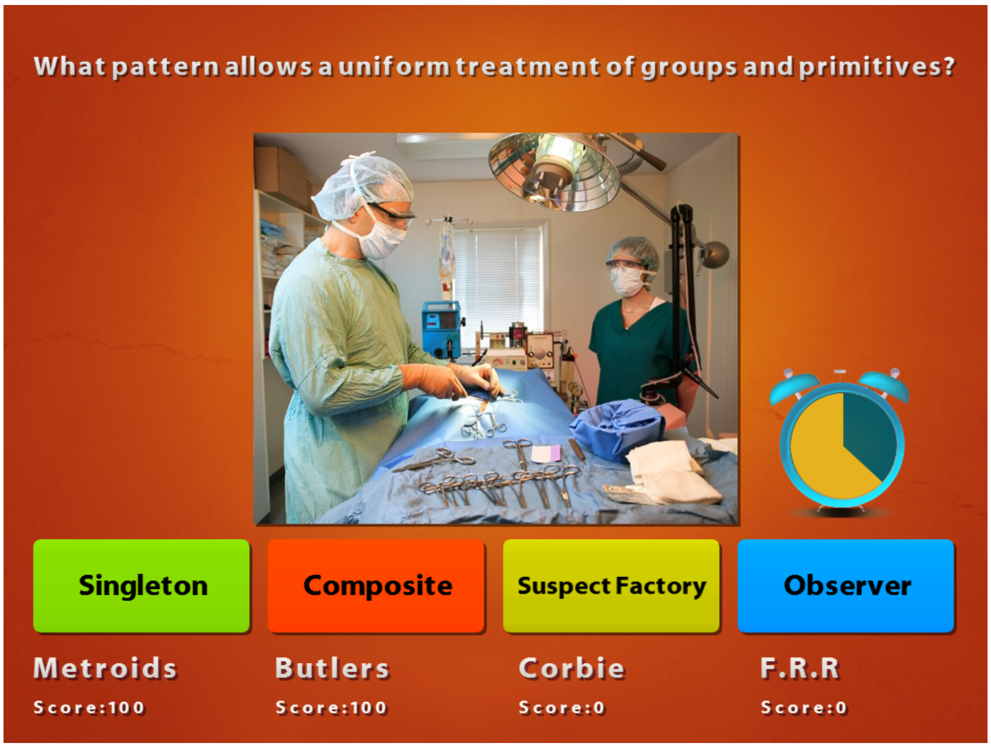


Figure 8.5: Presentation Client question screen.

animated pie countdown, which is used for fill the inside of the clock, as shown in Figure 8.5. When the question timer runs out, the clock will start to ring. This shows by shaking the alarm clock, the bells, and changing the image of the hammer (on top of the clock) to look like it is moving really fast. We have separated each of the bells from the clock, to be able to move them independently from the rest of the clock.

To create the feeling of depth in the GUI, the Presentation Client can apply a shadow effect to components. The shadow effect is performed by rendering the component without color, and shifting its location. Then the component is drawn with color on its original location. For example, the question image shown in Figure 8.5, uses the shadow effect.

8.2.2 Player Client

The Player Client is used by players when playing Lecture Quiz games. It uses HTML web interface, suited for both point and click, and touch input. To join a Lecture Quiz game, a player needs to specify a username. This is done in the first screen, which is the login screen, shown in Figure 8.6. The login screen contains the Lecture Quiz logo and developer info, similar to the splash screen in the Presentation Client. This should be easily recognizable by the player to avoid confusion.

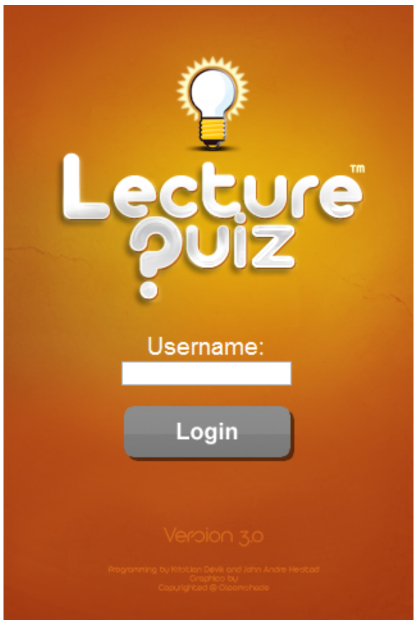


Figure 8.6: Player Client login screen.

During team selection, the Player Client can present a team selection screen, as shown in Figure 8.7. This screen shows four unique mascots, that are associated with the different teams presented by the Presentation Client. Lecture Quiz uses mascots to give players something to relate to their team. Developers have the ability to change the appearance of the Player Client based on a player's team/-mascot.

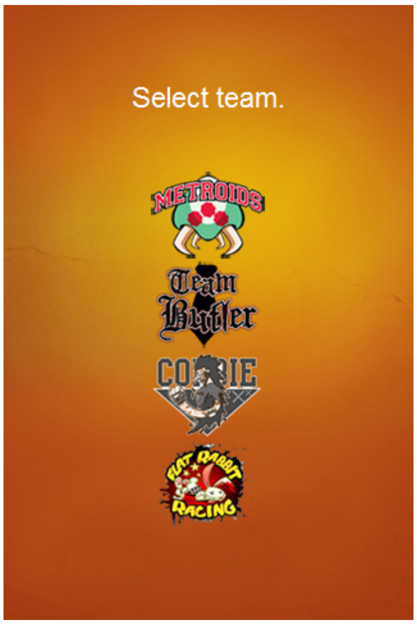


Figure 8.7: Player Client team select screen.

The “waiting for round screen” is presented between answering questions, as shown in Figure 8.8. If the player answered the previous question, the Player Client displays the alternative the player chose. This is to allow the player to keep track of responses during gameplay. Note that the answer will not be presented until the question timer is finished. This prevents people from cheating.

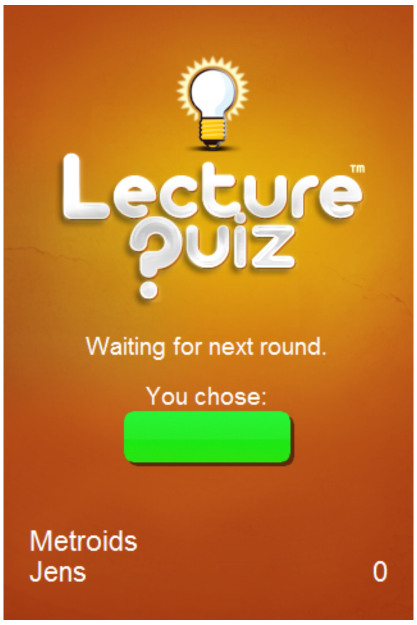


Figure 8.8: Player Client waiting for round screen.

When a question is presented on the Presentation Client, the Player Client will display the answer question screen, as shown in Figure 8.9. This screen contains four buttons with different colors, and allows the player to select one of the buttons. Each of the buttons represent one of the question alternatives presented by the Presentation Client, as shown in Figure 8.4. The buttons contain no text on the Player Client, which is to keep the players focused on the Presentation Client. In Lecture Quiz, each question has four alternatives, and more than one of the alternatives can be correct.

8.2.3 Quiz Server

The Quiz Server is created with JSF and is a Web application. It uses simple HTML structure combined with Cascading Style Sheets (CSS). This allows easy modification of the interface composition, and separates the styling from the HTML elements. For example, when listing items, each item use either the “oddItem” class or the “evenItem” class, depending on what number in the list the item is. This allows CSS to handle odd or even items in the list separately. The Quiz Server uses this to create a zebra like pattern in lists, as shown in Figure 8.10. This

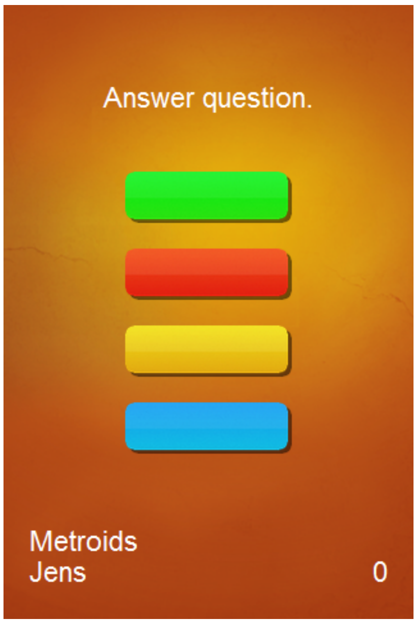


Figure 8.9: Player Client answer question screen.

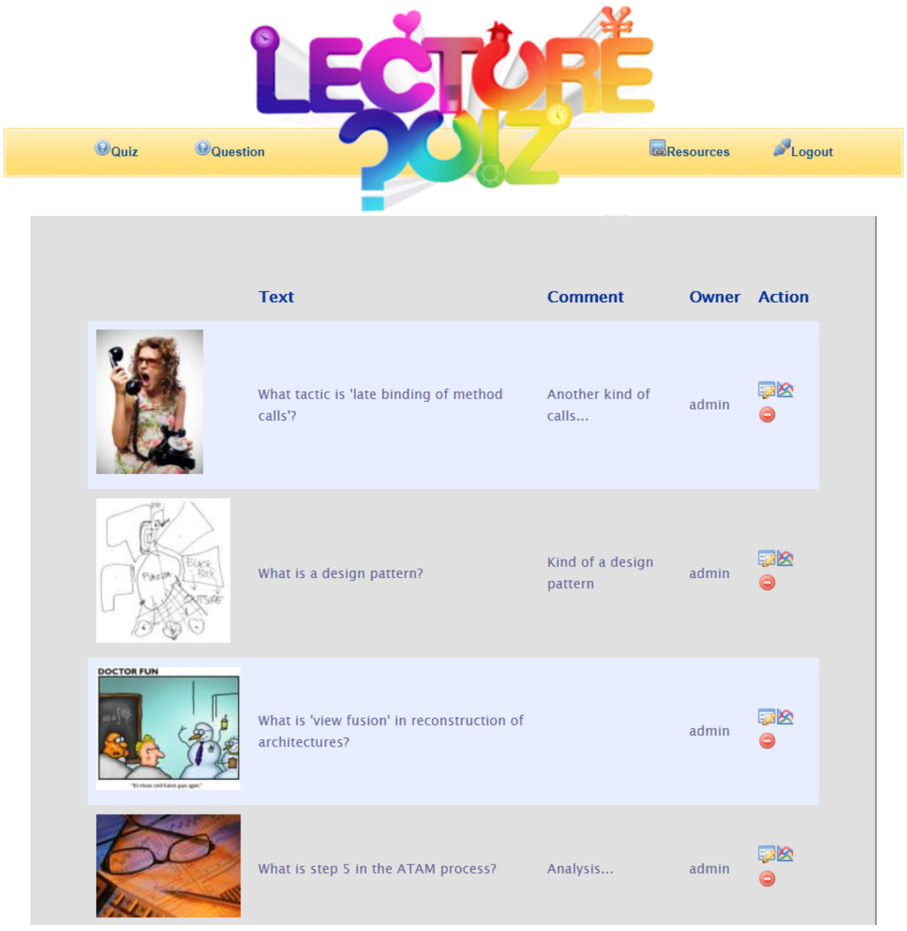
pattern gives a discrete separation of elements, to make the list more readable to the end-user.

CSS supports more than one class for each HTML element. In addition to the odd/even classes, each element in lists uses the listItem class. This applies the same basic style for each list item, while keeping the zebra pattern. For example, the quiz list and the question list both use the listItem class, but contain different content for the items. This is shown in Figure 8.10 and Figure 8.11.

The Quiz Server uses web forms to edit information. To enhance the user experience, these pages utilize AJAX. This allows information to be updated without reloading the web page. For example, when selecting an icon for a quiz, the preview image will be updated, as shown in Figure 8.12.

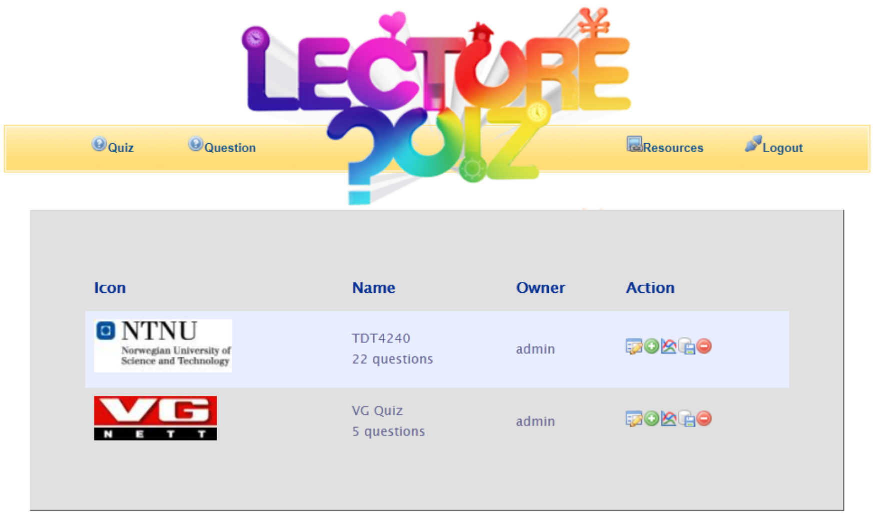
When creating quizzes and questions, the user can add images. These images are used to give additional information to a question, or as an icon for a quiz. The Quiz Server allows users to upload their own image. To give an overview of available images, the Quiz Server uses an image grid, as shown in Figure 8.13. Images can vary in size, and may take some time to display. To make loading of images faster for the end-user, the Quiz Server creates thumbnails of every uploaded image.

The Quiz Server can be navigated with the “Quiz”, “Question” and “Resources” buttons. Each of the buttons displays a menu list when hovered, as shown in Figure 8.14. This makes the web page more interactive and allows for grouping of navigational elements.



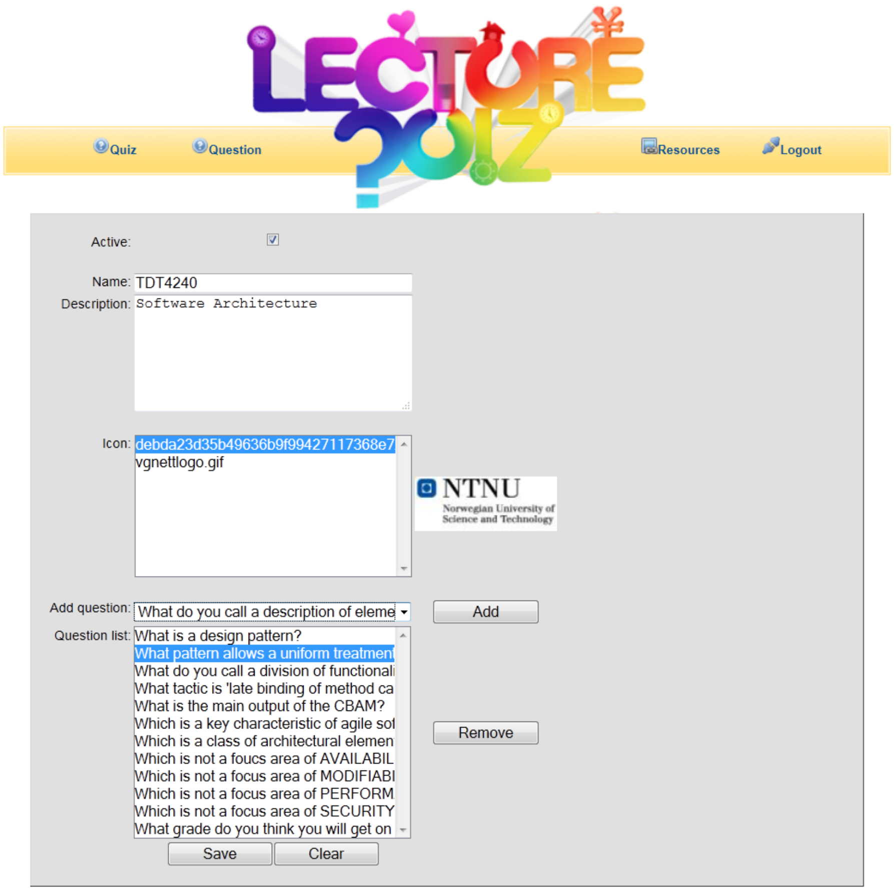
	Text	Comment	Owner	Action
	What tactic is 'late binding of method calls'?	Another kind of calls...	admin	
	What is a design pattern?	Kind of a design pattern	admin	
	What is 'view fusion' in reconstruction of architectures?		admin	
	What is step 5 in the ATAM process?	Analysis...	admin	

Figure 8.10: Quiz Server question list screen.



Icon	Name	Owner	Action
NTNU Norwegian University of Science and Technology	TDT4240 22 questions	admin	
VG NETT	VG Quiz 5 questions	admin	

Figure 8.11: Quiz Server quiz list screen.



Active:	✓
Name:	TDT4240
Description:	Software Architecture
Icon:	debda23d35b49636b9f99427117368e7 vgnettlogo.gif
Add question:	What do you call a description of eleme
Question list:	What is a design pattern? What pattern allows a uniform treatment What do you call a division of functionali What tactic is 'late binding of method ca What is the main output of the CBAM? Which is a key characteristic of agile sof Which is a class of architectural elemen Which is not a foucs area of AVAILABIL Which is not a focus area of MODIFIABI Which is not a focus area of PERFORM Which is not a focus area of SECURITY What grade do you think you will get on

Figure 8.12: Quiz Server edit quiz screen.

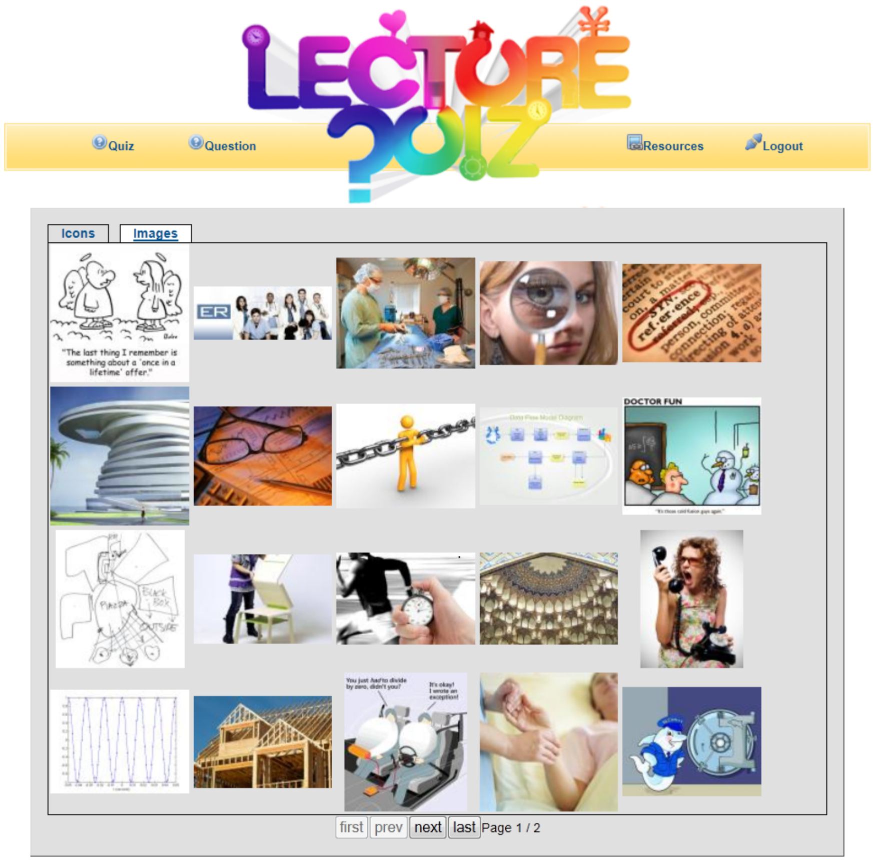


Figure 8.13: Quiz Server image grid screen.

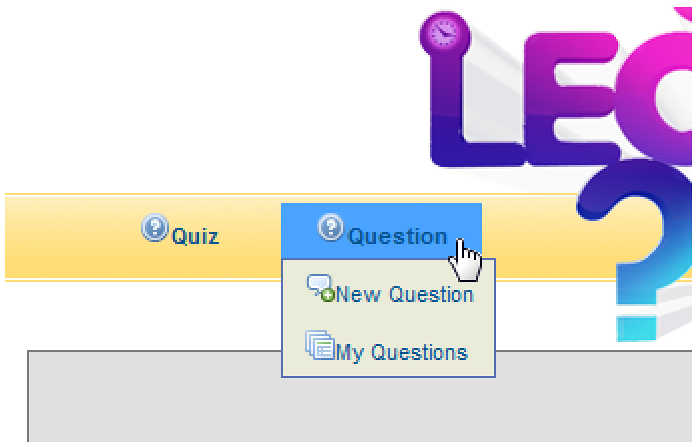


Figure 8.14: Quiz Server question hover menu.

Chapter 9

Implementation

We created a set of quality requirements for our system. Based on these requirements, we designed our software architecture. By combining the architecture and the functional requirements we developed the Lecture Quiz implementation.

In this chapter, we present our implementation of Lecture Quiz 3.0. All the libraries are presented in their own sections below.

9.1 Distributed Message Queue

Based on our previous experience from Lecture Quiz 2.5, we wanted a stable and fast way of communicating between different parts of the framework. We early decided to use sockets as they are both simple and fast; this was also mentioned in the further work section of our Specialization Project. This meant that we had to do some data interpretation, as sockets only handle binary streams.

9.1.1 Network Packets

We created a way to package data and a simple protocol for interpreting this. A network packet contains an identifier, the length of the packet, a checksum and the package data. The identifier makes sure that a valid header is being sent from the client. To validate the content of the packet, we use the received checksum. This checksum is compared to the checksum of the packet content. For checksum calculation, we use 32 bits cyclic redundancy check (CRC-32), as this is a fast way to create a checksum and sufficient for our use.

9.1.2 Message Queue Abstraction

The communication logic needed a more generic way of interpreting the content of a packet. We decided to expand our use of the message queue pattern, shown in Figure 7.25. To achieve this, every message to be sent over sockets needs to be serializable. Our system already supported this extension, as every message was implementing the IMessage interface. To obtain serialization, we created a message Serializer class. This class uses Java object serialization to convert between messages and binary packet content. One of the problems encountered using this kind of serialization, was that each communicating application, needs to have a definition for each message class. To solve this, we created the Lecture Quiz Messages library, which is a shared library containing the messages used by the Lecture Quiz system. This is further described in Section 9.2.

Our goal with the Distributed Message Queue is to create an abstraction for the network communication logic. Using the message queue makes the individual component only care about sending and receiving messages, and not to whom or where. For example, when statistics are collected after a game round, a statistical message is queued to the message queue. This message will be sent to both the server and the local class, since both are registered for statistical messages. The Local class saves statistics to the file-system, while the Quiz Server persists it. In this case, the sender of the statistics is not concerned about with whom or where the statistics are saved.

9.1.3 Connection Management

To utilize the communication logic combined with the message queue pattern, the Distributed Message Queue uses sockets. This means that to create a message queue, the application needs to define if it is a server or a client, as shown in Figure 7.24. When running the message queue as a client, the application needs to know the server address and the application port. Running the message queue in server mode requires the application to specify a listener port and a backlog count. The backlog count is used to specify how many unprocessed connection requests the server socket can queue, before starting to refuse connections. After a message queue instance is obtained, new connections appear on the message queue as regular messages. For example, when a client connects to the Quiz Server, the server receives a NewConnectionMessage on the message queue. The NewConnectionMessage contains a ClientConnection object. This object is the application's reference to the client and can be used to send messages. The ClientConnection object implements IMessageReceiver, which allows the server to register it on the message queue, as shown in Figure 7.25.

The ConnectionHandler component is used along with the message queue. When creating a Distributed Message Queue instance, communication related messages are registered to the ConnectionHandler. This allows the ConnectionHandler to manage messages like NewConnectionMessage, ConnectionLostMessage and Host-NotFoundMessage. These are status messages used by the Distributed Message Queue to indicate its state. If the user has specified a server connection, the ConnectionHandler will take care of the authentication. The ConnectionHandler makes sure that the Presentation Client is identified by the Quiz Server.

9.1.4 Stable Connection

To make sure the message queue have a stable execution, we have implemented a layer between the message queue and the socket. This will change depending on whether the queue is run as a server or as a client. If the message queue is running as a client, it will create a SocketConnector. This object runs in its own thread and is responsible for establishing contact with the server. In addition to establishing contact, it makes sure the client stays connected, while it actively checks for errors. For example, if the connection is lost, it will queue a ConnectionLostMessage to the message queue. It will then try to reestablish a connection to the server, in intervals. If it can establish a new connection, it sends a NewConnectionMessage to the message queue to restore the ClientConnection. In practice this means that classes using the ClientConnection will just update their reference to the client and keep sending messages. A SocketListener will be created when creating the message queue, for servers. This works in the same way as for clients, but since it is a listening port, it does not deal with reconnection. If a client sends invalid data to the server, the SocketListener will queue an InvalidPacketMessage to the message queue and disconnect the client. This is to prevent spamming of random data to the server.

To make the Distributed Message Queue run smoothly, it uses several threads. This is to make sure the application does not block while communicating with other applications. It also allows for the SocketListener and SocketConnector to run in the background. As mentioned earlier, we use an abstract factory pattern to create the message queue. Since the queue uses threads, we wanted the dispose logic to be controlled in the same manner as it is created. We created a recycler, which shuts down the queue and cleans all the thread usage.

9.2 Lecture Quiz Messaging

Lecture Quiz Messages is a shared library used by the Presentation Client and the Quiz Server. It contains message definitions that are used when communicating between the client and server. The reason for having a separate library is that both instances needs to have the same message definitions.

The library contains messages for handling authentication and identification of instances. Since we are using the Distributed Message Queue for client-server communication, we need to establish a connection role. This is achieved when a new connection is created. Upon receiving a NewConnectionMessage, the client is to identify itself by sending an identification message. This message is either a PresentationClientIdentificationMessage or a QuizServerIdentificationMessage. For a client to gain access to Quiz Server content, it requires user authentication. This login process is equal to a user login directly on the web server. The Presentation Client uses the UserLoginMessage to send user authentication to the Quiz Server. Upon receiving a UserLoginMessage, the Quiz Server will send either a UserLoginFailedMessage or a UserLoginSuccessMessage. If the login is successful, the client will be able to send messages to send and request content from the server.

The Presentation Client will be able to list quizzes from the Quiz Server by sending a RequestQuizListMessage. The requested list will be sent from the Quiz Server as a QuizListMessage. This message contains a list of unique identifications for the quizzes stored on the server. This is to make the Presentation Client able to cache quizzes by identification. A quiz' identification is the Secure Hash Algorithm 1 (SHA-1) checksum of its content. To receive information about a quiz from the server, the Presentation Client can send a RequestQuizInfoMessage with the quiz' identification. This will make the Quiz Server reply with a QuizInfoMessage containing quiz information like name, author, question count and description.

Quiz content are contained inside quiz packages. To obtain a quiz package from the Quiz Server, the Presentation Client sends a RequestQuizDownloadMessage. This message needs a unique quiz identification for the quiz that is to be downloaded. The Quiz Server will send a QuizDownloadMessage with the quiz identification and a URL for download. The Presentation Client uses a package downloader which downloads the package from the URL. The reason why we do not send quiz packages over the message queue is the size of the packages. Sending several megabytes over the message queue will result in the client not being able to receive other messages while downloading. Another reason is that using HTTP transfer from a web server will give more control to the client as the communication logic is not hidden.

When running games, the Presentation Client will send statistics to the Quiz Server. These statistics are sent in the form of one or more QuestionResultMessage. A QuestionResultMessage contains the quiz identification of the quiz that the question belongs to. It also contains the question identification within the quiz, and the result of how players answered that question.

9.3 Lecture Quiz Packaging

Quizzes are sent from the Quiz Server to the Presentation Client as quiz packages. This allows the server to send quizzes, including both questions and resources, as one package. The Quiz Server supports direct download of these packages. Once a package is downloaded, it can be run offline without a connection to the server. This is one of the main features we have added to make Lecture Quiz more flexible.

To create quiz packages, the Quiz Server uses the Lecture Quiz Packaging library. This library allows for both creation and reading of quiz packages. A quiz package is a ZIP file that contains a quiz.xml file and image files. The XML file contains the quiz information as well as the quiz questions. Image references in the XML file, refer to the images contained in the package ZIP. For example, a question contains an image reference. The Presentation Client will use this reference to read the image data from the package. The resulting image is presented alongside the question during a game. The Lecture Quiz Packaging library utilizes Java's standard ZIP support to read the quiz packages.

To manage XML, we have created a serializer that converts between an XML document and data models. A quiz' unique identification is generated by the Lecture Quiz Packaging library. At this stage, we had gotten all the quiz information gathered in one string, which makes it easy to calculate a checksum. We created a hash string generator, that uses SHA-1 to calculate a checksum and convert it to a hexadecimal formatted string. A hexadecimal formatted string is a way to represent binary data using hexadecimal numbers. Each byte value in the binary data is represented by a pair of hexadecimal characters. This is to make the checksum human-readable and easy to compare to other checksums.

The Lecture Quiz Packaging library also handles statistical packages. These are created in the same way as quiz packages, but contain no other resources than the statistics XML. The Presentation Client uses these packages when run in offline mode. This allows a lecturer to copy a quiz to his laptop, run a game on a local network and still be able to bring back the results.

9.4 Presentation Client

The Presentation Client is a standalone Java SE application. Its main purpose is to be used during lectures to run Lecture Quiz games.

We have separated the Presentation Client into six parts. Each part has its own responsibilities and components. Communication between the different parts is done either through the message queue or via the service locator. The different parts are infrastructure, graphics, sound, input, GUI and logic. We have created a manager class for each part. A manager handles the initialization and disposing of its sub-components [43]. Managers are initialized at the start of the application through the IManager interface, as shown in Figure 7.6. During initialization, each manager registers the services they want to offer on other parts of the application, on the service locator.

9.4.1 Infrastructure

The infrastructure part of the Presentation Client contains support components. The components managed by the InfrastructureManager are Configuration, ConnectionHandler, ComponentUpdater, ThreadDisposer and MessageQueue. The MessageQueue object is a client instance of the Distributed Message Queue, as described in Section 9.2.

Configuration

The Configuration object is registered as a service through the IConfiguration interface. This interface allows for easy reading of configuration properties, by using generics and default values, as shown in Figure 7.8. At the start of the application, the configuration settings are loaded from the configuration.xml file. This is done through the built-in Java properties object. Examples of settings that are read from the configuration are: DisplayFullscreen, DisplayWidth, DisplayHeight, ClientPort and ServerAddress. We wanted the application to support command-line arguments, to quickly test settings. This is implemented by reading the command-line arguments as key-value pairs. These values are written to the configuration object, and overrides the settings that are read from file.

ConnectionHandler

The ConnectionHandler component distributes incoming messages to their respective handlers, for further description, see Section 9.1. The Presentation Client identifies itself to the Quiz Server, by sending a PresentationClientIdentificationMessage. In the same manner, it makes sure that the machine it is connected to, identifies itself as a Quiz Server. After identification, the ConnectionHandler will try to authenticate to the Quiz Server. The authentication credentials are read as properties from the configuration, as described earlier. If the user is successfully authenticated, we want the message queue to send messages to the server. The ConnectionHandler will register the message types that should be sent to the server, over the message queue. Examples of messages that are sent to the server, are RequestQuizListMessage, RequestQuizDownloadMessage and QuestionResultMessage.

ThreadDisposer

The ThreadDisposer implements the IThreadDisposer interface. Threads are used in several of the application's components. We want to make sure that they are correctly disposed when the application shuts down. This is the goal of the ThreadDisposer. Other components can access the ThreadDisposer through the service locator, then register their threads. When the InfrastructureManager is called to dispose its components, it will call dispose on the ThreadDisposer. This will make the ThreadDisposer stop any registered threads that are still running.

ComponentUpdater

The last component in the infrastructure is the ComponentUpdater. The ComponentUpdater is registered in the service locator through the IComponentUpdater interface. This allows classes that implement the IUpdateable interface to register for updates. The ComponentUpdater will call the update() function on each registered component. This is done whenever the ComponentUpdater thread gets processing time. The call to the update function includes a tick count. The tick count is the number of milliseconds passed since the last call to the update function. This is important in games, as both timers and animations need to be updated continuously. By using the ComponentUpdater, we save ourselves the trouble of having to deal with individual threads for timed tasks.

9.4.2 Graphics

The GraphicsManager is responsible for creating the application window. The display resolution and fullscreen settings are read from the Configuration object. The GraphicsManager creates an instance of the GraphicsDisplay class. This object creates the application window.

GraphicsDisplay

The GraphicsDisplay object initializes an OpenGL window through LWJGL. Graphics loaded with LWJGL needs to be loaded using the same thread as was used to create the application window. To solve this, we let the GraphicsDisplay object run its own thread. This thread initializes the OpenGL window, a TextureLoader and a GraphicsRenderer.

TextureLoader

The TextureLoader is an abstraction used for loading graphics in the Presentation Client. It implements the ITextureLoader interface and is registered on the service locator. Textures loaded with the TextureLoader from other classes will only create a model of the texture. This model implements the ITexture interface, as shown in Figure 7.10 The first time the texture is being rendered by the GraphicsRenderer it will be loaded by the correct thread.

GraphicsRenderer

The GraphicsRenderer implements the actual rendering logic. It is the GraphicsRenderer that uses LWJGL to render graphics. The rendering of graphics needs to be isolated from the rest of the application. This is to avoid problems with concurrency and the rendering thread. All graphics in the Presentation Client are defined as GUI components. The collection of GUI elements are called scenes, and will be covered later in the following Section 9.4.5. To allow the GraphicsRenderer to render a scene, we make the GraphicsRenderer register for SceneChangedMessages. These are messages sent to the message queue when a scene is changed. This allows the GraphicsRenderer to get the current scene and render it.

9.4.3 Sound

The SoundManager is responsible for playing sound effects in the Presentation Client. Playing sound effects is done through the LWJGL library. The LWJGL library utilizes OpenAL to play sounds. Unlike the OpenGL implementation, the OpenAL functionality can be called from any thread. Despite this, we decided to isolate this functionality from the rest of the application. This was to be able to change the implementation in the future, without worrying about restrictions.

Since we have a limited number of sound effects for our game, the SoundManager uses the SoundEffect Enum to decide what sound to play. This Enum contains Click, GameMode, Tick, Ring and Finalé; which are associated with the sound effect files included with the game. The SoundManager will register itself to the message queue, to listen for PlaySoundEffectMessage and StopSoundEffectMessage. When receiving a PlaySoundEffectMessage, it will extract the SoundEffect value from the message. It will use this value to decide which sound effect to play. The same applies to the StopSoundEffectMessage, which does the reverse: it stops the sound effect, if it is playing.

9.4.4 Input

The InputManager handles the initialization of components used for controlling the application. In the current implementation, the InputManager will check for mouse and keyboard support. The communication with these devices are done through the LWJGL library.

We separate the devices into pointer devices and input devices. If a mouse is connected the InputManager will create an instance of the LWJGLMouse class. This class implements the IPointerDevice interface so one can easily change the implementation later. The same goes for the LWJGLKeyboard class which implements the IInputDevice interface. IInputDevice is intended for application control and not text input, and may in the future be used for game-pads and joysticks. In our implementation, the input devices are passive, which means that there are no events linked to buttons being pressed. Instead, the application needs to ask for a device state. A device state is different from a pointer device to an input device. An IPointerDevice will return an InputPointerState. The InputPointerState includes values like PositionX, PositionY, IsLeftButtonPressed and MouseWheelDelta. An IInputDevice will return an InputDeviceState. The InputDeviceState includes values like Left, Right, Accept and Cancel.

To give other parts of the application access to the input devices, the InputManager uses an InputDeviceContainer. The InputDeviceContainer is a container class that implements the IInputDeviceContainter interface. Both the keyboard and the mouse objects are added to the InputDeviceContainer. The container is registered as a service on the service locator to allow access by the rest of the application.

9.4.5 Graphical User Interface

The GuiManager is responsible for the applications graphical interface logic. It contains two components, the SceneHandler and the SpriteFontCreator.

As the framework uses OpenGL for rendering graphics it has only access to functions that render textures. To make the application able to render text, we created the SpriteFontCreator. The SpriteFontCreator load font images from the resource folder and associates them with our FontType Enum. Font images are loaded as textures and passed to SpriteFont objects. We defined our own format for these font images. Each image is divided into a 16x16 grid where each cell in the grid is a character. This sums up to 256 different characters and are associated with char values in strings. For example, the letter **A** has American Standard Code for Information Interchange (ASCII) value 65, which will associate it with cell number 65 in the image.

When a SpriteFont object is first created, it will use an algorithm to calculate font metrics. These metrics are measures of the individual sizes of each character in the texture. These values are important when creating textures from a font.

We want the character spacing to be relative to each character's size and not just a fixed size. This makes the result proportional and more readable by the end-users. Our SpriteFont class contains a method for generating text textures. The method is called generateTexture() and takes two arguments. The first argument is the text to be displayed in the texture. The second argument is an alignment Enum, which allows for aligning the text left, right or center, inside the texture. An advantage of using this system to generate text, is that it allows easy caching. We can store textures for later use, instead of synthesizing the text output from a font, each time the application renders the screen.

The other component of the GuiManager, is the SceneHandler. The SceneHandler is responsible for managing scenes and transitions. It keeps track of which scene is currently presented by the application. The SceneHandler listens for QueueSceneChangeMessages. These messages are sent when changing scenes. Upon receiving this message the SceneHandler will call the end transition effect of the current scene. When the end transition has completed, it will change its current scene to the scene supplied in the QueueSceneChangeMessage. This change is done by sending a SceneChangeMessage to the message queue, and calling the start transition of the new scene.

The SceneHandler contains a SceneNavigator. The SceneNavigator is responsible for handling user input and navigation of the current scene. To be able to do live updates of the interface, the SceneNavigator is registered on the ComponentUpdater, as described in previous section. During updates, the SceneNavigator will get input states from the devices registered with the InputDeviceContainer. These states are combined into a NavigationInputState object. This object is then passed to the current scene. This gives GUI components within the scene the ability to run logic based on the current input states. For example, to check if the mouse pointer is hovering over a component, or if a button is pressed.

Scenes in the Presentation Client are responsible for handling high level logic. The GameTypeSelectScene creates an instance of the Game class. The Game object contains the settings to run a game. These settings are set by different scenes, before the game starts. Examples of scenes used to configure the game, are the QuizSelectScene, GameModeSetupScene and TeamSelectScene. The different scenes are used to satisfy most of the Presentation Client's functional requirements, as shown in Table 6.1. By handling functional requirements on a high level, the Presentation Client can easily be changed to meet new requirements.

9.4.6 Logic

The LogicManager is responsible for handling game related logic. It manages the following three components; QuizProvider, GameModeProvider and PlayerClient.

The QuizProvider object consists of two sub components. Both are used to provide quizzes that can be run in a quiz game. These components are the LocalQuizProvider and the RemoteQuizProvider.

LocalQuizProvider

The LocalQuizProvider will read quizzes from the local file system. To locate the quizzes it will search the quizzes sub folder in the Presentation Client directory. When creating an instance of the RemoteQuizProvider, we pass a reference to the LocalQuizProvider. This is to allow the RemoteQuizProvider to use the LocalQuizProvider for caching quizzes. For example, if the Quiz Server offers a quiz, the RemoteQuizProvider will ask the LocalQuizProvider if this quiz is cached. If the LocalQuizProvider matches the quiz identification to one of the local quizzes, the current cached version is used. This saves download time, as quizzes often are of several megabytes in size, even tenfold if it includes large images.

RemoteQuizProvider

The RemoteQuizProvider will as the name suggests, provide quizzes that are not represented on the local computer. To receive quiz information, the RemoteQuizProvider needs access to a Quiz Server. This is handled by the ConnectionHandler in the infrastructure part, as described in Section 9.4.1. Communication with the server is done through the message queue. The RemoteQuizProvider will send a RequestQuizListMessage when the server is connected. The Quiz Server will reply to this message with a QuizListMessage. The QuizListMessage contains a list of quiz identifications. The RemoteQuizProvider will send one RequestQuizInfoMessage for each message identification not located in the LocalQuizProvider. These messages will be replied to by the Quiz Server with QuizInfoMessages. A QuizInfoMessage contains general information about a quiz and not its content.

When the user selects a quiz to be used with a game, the application will send a RequestQuizPackageMessage. This message will be received by both the LocalQuizProvider and the RemoteQuizProvider. If the LocalQuizProvider already has the quiz, it will respond with a QuizPackageMessage containing the quiz. As the RemoteQuizProvider keeps track of the LocalQuizProvider it will know if the request will be handled locally. If the requested quiz is not contained in the local quiz provider, the RemoteQuizProvider sends a RequestQuizDownloadMessage to the Quiz Server. The Quiz Server will respond with a QuizDownloadMessage. The QuizDownloadMessage contains a valid URL, that the application use to download the quiz. Quizzes are downloaded via HTTP as it saves traffic on the message queue. When successfully downloaded, the RemoteQuizProvider will send a QuizPackageMessage with the downloaded quiz. The RemoteQuizProvider will tell the LocalQuizProvider to store the quiz for future use. Only the RemoteQuizProvider component is managed by the LogicManager, as the LocalQuizProvider is contained within the RemoteQuizProvider.

GameModeProvider

The GameModeProvider is as the name conveys, responsible for providing game modes. A game mode contains the logic for running quiz questions. Games are divided into several game types. The different types are Two Teams, Three Teams, Four Teams and Free For All. Not every game mode can run as a game type and this is decided by the GameModeProvider. When listing game modes from the GameModeProvider a GameType Enum needs to be specified. The Enum contains our different game types, and will make the GameModeProvider only list supported game modes. The GameModeProvider is registered on the service locator via the IGameModeProvider interface.

PlayerClient

Interaction with the Player Client library is managed by the LogicManager. To create an instance of the PlayerClient class, a port number needs to be supplied. This number decides which port the Player Client web server should listen on. The LogicManager reads two port numbers from Configuration at initialization. The values are ClientPort and AltClientPort. The reason why we chose two values for this, is that some computers may already be using one of the ports. Thus, if the creation of a PlayerClient instance fails for the first port it will try the second. If both ports are unavailable it will alert the user that it has to specify a free port.

Input from players will be handled by Java's internal ActionListener interface. Callbacks are done by the same threads that handle the HTTP communication with the clients. Since using these threads directly can easily lead to deadlocks, we created the PlayerClientWrapper. The PlayerClientWrapper manages callbacks from the PlayerClient. It will gather input into a thread safe queue. To process this queue, the PlayerClientWrapper is registered with the ComponentUpdater. This will make the callbacks run on the same thread as the rest of the game logic. The PlayerClientWrapper is registered with the service locator through the IPlayerClient interface.

9.5 Player Client

Players use a web interface to communicate with the Presentation Client. We did not want the Presentation Client to have to deal with HTML directly, so we created the Player Client library. The library enables the creation of a web server from inside another Java application. Interaction with the players is done through the IPlayerClient interface. This interface abstracts away the communication logic between the Presentation Client and the players.

9.5.1 Content Management

To start the Player Client, the Presentation Client needs to specify the port in which the HTTP server will be run. It also requires a resource handler, as shown in Figure 7.20. The resource handler provides HTML content, scripts, stylesheets and images to the end-user. At start-up, the Player Client spawns an instance of the Java Sun HttpServer. This class implements a simple HTTP server [22].

9.5.2 Sessions

The HTTP server does not include support for neither cookies nor sessions. We wanted both these features to allow players to login to the server and be remembered. In this version of Lecture Quiz, the Player Client contains a SessionManager for handling player sessions. When a player first connects to the HTTP server, the server checks to see if they offer a session cookie. Cookies are read using our HttpUtils class from the binary input steam sent by the client. If the client does not offer a session cookie, the SessionManager generates a session object for that client. The session object is identified by a unique identification string generated from the client's host, current date and a random salt. After creating the session object, the cookie information is written to the client response. The client will store the session cookie, and it is sent as part of future HTTP requests. This allows the Player Client to identify the client's session, and use it to store user specific information.

9.5.3 Threading

To allow multiple clients to use the HTTP server simultaneously it needs several threads. The Java HttpServer object supports the use of executor services. An executor service is a definition of whom executes the given tasks. In this context, the executor will be given a task for each connection it receives from a client. The Player Client is aimed toward multiple users with several small requests. With this in mind, we decided to use a cached thread pool for the executor service. This means that it will try to reuse existing threads if available. If no existing thread is available, a new thread will be created and added to the pool. Threads that have not been used for sixty seconds are terminated and removed from the cache.

9.5.4 Player Information

To store player information, the Player Client uses the PlayerManager. This manager handles login requests from players. It validates whether the username is unique, and whether it meets the username requirements. If the login is successful, a Player object is created and associated with the client's session. This will make the Player Client call the playerJoined() ActionListeners, which allows the Presentation Client to listen for joining players. The player Client supports playerLeave() ActionListeners to listen for leaving players.

This Player object contains the player's name, score, state, last answer and team. The state field is used to specify the users current state in the game. This controls what the player will see on his client. There is one state for each different client screen. States include: "login", "answer question", "waiting for round", and "game summary". To notice changes in players' state, the web page received by the client, uses JavaScript and AJAX requests. These are short requests that check the current state of the player. If the state stored on the client side is not equal to the received state the web page will be refreshed. This means that if you change the state on a Player object, the client will change its visuals.

9.5.5 Templating

To be able to create more generic web pages for the client, we created a simple template system. The system allows us to use Player variables in the client HTML. For example, adding %playerName% will insert the "Player.Name" variable of the client's player. For fast execution, pages are cached at startup, where each page is cached into its own ContentView. A ContentView represents a Player state and has several ViewParts that builds the output HTML. These parts contain either static HTML data or a reference to a variable. This allows the use of Java StringBuilder, to quick create the output HTML without string search/replace.

9.6 Quiz Server

The Quiz Server runs as a Java EE web application, implemented with technologies such as Java Server Faces (JSF) and Java Persistance API (JPA). It is deployed on a GlassFish Server and functions as a quiz management, statistical tool, and as a quiz database for Presentation Clients. Presentation Client access it over the network with the help of network sockets; such that online Presentation Clients can obtain live quizzes on the fly and commit statistics as the running quiz progresses. This Quiz Server implementation section is divided into; server initialization, persistence, packaging, messaging, http and caching. The structure is not necessarily the same as in the code-base, but is presented in a more readable manner. The following sections describe them respectively.

9.6.1 Server Initialization

Because the Quiz Server is a web application, no main class exists per say. When the web application is deployed, IndexServlet is selected as the first servlet to initialize. The IndexServlet also serves as the welcome file, for default access to the web server. This class overrides the init() method from Servlet, which in turn initialize the configuration, persistence and the message queue; and registers it with the ServiceLocator. The ServiceLocator, as described in Section 7.3, centralizes distributed service object lookups provides a centralized point of control, and may act as a cache that eliminates redundant lookups [55].

The initializeConfiguration() method tries to locate the default configuration file, or otherwise create it. The different settings for each part of web application are loaded into the Configuration object at startup. If the configuration file is created for the first time, the initializeDefaultConfigurationValues() is called. This method creates the default settings and writes them to the configuration file. This file is called configuration.xml and is stored in the same folder as the web application's Web application Archive (WAR) file. When this part has registered all necessary settings, it is registered at the ServiceLocator for further access.

The initializePersistence() method registers all the persistence managers to the service locator. The current managers are: IQuestionManager, IQuizManager, IIconResourceManager, IImageResourceManager and QuizResultManager and IUserManager. These are further described in Section 9.6.2. If default persistence data is needed in a fresh database, it should be initialized via this method.

The initializeMessageQueue() method contains logic for configuring, creating and starting the Distributed Message Queue on the Quiz Server. It obtains the listener port, and the listener backlog, via the configuration service. It creates a ConnectionHandler and a DistributedMessageQueueFactory object, which is required to create an instance of the Distributed Message Queue, as explained in Section 9.1. Further, it calls the createServer() method in the DistributedMessageQueueFactory object with the listener port, listener backlog and connection handler as arguments. This method creates and starts the server, unless the port is unavailable, which yields a PortAlreadyInUseException. The initializeMessageQueue() method initializes the connection handler, and registers all the components mentioned to the service locator.

If all the aforementioned initialization processes are successful, the web application will start. In all other cases, the system administrator will get meaningful responses in the server log on what could have gone wrong. Deployment information, possible errors and other runtime problems are described later in Appendix C.

9.6.2 Persistence

As described in Chapter 8, the Quiz Server manages a storage of quiz data and statistics. This includes users, quizzes, questions, resources, statistics; which is represented as persistence entities. The Quiz Server's "persistence.entities" package contains all entities regarding the persistence. An Entity is a lightweight persistence domain object [22]. Typically, an entity represents a table in a relational database, and each entity instance corresponds to a row in that table [22]. These entities are persisted back and forth from the database via managers, which all extend the ManagerBase. Each entity type is managed by its respective manager. Each manager implements a different interface for registering the manager as a service with the service locator. All our entities extend the EntityBase class, which is an abstract class containing shared methods and fields. The ResourceEntityBase is a shared abstract class which all the resource entities extend. This class contains the shared logic for the icon and image resource entities, and the future audio and video entities.

The "persistence.managers" package contains all the managers and their interfaces. These managers persists entities to the persistence. We describe some of these managers in this section. The IIconResourceManager and IImageResourceManager extend the IResourceManager. The IResourceManager interface uses a generic template named ResourceEntityBase, as seen in Listing 9.1.

```
public interface IResourceManager <T extends
  ResourceEntityBase>{
    List<T> getResourceList();
    T getResourceByUID(String resourceUID);
     ...
}
```

Listing 9.1: IResourceManager interface

The reason why we chose to implement it in this manner, is that it allows uniform treatment of resource data between different resource types. The two current supported resource types are icons and images, but could easily be expanded to video and audio. We chose to make this logical grouping, because resources are handled as binaries with shared meta data.

The ManagerBase, that all the managers extend, contains logic for persisting transactions, both with and without results. Examples on these transactions are methods like persist(), refresh(), merge(), remove() and find(). Following, the ManagerBase utilizes the PersistenceManager which again holds the reference to the main persistence object; the EntityManagerFactory. All managers and the PersistenceManager are created just once at server initialization, and can be located using the Service Locator.

The Lecture Quiz system contains more than six libraries and projects, and uses the shared models in the Lecture Quiz Packaging library, as described in Section 9.3. To convert between these models and the Quiz Server entities, we have created four converters. The converters include two classes for exporting and importing QuizEntity objects; and two for exporting and importing QuizResultEntity models.

9.6.3 Packaging

Packaging is used in the distribution of quizzes. It is also related to the process of importing-exporting statistics.

As described in Section 7.2 and Section 9.1, most communication is done through the Distributed Message Queue. One exception is the distribution of quiz packages, which would require unnecessary resources from the Distributed Message Queue. To relieve the Distributed Message Queue of heavy traffic due to downloading quiz packages, the Quiz Server supports HTTP download of these. To gain access to the Quiz Server via HTTP, we have introduced the QuizPackageLeaser. This class enables non-authenticated access to quiz packages. To download a quiz package, you will need the correct URL and a unique identifier which is received via a QuizDownloadMessage. The unique identifier is generated by the QuizPackageLeaser, and is sent via the message queue to an authorized Presentation Client upon request. The identifier is linked to one unique quiz package, has a lifetime of 10 minutes and will only work one time. The QuizDownloadMessage is located in the Lecture Quiz Messages library, as described in Section 9.2.

The process of importing and exporting quizzes to and from the persistence is handled by the PersistanceManager. Quizzes are stored in the persistence as a hierarchy, as explained in Section 9.6.2. The main entity is the QuizEntity, and its respected manager takes care of insertion and extraction of data from the persistence. A QuizEntity is only represented in the Quiz Server and is unknown to other parts of Lecture Quiz. We have created an importer to obtain a QuizEntity from a QuizPackage; and an exporter to obtain a QuizPackage from a QuizEntity. This is done in two steps. The first step is the conversion between quiz packages and quiz models, and is described in Section 9.3. The second step is the conversion between quiz models and quiz entities and was described in the previous section.

Statistics can be imported to the Quiz Server in two ways; where the first is via Presentation clients over the Distributed Message Queue; and the second is to import statistical packages via the web interface. The first way is described in Section 9.2 and the second way is described in this section. It is possible to import quiz packages via the web interface, the same way as statistical packages are imported. Because of the similarities in the process of importing packages, we will only describe the way statistics packages are imported. When a statistical package is sent to the server, the MultiPart filter will preprocess the information and wrap the request in a MultiPartRequestWrapper object. The reason we use a MultiPart filter, is that JSF does not support the Multipurpose Internet Mail Extensions (MIME) multipart message form of posting web data. This is the filters' only task, and it will pass the request to the next in the filter chain. The MultiPartRequestWrapper extends the HttpServletRequestWrapper, which is a Java Servlet 3.0 class. HttpServletRequestWrapper provides a convenient implementation of the HttpServletRequest interface. This class can be extended by developers wishing to adapt the request to a Servlet [22]. The HttpServletRequestWrapper separates the different parts in the multipart form and stores them in a MultiPart List. The MultiPart class is represented as a model which contains: a Part [22], the file data stream and the filename. Parameters are stored in a HashMap so that meta data can be cached, e.g. filename. After the filters have finished processing the request, the StatisticsBean's uploadStatisticsPackage() method called. This method obtains the QuizResultManager object from the service locator. It loops through a MultiPart list, which is obtained via our static FacesUtils' getMultiParts() method. And it persists each MultiPart object to the database via a

QuizResultManager's createQuizResultEntityFromMultiPart() method. The conversion from MultiPart to statistics and quiz packages is handled by the StatisticsPackageUtils and QuizPackageUtils classes.

9.6.4 Messaging

The messaging package contains a number of message handlers and one connection handler. The ConnectionHandler class is used to receive incoming connections, and register outgoing messages to these. For more information about this class, see Section 9.1. Both the ConnectionHandler and MessageQueue objects are initialized and registered to the service locator on server start up. In this version of the Quiz Server, we have created three message handlers: QuizRequestHandler, ResourceRequestHandler, StatisticsRequestHandler and the UserRequestHandler. All extend the RequestHandlerBase class which contains information about the MessageQueue, ConnectionHandler and keeps an overview over connected Presentation Clients. Each of these handlers is explained in the following sections. We will elaborate on how they process message requests and respond where applicable.

The QuizRequestHandler handles quiz related messages and contains the four following messages. The RequestQuizListMessage is a request message without additional data. It is used to request a list of which quiz packages that are available on the server. The message is answered with a QuizListMessage. This response message contains a list of unique quiz identifiers.

A RequestQuizInfoMessage is sent from a Presentation Client to obtain information from a single Quiz. The message contains a unique identifier for the quiz, and the response is of the type QuizInfoMessage. A QuizInfoMessage contains the name, description, author, a unique identifier, the question count and an icon resource. This request is used in connection to presenting quiz information to the lecturer.

A RequestQuizDownloadMessage is a request message containing a unique quiz identifier. The response to this message is a QuizDownloadMessage. Its purpose is to send a one-time download URL where the Presentation Client can download a quiz package. For more information about the packaging of these quizzes, see Section 9.6.3. This operation is handled by the leaseNewPackageAndSendMessage() method, which in turn utilizes functions described in Section 9.6.3.

The IsQuizPackageValidMessage is a request message which confirms whether a quiz still exists at the Quiz Server. Presentation Clients caches quiz packages and can find out whether a quiz package has been updated. To confirm this, it sends an IsQuizPackageValidMessage. This message contains the quiz' database reference and the current unique identifier of the quiz package. If these unique identifiers matches, there is no response. On the other hand, if the quiz has been updated, it sends a QuizDownloadMessage.

The ResourceRequestHandler handles message requests related to HTML, CSS and graphics storage. It processes messages of the RequestResourcesLocationMessage type. This can relieve Presentation Clients of bandwidth, by providing players with an alternative location for requesting resources. The response is of the ResourcesLocationMessage type, and contains the URL to the resources. This URL is created in the same manner as the QuizDownloadMessage.

The StatisticsMessageHandler is responsible for processing messages of the type QuestionResultMessage. These messages supply the Quiz Server with quiz results. The Quiz Server use the quiz results to create statistics. An alternative to this is to upload the StatisticsPackage, generated by the Presentation Client, directly via the web interface.

The UserRequestHandler is responsible for handling user authentication. It processes login and logout requests from Presentation Clients. The Quiz Server will only supply services to authorized users. A Presentation Client achieves this by sending a UserLoginMessage with a username and password. The Quiz Server responds with either a UserLoginSuccessMessage or a UserLoginFailedMessage.

9.6.5 Java Server Faces

The presentation of web pages is done with the help of the JSF framework. JSF is a Java-based Web application framework intended to simplify development integration of web-based user interfaces.

The Quiz Server has been created according to the MVC architecture pattern. We base each page on a template. This template contains a header, body and footer HTML structure. The template allows each page to change content in each of the structural elements. A default header with navigation bar is included in the template, to achieve a consistent look on the web page.

JSF is based on the FacesServlet, which works as a regular Java Servlet. It is controlled by the framework, which filters and redirects traffic to template or facelets views. These views are represented by Extensible HyperText Markup Language (XHTML) files. Facelets is a simple and effective view description language, and is rather an extension of the JSTL than JSP [19]. This framework is further described in Chapter 5.

Each web page has been created to represent a persistence entity. Because of this, the entities represent models, and the respective web page represents the view. These entities are described in Section 9.6.2.

Beans

The “faces.beans” package contains the controller classes for the Quiz Server web pages. These classes function as the controllers in the model-view-controller (MVC) pattern, with web pages as views. This is described in Section 7.3.3. We have created five classes, where all extend the abstract BeanBase class. The abstract class BeanBase, contains shared methods used by other beans. Beans extend this class to apply construction logic and manage parameters. The five bean classes are the QuestionBean, QuizBean, ResourceBean, StatisticsBean and the UserBean. We describe the QuizBean as an example.

Quiz related views use the QuizBean to get access to quiz management. The QuizBean class offers methods for saving, uploading, and removal, of quizzes.

Properties

The “faces.properties” package contains the files; “constants.properties”, “links.properties”, and “messages.properties”. We have created these files to allow non-developers to change the textual language of the Quiz Server. It also enables localization, which is an important feature for multi lingual support.

9.6.6 Caching

The caching package contains the MemoryCache class and the IDataCache interface. We have created the MemoryCache class to enable intermediate storage of resources. It functions as an intermediate, when users request image/icon resources as thumbnails, as shown in Figure 7.30, Section 7.4.3. While the Quiz Server is running, the getData() method is called for every thumbnail request. A resource identifier is passed as a parameter and the thumbnail is returned if it is stored in the MemoryCache object. If the resource is not available, it is fetched from persistence, and the thumbnail is stored in the MemoryCache object via the setData() method. The next time this resource is requested, the getData() method returns it directly.

9.6.7 Servlets

The servlet package contains sub-packages such as “servlet.servlets” and “servlet.filters”. In addition to JSF, the Quiz Server use custom servlets, such as the DownloadImageServlet and DownloadPackageServlet classes. These work beside JSF and is handled by Java’s core servlet functionality.

The DownloadImageServlet handles icon/image requests on the Quiz Server web page. When an image is requested, this servlet is responsible for providing the image data.

The DownloadPackageServlet is responsible for serving quiz packages over HTTP. Both the Quiz Server and the Presentation Client utilize this.

Filters

The "servlet.filters" package contains the GeneralFilter and MultipartFilter classes. Both uses the @WebFilter Java annotation, as described in the Java API.

The GeneralFilter class is set up to listen to all web page requests regarding "quiz", "question", "statistics", "users" and "resource". This is the first filter in line, and is used to redirect a user to the login page if the supplied session cookie is invalid. In all other cases, it will call the chain.doFilter() method which send the request to the next filter in the chain.

The MultipartFilter class listens to three specific web page requests; namely the "uploadimage.xhtml", "uploadicon.xhtml", and "uploadpackage.xhtml". As described in Section 9.6.3, it handles file uploads, where it wraps the HttpRequests inside the MultipartRequestWrapper class. This is a necessity, since JSF does not come with logic for handling Multipart forms.

9.7 Summary

During the master thesis, we spent four months developing Lecture Quiz 3.0. This resulted in seven different code projects and three usable applications. These projects contain about 75000 lines of code, divided between 900 code files. We focused on keeping the code-base as clean and free for duplications as possible.

Chapter 10

Lecture Experiment

In this chapter, we present our experiment where Lecture Quiz was tried out in a lecture environment. For details about the experiment method used, see Section 3.1.

In the first sections, we present the context, delimitation, participants and environment of the experiment. We made five success criteria, and these are presented as SC1 to SC5. Finally, we describe the execution of the experiment. In Chapter 11, we present the results, evaluation and our conclusion of the experiment.

10.1 Delimitation

The goal of this experiment was to test Lecture Quiz in a real lecture environment. This way we could observe the system's pros and cons. In addition we could obtain written evaluations from the students participating via a questionnaire. This questionnaire can be seen in Appendix B. Complex statistical analysis and comprehensive psychological analysis, were out of the scope in this project, since it is not only a research project, but also a development project. Because of this, the results cannot be generalized. However, trying out the system, and asking potential future users what they think, could reveal useful information and arguments that we might use for continued work on finishing and commercializing Lecture Quiz. The results also give indications for use in various lecture environments. We tested usability, functionality and to some degree performance of the Lecture Quiz 3.0 system. In addition to this, we tried to measure whether the system had an increased effect on students' learning, during the lecture.

10.2 Experiment Context, Environment and Participants

The experiment took place on May 10th, 2011, at lecture hall KJ2 at NTNU, Trondheim, Norway. The experiment was prepared by us, and led by our supervisor and associate professor Alf Inge Wang. Wang had the role as the lecturer, and held a summary lecture the hour preceding our experiment. Thus the students were warmed up on the theme of the questions, which is a preferable situation for executing a lecture quiz session. The lecture hall had one wireless access point available to the devices of the students, as well as the lecturer. A modular connector was also available for local area network (LAN) connection, but was not used. A Projector and big-screen were a necessity, and also available in the lecture hall.

In addition to this, the students were sitting together in a way that made it possible to discuss, and communicate with the lecturer and ourselves. We believe that, e.g. the font size on the big screen was suitable for the specific size of the group of students, and thus size of the lecture hall.

All the participants were students taking the TDT4240 Software Architecture course. 75 students attended the lecture, and 62 of these were equipped with devices that managed to access our Player Client. They used their own mobile equipment - mostly smart phones, tablets and notebooks - to participate through a JavaScript enabled web browser.

Because the WLAN infrastructure did not withstand the amount of requests, we had to ask the students to form groups with one to four students on each team. This resulted in 18 mobile clients connected, instead of the first 62.

We assume that the participants had experience with a wide range of different computer software, since most of them study computer science. None of them had tried Lecture Quiz in advance of the experiment.

10.3 Success Criteria

In this section, we present a set of success criteria, describing the most important goals of our experiment. Confirmation of the criteria was regarded as success.

SC1 - The client software should run on 90 percent of the clients.

SC2 - The system is considered user-friendly.

SC3 - The system is not considered having a distracting effect on the lecture.

SC4 - The students think the system has a positive effect on the lecture and learning.

SC5 - The students find the system inspiring and fun.

10.4 Experiment Execution

As mentioned, theory from the semester was summarized and discussed in the first part of the lecture. The experiment took place in the second part of the lecture, after a short break. The students were allowed to ask questions any time. The experiment was videotaped, for documentation and analysis.

10.4.1 Network Infrastructure Problems

The lecturer started Lecture Quiz' Presentation Client and connected it to the projector. The URL of the Player Client is shown in the bottom left corner of the Presentation Client, as shown in Appendix E. We also wrote this address down on the blackboard. As this was in the break, we had not announced anything yet. Students started to connect to the Player Client with their devices; they logged in and students recognized their username on the projector screen. When all students were back from the break, we announced how to connect to the Player Client. This was basically unnecessary, as almost all the students were already connected and waited for the next step.

Network issues occurred when the lecturer started the quiz. All the student equipment and the laptop running the Presentation Client were connected to the same WLAN. This caused an amount of requests and responses that the network equipment did not support, and made the game very sluggish. In addition to this, the lecturer's Wi-Fi equipment was bombarded with requests, far more than it supports. This led to that both the lecturer's Wi-Fi and the WLAN did not cope with the traffic. We tried to start the game, but none of the students' clients got any responses, and we had to turn off the Presentation Client software. We asked the students to team up on fewer devices, as described in Section 10.2. The result: the previously 62 connected devices were now reduced to 18. This was not the original intent, but merely a necessary adaptation as it would be too cumbersome to move to a lecture hall with optimal wireless infrastructure.

10.4.2 Running Lecture Quiz

The lecturer picked the *Software Architecture* quiz, containing 22 questions. He selected the "Free for all" game type, which means that the 18 players compete with each other. Because we had to limit the number of clients, multiple students shared one device. Further, six game modes were added, mixing Point Building and Last Man Standing. He gave a short introduction to what were to happen next and started the quiz.

The students played through the software architecture questions. After the gaming session, we handed out questionnaires to the students who participated in the quiz game. The results from the questionnaires and observation are presented in Chapter 11.

10.5 Follow-up Test

After analyzing our results, we improved Lecture Quiz with regards to the problems found during the experiment. We ran a small test in lecture hall F1, to evaluate these improvements. This lecture hall supports more than 500 participants and has six WLAN access points.

The Presentation Client ran on our laptop and was connected to the network, via a modular connector. We connected it to the lecture hall's audio and video equipment, and it was shown on a big-screen. Our supervisor lent us 18 devices which we connected to the Presentation Client with the help of four colleagues. Totally, we used 22 devices: 15 iPods, 3 iPhones, 4 HTC phones, and two laptops.

We ran all five available quizzes, with a total of 62 questions. The session lasted more than one and a half hour, yet our colleagues did not want to stop playing. This time, both the Presentation Client and the player devices ran smoothly, and there were no network infrastructure issues. The results are presented in Chapter 11.

Part V

Evaluation

Chapter 11

Experiment Summary

In Chapter 10, we presented our lecture experiment. In this chapter, we present the results, evaluate, and conclude the experiment based on some success criteria.

11.1 Results

In this section, we present the results from the lecture experiment, as described in Chapter 10. These results are divided into three sections: the questionnaire, the video material, and the follow-up test.

11.1.1 Questionnaire

In this section, we present the results from the questionnaire. We have divided the questionnaire into four sections: SUS score, technical, client and learning. This is the same breakdown as in the evaluation form, as seen in Appendix B.

SUS Score

The second page of the questionnaire contains a System Usability Scale (SUS). Some of the participants answering our questionnaire only answered the first page, leaving the second page blank. We had to omit these from the result, giving us 30 valid questionnaires for the SUS calculation. To calculate Lecture Quiz' SUS score, we decided that unanswered questions will give the score 2, which represents neutral. This is also recommended by the authors of the SUS system [36]. In addition to this, even and odd questions are calculated differently, this is explained in detail in Section 3.1.1.

The Lecture Quiz game received a SUS score of 81 out of 100, despite network infrastructure issues. This score has been calculating by summing each of the ten questions, as shown in Table 11.1. We have also added a column which represents the average of the raw data.

#	Question	Avg	Score
1	I think that I would like to use this system frequently	3.57	6.42
2	I found the system unnecessarily complex	1.6	8.50
3	I thought the system was easy to use	4.3	8.25
4	I think that I would need support of a technical person to be able to use this system	1.23	9.42
5	I found the various functions in this system were well integrated	3.47	6.17
6	I thought there was too much inconsistency in this system	1.87	7.83
7	I would imagine that most people would learn to use this system very quickly	4.67	9.17
8	I found the system very cumbersome to use	1.67	8.33
9	I felt very confident using the system	4.03	7.58
10	I needed to learn a lot of things before I could get going with this system	1.27	9.33
	Total SUS Score		81.00

Table 11.1: Lecture Quiz SUS scores.

Technical

The technical part of the questionnaire contains questions regarding connection, mobile brand, etc., as shown in Appendix B. In this section, we present the results from this part of the questionnaire. The results are divided into four questions, where each is numbered as in the questionnaire:

4. What connection did you use?

Before we held the experiment, we assumed most of the students would use a Wireless LAN connection. The lecture hall has support for WLAN, but other connections, such as 3G, is also a decent alternative. We divided the question into four different connections: Wireless, Cable, 3G and GPRS/EDGE. Based on 34 student answers, 32 used WLAN and one used 3G.

5. What brand is your mobile/computer?]

This question is rooted in the versatility of our software and is described in Section 3.1. We assumed that most of the students would use notebooks or some variant of smartphone, but we wanted to test our software on as many mobile platforms as possible. The results showed that 17 of the students used notebooks, where Linux, Mac OS X and Microsoft Windows were fairly distributed. 10 of the students used HTC and 2 used iPhone. There were four other brands used, with only one of each. The devices were: iPod, iPad, Nexus and Samsung. One of the students chose not to answer this part of the questionnaire. A composition of the different brands can be seen in Figure 11.1.

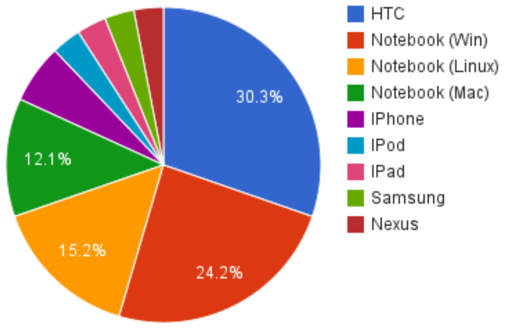


Figure 11.1: Device brand pie chart.

6. What operating system did you use?

This question is meant as a follow up question. It is used to give additional information in cases where the user reports client trouble. This is to increase the chance of reproducing eventual client bugs or errors. As seen in the previous question, the notebook users were distributed between the Linux, Microsoft Windows and Mac OS X OSs. The OS distribution of the notebooks can be seen in Figure 11.2 and smartphones in Figure 11.3.

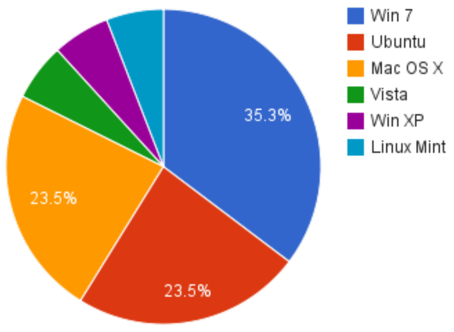


Figure 11.2: Notebook OS distribution.

7. What web browser (and version) did you use during the test?

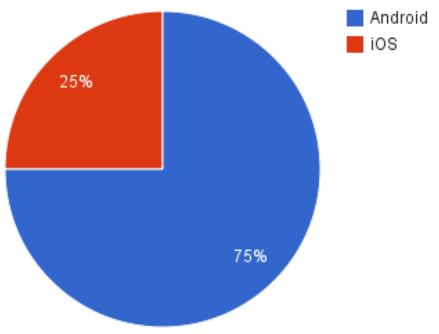


Figure 11.3: Smartphone OS distribution.

This question is related to the usability of our system. We want to support as many web browsers as possible. The question is tightly connected with Question 16a, as the combination allows us to see which web browsers work with Lecture Quiz. The web browser distribution of the notebooks can be seen in Figure 11.4 and smartphones in Figure 11.5.

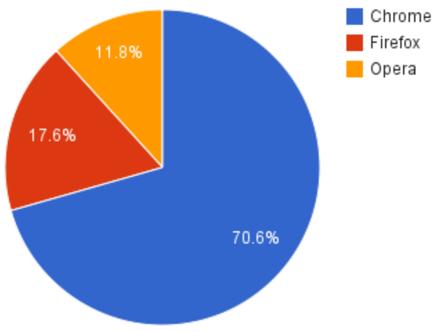


Figure 11.4: Notebook browsers pie chart.

Client

The Client part of the questionnaire contains questions regarding client issues. It gives us an indication if the client software worked properly on the students' devices. This part has one yes/no question and two qualitative questions. We will

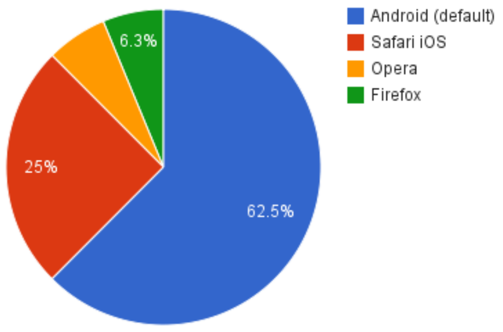


Figure 11.5: Smartphone browsers pie chart

try to summarize the most important qualitative contributions.

16a. Did the client software work properly on your phone/computer?

Because of the network issues during the initializing of our experiment, many students have answered that the client software did not work properly. The students have commented that the software did not work properly because of technical errors, such as delay and WLAN faults. There were no issues apart from the network issues, and from this point we will only address network related issues. To give more depth, we have separated the results where the students said the software worked with network issues and without. The distribution of the notebook users can be seen in Figure 11.6, while Figure 11.7 represents the smartphone users. We created a combined distribution of both notebooks and smartphones, as seen in Figure 11.8.

16.b If no; please describe the problem?

This is the first of two closed-ended questions in our questionnaire. We are interested in finding out why the software might not work properly, and being able to reproduce/fix the problem? Of the 34 students, 16 answered this question. 10 of these replies stated in few words, that the WLAN was overloaded. Two of the students complained that the client felt “a bit slow” and “some delay”; both using notebooks. Two students had problems with auto-refreshing of the page. One replied “bit buggy during refresh”, where one stated that “one time I needed to refresh”. One student lost his connection during the quiz, but no more information is given. One student answered “No text on buttons?”, which was how the client was purposefully designed.

17. Are there anything else you would like to comment?

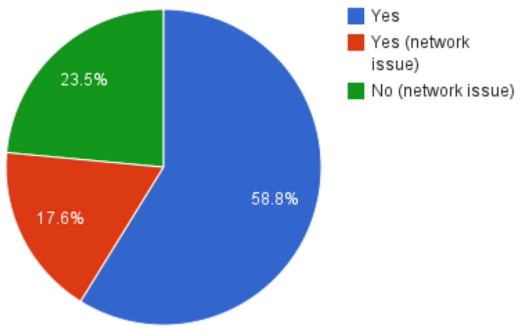


Figure 11.6: Notebook work properly distribution pie chart.

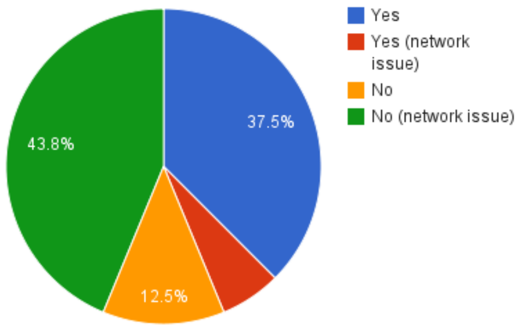


Figure 11.7: Smartphone work properly distribution pie chart.

This is the second and final closed-ended question from this questionnaire. 12 of the 34 students replied to this question. It encourages the students to give their final marks about the system, in their own words. Many uses this for suggesting further improvements, while others just state their feeling about the system. Responses of this type were “Nice!”, “A very good idea. I like it!” and “Great system”. Several mention that the green and yellow button on the Presentation Client is almost indistinguishable. Another student goes in depth of this problem and suggests to use some kind of number or letters to represent the buttons. He adds that color blind players would otherwise have a hard time distinguish the answers. One student mentions that sound would be nice. Two students elaborate the network problems, and both propose to use an external server to solve these problems. We only received

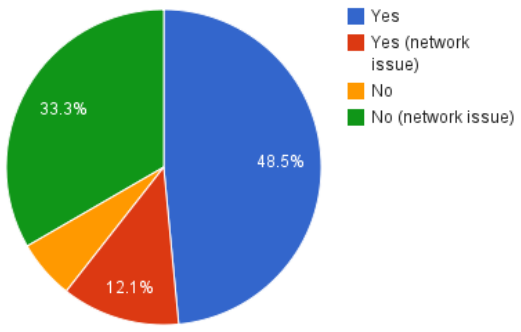


Figure 11.8: Overall work properly distribution pie chart.

one comment about learning, and it is so remarkable that we cite it: “Nice way to make students aware that they know less than they think”.

Learning

The Learning part of the questionnaire contains eight questions, and addresses Lecture Quiz’ effect on learning. These questions are numbered 8 to 15, as seen in Appendix B. All questions are on the same notation as the System Usability Scale (SUS), which is described in Section 11.1.1. Scores are ranked from 1 to 5, which is represented as “strongly disagree”, “disagree”, “neutral”, “agree”, and “strongly agree”. We have summarized all the questions and calculated their unique average scores, as shown in Table 11.2.

#	**Question**	**Average**
8	I like to compete	4.15
9	I think I paid closer attention during the lecture because of the system	3.88
10	I found the system had a distracting effect on the lecture	1.94
11	I think I learn more during a traditional lecture	2.5
12	I found the system made me learn more	3.38
13	I found the system made the lecture more fun	4.41
14	I think regular use of the system will make me attend more lectures	3.62
15	I like the way I can interact in the lecture using the system	3.97

Table 11.2: Learning scale average distribution.

11.1.2 Observation

We watched the video to observe the reactions of the participants. In the video, the students are looking toward the big-screen for long periods of time, and then at each other after the questions are given by the lecturer. They are discussing eagerly with one another, and laughing. They seem to be laughing in the "right" places, i.e. not like if they find Lecture Quiz ridiculous, but like if they find it enjoyable. For example, they laugh as the scores change in the big-screen, in front of everyone. Thus it seems like they experience it as a competition, or game. In general we get the impression that they are engaged and participate with pleasure. They seem to take it seriously, as if it might have some academic value. We assume that the lecturer represents a kind of authority, that the students "trusts" to give them academic quality. The questions used in the specific quiz are possibly also a "proof" to them that participation might be useful.

11.1.3 Follow-up Test

In our second test, we found out that the system supported all player clients used during the test. 15 iPods, 4 HTC phones, 3 iPhones, and two notebooks were represented. None of the clients had to relog in between the different quizzes. The notebook running the Presentation Client ran stable during the whole test, and no application restart was needed.

21 of the devices was connected though WLAN and one via 3G. During this test, none of the participants noticed any WLAN or 3G issues.

Based on the observations we did during the test and in the aftermath, the participants stated that they found the session both educational and fun. Two of our colleagues even told us they would appreciate a new Lecture Quiz gaming session the following week, but unfortunately we did not have time for this.

11.2 Evaluation

In this section, we evaluate the results from Section 11.1. In Section 10.3 we presented five success criteria, SC1 to SC5. These will be evaluated based on the results obtained from the questionnaire, as shown in Appendix B.

SC1 - The client software should run on 90 percent of the clients.

Based on the findings in Section 11.1.1, only two students stated that the client software did not work properly. We excluded the 11 students who answered no because the WLAN broke down. Based on this, only 2 out of 34 participants could not get the client to work. This gives a coverage of 94.1 percent, which satisfy this success criterion. In the follow-up test, all 22 clients worked properly, which give additional support to this success criterion.

SC2 - The system is considered user-friendly.

This success criterion is related to the usability of the system, and we have measured this utilizing a System Usability Scale (SUS) score. The Lecture Quiz game received a SUS score of 81 out of 100, as described in Section 11.1.1. The average of SUS score results have been thoroughly researched the past two decades. This research shows that the average SUS score lies between 52 and 68 [60]. A SUS score of 81 suggests that our system can be considered user-friendly. We had WLAN trouble during the initializing of our experiment, and this seems to have affected the test data. We believe that the SUS score would be higher in an experiment without network trouble.

SC3 - The system is not considered having a distracting effect on the lecture.

Statement 10 in Table 11.2 states that “I found the system had a distracting effect on the lecture”. The average score is 1.94. All 34 participants answered this statement, and the score represents that the students disagree. These results can be seen as a clear indicator that the criterion has been satisfied. In future releases, we would like to decrease this score even more.

SC4 - The students think the system has a positive effect on the lecture and learning.

This success criterion is related to the learning part of the experiment results, as seen in Section 11.1.1. Five of the statements in Table 11.2 can be considered relevant to this criterion.

As seen in the evaluation of H3 and statement 10, the students disagree that the system had a distracting effect on the lecture. In addition to this, statement 9, states “I think I paid closer attention during the lecture because of the system”. This statement received an average score of 3.88, which means the consensus is closer to agree than neutral.

Statement 12, states “I found the system made me learn more” and received an average score of 3.38. This is directly related to learning, but the word “more” is ambiguous and can have affected the score. Either how, the average student is more positive than neutral, toward the statement. This statement is closely related to statement 11, which states “I think I learn more during a traditional lecture”. The average score of this statement is 2.5, which indicates that the student consensus is between disagree and neutral. If compared to statement 12, similarities indicate that the Lecture Quiz system has a positive effect on learning.

We created statement 14 to find out whether students think regular use of the system will make them attend more lectures. This statement got an average score of 3.62, which means that the consensus is closer to agree than neutral. This result indicates that slightly more students will attend lectures.

The average score of the learning statements of the questionnaire has positive results, which indicates that the system has a positive effect on the lecture and learning.

SC5 - The students find the system inspiring and fun.

Statement 13 in Table 11.2 gave an average score of 4.41 out of 5, where "5" represent strongly agree. The statement "I found the system made the lecture more fun" was answered by all 34 participants. This result can be seen as a clear indicator that the Lecture Quiz game had a positive effect on how fun the students found the lecture.

Statement 8 states "I like to compete" and this can be related to our goal of making the system inspiring. All students answered this statement, and the average score was 4.15. Statement 15 is also related to the inspiration of students and states "I like the way I can interact in the lecture using the system.". This got an average score of 3.97. This places the consensus between agree and strongly agree, which makes us reach our goal of making the system inspiring.

In addition to this, during the follow-up test our colleagues said they enjoyed the quiz session and requested a recurring event.

11.3 Conclusion

In the first phase of our experiment, we experienced technical problems with the network infrastructure, and the result from the client part of the questionnaire support this. As presented in Section 11.1.1, 12 of the 34 participants gave a comment on why the client software didn't work properly. All of them replied in the lines of network infrastructure problems. Despite these problems, we have received high scores from both the SUS and the learning part of our questionnaire. However, in the follow-up test, all represented client devices worked properly, and none of our colleagues had any technical issues. The next time we do a large scale experiment of the Lecture Quiz game, we will ensure that the network infrastructure will support the amount of participants. Given a suitable network infrastructure, we feel confident that the Lecture Quiz would get an even higher score.

The evaluation of the success criteria, indicated the following trends:

- The software works on at least 90 percent of the clients.
- The system supports running games in a medium sized lecture hall, with 30 to 80 students.

- Most students considered the system user-friendly, and easy to start.
- The system did not have a distracting effect on the lecture.
- Most students found the system had a positive effect on the lecture and learning.
- The students found the system both inspiring and fun.

Chapter 12

Project Evaluation

In this chapter, we evaluate our research, development and documentation process. We reflect on whether we have managed to follow the research method we chose initially, the Scientific method. In addition we analyze the functional requirements and quality requirements, and discuss how well these are met in our system.

12.1 Project Evaluation

In this section, we give an overall evaluation of our own contribution, described in Part IV.

12.1.1 Research

We started this master thesis evaluating our specialization project, by analyzing the conclusion and further work. We had several new ideas for improvements, so we created the technology chapter by testing and comparing technologies and frameworks. This proved out to be a good baseline, because we described our research questions, research method and our lecture experiment.

In our specialization project, we looked at similar solutions that are used in educational environments. This information has been of good use to us in this master thesis as well, although some of it had to be updated to cover the last six months. By examining similar solutions' structure and approach, we found that there are no related solutions with the same game concept using our technical infrastructure in lectures. This gave us motivation and belief that this system can be used, not only in lecture environments, but also commercially when it is finished.

This master thesis is based on the work we did in our specialization project. However, our code-base is new and has no elements of code created by the previous lecture quiz groups. We reused some parts that we wrote during the specialization project, but the overall architecture and implementation can be seen as completely re-hauled.

12.1.2 Development

Concretizing the requirements, was an important step toward dividing functionality between the different components. Knowing which features each component would support, we could divide the development between the two of us. This improved the development pace as we could work independently where pair programming did not give a bonus.

In comparison to the specialization project, we have learned each other coding techniques better, leading to increased synchronization. This has led to a better understanding of each other's ideas and mindset. Because of this, we have managed to do a lot of brainstorming, which has led to a flexible architecture and good overall quality. Pair programming has been an important part of our development process, and this has helped us overcome the more complex parts.

Motivation has been a key factor in the development of Lecture Quiz 3.0. Our goal was to create a system that people would use and this had a positive effect on our motivation and effort. The use of exciting technologies has also helped keep us motivated during the development process.

12.1.3 Documentation

Documentation is an important factor of any master thesis, and this report is the main input of evaluation. We decided to set a code freeze date to ensure we got enough hours to work with this report. This gave us a good start in getting solid documentation of our system.

Due to our extensive architecture, we have created several architectural diagrams. These are represented in the documentation with different views. This helped create an easy separation for different stakeholders when it comes to finding relevant information.

We spent most of the project time developing the components for the game. This means that we had to create documentation that reflects the hours of work we put into the project. This resulted in a large own contribution part; Part IV. In addition we developed a user guide, using screenshots to illustrate some of the steps.

To increase the quality of the report, we interchanged the parts we have written individually. By iterating this process we had a new pair of eyes with a potentially different mindset on all parts of the documentation. Following discussions and comments improved the thesis.

12.2 Research Method

As described in Chapter 3, we chose to follow the scientific research approach. We have followed the engineering method, but have also used the empirical method as a part of the process. The engineering method has made it possible to experiment with the combination of new ideas, and things that have been done earlier. Our technology choices are based on the evaluation and testing of previous work, platform compatibility and personal preferences. We have tested several implementation ideas, and evaluated which ideas we believe improve the framework the most.

Using a lecture experiment, we received feedback on the usability of Lecture Quiz. To measure the usability, we used the System Usability Scale. The SUS result was presented in Section 11.1.1, and suggest that Lecture Quiz works on most of the commonly used devices. The experiment revealed issues with the network infrastructure, which has been an important factor in defining further work, as seen in Section 14.1.1. After the experiment, we evaluated its results and observations, and improved the Lecture Quiz code-base according to this. A few weeks later, we performed a follow-up test where Lecture Quiz ran smoothly on all present devices. In addition to this, there were no network infrastructure issues.

12.3 Functional Requirements

After the prestudy, we created functional requirements, as described in Section 6.1. In this section, we give a short description on how each functional requirement is made possible for the user. The requirements consist of the most important functionality we want Lecture Quiz 3.0 to have. They are formatted as user stories, and we present all the requirements as a whole.

During developing Lecture Quiz, these requirements have functioned as a red thread, and are aimed at students, lecturers and developers. All of our functional requirements have been satisfied, which is a 100% coverage. In addition to covering the functional requirements, we have created a good basis for meeting future requirements.

12.3.1 Presentation Client

The Presentation Client is responsible for running quiz games. This means that most of the functional requirements for the Presentation Client involve setup and execution of a quiz game. The Presentation Client is intended for the lecturer, and the requirements are mostly based on the lecturer's needs.

FR1 As a lecturer, I want to run quiz games, so that I can test the knowledge level of my students.

By using the Presentation Client, a lecturer can run games based on a quiz and a composition of game modes. The Presentation Client offers game statistics to the Quiz Server at the end of each game. These statistics can be viewed on the Quiz Server client, to evaluate the knowledge level of the students.

FR2 As a lecturer, I want to start team games, so that I can divide the students into groups.

The Presentation Client offers the lecturer a choice of three different team modes. The different modes are two teams, three teams and four teams. Each mode allows the lecturer to divide students into teams.

FR3 As a lecturer, I want to start free for all games, so that I can put each student up against each other.

When starting a quiz game with the Presentation Client, the lecturer can select the "free for all" mode. This mode does not divide students into teams, but instead let them play against each other.

FR4 As a lecturer, I want to display my computer address, so that I can tell students where they can access the game.

The Presentation Client reads the computer's address from the Player Client. The address is presented in the lower left corner of the Presentation Client. This address is displayed in every screen before the game starts, so that students can join while the lecturer is setting up the game.

FR5 As a lecturer, I want to choose a quiz, so that I can decide what type of questions the students have to answer.

The Presentation Client reads quizzes from the local file system and receives quiz lists from the Quiz Server. When the lecturer creates a game, the combined set of quizzes are available.

FR6 As a lecturer, I want to view a quiz, so that I can get a description of what the quiz contains.

The lecturer selects a quiz which is used during a game, in the Presentation Client. In this screen, the Presentation Client displays extended information about each quiz. The information shown is description, author, question count and more.

FR7 As a lecturer, I want to select one or more game modes, so that I can create variation in the game play.

The Presentation Client allows the lecturer to select a combination of game modes. This is done by adding game modes to a list, and sorting the list to create game variation.

FR8 As a lecturer, I want to see how many students are on each team, so that I can decide if the teams are even.

When a lecturer is creating a team game in the Presentation Client, a team setup screen is displayed. This screen shows a list of players for each team and the number of players on each team.

FR9 As a lecturer, I want to see how many students are connected, so that I can check if everybody is playing.

The team setup screen in the Presentation Client, contains information about how many students are connected. The number of connected students is the sum of the number of players on each team.

FR10 As a lecturer, I want to move a student from one team to another, so that I can decide who plays on which team.

The lecturer can move players from one team to another in the Presentation Client. This is done by selecting a player in one of the team lists, then clicking the move left or move right button. By clicking one of the move buttons, the player is moved to the team to the left or to the right of the player's current team.

FR11 As a lecturer, I want to automatically even teams, so that I don't have to move a student from one team to another.

The team setup screen in the Presentation Client contains a checkbox called "Automatically even teams". This checkbox makes sure that the teams are even, as long as it is checked. If the teams are uneven, and the lecturer checks the checkbox, players will automatically be uniformly distributed.

FR12 As a lecturer, I want to let the students select teams, so that I can tell students to pick their own team.

The Presentation Client and the Player Client allows players to choose what team they want to play with. This is done in the Presentation Client's "team select" screen, by checking the "Allow clients team select" checkbox. While this checkbox is checked, the players are able to select team on their client device. If players choose another team, they are moved to the correct team list in the Presentation Client.

FR13 As a lecturer, I want to display information about the game mode, so that I don't have to tell the students how the game works.

When the Presentation Client runs a quiz game, it uses one or more game modes. Each time a new game mode is started, the Presentation Client will display information about that game mode. The information contains an explanation on how the game mode works for the end-user.

FR14 As a lecturer, I want the game to display questions and alternatives, so that I don't have to tell students the question and what they can answer.

During game play, the Presentation Client runs question rounds. At the start of each round, the question is presented to the students. After 5 seconds, the Presentation Client displays four buttons, which represents the different alternatives. The students see the same buttons on their device, except that they do not contain the alternative text. By clicking one of the buttons, the student gives his one and final answer to that question.

FR15 As a lecturer, I want to decide when the next question appears, so that I can speak about the current question.

The game is paused when a question has timed out. During this pause, it displays the question, the alternatives, and if present, the question image. This pause gives the lecturer time to lecture about the question or topic. To resume gameplay, the lecturer can press enter, or click the left mouse button.

FR16 As a lecturer, I want to export statistics after a quiz, so that I can review them later.

After a game is finished, and the Presentation Client is not connected to the Quiz Server, it will offer the user to export statistics to file. This file can later be uploaded to the Quiz Server, where it can be evaluated to test the knowledge level of the students.

FR17 As a lecturer, I want statistics to automatically be uploaded to the server when connected, so that I don't have to export.

The Presentation Client sends game statistics to the Quiz Server after a game has finished. This is done automatically, if the Presentation Client is connected to the Quiz Server.

FR18 As a lecturer, I want to see student rankings, so that I can reward the best team or student.

When a game is finished in the Presentation Client, it displays player names, ranks and the score. The list is sorted by score, and allows both the students and the lecturer to see the student rankings.

FR19 As a developer, I want to add a new game mode, so that I can change the game logic.

This is supported by the Presentation Client, by extending the existing GameModeBase class. The class contains general methods for deciding game mode logic, which makes it easy for a developer to create new game modes.

12.3.2 Player Client

The Player Client is the part of Lecture Quiz that involves the players. While being run on the Presentation Client, it is presented on the students' client devices. This chapter involves the students' functional requirements, to be able to play games.

FR20 As a student, I want to log in, so that I can participate in a game.

When a student opens the Player Client, a login screen is presented. This screen allows students to login and join games.

FR21 As a student, I want to specify a username when logging in, so that I can identify my player.

The Player Client login screen contains a username textbox. This textbox is used to specify the student's username. The specified name is presented by the Presentation Client during gameplay. By looking for the username, the students can identify themselves in-game.

FR22 As a student, I want to pick an alternative, so that I can answer questions.

The Player Client presents alternative buttons during question rounds. These buttons look similar to the buttons on the Presentation Client, except that they do not contain the alternative text. By clicking a button, the student gives his one and final answer to the current question.

FR23 As a student, I want to see my score, so that I can track my progress.

Between and during question rounds, the Player Client displays the players' current score. This allows students to keep track of their progress in the game.

FR24 As a student, I want to select a team, so that I can join the team I want to play together with.

Before starting a team game, the Presentation Client can allow players to select team. This makes the Player Client display four team buttons, which the students can use to change team.

FR25 As a student, I want to pick a vote selection, so that I can affect my team's choices.

Some game modes in Lecture Quiz, requires users to vote. This is to allow each player to have a say in decisions made by the team. During votes, the Player Client displays vote buttons. These buttons are similar to the alternative buttons and allow the student to affect the team's choice.

FR26 As a student, I want to see my rank at the end of the game, so that I can determine how good I did in the quiz.

When a game is finished, the Player Client displays a summary screen. This screen contains information about the students' score and rank. Students can use this screen to determine how well they did on the quiz.

FR27 As a student, I want to see what I answered on a question, so that I can compare it to the results on the screen.

After a question has been answered by all the students, or the timer for the question has run out, the Player Client displays the students' answers. This allows the students to compare their choices to the alternatives being presented by the Presentation Client.

FR28 As a student, I want to see my nickname associated with a team, so that I know which team I am playing on.

The Player Client displays the player nickname on each screen after logging in. During team gameplay, the Player Client displays the students' team name besides the player nickname. This allows the students' to identify which team they are playing on.

12.3.3 Quiz Server

The Quiz Server is an administrative tool, that allows lecturers to manage quizzes and view statistics. The Presentation Client can be connected to the Quiz Server, when running quiz games. The functional requirements are intended for lecturers, as lecturers are the main users of the Quiz Server.

FR29 As a lecturer, I want to create quizzes, so that I can use them in lectures.

The Quiz Server client has a separate page for creating quizzes. Quizzes that are created with the Quiz Server, is automatically available in the Presentation Client. This allows the lecturer to use them during lectures.

FR30 As a lecturer, I want to edit a quiz, so that I can correct errors.

After a quiz is created with the Quiz Server, it is available in the quiz list. By selecting edit in the quiz list, the lecturer can make changes to a chosen quiz. This allows the lecturer to correct errors.

FR31 As a lecturer, I want to delete quizzes, so that I can remove unwanted quizzes.

The quiz list in the Quiz Server gives an overview of already existing quizzes. To delete a quiz, the lecturer can click the delete button in the quiz list. This allows the lecturer to remove unwanted quizzes.

FR32 As a lecturer, I want to log in, so that I can have a private account.

To access the Quiz Server, the lecturer needs to have a user account. The first page that appears in the Quiz Server client, is the login page. This allows lecturers to login using their private user account information.

FR33 As a lecturer, I want to see statistics for a quiz, so that I can evaluate student progression.

In the quiz list on the Quiz Server client, the lecturer can click the "Show statistics" button. This allows the lecturer to view statistics associated with the selected quiz, and evaluate student progression.

FR34 As a lecturer, I want to assign an icon to a quiz, so that I can create questions involving images.

The Quiz Server supports uploading of icons and images. Images can be assigned to quizzes, when they are created, or when editing a quiz. By adding an icon to a quiz, the lecturer can create questions involving images.

FR35 As a lecturer, I want to download a quiz, so that I can run the quiz without having a connection to the server.

The Quiz Server allows the lecturer to download quizzes. Downloaded quizzes are stored as quiz packages, containing all the information needed to run the quiz. Lecturers can use these packages to run quiz games, without being connected to the Quiz Server.

FR36 As a lecturer, I want to upload a quiz, so that I can get other people's quizzes.

To add quizzes from other servers, the Quiz Server supports uploading of quiz packages. By uploading quiz packages, the lecturer can use other people's quizzes.

FR37 As a lecturer, I want to be able to upload statistics, so that I don't have to be connected to the server when generating statistics.

Statistical packages are created when a lecturer has run an offline quiz game on the Presentation Client. The lecturer can upload this package to the Quiz Server client. The content of the statistical package will be added to its respected quiz. This allows the lecturer to run offline games, and still receive statistics saved on the Quiz Server.

12.4 Quality Requirements

In addition to functional requirements, we have created quality (non-functional) requirements, which are described in Section 6.2. These quality requirements are divided into three sections; modifiability, usability and availability. In this section, we describe how each requirement is integrated into our solution, and a test to confirm that they are fulfilled by our system. We include the headline of each quality attribute scenario.

In total, we have eight modifiability, two usability, and three availability requirements, where each has been confirmed by tests. Thus, all of our quality requirements have 100% coverage.

12.4.1 Modifiability

Lecture Quiz needs to be modifiable. This is because it is a new software in the lecture context and its feature demands may change. The modifiability quality requirements set for Lecture Quiz has been fulfilled. Testing has been done by us the developers, and can be regarded a bit biased, when it comes to timing. We still think that other developers will be able to extend Lecture Quiz within the response measure.

M1 Create a new game mode.

Game modes in Lecture Quiz 3.0 are created in the Presentation Client project. To make a game mode, the developer needs to create a class that implements the IGameMode interface. The current game modes included in the Presentation Client extend the GameModeBase class, as shown in Figure 7.17. By extending this class, the developer can override the methods he wants to modify the logic for. The GameModeBase class can be found in the "lecturequiz.presentationclient.logic.gamemodes" package. The game mode needs to be enabled in the game mode selection screen. This is done by creating an instance of the game mode in the GameModeProvider class, and then adding it to the game mode map. This is further explained in Appendix D.

To test this quality requirement, we created a sample team game mode. The game mode would run games, e.g., the "Point builder" game mode. The difference in gameplay, is that the scoring for each team depend on its players, to be able to answer fast and correct. For each correct player answer, the game mode will add this score to the team score. This also applies for wrong answers, but will result in decreasing the score. The creation of this game mode was completed in 20 minutes, which satisfies this quality scenario requirement.

M2 Multiple game modes.

This modifiability requirement is aimed at the lecturer. We have created a game mode selection screen in the Presentation Client, that allows the lecturer to select multiple game modes. The game modes are combined in a GameModeQuestionChain, as shown in Figure 7.16. This allows the quiz game to run different game modes for the same quiz, by dividing the questions between the game modes.

This quality requirement was met during the lecture experiment. The experiment ran two different game modes with success.

M3 Creating a new player client.

The Presentation Client uses the IPlayerClient interface to communicate with the Player Client. By implementing this interface to a new class, the developer is able to change the player client logic. To make the Presentation Client use the new Player Client, it has to be registered with the service locator. This is done in the LogicManager, in the initialize() method, as described in Appendix D.

To meet this quality requirement, we created a Player Client we could use for testing purposes. Our client implements the IPlayerClient interface, and runs in a separate thread. The new Player Client will add non-playable debug players, and make these players answer questions at random. By creating this class, we can test how many users the Presentation Client can handle at once. The new player client was created in one hour, which satisfies the response measure in the quality attribute scenario.

M4 Change client skin.

The Player Client used by Lecture Quiz 3.0 uses web pages and JavaScript to create the interface. We want this to be changed by a graphical artist, as there are a lot of design and graphics involved. To change the client skin, the artist can replace the images included in the client folder, in the Presentation Client project.

This quality requirement was met by replacing some of the new graphics with older, outdated graphics. The change was not pretty in any way, and cannot be compared to the time it takes to create real graphics. Pretty or not; it proves that it is possible to change the skin of the Player Client, within the requirement limits.

M5 Change the database back-end.

The Quiz Server use EclipseLink and JPA to manage the data storage. EclipseLink supports all commonly used databases, which is supported by JDBC. To change database back-end, the developer needs to edit the “persistence.xml” file. The “persistence.xml” file contains the settings EclipseLink use to communicate with the database. The Quiz Server project needs to include a reference to the JDBC connection drivers for the chosen database.

We tested this quality requirement by installing PostgreSQL. To connect EclipseLink with the database, we downloaded the PostgreSQL JDBC drivers. After including the drivers in the Quiz Server project, we configured the “persistence.xml” to point to the PostgreSQL driver and database. We chose to create a new database user for Lecture Quiz purposes, instead of using the PostgreSQL root user. As with any application using a database, EclipseLink requires these credentials to connect to the database. This took us 30 minutes to complete, and this quality attribute scenario is met.

M6 Change GUI composition.

The Presentation Client uses components and scenes to present the GUI. The composition of components on a scene is structured using the composite pattern. To change the GUI composition, a developer can add and remove components to a scene, and its sub components.

Close to the end of the process, we received help from a graphics artist to change the visual impression of Lecture Quiz. During the time, we worked with the design, we changed GUI composition several times, without any difficulties. This quality requirement is therefore met.

M7 Add support for a new input device.

The Presentation Client utilizes both keyboard and mouse input devices. This is done through the LWJGL library, encapsulated with well defined interfaces. By using the IInputDevice or IPointerDevice interfaces, a developer should be able to create support for another input device.

We tested this requirement by creating support for joysticks. To add support for a joystick device, we used LWJGL. The logic was included in a class that implements the IInputDevice interface. This was done in 20 minutes, and satisfies the quality requirement.

M8 Add support for Java SE compatible OS.

Lecture Quiz 3.0 was created with support for Microsoft Windows, Mac OS X and Linux. To extend the support to another platform, the developer needs to manage the graphics rendering part of the Presentation Client. This can be done by adding system files compatible with LWJGL for the new operating system, or changing the render implementation. The rendering implementation for the Presentation Client, has been separated from the rest of the logic, with interfaces. This makes it easy to create support for another graphics library.

We tested this requirement as a part of the development process. Testing has been done on Microsoft Windows and Mac OS X. To meet this requirement, we added driver support for Linux. The Linux version did not start as it could not find the drivers at first. This is because Java does not add the current folder to the class path as in Microsoft Windows and Mac OS X. To solve this, we added "." to the class path and it was working with Linux. This was done in 10 minutes and fulfills the quality requirement.

12.4.2 Usability

Lecture Quiz needs to be user-friendly. This is because it is aimed to be used in a context where there are already working solutions. To make people choose Lecture Quiz over the traditional lecture it has to be easy to use. The usability requirements have been tested by running a lecture experiment. Based on these results, Lecture Quiz meets the usability quality requirements.

U1 Support visual gaming interface.

The Presentation Client is responsible for running games in Lecture Quiz. We have created a visual interface that is intended for playing quiz games. The interface consists of a fullscreen graphical display, with sprites representing the GUI elements. We have created animations, ambient light effects, hovering effects, and transitions. These are common elements in gaming environments. The graphics and design have been enhanced by a graphical artist, to push the visuals even further toward a commercial gaming interface.

We tested the quality requirement during the user experiment. With respect to the experiment results, students agreed that using Lecture Quiz made the lecture more fun. Based on these results, we have satisfied the quality requirement.

U2 A common input interface.

The Player Client is responsible for collecting player input, when playing Lecture Quiz games. We have developed the Player Client interface to be as user-friendly and self explanatory as possible. For example, when answering questions, the Player Clients presents the alternative buttons as they would be presented on commercial quiz games, like Scene it? and Buzz!.

According to the experiment results, the students did not need any help using the client interface. The students agree with the statement “I felt very confident using the system”. The experiment results show that the students “did not need to learn a lot of things before they could use the client”. Based on these results, the Player Client satisfies this quality requirement.

12.4.3 Availability

Lecture Quiz needs to be stable and available on a wide range of client devices. If the application crashes during a lecture, it would waste both students and lecturers time. To avoid custom hardware to play the game, the client should work on the most common devices owned by students. The availability quality requirements set for Lecture Quiz has been fulfilled. Testing the client was done in the lecture experiment. Based on the results, the client worked on most devices and did not crash during the experiment.

A1 Reconnect to server.

The Presentation Client will establish a connection to the Quiz Server at startup. This connection may be lost during gameplay, and the Presenation Client needs a way to re-establish this connection. The Distributed Message Queue runs in a separate thread, and this thread checks if the connection to the server is lost. If the connection is lost, the Distributed Message Queue tries reconnecting to the Quiz Server.

We have tested this by running the Presentation Client connected to the Quiz Server. During gameplay, we stopped the server, and then restarted it, to make sure the connection was lost. The Presentation Client was able to reconnect to the Quiz Server and post the complete game statistics. This availability quality requirement is with basis in this, met.

A2 Simple web client.

The Player Client runs as an embedded web server on the Presentation Client. It is important that this web server does not crash, as this will disable communication between players and the game. The Player Client handles incoming connections with the use of a thread pool. This allows client requests to be processed in parallel, and helps the stability of the Player Client. The client returns a simple web page, that uses AJAX to update the interface.

To test this requirement, we used Httperf to simulate 1000 users. Httperf is a tool for measuring web server performance [49]. The Player Client spawned a total of 18 threads to handle the requests. The user interface was slower, but the Player Client did not crash. After we stopped the stress tool, the Player Client cleaned up all the threads as intended. According to this test, this quality requirement is met.

We also tested this requirement in a real lecture. The response measure for this requirement is that “the Player Client should not crash during a game session with at least 20 clients”. The first phase of the test was done with about 70 users. Given the network equipment used for the test, this many users made it error prone. In the second phase, we ran the test with about 20 clients and the game worked properly. This confirms that the client can run with 20 clients, without crashing.

A3 Run on different client types.

Lecture Quiz is developed to run on devices commonly used by students. This is to save the expense of having to buy custom hardware. To allow students to participate in Lecture Quiz games, the client needs to support a wide range of devices. We have developed the client with focus on multi-browser support.

This requirement was tested during the user experiment. According to the experiment results, 2 out of 34 students had problems running the client.

The experiment was performed on a wide range of devices, and the two devices with problems were iPod and iPhone. The related results are evaluated in Section 11.2, under success criterion SC1. We performed a follow-up test with both iPhones and iPods. This test shows that 16 out of 16 iPods, and the 3 iPhones worked. Confirmed devices are Android phones, iPhones, iPods, Windows notebooks, Mac notebooks and Linux notebooks. Confirmed browsers include Chrome, Firefox, Opera, Android and Safari. The experiment result yields no test data about Internet Explorer, but we have tested the client on IE 5, 6, 7, 8 and 9, and are quite confident that it has IE support. Based on these results we conclude that this quality requirement is fulfilled.

Part VI

Conclusion

Chapter 13

Conclusion

At the start of this master thesis, we defined four research questions. This was followed by searching for previous work, both regarding characteristics, research and related educational software, which gave us a good basis for our implementation. We did a thorough technology research, and chose technologies with this in mind.

We are satisfied with how our Lecture Quiz solution turned out, and have concluded the following. The Quiz Server is now a stable environment for creating quizzes and reviewing statistics. The Presentation Client can run fully playable quiz games, with different game mode logic, and looks and feels like a game. The Player Client runs on all intended devices, and is working great given the right network infrastructure. The overall Lecture Quiz solution is solid and fully playable.

During our master thesis, we have made a flexible architecture for use with the Lecture Quiz solution. We have created a new and better version, that is easy to use, modifiable and is more visually appealing.

The research questions have functioned as a guideline during the development of Lecture Quiz 3.0. In this chapter, we answer the four research questions, described in Chapter 2:

RQ1 Which systems related to Lecture Quiz exist today, and what differentiate them from ours?

In Section 4.3, we presented existing solutions that are related to Lecture Quiz. We summarized the features and concluded that i>clicker, JustVote and Buzz!: The Schools Quiz has elements that can enrich our framework.

The i>clicker software is the only other related system we looked at, that requires no custom hardware. They have created the web>clicker client that runs on the students' devices. This is a feature that is critical to our framework, and should continue to be a requirement. They have wide support for network infrastructure, which is also a goal for this framework. The drawback with web>clicker, is that it requires connection with the i>clicker hosting service, to run. Another drawback is the large infrastructure involved to use the software. i>clicker is not intended as a game, and thus not directly compete with the Lecture Quiz game.

JustVote is one of the solutions that is related to the Lecture Quiz framework. Their presentation client has animated graphics, several game modes, supports easy loading of prepared material, and handles up to 1000 users at the same time. Their quiz editor focuses on drag and drop, multiple question layout templates, and has extensive support for multimedia. Their game server supports an extensive range of statistical data, monitors progression on many levels, has the ability to review past sessions, and to print and export data. These are the features of JustVote that we want to embrace. They also have some drawbacks. Their animated graphics are plain, and look like a presentation slide-show. Their handheld devices are simple controllers, with no display, and few buttons for input. Custom hardware costs money, which makes it less interesting for educational use.

Buzz!: The Schools Quiz has a bigger focus on playability and the audio-visual. They have animated 2D and 3D graphics and voice-overs. In addition to this, they use in-game avatars. We found these features interesting when it comes to making Lecture Quiz feel like a game. They have a large integrated question base, but it is not modifiable, and you have to buy a new game to get additional content. Another drawback is that they have limited the player base to 8. They also require a Playstation 2 and custom hardware to play.

RQ2 How can we make Lecture Quiz easier for the end-user to install and use?

We have limited the number of enterprise applications needed to run Lecture Quiz. Application servers require additional work to set up and configure. Configuring applications to run on these servers, requires even more work. Lecture Quiz 3.0 is not dependent on any application servers to run quiz games.

We have created executable installation packages for Microsoft Windows and Mac OS X. This allows users to easy install the software needed, to run quiz games on their computers. The installation package will create launchers, so that end-users do not have to worry about executing the application through Java.

We have added support for mouse and keyboard devices, and they have been integrated with the graphical user interface. They have been seamlessly integrated into the application, with support for features like the mouse wheel. The application flow is controlled by common GUI elements that people encounter every day. This allows an easy understanding of how the application works without having to read any user guide or the alike.

We have embedded the Player Client into the Presentation Client, so that people do not have to start two different applications. This allows the Presentation Client to display the address of the Player Client, to the students, while the lecturer configures the game. This saves time when it comes to connecting players to the Lecture Quiz game, and makes the process more fluent.

Once deployed, the Quiz Server allows lecturers to create quizzes in the comfort of their own office. The quizzes will be automatically available to the lecturer when running the Presentation Client connected to the Quiz Server. It is also possible to download the quiz, and run it on locations without connection to the Quiz Server. This makes it easier to prepare quizzes and run them wherever needed.

RQ3 How can using a quiz game in teaching environments affect the students?

During our thesis, we ran a lecture experiment, where 34 students participated, and answered our questionnaire. This questionnaire can be seen in Appendix B. The results from this lecture experiment were presented in Section 11.1. This research question is related to the learning part of these results, and was presented in Section 11.1.1.

According to the audience that participated in the experiment, the consensus is that they like to compete. When people are competing, their senses are sharpened, which leads to increased attention and brain activity [37]. This is an important trait that the Lecture Quiz game brings to the lecture.

The average student agreed that they paid closer attention during the lecture, because of the Lecture Quiz game. The students reported that using Lecture Quiz was a fun alternative to the traditional lectures. More than half of the students, reports that they would attend more lectures if the system is regularly used.

RQ4 How can we make Lecture Quiz 3.0 look, act and feel like a game?

The Presentation Client is responsible for running quiz games. With the help of a graphical artist, we have enhanced the visual impression of Lecture Quiz games. We have aimed at a more commercial game look, when it comes to the overall layout and design of the graphical user interface.

We have added an ambient background effect. This effect makes the game feel more “alive”, as there will be constant subtle movement in the image. The Presentation Client uses transitions, when changing from one screen to another. The transition effect is a simple fade effect, and is commonly found in games. We have added animations and icons to the Presentation Client and the Player Client. This makes it look less like a regular desktop application.

To give the players a gaming experience, we have added several game modes. This is to create variation in the game play, and differentiate the application from a regular questionnaire/presentation. Quiz games are won by scoring the most points. Game modes like “Point stealing”, brings depth to the competition. To make the game play different from a questionnaire, each question is timed. This puts pressure on the players and further increase their competitive instincts.

Chapter 14

Further Work

In this chapter, we present our suggestions to finalize and further improve the Lecture Quiz game. We have divided the suggestions into two sections: technical and research aspects.

14.1 Technical

In this section, we try to summarize the technical changes which need to be done regarding a completed system. In addition to this we present a bulletin list of features that might improve Lecture Quiz even further.

14.1.1 Completing the System

During our testing of Lecture Quiz, we have encountered some network issues. A main concern is the amount of clients sharing the same WLAN. We overloaded the wireless local area network when running Lecture Quiz in a real lecture. This was partly because of the network configuration in the lecture hall, and the fact that the computer running the presentation used the same WLAN. This could have been solved by running the presentation on cable and inserting additional wireless access points. Further work with the system will include solving issues related to the network infrastructure.

We have some suggestions that might reduce the need for extra network equipment.

- Make the Player Client load graphics from a Quiz Server instead of the Presentation Client. This will reduce the initial load on the network communication to the Presentation Client.

- Extend caching on the Player Client and pre-load graphics.

- Reduce graphics, script and HTML size.
- Use web sockets instead of AJAX state polling.
- Tweak the time between each AJAX state request and create logic to adjust timing.
- Ad hoc wireless network branching, using notebooks that are connected with cable to the presentation. Note that it is essential to use different WLAN channels.

14.1.2 Other Suggestions

In this section, we list features that we think would improve Lecture Quiz. Some of the suggested features might not directly improve the game, but add versatility to the system.

- The use of Quick Response codes (QR codes), allowing clients to take a picture of the Presentation Client, that directs them to the Player Client web page.
- Allow the Player Client to be run on the Quiz Server to ease local WLAN traffic.
- Add different skins to the Player Client, depending on the player's team.
- Allow players to use avatars, possibly from camera images.
- Enable audio and video playback for questions.
- Create answer icons that link an alternative to a specific icon on the player's device.
- Visualized statistics for question results.
- The possibility to play quiz games, where players on separate physical locations compete with each other.
- Localization support on all the clients.
- Support for rating the difficulty of questions, based on statistics.
- Add achievements to the game summary screen.
- Add drag and drop support on the Presentation Client.

14.2 Research

In this section, we present suggestions for further research regarding the educational benefit from using Lecture Quiz. This includes research questions, conduction of experiments and different testing contexts.

We list several possible research questions for further work.

- How involved and committed are students when playing Lecture Quiz?
- How does number of students affect the gaming experience, when it comes to the social aspects, e.g. competition, communication, and cooperation?
- How does Lecture Quiz affect the student-teacher interaction?
- In which ways can Lecture Quiz affect the learning effectiveness?
- Which game modes and game types are most effective?

These questions can be answered by conducting several scientific experiments, with control groups, etc., to find more certain evidence. An example of a scientific experiment is:

Create two different conditions, where one group of students is exposed to Lecture Quiz and one is not. Measure level of knowledge before and after, e.g. by handing out a test and giving grades. The groups should have the same average grade, and then after they have been exposed to Lecture Qiuz, measure them again. During the period between the measurements, there should not be anything else than Lecture Quiz affecting the learning of the groups as a whole. If the Lecture Quiz group has a higher average grade, then one could say that “Yes, Lecture Quiz has a positive effect on learning”.

To obtain generalizability, Lecture Quiz has to be tested in different contexts, i.e. outside lecture environments. Examples of this can be: specialist courses, training, hospitals, and business meetings.

Bibliography

[1] Sun Microsystems a Oracle Corporation subsidiary. Java document type definition. `"http://java.sun.com/dtd/"`, January 2005. Accessed 26. June, 2011.

[2] Elie A. Akl, Richard W. Pretorius, Kay Sackett, W. Scott Erdley, Paranthaman S. Bhoopathi, Ziad Alfarah, and Holger J. Schünemann. The effect of educational games on medical students' learning outcomes: A systematic review: BEME Guide No 14. *Medical Teacher*, 32(1):16–27, January 2010.

[3] Alex. Streaming in openal. `http://www.alexcurylo.com/blog/2010/05/15/streaming-in-openal/`, May 2010. Accessed 26. June, 2011.

[4] Anand. JavaServer Faces (JSF) Tutorial. `http://www.developersbook.com/jsf/jsf-tutorials/jsf-tutorials.php`, January 2008. Accessed 26. June, 2011.

[5] Henning Bär, Erik Tews, and Guido Rößling. Improving feedback and classroom interaction using mobile phones. Master's thesis, Darmstadt University of Technology, 2005.

[6] Paul Barribeau, Bonnie Butler, Jeff Corney, Megan Doney, Jennifer Gault, Jane Gordon, Randy Fetzer, Allyson Klein, Cathy Ackerson Rogers, Irene F. Stein, Carroll Steiner, Heather Urschel, Theresa Waggoner, and Mike Palmquist. Survey research. writing@csu. colorado state university department of english. `http://writing.colostate.edu/guides/research/survey/com4a2a.cfm`, 2005. Accessed 26. June, 2011.

[7] Victor R. Basili. The experimental paradigm in software engineering. In *Proceedings of the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 3–12, London, UK, 1993. Springer-Verlag. Available from: `http://portal.acm.org/citation.cfm?id=647362.725507`.

[8] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, April 2003.

[9] Mark Bates. *Distributed Programming with Ruby.* Addison-Wesley Professional, 1st edition, 2009.

[10] Gillian Blakely, Heather Skirton, Simon Cooper, Peter Allum, and Pam Nelmes. Educational gaming in the health sciences: systematic review. *Journal of Advanced Nursing*, 65:259–269, August 2008.

[11] Tom Bramwell. Buzz!: The schools quiz - interview. `http://www.eurogamer.net/articles/buzz-the-schools-quiz-interview`, 2008. Accessed 26. June, 2011.

[12] Erling A. Børresen and Knut A. Tidemann. Lecture quiz 2.0 - a service oriented architecture for educational games. Master's thesis, NTNU, June 2010.

[13] K.L. Calvert and M.J. Donahoo. *TCP/IP sockets in Java: practical guide for programmers.* Morgan Kaufmann practical guides series. Elsevier/Morgan Kaufmann, 2008. Available from: `http://books.google.com/books?id=lfHo7uMk7r4C`.

[14] Scott Carlson. The Net Generation Goes to College. *The Chronicle of Higher Education*, October 2005.

[15] Oracle Corporation. Jsr 52: A standard tag library for javaserver pages (maintenance draft review 3). `http://jcp.org/en/jsr/detail?id=52`, October 2006. Accessed 26. June, 2011.

[16] Oracle Corporation. Java se community - open-source jdk. `http://www.oracle.com/technetwork/java/javase/community/opensourcejdk-jsp-136417.html`, May 2007. Accessed 26. June, 2011.

[17] Oracle Corporation. Java persistence 2.0. `http://jcp.org/aboutJava/communityprocess/final/jsr317/index.html`, December 2010. Accessed 26. June, 2011.

[18] Oracle Corporation. Java: The best environment for network-based applications. `http://www.oracle.com/us/technologies/java/10045230-br-java-c17307-187867.pdf`, 2010. Accessed 26. June, 2011.

[19] Oracle Corporation. Jsr-000314 javaserver faces 2.1 (maintenance release 2). `http://jcp.org/aboutJava/communityprocess/mrel/jsr314/index2.html`, November 2010. Accessed 26. June, 2011.

[20] Oracle Corporation. Jdk 6 java database connectivity. `http://download.oracle.com/javase/6/docs/technotes/guides/jdbc/`, 2011. Accessed 26. June, 2011.

[21] Oracle Corporation. Oracle glassfish server. `http://www.oracle.com/us/products/middleware/application-server/oracle-glassfish-server/index.html`, 2011. Accessed 26. June, 2011.

[22] Oracle Corporation. Overview (java platform se 6). `http://download.oracle.com/javase/6/docs/api/`, 2011. Accessed 26. June, 2011.

[23] University of Washington Department of Computer Science & Engineering. Uw classroom presenter. `http://classroompresenter.cs.washington.edu/`, 2008. Accessed 26. June, 2011.

[24] University of Mannheim Department of Computer Science IV. Uce project. `http://pi4.informatik.uni-mannheim.de/pi4.data/content/projects/wil-ma/`. Accessed 26. June, 2011.

[25] Kristian Døvik and John Andre Hestad. Lecture quiz 2.5 - combining education, usability and gaming experience. Master's thesis, NTNU, December 2010.

[26] T. Erl, D. Chou, J. deVadoss, N. Gandhi, and H. Kommapalati. *SOA with .NET and Windows Azure: Realizing Service-Orientation with the Microsoft Platform.* Prentice Hall Service-Oriented Computing Series from Thomas ERL. Prentice Hall, 2010. Available from: `http://books.google.com/books?id=gm3bPAAACAAJ`.

[27] Sony Computer Entertainment Europe. What is buzz! `http://www.buzzthegame.com/en-gb/What-is-Buzz/`, 2008. Accessed 26. June, 2011.

[28] The Eclipse Foundation. Eclipse announces eclipselink project to deliver jpa 2.0 reference implementation. `http://www.eclipse.org/org/press-release/20080317_Eclipselink.php`, March 2008. Accessed 26. June, 2011.

[29] The Eclipse Foundation. Eclipselink jpa. `http://www.eclipse.org/eclipselink/jpa.php`, 2011. Accessed 26. June, 2011.

[30] The Eclipse Foundation. Eclipselink/faq/jpa. `http://wiki.eclipse.org/EclipseLink/FAQ/JPA`, May 2011. Accessed 26. June, 2011.

[31] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[32] Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm. *Design Patterns CD: Elements of Reusable Object-Oriented Software, (CD-ROM).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[33] Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli, Kate Stout, and Inc. Sun Microsystem. Java message service. `http://jcp.org/aboutJava/communityprocess/final/jsr914/index.html`, April 2002. Accessed 26. June, 2011.

[34] i>clicker. web>clicker. `http://www.iclicker.com/dnn/Products/webclicker/tabid/156/Default.aspx`, 2011. Accessed 26. June, 2011.

[35] i>clicker. What is a clicker? `http://www.iclicker.com/dnn/Abouticlicker/WhatisaClicker/tabid/143/Default.aspx`, 2011. Accessed 26. June, 2011.

[36] Patrick W. Jordan, B. Thomas, Ian Lyall McCelland, and Bernard Weerdmeester. *Usability Evaluation in Industry.* CRC Press, June 1996.

[37] Matias Kivikangas, Inger Ekman, Guillaume Chanel, Simo Järvelä, Ben Cowley, Mikko Salminen, Pentti Henttonen, and Niklas Ravaja. Review on psychophysiological methods in game research. In Lankoski Petri, Thorhauge Anne Mette, Verhagen Harko, and Waern Annika, editors, *Proceedings of DiGRA Nordic 2010: Experiencing Games: Games, Play, and Players*, Stockholm, January 2010. University of Stockholm. Available from: `http://www.digra.org/dl/display_html?chid=10343.06308.pdf`.

[38] S. Kopf, N. Scheele, L. Winschel, and W. Effelsberg. Improving activity and motivation of students with innovative teaching and learning technologies. *Dept. of Computer Science IV, University of Mannheim, Germany*, 2005.

[39] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12:42–50, November 1995. Available from: `http://portal.acm.org/citation.cfm?id=624610.625529`, `doi:10.1109/52.469759`.

[40] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition).* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[41] Doug Lea. *Concurrent Programming in Java. Second Edition: Design Principles and Patterns.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1999.

[42] Dean Leffingwell. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Addison-Wesley Professional, 1st edition, 2011.

[43] David L. Levine, Christopher D. Gill, and Douglas C. Schmidt. *Object lifetime manager a complementary pattern for controlling object creation and destruction*, pages 495–534. Cambridge University Press, New York, NY, USA, 2001. Available from: `http://portal.acm.org/citation.cfm?id=566110.566135`.

[44] Leyland, B. How can computer games offer deep learning and still be fun? *In The conference Ascilite, December 2-4, 1996*, 1996. Available from: `http://www.ascilite.org.au/conferences/adelaide96/papers/14.html`.

[45] Avrio Ideas Ltd. Justvote. `http://www.just-vote.co.uk/`, 2008. Accessed 26. June, 2011.

[46] Avrio Ideas Ltd. Justvote specification sheet. `http://www.just-vote.co.uk/downloads/JustVote-Education.pdf`, 2008. Accessed 26. June, 2011.

[47] Silicon Graphics (managed by the non-profit technology consortium Khronos Group). Opengl overview. `http://www.opengl.org/about/overview/`, 2011. Accessed 26. June, 2011.

[48] Merrilea J Mayo. Video games: a route to large-scale stem education? *Science*, 323:79–82, 2009.

[49] David Mosberger and Tai Jin. httperf - a tool for measuring web server performance. In *In First Workshop on Internet Server Performance*, pages 59–67. ACM, 1998.

[50] Ole Kristian Mørch-Storstein and Terje Øfsdahl. Game enhanced lectures - an implementation and analysis of a lecture game. Master's thesis, NTNU, June 2007.

[51] Netcraft. November 2010 web server survey. `http://news.netcraft.com/archives/2010/11/05/november-2010-web-server-survey.html`, November 2010. Accessed 26. June, 2011.

[52] Diana Oblinger. Boomers, Gen-Xers, and Millennials: Understanding the "New Students". *EDUCASE Review*, 38(4):37–47, July 2003. Available from: `http://www.educause.edu/EDUCAUSE+Review/EDUCAUSEReviewMagazineVolume38/BoomersGenXersandMillennialsUn/157842`.

[53] British Academy of Film and Television Arts. Games nominations 2006. `http://www.bafta.org/awards/video-games/nominations/?year=2006`, 2006. Accessed 26. June, 2011.

[54] International Standardation Organisation, editor. *ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs), Part 11: Guidance on usability.* International Standardation Organisation, March 1998.

[55] Sun Microsystems (Now owned by Oracle Corporation). Design pattern: Service locator. `http://java.sun.com/blueprints/patterns/ServiceLocator.html`, 2002. Accessed 26. June, 2011.

[56] Lightweight Java Game Library Project. Lightweight java game library. `http://www.lwjgl.org/index.php`, February 2011. Accessed 26. June, 2011.

[57] T. Reenskaug. Models-views-controllers. *Technical note, Xerox PARC, December*, 1979. Available from: `http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf`.

[58] G. Reese. *Database programming with JDBC and Java*. Java series. O'Reilly, 2000. Available from: `http://books.google.com/books?id=oPbGiOlOZHEC`.

[59] Research and Development team from Wake Forest University. Classinhand. `http://classinhand.wfu.edu/`, June 2007. Accessed 26. June, 2011.

[60] Jeff Sauro and James R. Lewis. Correlations among prototypical usability metrics: evidence for the construct of usability. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 1609–1618, New York, NY, USA, 2009. ACM. Available from: `http://doi.acm.org/10.1145/1518701.1518947`, `doi:http://doi.acm.org/10.1145/1518701.1518947`.

[61] L. Schuh, D. E. Burdette, L. Schultz, and B. Silver. Learning clinical neurophysiology: gaming is better than lectures. *Journal of clinical neurophysiology : official publication of the American Electroencephalographic Society*, 25(3):167–169, June 2008.

[62] Screenlife. Screenlife games overview. `http://www.screenlifegames.com/scene-it/scene-it-console/overview.htm`, 2011. Accessed 26. June, 2011.

[63] Loki Software. Openal specification and reference. `http://connect.creativelabs.com/openal/Documentation/oalspecs-specs.pdf`, June 2000. Accessed 26. June, 2011.

[64] Daniel E. Stevenson and Andrew T. Phillips. Implementing object equivalence in java using the template method design pattern. In *Technical Symposium on Computer Science Education*, volume 35, pages 278–282, 2003.

[65] MySQL AB (A subsidiary of Oracle). About mysql. `http://www.mysql.com/about/`, 2010. Accessed 26. June, 2011.

[66] W.M.K. Trochim and J.P. Donnelly. *Research methods knowledge base*, chapter Experimental Design. Cengage Learning, 2006. Available from: `http://www.socialresearchmethods.net/kb/desexper.php`.

[67] E.R. Tufte. *Beautiful evidence*. Graphics Press, 2006. Available from: `http://books.google.com/books?id=v302PAAACAAJ`.

[68] A.N. Whitehead and B. Russell. *Principia Mathematica to *56*. Cambridge Mathematical Library. Cambridge University Press, 1997. Available from: `http://books.google.com/books?id=rdMgDpNSdLsC`.

Part VII

Appendices

Appendix A

Acronyms

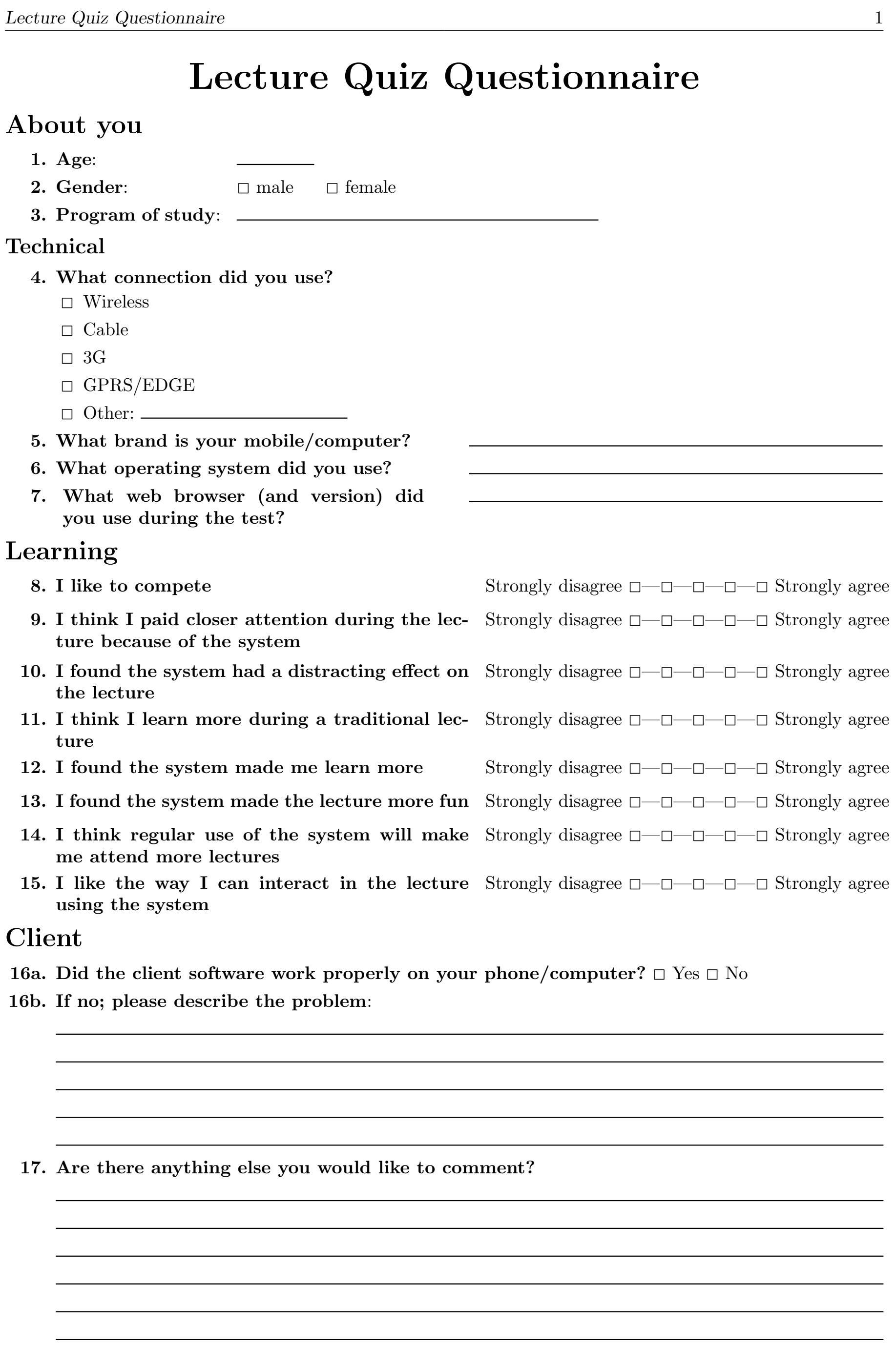
3G	3rd generation mobile telecommunications
AJAX	asynchronous JavaScript and XML
API	application programming interface
ASCII	American Standard Code for Information Interchange
BAFTA	British Academy of Film and Television Arts
CDDL	Common Development and Distribution License
CRC-32	32 bits cyclic redundancy check
CSS	Cascading Style Sheets
DBMS	Database Management System
DTD	Document Type Definition
EDGE	Enhanced Data rates for GSM Evolution
EL	Expression Language
EULA	end-user licensing agreement
GPL	General Public License
GPRS	general packet radio service
GPU	graphics processing unit
GUI	graphical user interface
GWT	Google Web Toolkit

HSCSD	high-speed circuit-switched data
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
IDE	integrated development environment
IIS	Internet Information Services
IP	Internet Protocol
ISM	industrial, scientific and medical
ISO	International Organization for Standardization
J2ME	Java 2 Platform, Micro Edition
JDBC	Java DataBase Connectivity
JME	Java Micro Edition
JMS	Java Message Service
JOGL	Java OpenGL
JPA	Java Persistance API
JSF	Java Server Faces
JSP	JavaServer Pages
JSR	Java Specification Request
JSTL	JavaServer Pages Standard Tag Library
JVM	Java Virtual Machine
Java EE	Java Enterprise Edition
Java ME	Java Micro Edition
Java SE	Java Standard Edition
JMS	Java Message Service
LAN	local area network
GNU LGPL	GNU Lesser General Public License
LWJGL	Lightweight Java Game Library
MIME	Multipurpose Internet Mail Extensions

MVC	model-view-controller
NTNU	Norwegian University of Science and Technology
OS	operating system
OpenAL	Open Audio Library
OpenGL	Open Graphics Library
PC	personal computer
QR code	Quick Response code
RDBMS	relational database management system
RITE	Residency In-service Training Exam
SHA-1	Secure Hash Algorithm 1
SQL	Structured Query Language
SUS	System Usability Scale
SOA	service-oriented architecture
TCP	Transmission Control Protocol
TTO	NTNU Technology Transfer AS
UI	user interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
USB	Universal Serial Bus
WAR	Web application Archive
WIL-MA	Wireless Interactive Learning - Mannheim
WLAN	wireless local area network
WSDL	Web Services Description Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Appendix B

Questionnaire



Lecture Quiz Questionnaire 1

Lecture Quiz Questionnaire

About you

1. **Age**: ________
2. **Gender**: ☐ male ☐ female
3. **Program of study**: ____________________

Technical

4. **What connection did you use?**
   - ☐ Wireless
   - ☐ Cable
   - ☐ 3G
   - ☐ GPRS/EDGE
   - ☐ Other: ____________
5. **What brand is your mobile/computer?** ____________________
6. **What operating system did you use?** ____________________
7. **What web browser (and version) did you use during the test?** ____________________

Learning

8. **I like to compete**	Strongly disagree ☐—☐—☐—☐—☐ Strongly agree
9. **I think I paid closer attention during the lecture because of the system**	Strongly disagree ☐—☐—☐—☐—☐ Strongly agree
10. **I found the system had a distracting effect on the lecture**	Strongly disagree ☐—☐—☐—☐—☐ Strongly agree
11. **I think I learn more during a traditional lecture**	Strongly disagree ☐—☐—☐—☐—☐ Strongly agree
12. **I found the system made me learn more**	Strongly disagree ☐—☐—☐—☐—☐ Strongly agree
13. **I found the system made the lecture more fun**	Strongly disagree ☐—☐—☐—☐—☐ Strongly agree
14. **I think regular use of the system will make me attend more lectures**	Strongly disagree ☐—☐—☐—☐—☐ Strongly agree
15. **I like the way I can interact in the lecture using the system**	Strongly disagree ☐—☐—☐—☐—☐ Strongly agree

Client

16a. **Did the client software work properly on your phone/computer?** ☐ Yes ☐ No

16b. **If no; please describe the problem**:

____
____
____
____
____

17. **Are there anything else you would like to comment?**

____
____
____
____
____
____

System Usability Scale

© Digital Equipment Corporation, 1986.

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Appendix C

Deployment Guide

In this appendix, we go though the steps on how to configure and deploy the Quiz Server; including a database back-end.

The Quiz Server works as a centralized data storage server, and needs to run continuously on some server connected to the Internet. In this section we will explain how the Quiz Server can be deployed with a database back-end in order to get it up and running. We assume the user has previous knowledge about both Database Management System (DBMS) and Web application servers. Knowledge about MySQL and GlassFish would be preferred.

C.1 Requirements

The Quiz Server is written with Java EE which is multi-platform. Because of this, several operating systems is supported, like standard Linux and Microsoft Windows servers. The server is not required to be physical, thus a virtual machine would suffice.

The deployment of the Quiz Server is done on a Java application server. We have used GlassFish 3.0 and 3.1 through our development, but similar Java application servers like Apache Tomcat is supported. This guide will only give guidance on how to use the Quiz Server deployed on a GlassFish server. We have not tested the Quiz Server on earlier GlassFish versions than 3.0 and because of this we recommend version 3.0 and newer.

Java EE is required to deploy the Quiz Server. You should use the newest version available, and versions older than Java EE 6 is not supported.

The Quiz Server is dependant on a data storage, most preferably a DBMS. With the use of EclipseLink and JPA the database type is abstracted away. This means that you can use any relational database that is compliant with SQL and has a compliant JDBC driver. In this guide we use MySQL as our reference implementation. We have used MySQL version 5 during development. Older versions are supported, but MySQL version 5.0 or higher is recommended

C.2 Database Setup

The database server does not need to be on the same server as the GlassFish server. The workload can be distributed among several physical servers, or logically on two different virtual machines. Despite this, we recommend installing both on the same server.

As described in section C.1, we have chosen MySQL as out back-end database server. Other database servers can be used, but we will use MySQL as an example in this guide. We suggest creating a separate user, that the web service will use to connect to the database, but this is not required.

The creation of the database structure will be handled by the persistence itself and because of this there is no need to import any SQL manually. The persistence will also create an administrative user named "admin" with the password "12345". This can be changed later, after deploying the Quiz Server. In this step it is important to remember the specific setting for the MySQL user, the port and the location of the MySQL server. This will be needed for the next step - deployment.

C.3 Quiz Server Deployment

The deployment of the Quiz Server is done on a Java application server. During the development of the Lecture Quiz game, we used GlassFish 3.0 and 3.1. Other application servers should be usable, but we do not cover that in this guide.

The deployment consists of two steps, configuration and deployment of the WAR file.

C.3.1 Step 1

Database settings alter from server to server, and because of this they have to be altered before deploying the Quiz Server. Open the WAR file in a file archiver/-compressor application, e.g. 7-Zip, to access the persistence.xml file. When you have done this, open the *WEB-INF/classes/META-INF* archives and edit the persistence.xml file. You have to change the driver property, if you want to use another JDBC driver than MySQL. Supply the database server URL and port in the URL property, e.g. "jdbc:mysql://localhost:3306/lecturequiz". In the user and password properties you have to supply the credentials belonging to the MySQL user you made, or the root credentials. The generation property is required to be "create-tables" the first time Quiz Server is deployed. In consecutive deployments, it can be turned off. The other properties are for advanced users and should be left as they are.

C.3.2 Step 2

Now that your WAR file is configured correctly, it is time to deploy it on the GlassFish server. We recommend using the auto-deploy functionality which is supplied by GlassFish. Goto the GlassFish install directory; then goto *GlassFish/domain/your_domain/autodeploy* directory. Copy the WAR file into this directory. When this is done, Quiz Server should be running after a little while. For more information about configuring and deploying WAR files on a GlassFish server, see the GlassFish community web page[1].

[1] http://glassfish.java.net/docs/project.html

Appendix D

Development Guide

In this appendix we present our development guide. We explain how new developers can create game modes and player clients.

We have used NetBeans integrated development environment (IDE) 7.0 as our development environment to develop Lecture Quiz 3.0. The latest version of NetBeans can be found at the NetBeans web page[1]. All libraries and dependencies have been included with the source code. This means, to start developing for Lecture Quiz, all you need is to open the projects in NetBeans.

D.1 Creating a Game Mode

Game Modes are located in the Presentation Client project, inside the lecturequiz.presentationclient.logic.gamemodes package. There are several game modes in this package that can act as samples when creating a game mode. Every game mode implements the IGameMode interface, as seen in Figure 7.17.

```
public interface IGameMode {

    public String getName();
    public String getDescription();
    public ITexture getIcon();

    public int getMinQuestions(GameType gameType);
    public int getMaxQuestions(GameType gameType);

    public void start(Game game,
    GameModeQuestionChain gameModeQuestionChain);
    public void end(Game game,
```

[1]www.netbeans.org

```
    GameModeQuestionChain gameModeQuestionChain);

    public void abortGame();
}
```

Listing D.1: IGameMode interface

The interface contains methods that provide general information about a game mode. These methods are getName(), getDescription(), getIcon(), getMinQuestions() and getMaxQuestions(). The getName() method can return a short text containing the name of the game mode being created. We recommend a descriptive text is returned by the getDescription() method, as this will be all the explanation the users will get. The getIcon() method will allow the creator to add a small graphic to the interface, representing the game mode. If no icon texture is returned, the Presentation Client will use a generic image for the game mode. The getMinQuestions() and getMaxQuestions() methods allow the creator to specify how many questions the game mode is designed for based on the game type. For example in the Three Strikes game mode, the users play until one team has three strikes. This will require a maximum of 9 questions when playing four teams, before one team has three strikes. If playing with only two teams, the maximum will change to 5, as this is the maximum number of questions that it will be able to run.

When a lecturer starts a game with the new game mode, the Presentation Client will start by calling the start() method. This will give full control to the game mode, and it will be responsible for starting question rounds, and let the next game mode take over when its done. To get information about the current game, the game mode can use the Game object. This object contains information about the chosen quiz, the team setup, game statistics and game modes. If the game mode wants to stop a game it can call the stop() method on the Game object, this will make the Game object cleanup and return to the main screen. We recommend that the abortGame() method, on the IGameMode interface, should include this logic to allow the lecturer to stop games, but can of course run some other logic first.

To retrieve questions the game mode can use the GameModeQuestionChain object. This object contains the game modes assigned questions in a stack. The number of questions have been divided according to the getMinQuestions() and getMaxQuestions() methods in the IGameMode interface. If the GameModeQuestionChain does not contain any more questions, it will call the end() method on the game mode. The end() method should contain code to cleanup object created by the game mode during game play. The Game object will call start on the next game mode in the GameModeQuestionChain after it has called end() on the game mode. This will start the next game mode in the game or display the game summary if there are no more game modes.

We have constructed an abstract GameModeBase class. This class can be extended by the new game mode to give easy access to common tasks.

```
public GameModeBase(String modeName,
    String modeDescription, String iconFilename) {
  this.modeName = modeName;
  this.modeDescription = modeDescription;
  this.iconFilename = iconFilename;
}
```

Listing D.2: GameModeBase constructor

The GameModeBase constructor takes three arguments, as shown above. This will require the game mode name, the game mode description and the filename of the game mode icon. The GameModeBase will then handle the getName(), getDescription() and getIcon() methods. This saves unnecessary repetition of code, like loading of the icon texture. The GameModeBase contains code for showing information about a game mode. The information will be presented on screen based on the general information received from the IGameMode interface. To start a question round, the game mode can call the startRound() method, as shown in Figure 7.17. This will run a normal question round, with the next question in the GameModeQuestionChain. To implement logic between rounds, the game mode can override the roundEnd() method, shown in Figure 7.17. To specify player or team score after each round, the game mode can override the changeTeamScore() and changePlayerScore() methods. This will allow the game mode to change score, based on player's/team's answer. Some game modes will restrict which players are able to answer during a question round. This can be decided by overriding the canPlayerAnswer() method.

Game modes are chosen from a game mode list when running the Presentation Client. To make the new game mode appear in this list, the developer needs to add an instance of the game mode class to the GameModeProvider. Game modes are contained in a Java EnumMap inside the GameModeProvider. The game mode has to be added for each game type it supports. When this is done, the new game mode is ready for use.

D.2 Creating a new Player Client

The current implementation of the player client is based on the IPlayerClient interface. This interface contains the methods used to communicate with the players. When creating a new Player Client the developer will need to implement this class.

```
public interface IPlayerClient {
    public List<Player> getPlayers();
    public void setState(int playerState);

    public String getAddress(boolean resolveName);

    public void addPlayerAnswerActionListener(
    ActionListener actionListener);
    public void removePlayerAnswerActionListener(
    ActionListener actionListener);

    public void addPlayerJoinActionListener(
    ActionListener actionListener);
    public void removePlayerJoinActionListener(
    ActionListener actionListener);

    public void addPlayerLeaveActionListener(
    ActionListener actionListener);
    public void removePlayerLeaveActionListener(
    ActionListener actionListener);
}
```

Listing D.3: IPlayerClient interface

The Player Client needs to manage players and allow listing through the getPlayers() method. The listed players need to be a list of Player models. A Player model includes information like name, score, state and team name. Player states are defined in the model, and represent the players' current state in the game.

```
public static final int STATE_LOGIN = 0;
public static final int STATE_WAITING_FOR_GAME = 1;
public static final int STATE_WAITING_FOR_ROUND = 2;
public static final int STATE_SELECT_TEAM = 3;
public static final int STATE_ANSWER_QUESTION = 4;
public static final int STATE_QUESTION_ANWERED = 5;
public static final int STATE_VOTE_SCREEN = 6;
public static final int STATE_NOT_PARTICIPATING = 7;
public static final int STATE_GAME_SUMMARY = 8;
```

Listing D.4: Player state constants

New players will start with the login state. The Presentation Client will change player states according to the game play. For example when a new question is presented, the player states are changed to STATE_ANSWER_QUESTION to allow players to answer. The setState() method on the IPlayerClient should be used to change every players state.

The players are not playing directly on the Presentation Client. This means that the new player client class needs to specify an address with the getAddress() method. The address for a game will be presented on the Presentation Client, so that the players can access the game.

To inform the Presentation Client of player actions, we use Java's ActionListeners. These are registered to the player client, with add and remove methods. The different ActionListener types are: player answer, player join and player leave. Every call to an ActionListener's performAction method(), should have the affected Player model as source object. The Presentation Client will add and remove listeners during game play, this means that it is important that both work correctly.

Creating a new Player Client will require some work, depending on the type of client being created. To make the Presentation Client use the new client, the new player client class needs to be created in the LogicManager. After creating an instance of the player client, it has to be registered on the service locator through the IPlayerClient interface. This will make the rest of the application use the new player client. It is also preferable that the player client is unregistered in the dispose() method of the LogicManager. This is to ensure that no new calls to the client will be performed after the application has started its closing procedures.

Appendix E

User Guide

In this appendix we present our user guides. This will give an thorough explanation of how to use the Presentation Client, the Player Client, and the Quiz Server web page.

E.1 Presentation Client

In this section we present the Presentation Client user guide. The Presentation Client is the part of Lecture Quiz that is responsible for running quiz games.

At application start, it displays a splash screen, as shown in Figure E.1. If you see the splash screen, it means that the Presentation Client has started successfully.

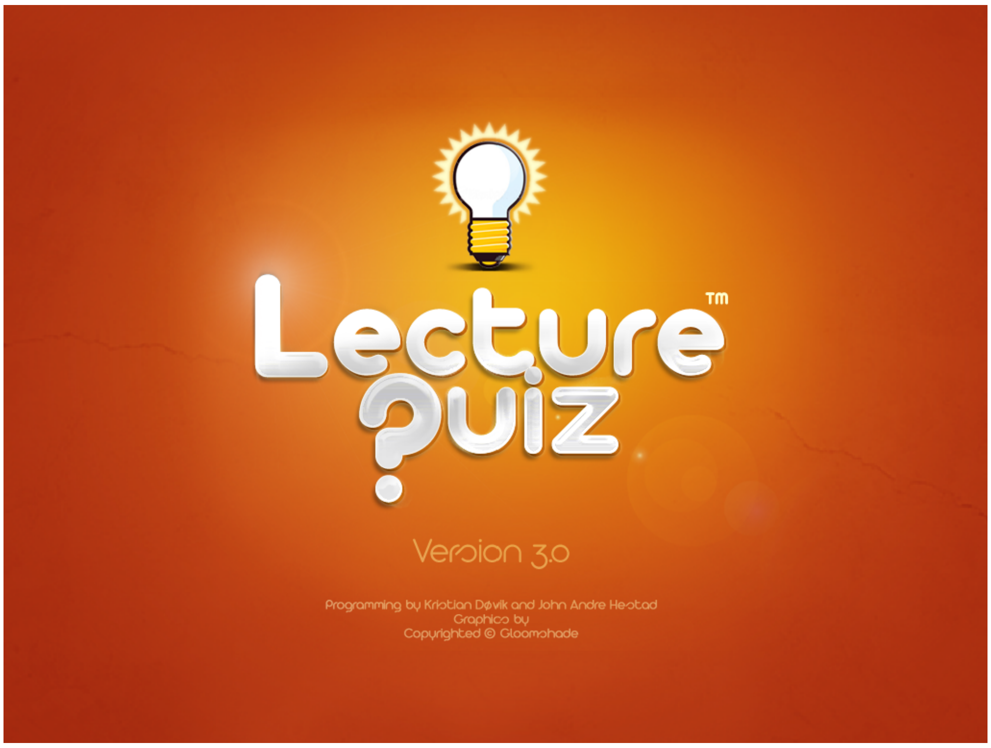


Figure E.1: Presentation Client splash screen.

E.1.1 Game Type Selection

The first screen you can interact with is the game type selection screen, as shown in Figure E.2. This screen allows you to configure what kind of game you want to play. The different types of games are two teams, three teams, four teams and free for all. To allow players to compete as teams, you can select either one of the team game types. If you want all the players to compete against each other, you select the free for all type.

To select a game type, you click on the graphic representing the type of game you want. The chosen type is indicated with an animated yellow arrow. At this stage, you can announce that the participants can join the game. The network address of the Player Client can be found in the lower left corner of the screen. To proceed with the game setup, you can click the "Next" button at the lower right corner of the screen.

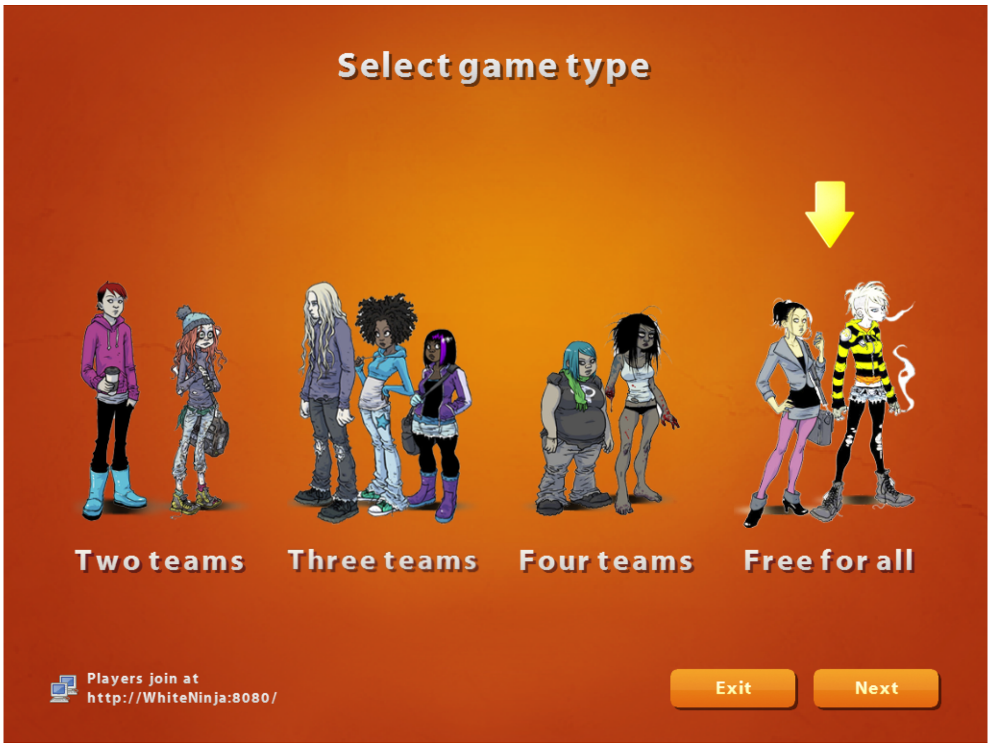


Figure E.2: Presentation Client game type select screen.

E.1.2 Exit the Presentation Client

If you want to exit the Presentation Client, you can click on the "Exit" button at the lower right corner of the select game type screen, as show in Figure E.2. This brings you to the exit screen, which contains two buttons; "Yes" and "No". By clicking the "Yes" button, you confirm that you want to close the application. If you click the "No" button, the Presentation Client takes you back to the select game type screen.

E.1.3 Quiz Select Screen

After selecting a game type, the Presentation Client takes you to the quiz select screen, as shown in Figure E.3. This screen allows you to select a quiz from the quiz list, located on the left side of the screen. By selecting a quiz in this list, you can get more detailed information about the quiz. The information is displayed on the right hand of the screen.

If you want to go back to change the game type, you can click the "Back" button. When you have found the quiz you want to use for your game, click the "Next" button to continue.

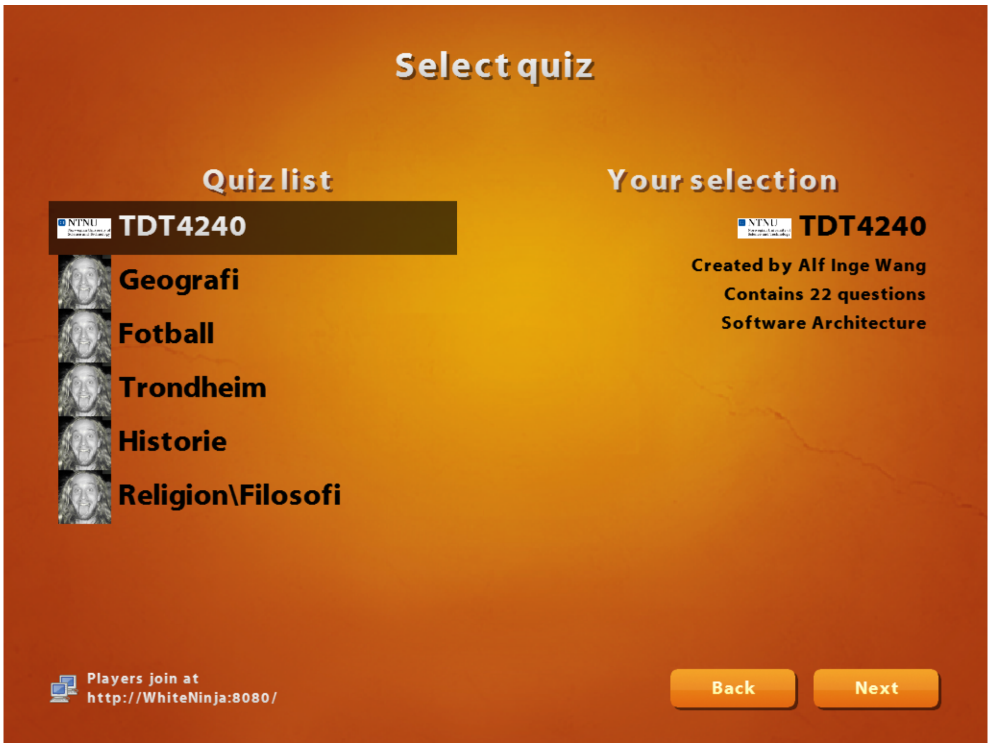


Figure E.3: Presentation Client quiz select screen.

E.1.4 Game Mode Composition

After selecting a quiz, the Presentation Client takes you to the game mode setup screen, as shown in Figure E.4. This screen allows you to use one or more game modes, which the game will be based on. The game modes are presented as a list, on the left side of the screen. The available game modes depends on the type of game specified in the select game type screen. The composition of game modes are displayed on the right side of the screen.

To use a game mode, double click on it. This adds one instance of the game mode to the mode setup list. To remove game modes from the mode setup list, double click the game mode you want to remove. A number is displayed to the right of each game mode in the mode setup list. This number represents the number of questions that is run by that game mode. If you want to change the quiz or game type, you can click the “Back” button. When you are satisfied with the composition of game modes, you can click “Next” button to continue.

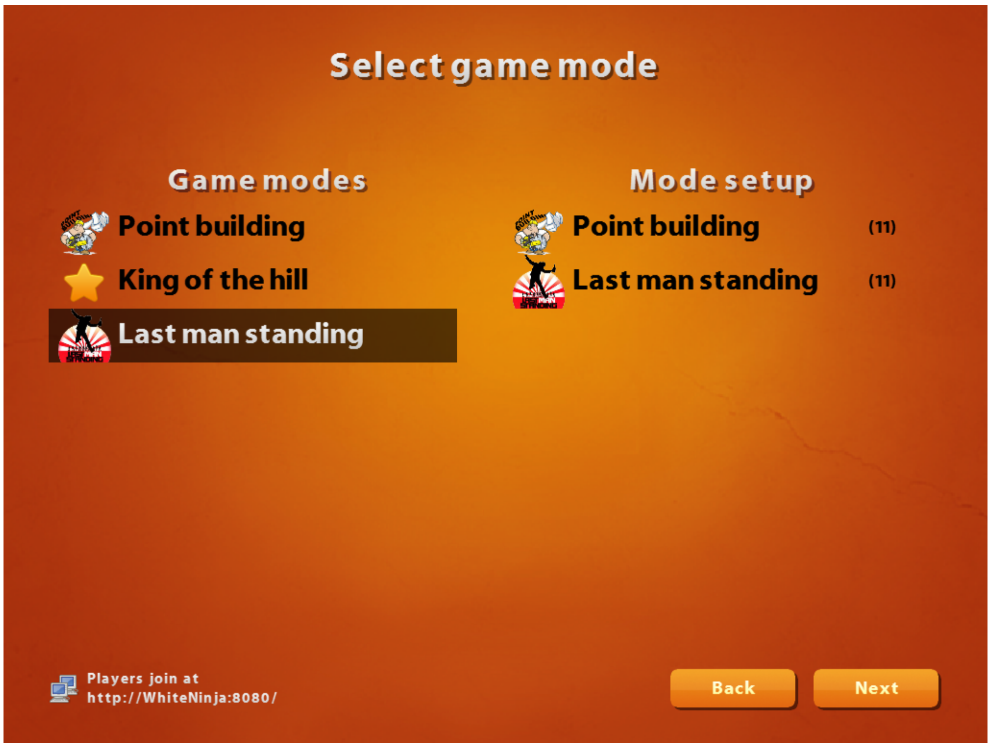


Figure E.4: Presentation Client game mode setup screen.

E.1.5 Team Selection

After creating the mode setup, the Presentation Client takes you either to the team select screen, or the join game screen. The choice depends on the type of game you have selected earlier. If you have selected the "Free for all" game type, the join game screen is displayed, as shown in Figure E.6. The join game screen allows you to see which players have joined the game. If you selected a team game type, the Presentation Client takes you to the team select screen, as shown in Figure E.5. The team selection screen displays player lists for each team, the number of lists depend on the number of teams you have selected in the game type select screen.

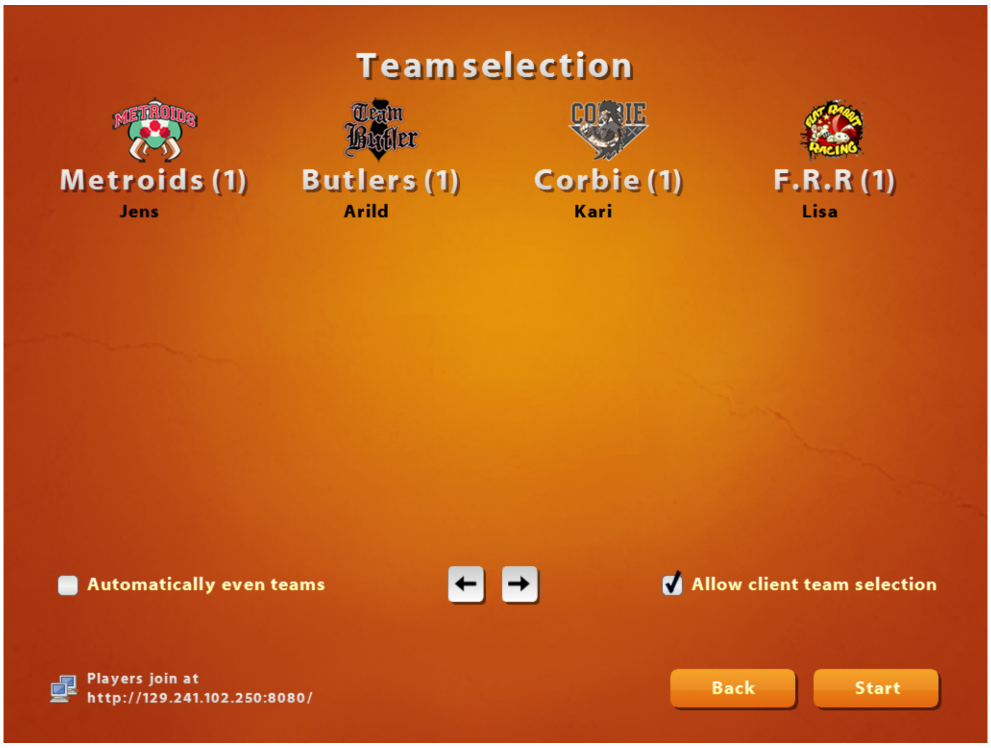
Above each list is an icon of the team mascot. This mascot is used to associate a player with a team.

Below the lists to the right you can find a checkbox called "Allow client team selection". Checking this box will allow the players to choose what team they want to play with.

Below the lists to the left of the screen you can find a checkbox named "Automatically even teams". If you check this checkbox, players are divided evenly on each team. While this checkbox is checked, players are not allowed to change to a team with more players than the team they are currently on.

There are two buttons below the team lists, one with an arrow pointing left and one with an arrow pointing right. These are used to move players to other teams. To manually move a player from one team to another, select the player in the team list. Then click the button representing the direction of the team you want to move the player to.

When you are ready to start the game, click the “Next” button. This starts the game with the configuration you have chosen.



Players join at
http://129.241.102.250:8080/

Figure E.5: Presentation Client team select screen.

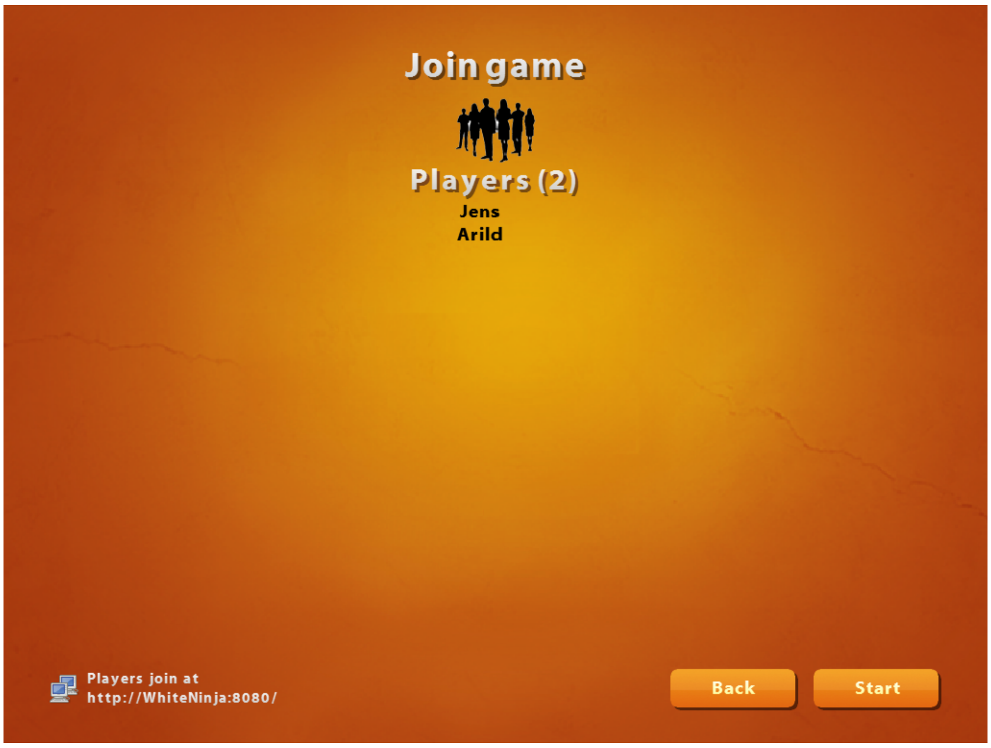


Figure E.6: Presentation Client join game screen.

E.1.6 Running a Quiz

During game play, the Presentation Client uses different game modes according to your setup. When a new game mode is used, a game mode information screen will be displayed, as shown in Figure E.7. This screen displays information about the current game mode, allowing players to read the rules of the game mode. To continue with the described game mode, you can click the start button.

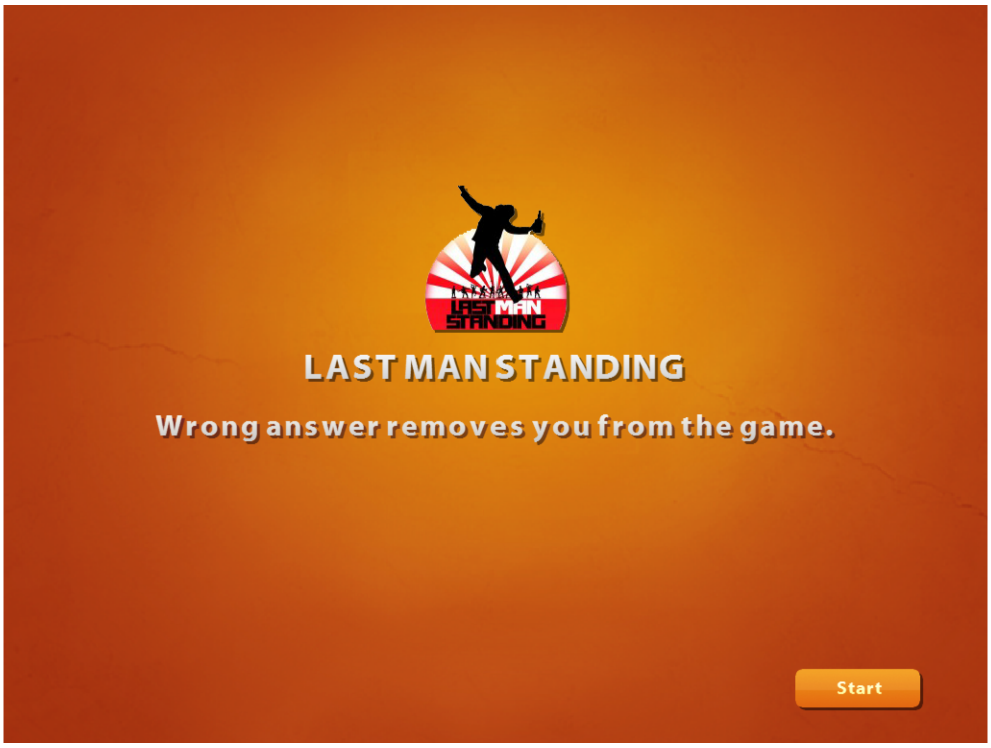
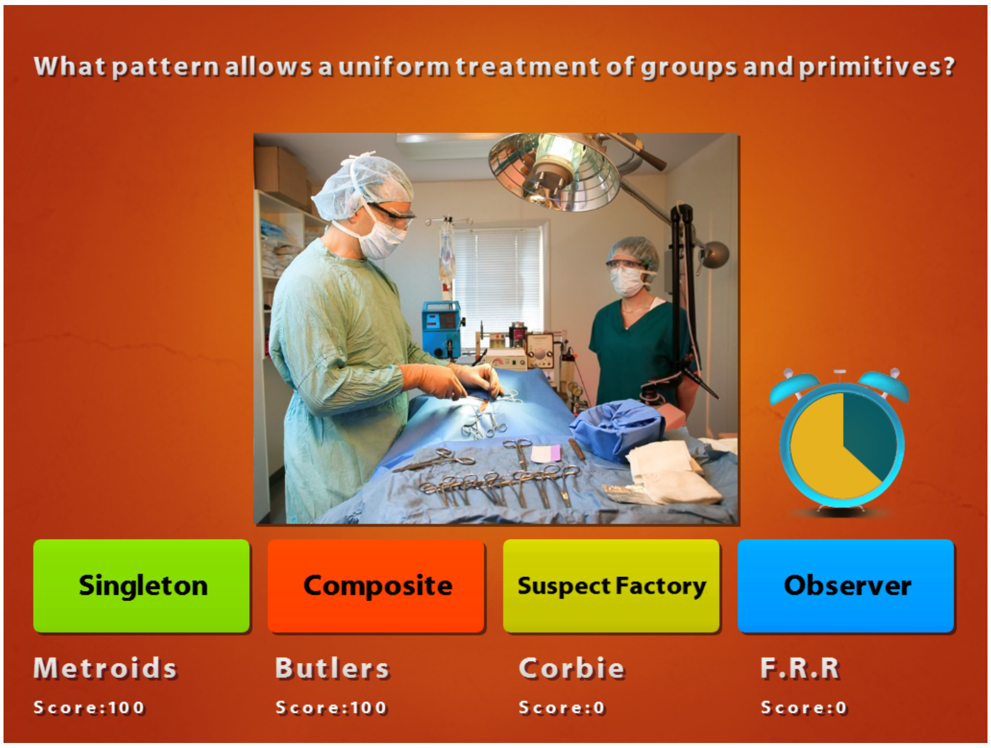


Figure E.7: Presentation Client game mode information screen.

Quizzes contains questions, and the Presentation Client presents these with the question screen, as shown in Figure E.8. The question text is presented at the top of the screen. If the question has an assigned image, this image will be displayed in the center of the screen.

Each question in Lecture Quiz has four alternatives. These are presented as buttons in the question screen. When a question starts, the players have a limited time period to answer the question. To keep track of the time they have left, the question screen contains a ticking clock. When the timer runs out, the players can not answer any longer, and the correct answer will be revealed.

To reveal the correct answer, the question screen will highlight the correct alternative. When playing a team game type, the team scores will be presented at the bottom of the screen. The same applies when playing free for all, but only the top four players will be displayed. After a question has timed out, the teams/players answer is displayed at the bottom of the screen. This is displayed as a little colored rectangle to the right of teams/players score. This rectangle has the same color as the given answer. After everyone has answered, or the time is up, continue by clicking anywhere on the screen with your mouse pointer.



What pattern allows a uniform treatment of groups and primitives?

Singleton	Composite	Suspect Factory	Observer
Metroids	Butlers	Corbie	F.R.R
Score:100	Score:100	Score:0	Score:0

Figure E.8: Presentation Client question screen.

Some game modes use special screens, like the point bet game mode, as shown in Figure E.9. This game mode is only available in team games, and will allow players to vote for their team's decision. In this case, they are voting on how much they want to bet on the question presented at the top of the screen. The players make their bets before the alternatives are shown. When everyone has voted, or the start button is pressed, the game will continue.

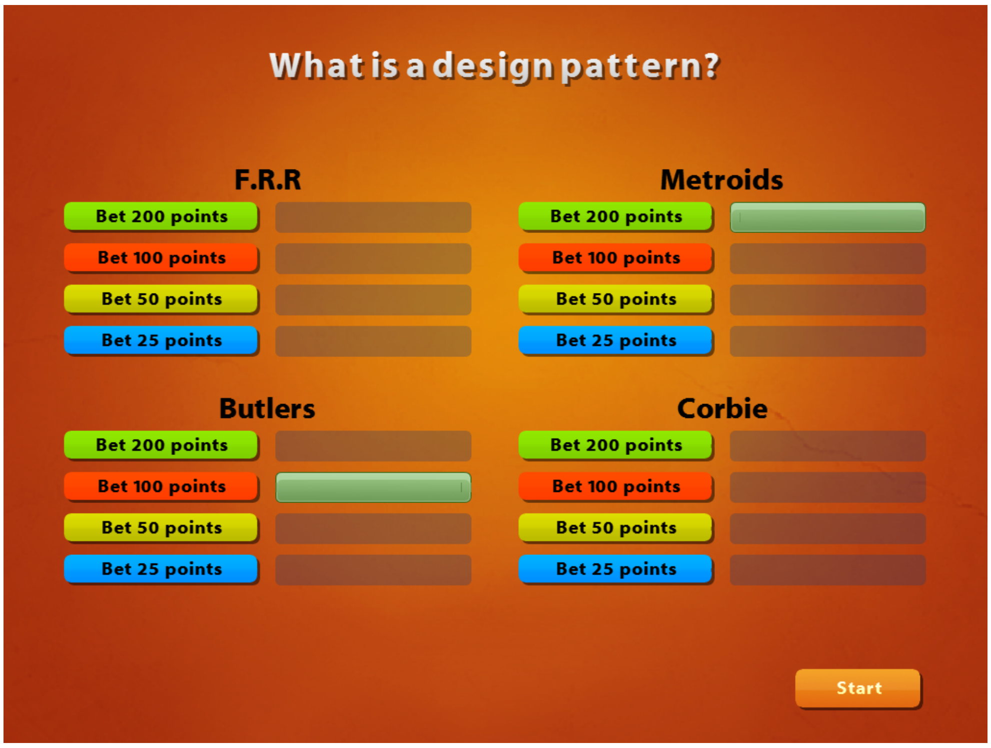


Figure E.9: Presentation Client point bet screen.

E.1.7 Game Summary

After all the questions have been answered, the Presentation Client will display a game summary, as shown in Figure E.10. If the selected game type is a team game, the summary screen will display team ranking on the left side of the screen.

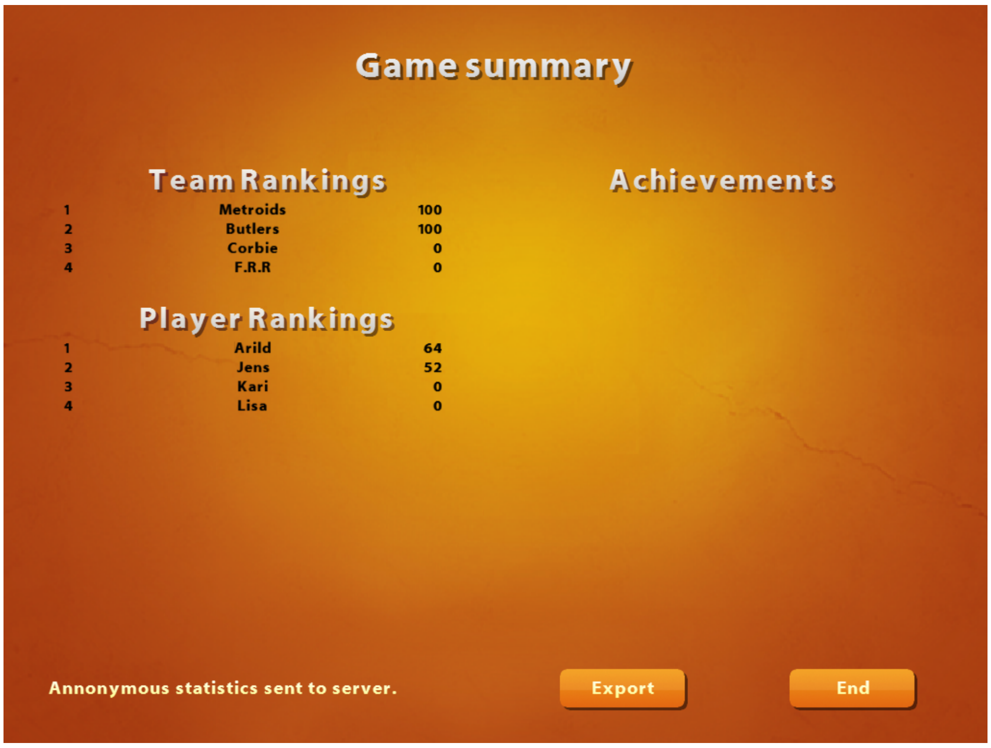
Player scores are different from team scores, but is present in both free for all and team games. This allows players to compete as part of the team and against each other, at the same time.

Player rankings are displayed on the left side of the screen. Both ranking lists can be used to see who won the game.

The right side of the screen is set aside to display achievements, like which player was the fastest to answer. This feature is not complete yet, but will be available in upcoming releases.

If the Presentation Client is connected to the Quiz Server, it will automatically send user statistics to the server. The same statistics can also be exported to a file, if you click the export button.

To end the game, you can click the end game button. This will return you to the game type select screen, and allow you to setup another game or exit.


Game summary

Team Rankings

1	Metroids	100
2	Butlers	100
3	Corbie	0
4	F.R.R	0

Player Rankings

1	Arild	64
2	Jens	52
3	Kari	0
4	Lisa	0

Achievements

Annonymous statistics sent to server.

Figure E.10: Presentation Client summery screen.

E.2 Player Client

In this section we present the Player Client user guide. The Player Client is the part of Lecture Quiz that is responsible for receiving player responses.

E.2.1 Login

When the Presentation Client is running, players can go to the Player Client web page. The first screen you see is the login screen, as shown in Figure E.11. This screen allows you to specify the username you want in the Lecture Quiz game. After typing in the username, you can click the login button to join the game.

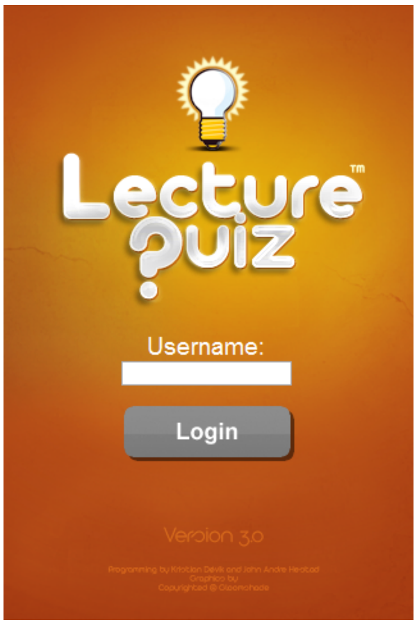
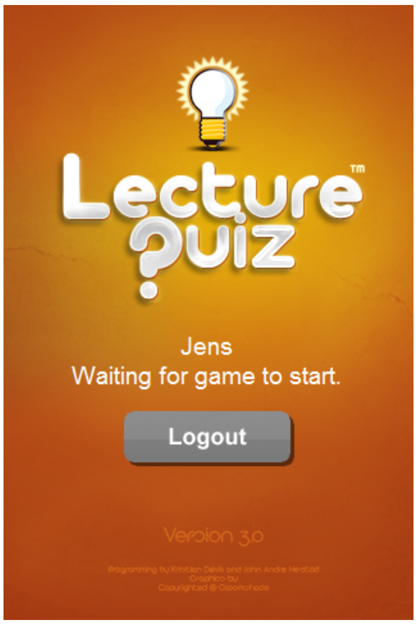


Figure E.11: Player Client login screen.

E.2.2 Waiting for Game

After logging into the game, you will see the "Waiting for game screen", as shown in Figure E.12. This screen will be displayed until the game starts, or the lecturer specifies that the clients are allowed to select a team. The wait for game screen allows you to logout, you can do this if you want to change username. It is not possible to change the username after the game starts.



Jens
Waiting for game to start.

Figure E.12: Player Client waiting for game to start screen.

E.2.3 Team Selection

If the lecturer specifies that players are allowed to change teams, the “team select screen” will be displayed, as shown in Figure E.13. In this screen, you can change team by clicking on the mascot, which represents the team you want to play with. The mascots is presented on the Presentation Client, above each team list, as seen in Figure E.5.

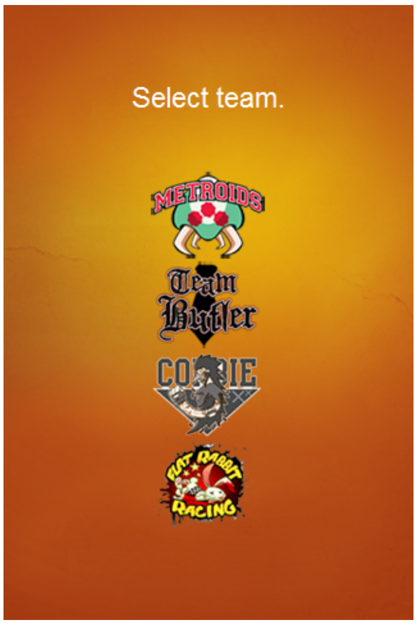


Figure E.13: Player Client team select screen.

E.2.4 Waiting for Round

When the lecturer starts the game, you will see the "waiting for round screen", as shown in Figure E.14. This screen will also be shown between each question round. The screen contains your team's name, your username and your current score. If you answered a question during the last question round, it will display a button representing the alternative you chose.

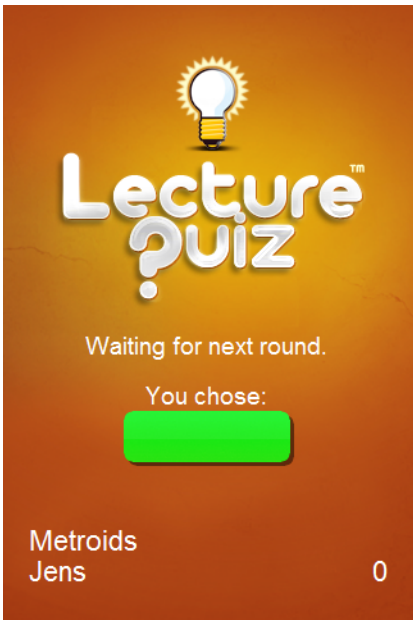


Figure E.14: Player Client waiting for next round screen.

E.2.5 Answering Questions

The Player Client will display the "answer question screen" during question rounds, as shown in Figure E.15. This screen allows you to answer the presented question, by pressing one of the alternative buttons. These buttons are of the same color as the alternative buttons presented on the Presentation Client, as shown in Figure E.8. The screen will display your team's name, your username and your current score. This allows you to keep track of your progress.

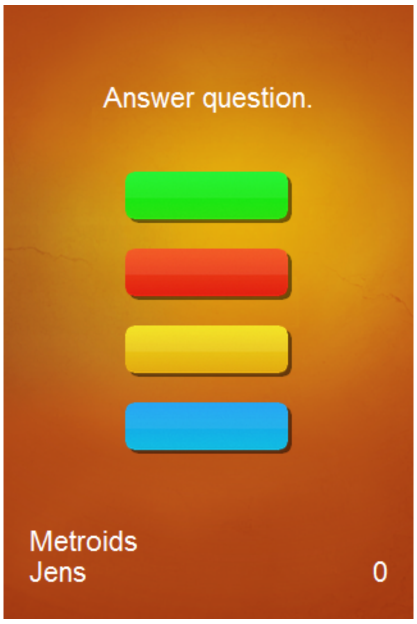


Figure E.15: Player Client answer question screen.

E.2.6 Question Answered

After answering a question, the Player Client displays the question answered screen, as shown in Figure E.16. This screen informs that you have to wait for other players, as the question round is not done yet. The screen will display information about your team's name, username and score.

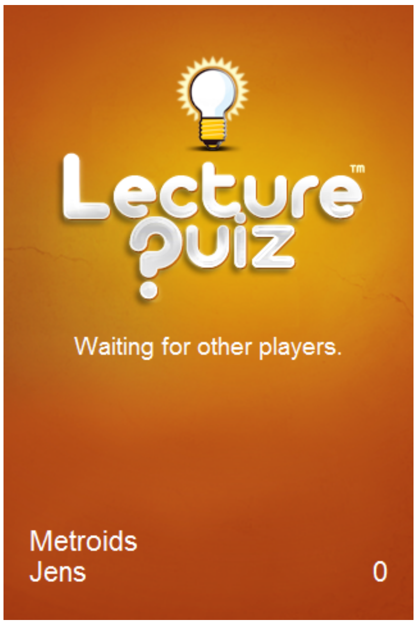


Figure E.16: Player Client question answered screen.

E.2.7 Not Participating

If the game mode logic prevents you from answering a question, the Presentation Client displays the "not participating screen", as shown in Figure E.17. This screen is currently only available when running the "last man standing" game mode. I.e., the "last man standing" game mode removes a player's opportunity to answer any more questions if they answer one question wrong. This effect is reset when the game enters the next game mode.

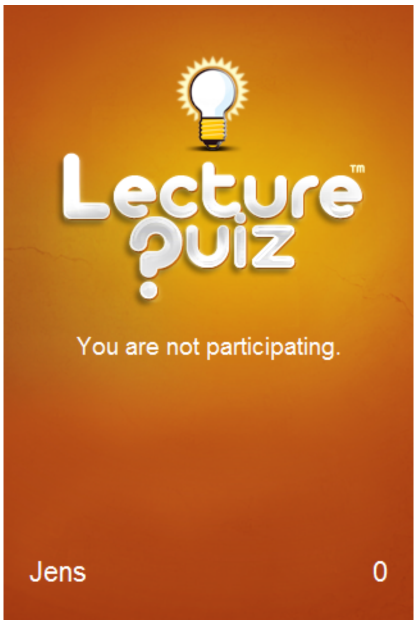
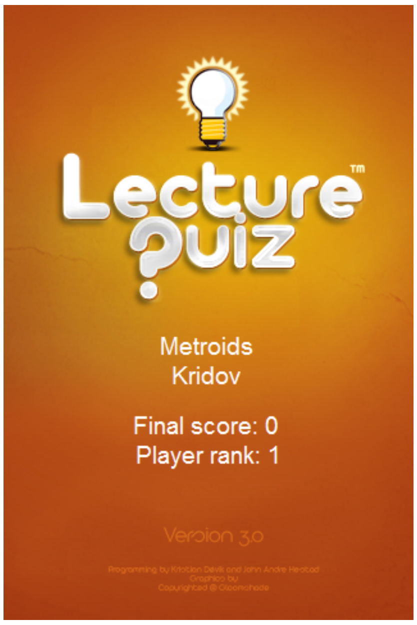


Figure E.17: Player Client not participating screen.

E.2.8 Game Summary

When a Lecture Quiz game is over, the Player Client displays the game summary screen, as shown in Figure E.18. The game summary screen contains information about how well you did in the quiz game. It will display your team's name, your username, your final score and your rank. Ranks let you know how high you scored, compared to the rest of the class. This means that if you are at rank one, you where the best player in the quiz.



Metroids
Kridov

Final score: 0
Player rank: 1

Figure E.18: Player Client summery screen.

E.3 Quiz Server

In this section we present the Quiz Server web page user guide. The Quiz Server is the part of Lecture Quiz where users can manage quizzes and statistics. It can be accessed from a web browser, so that lecturers and other privileged users can create and edit their quizzes.

To explain the design of the Quiz Server in a presentable way, we have chosen to split this guide into logical parts where each part resemble an actual user operation. We will guide you through the login process, creation of quizzes and questions, and showing of statistics after a successful quiz run.

E.3.1 Login

When the Quiz Server is running, users can connect to it from a graphical web browser. The first web page presented is the login screen, as shown in Figure E.19. To login, you have to supply a valid username and password; which has already been obtained from an administrator. Click login, and you will enter the main screen of the Quiz Server web page.

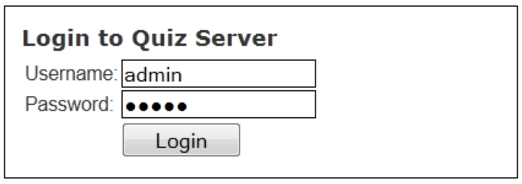
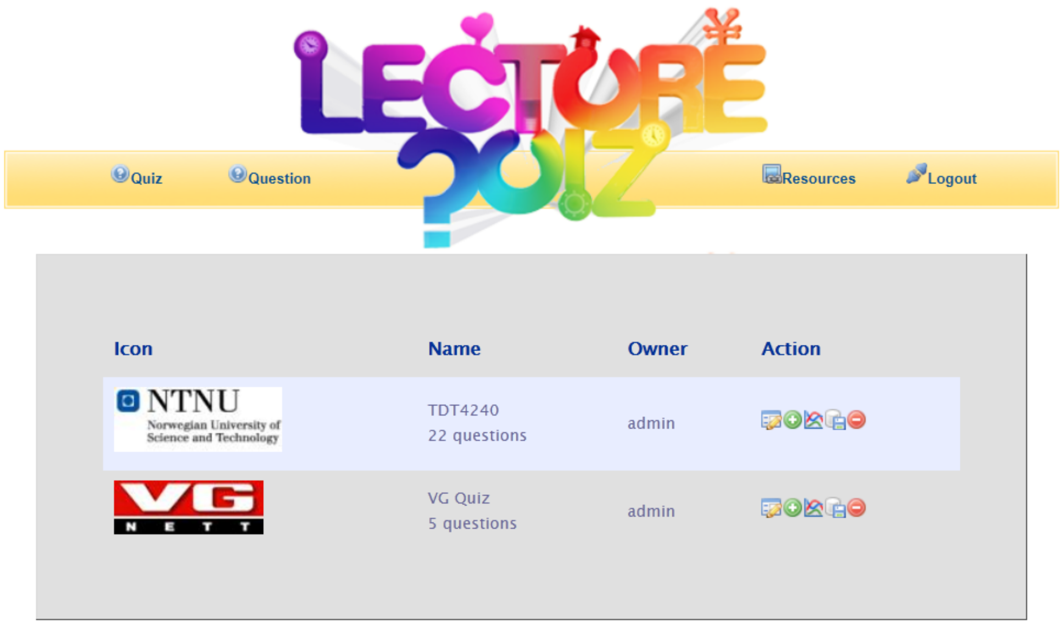


Figure E.19: Quiz Server login screen.

E.3.2 Navigation

After logging into the Quiz Server, you will be able to navigate the Quiz Server web page, as shown in Figure E.20. The top horizontal bar represents a navigational menu which interacts with the user. This navigational menu is called a header and will always be represented in the top of the web page. This is for easy access to all the functions of the Quiz Server web page despite your current web page location. The horizontal bar is divided into three categories; Quiz, Question, Resources; and a log out button. These categories are explained in the following sections. If you want to end the session or for some other reason quiz, please click the logout button to the far right on the navigational bar. This will remove your current session and you will be taken to the login web page.



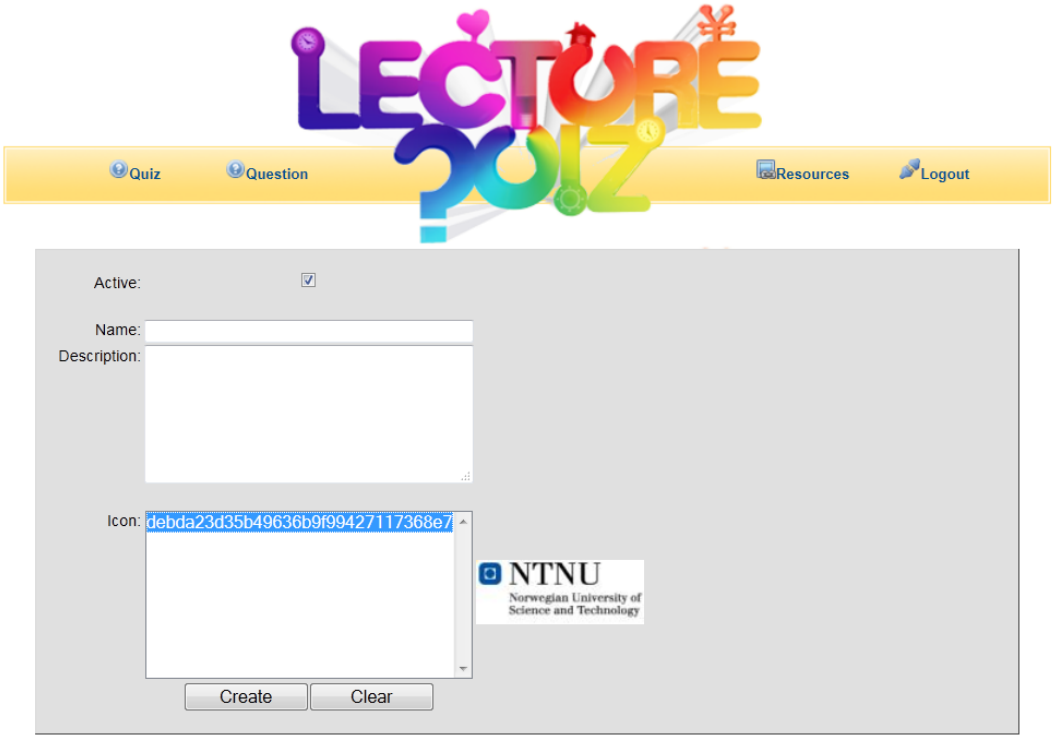
Icon	Name	Owner	Action
NTNU Norwegian University of Science and Technology	TDT4240 22 questions	admin	
VG NETT	VG Quiz 5 questions	admin	

Figure E.20: Quiz Server quiz list screen.

E.3.3 Quiz Management

The Quiz category contains a list of: “New Quiz”, “My Quizzes”, “Upload Quiz Package”, and “Upload Statistics Package”.

To create a new quiz, click the “New Quiz” button. You will now be taken to the create new quiz web page, as shown in Figure E.21. This web page lets you create a new quiz, after you fill in the necessary data. You have to specify a quiz name in the name field and for then to select an icon. If no icon is available, you have to upload one first, which is described in Section E.3.5. The description field is optional and is meant for additional textual information about the quiz. If you do not the quiz to be active (reachable from the Presentation Client), please un-check this checkbox.



Active: ✓

Name:

Description:

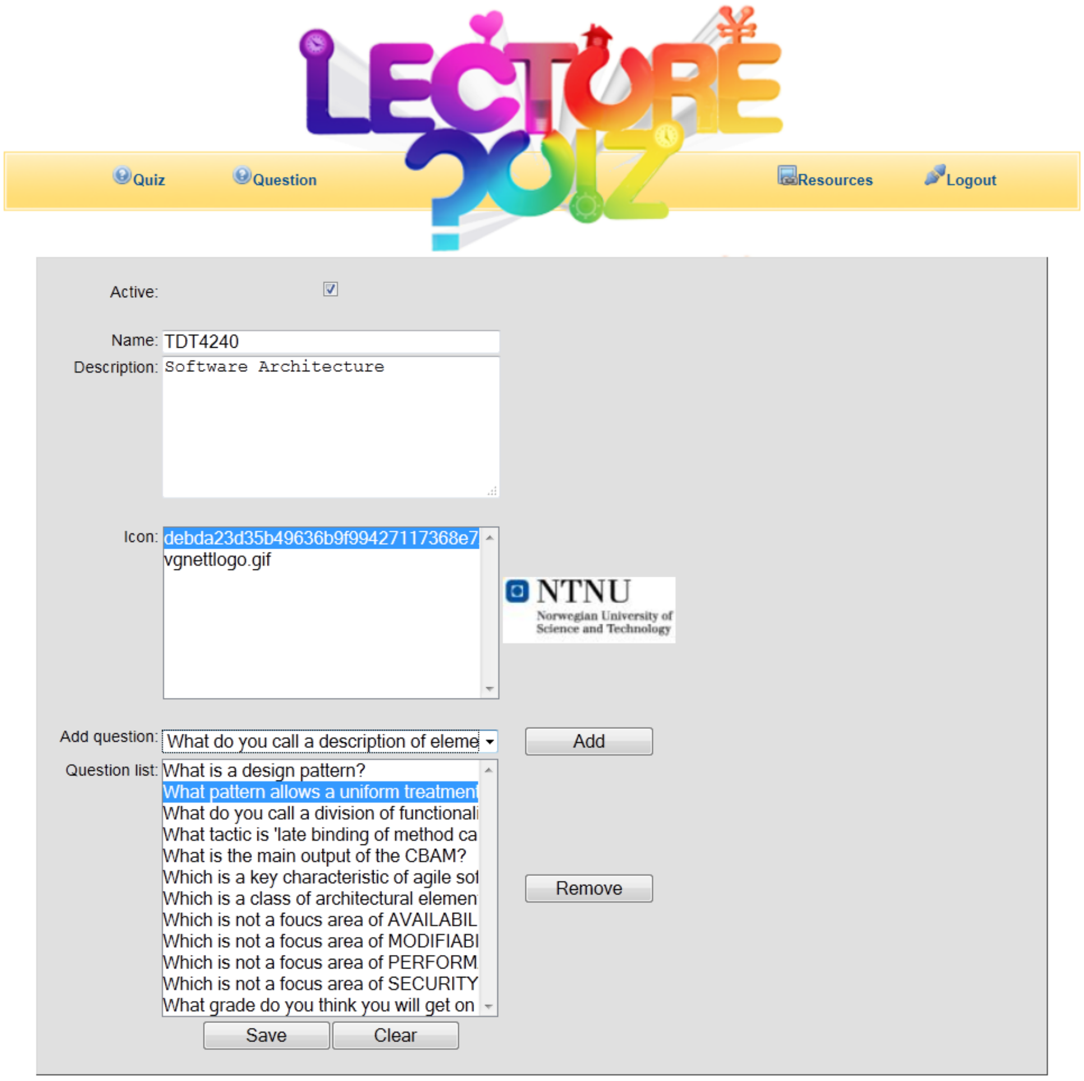
Icon: debda23d35b49636b9f99427117368e7

Figure E.21: Quiz Server create quiz screen.

To view all your quizzes, click the “My Quizzes” button. You will now be taken to the view quizzes web page, as shown in Figure E.20. This web page lets you view and manage all your current quizzes. Under the action tab you have the possibility to edit the quiz, add questions to the quiz, show statistics for the quiz, download it as a quiz package and delete the quiz.

Actions

If you click the edit quiz action, you will be taken to the edit quiz web page, as shown in Figure E.22. In this web page you have the possibility to edit all the quiz settings, including which questions that is connected to it. To add a question, select a question from the "Add question" drop-down menu, and then click the "Add" button. To remove a question, select a question from the "Question list" select list, and then click the "Remove" button. When you feel you have achieved what you want, click the save button. If you want to revert the changed, click the "Clear" button.



Active: ☑

Name: TDT4240

Description: Software Architecture

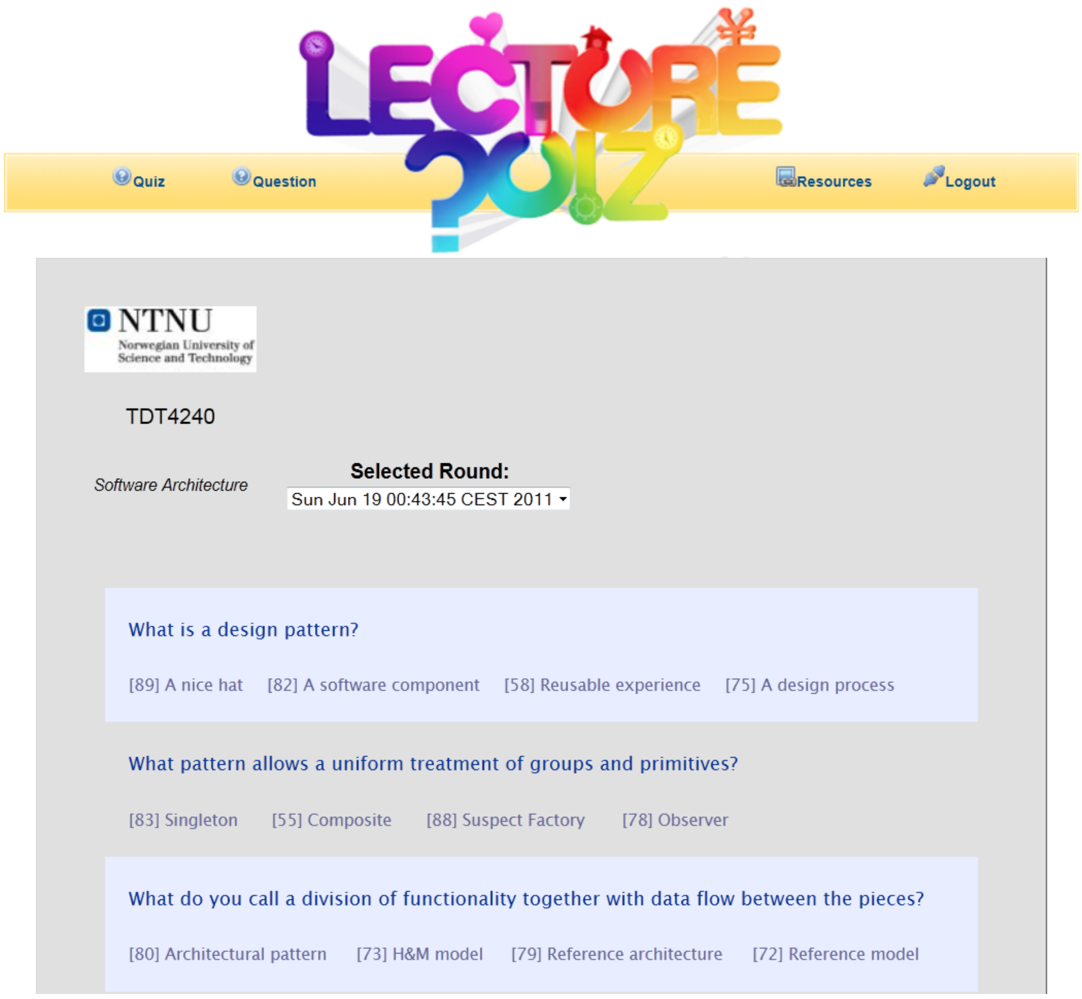
Icon: debda23d35b49636b9f99427117368e7
vgnettlogo.gif

Add question: What do you call a description of eleme

Question list:
What is a design pattern?
What pattern allows a uniform treatment
What do you call a division of functionali
What tactic is 'late binding of method ca
What is the main output of the CBAM?
Which is a key characteristic of agile sof
Which is a class of architectural elemen
Which is not a foucs area of AVAILABIL
Which is not a focus area of MODIFIABI
Which is not a focus area of PERFORM
Which is not a focus area of SECURITY
What grade do you think you will get on

Figure E.22: Quiz Server edit quiz screen.

If you click the show statistics action, you will be taken to the statistics web page, as shown in Figure E.23. In this web page you can view statistics related to a specific quiz. The “Selected Round” drop-down menu, contains the different quiz instances, sorted by the date and time the quiz was run.



TDT4240

Software Architecture

Selected Round:
Sun Jun 19 00:43:45 CEST 2011

What is a design pattern?

[89] A nice hat [82] A software component [58] Reusable experience [75] A design process

What pattern allows a uniform treatment of groups and primitives?

[83] Singleton [55] Composite [88] Suspect Factory [78] Observer

What do you call a division of functionality together with data flow between the pieces?

[80] Architectural pattern [73] H&M model [79] Reference architecture [72] Reference model

Figure E.23: Quiz Server statistics screen.

The third choice under the Quiz category is the ‘‘Upload Quiz Package” button. Click this to see the upload quiz package web page, as shown in Figure E.24. Simply select the browse button to choose a quiz package on your file system, click open, and then click “Upload Quiz Package”. The quiz will now be stored in the Quiz Server database and connected to your account. To browse it, select the “My Quizzes” tab.

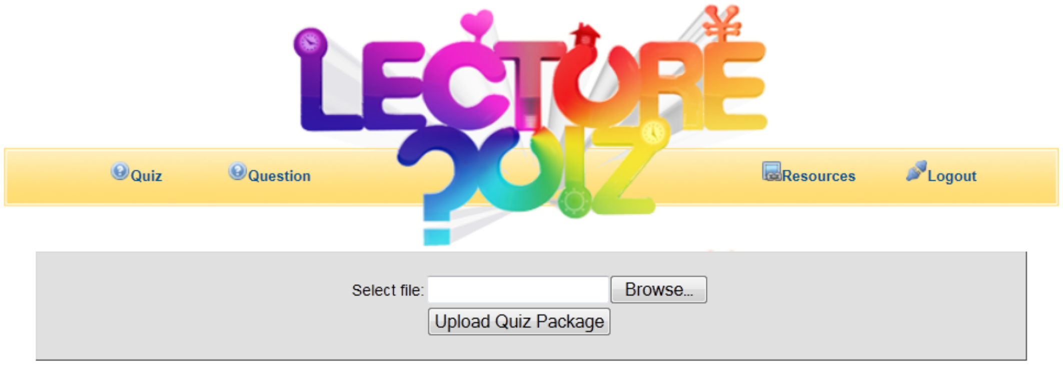


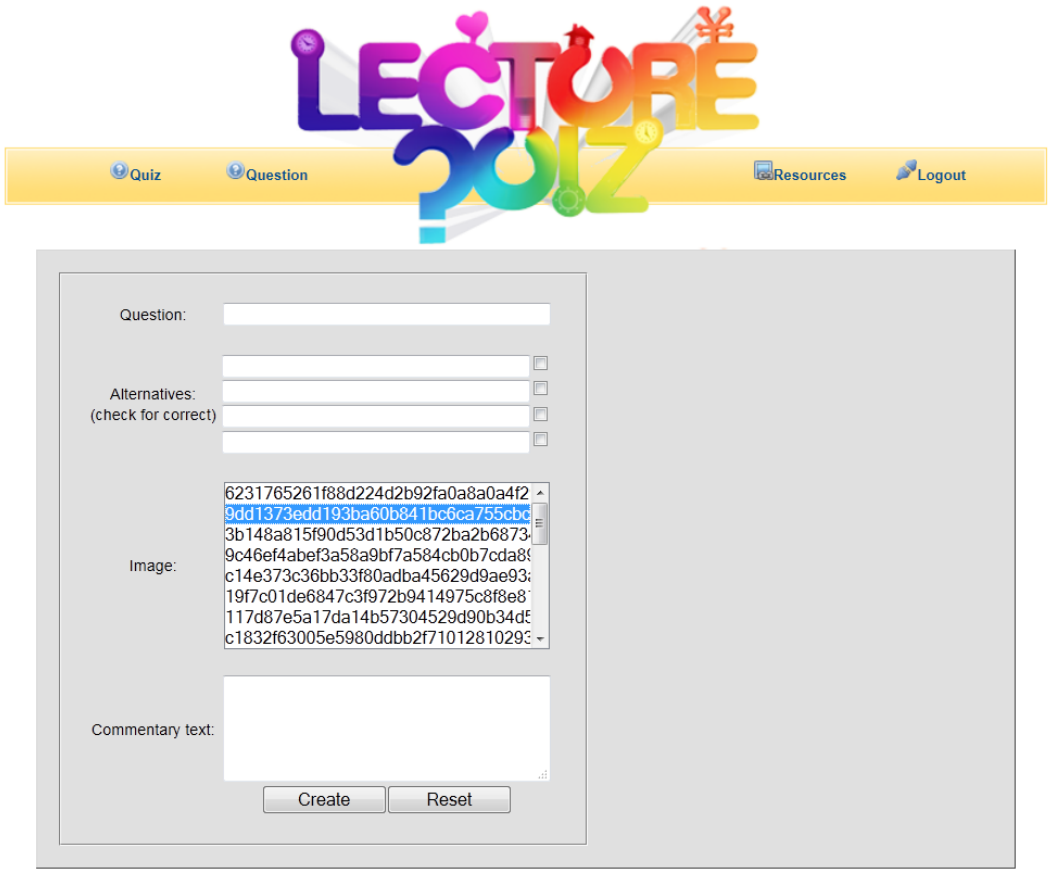
Figure E.24: Quiz Server upload package screen.

The last choice under the Quiz category is the “Upload Statistics Package” button. By clicking this, you will be taken to the upload statistics package web page. This web page works the same way as the previously explained upload quiz package web package, and needs no further explanation.

E.3.4 Question Management

The Question category contains: the “New Question” and “My Question” buttons.

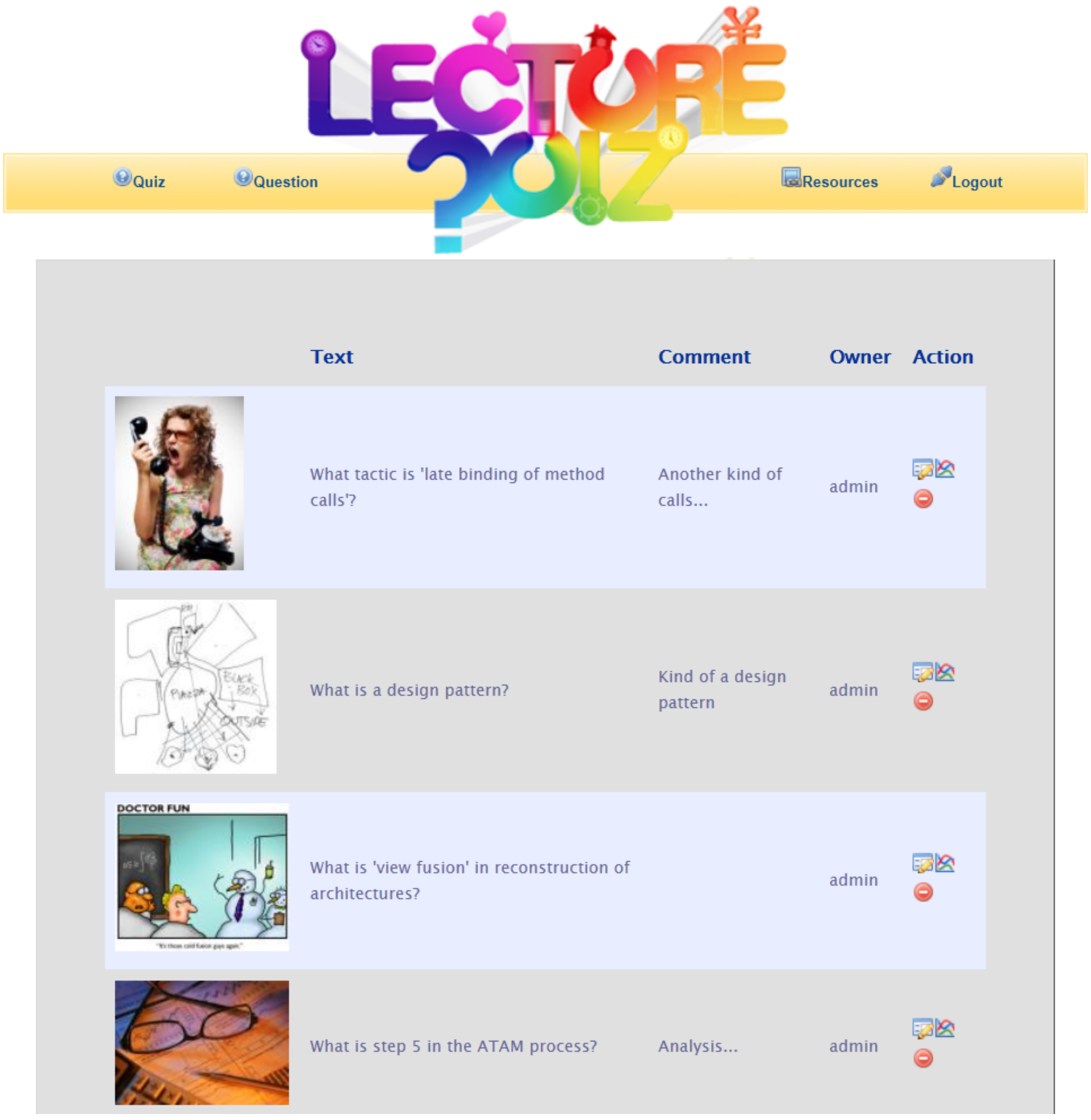
To create a new question, click the “New Question” button. You will now be taken to the create new question web page, as shown in Figure E.25. This web page lets you create a new question, after you fill in the necessary data. You have to specify a question and four alternatives. It is optional to check one or more correct alternatives, commentary text, and commentary image.



```
6231765261f88d224d2b92fa0a8a0a4f2
9dd1373edd193ba60b841bc6ca755cbc
3b148a815f90d53d1b50c872ba2b6873
9c46ef4abef3a58a9bf7a584cb0b7cda8
c14e373c36bb33f80adba45629d9ae93
19f7c01de6847c3f972b9414975c8f8e8
117d87e5a17da14b57304529d90b34d5
c1832f63005e5980ddbb2f71012810293
```

Figure E.25: Quiz Server create question screen.

To view all your questions, click the "My Question" button. You will now be taken to the view questions web page, as shown in Figure E.26. This web page lets you view and manage all your current questions. Under the action tab you have the possibility to edit the question, view question statistics and delete the question.



	Text	Comment	Owner	Action
	What tactic is 'late binding of method calls'?	Another kind of calls...	admin	
	What is a design pattern?	Kind of a design pattern	admin	
	What is 'view fusion' in reconstruction of architectures?		admin	
	What is step 5 in the ATAM process?	Analysis...	admin	

Figure E.26: Quiz Server question list screen.

E.3.5 Resources

The Resource category contains: the "Upload Image", "Upload Icon", and "My Resources" buttons.

To upload a new image or icon, click their respective buttons. You will now be taken to the upload resource web page. This web page has the same user interface as the upload quiz package web page, as seen in Figure E.24. For information on how to proceed from here, see the aforementioned section.

To view your resources, click the "My Resources" button. You will now be taken to the view resources web page, as shown in Figure E.27. This web page displays either the icons or images you have uploaded. You choose this view by either clicking the icons or images tab. This view supports 25 images/icons at a time, before you have to switch pages. To switch to the next web page use the four buttons below the images If you click on an image/icon, it will be shown full-scale in a new view.

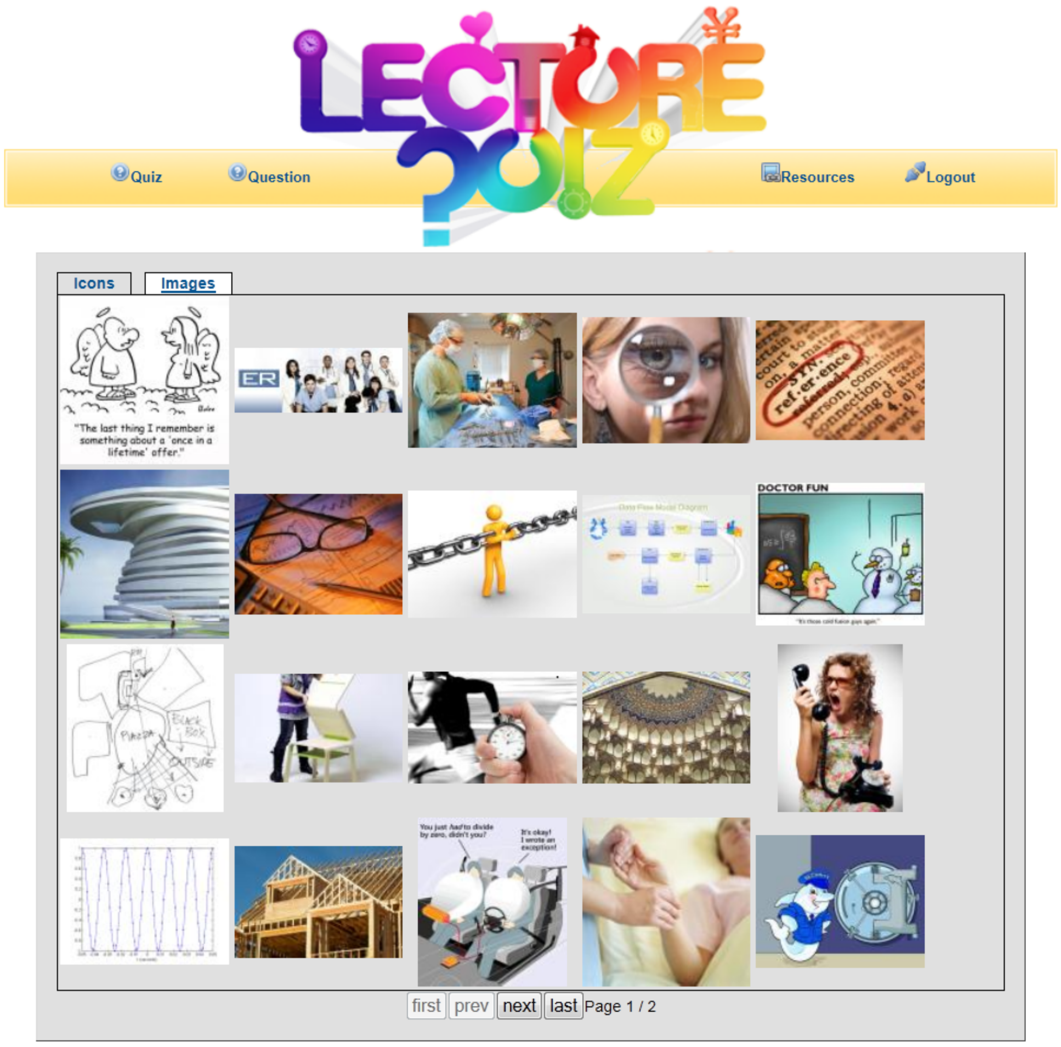


Figure E.27: Quiz Server image list screen.