Bjørn-Erik Ervik, Pernille Olsen, Espen Vad

User-friendly, full-stack table booking system for an Italian restaurant

Bachelor's project in Computer Engineering
Supervisor: Girts Strazdins
May 2021

Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Kunnskap for ei betre verd

Bjørn-Erik Ervik, Pernille Olsen, Espen Vad

User-friendly, full-stack table booking system for an Italian restaurant

Bachelor's project in Computer Engineering
Supervisor: Girts Strazdins
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Kunnskap for ei betre verd

NTNU
Kunnskap for en bedre verden

User-friendly, full-stack table booking system for an Italian restaurant

Bjørn-Erik Ervik

Pernille Olsen

Espen Vad

May 2021

BACHELOR THESIS

Department of ICT and Natural Sciences

Norwegian University of Science and Technology

Supervisor: Girts Strazdins

Mandatory group declaration

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:

1	Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	✓
2	Jeg/vi erklærer videre at denne besvarelsen: • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	✓
3	Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§14 og 15.	✓
4	Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver	✓
5	Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens studieforskrift §31	✓
6	Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider	✓

Acknowledgement

We would like to thank:

- Girts Strazdins, our supervisor which have guided the group through this project, providing us with great advice, assistance, and suggestions for the project, the project report, and client meetings.
- Enrico Agostinelli and Svetlana Dragieva, our clients that has given us specific feedback for improvement and change
- Friends and family for testing and providing feedback

Summary and Conclusions

Enrico Agostinelli and Svetlana Dragieva own an Italian restaurant, Cinque Minuti, in Ålesund, Norway. They are looking for a system that provides their restaurant and customers with an intuitive and easy-to-use table booking system.

The purpose of this thesis was to create a robust, interactive, intuitive, and user-friendly system where a customer can create a reservation, and the employees in the restaurant can have an overview of the current and future reservations.

The system was created with the MEAN-stack and a REST API, and a strong focus on designing an intuitive, user-friendly application with a consistent theme that matches the restaurant. Accessibility, particularly colorblindness, was considered when creating the application.

The solution to the stated problem was to create a full-stack system. The result includes a solution for both customers and employees, which has received great feedback. This includes a web application where customers can create a reservation and get an email confirmation afterward. Employees get an overview of where they can sort and filter through reservations. They get notified about new reservations, get alerted when a reservation starts within one hour, and can delete a reservation if needed. The personal information in a reservation gets removed one day after the reservation ends. The web application works on all devices and is downloadable as a Progressive Web Application.

Contents

Terminology

Angular A javascript front end framework that uses pre-built "components" to easier create a web application, these components are already styled and ready to use, thus making it easier in the development stage.

AppStore The iOS (Apple) platform for downloading more applications.

API Application Programming Interface, a back-end software for communicating between your application and your back-end logic, such as your database.

Back-end The logic of a web application that the user does not interact with. The back-end is everything that happens before the page is displayed in a web browser.

BSON User Datagram Protocol, non-connection-based transmission protocol of information.

Confluence A tool to store notes/documentation.

Database A organized collection of data, often stored and accessed electronically from a computer system that has high availability.

Environment To set the web application URL globally, so we dont have to hardcode the URL in each functions.

Express.js A server-side application framework for Node.js, written in typescript, to automate and deploy scaling API's.

Front end What the user interacts with, and the user interface. It is the interface design and the programming that makes the interface function.

Interface Mirroring the inputs in the browser, so the JSON know where to fetch the diffrent inputs.

JavaScript Program language used for interactive webpages. An essential part of the web application.

JIRA Online tool to track issues.

JSON JavaScript Object Notation. It is the structure of the way the object is going to build it.

Sourcetree Git client that offers visual representation of Git repositories.

MEAN-STACK A combination of different technologies for developing a web application, the technologies consist of M(MongoDB), E(Express.js), A(Angular), and N(Node.js).

MongoDB A NoSQL database for storing and retrieving large amounts of data.

Node.js A open-source, cross-platform, back-end JavaScript runtime environment.

PlayStore The Androids platform for downloading more applications.

Service A service is an option where a script can bring various features to the project. In our case, the service is what binds the front-end and back-end together. It calls the method in the front-end to route it correctly in the back-end.

SQL Structured Query Language (SQL) SQL is a domain-specific language used in programming and for managing data in a relational database management system (RDBMS).

TypeScript A extension of JavaScript that speeds up development by catching errors and providing fixes, made my Microsoft.

Web Application Application software which runs on a webserver.

Webserver Is a server that stores and sends information to the internet.

Abbreviations

API	Application Programming Interface
BSON	Binary JSON
CLI	Command Line Interface
CSS	Cascading Style Sheets
DNS	Domain Name Server
DOM	Document Object Model
GDPR	General Data Protection Regulation
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
MEAN	MongoDB, Express, Angular and Node.js
MERN	MongoDB, Express, React and Node.js
MEVN	MongoDB, Express, Vue and Node.js
MVP	Minimum Value Product
Navbar	Navigation bar
PWA	Progressive Web Application
RegEx	Regular Expressions

SVG	Scalable Vector Graphics
SSL	Secure Socket Layer
SQL	Structured Query Language
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
VSC	Visual Studio Code
VM	Virtual Machine
XML	Extensible Markup Language

List of Figures

Chapter 1

Introduction

This chapter will introduce the assignment and the project's structure, formulate the problem, limitations, approach, and design of the report.

1.1 Background

This bachelor is carried out by Bjørn-Erik Ervik, Espen Vad, and Pernille Olsen as students of computer engineering at NTNU Ålesund. Enrico gave the task to us based on a list of assignments. We got our first choice to create a table booking system for the restaurant Cinque Minuti in Ålesund.

The task for the bachelor was to develop a system to help optimize table booking and future reservations for the restaurant Cinque minuti. Optimizing table booking would be made by creating a Progressive Web Application (PWA) that the users can connect to in their browser and fill in the information for their booking. This application will allow the customers to book the table they want in advance, prepare the restaurant for reservations, and improve business flow.

The task was given to us by Enrico Agostinelli, the co-owner of the Italian restaurant Cinque minuti, located in central Ålesund (Gågata Ålesund). With Enrico and the second co-owner of the restaurant Svetlana Dragieva as our person of contact. Through several meetings at the restaurant, a requirement specification was created that detailed the system's functionality, which components should be designed, which user roles exist within the system and the goal of the entirety of the system. The requirement specification can be found in AppendixA.

The background for the task was that Enrico and Svetlana wanted better flow in their restaurant and a system that would help with table reservations.

This report includes the theoretical background, knowledge, and concepts used to solve the task; the tools, methods, and materials used to solve the task as well as the reasoning to why the group chose them; and the achieved result of the project with a reflection and discussion regarding results as mentioned earlier.

1.2 Problem Formulation

Usually, when a person wants to reserve a table for a restaurant at a particular time, they call the restaurant, which selects a table for them based on availability at the time. In most cases, the option of choice is minimal. With the technology and methods we have, we can create a visual booking system where the user can choose the specifics of the reservation and get a visual overview of the tables in the restaurant. However, these types of systems are not the standard for booking. Cinemas often have a similar system in place, but it is not the standard for restaurants. Therefore, to create a better user experience, we need a dynamic, user-friendly system available on almost every device. The system has many standards to be met and, as a consequence, many problems to be addressed.

1.2.0.1 Problems to be addressed

- Design
- Technology/architecture
- User-friendliness
- Development tools

1.3 Limitations

In this particular project, we will be limited to the tools we choose and their capability. An example of this would be HTML, CSS, and Javascript. This has shown to be one limitation that good development practice can break with enough skill and expertise. You can practically make whatever you want, as long as you are good enough, have enough time, and enough resources. An excellent example of this is that almost all tools we use today are made up of javascript or other more basic programming languages. This means that the tools we have are powerful and can be used to create robust solutions. Another limitation we have is time; the project is set to be finished on May 20th. This will give us a set amount of time where the requirements have to be met.

1.4 Approach

In this project, the group will approach the problem using planning tools, the MEAN stack, collaborative coding tools such as Github and Jira, and our vision for how the final project will look.

1.5 Structure of the Report

The report is structured as follows:

1. **Chapter 1 - Introduction**
   Introduction for the report with information about problem formulation, limitations, problems and the approach

2. **Chapter 2 - Theoretical basis**
   Chapter two gives an introduction to the theoretical background needed to understand different concepts and terminology for future use

3. **Chapter 3 - Method**
   Contains a description of the methodology and materials used throughout the project

4. **Chapter 4 - Result**

   Contains the results of the finished project and system

5. **Chapter 5 - Discussion**

   Contains discussion of the finished project and system

6. **Chapter 6 - Conclusions**

   This chapter present an overall conclusion of the project

Chapter 2

Theoretical basis

This chapter explains the theoretical basis for the project. It also describes many of the necessary principles and specifics of the techniques used to create the project.

2.1 Laws and Regulations

Our project involves the handling of personal data and is therefore subjected to several laws and regulations. These are described below.

2.1.1 GDPR

The General Data Protection Regulation (GDPR) is a privacy and security law that aims to control individuals over their data and simplify the regulatory environment for international business by unifying the regulation within the EU and the European Economic Area (EEA). The GDPR rules were put into effect on May 25th, 2018 (*General Data Protection Regulation* 2021).

2.1.1.1 Principles

According to the GDPR chapter 2, article 5, personal data should be processed in a fair and transparent matter where the purpose of collection should be specified and legitimate. It should be limited to only the necessary information, stored for only how long it is necessary, and uphold security with integrity and confidentiality. (*GDPR* 2021).

Personal Data

Personal data means any information that can be related to an identifiable natural person. (*GDPR* 2021Ch.1, Art.4).

2.1.2 Personal Data Act

The Personal Data Act is a Norwegian law for privacy protection that took effect on April 14th, 2001 (*Personopplysningsloven* 2021). It represents the principle that individuals should control how data relating to them is used (Datatilsynet, 2021). On July 20th, 2018, the law was updated to include the GDPR (beredskapsdepartementet, 2018).

2.2 MEAN stack

The MEAN stack is a full-stack web development framework based on JavaScript. The name is an acronym for MongoDB (2.5.1), Express.js (2.4.8), Angular (2.4.6), and Node.js (2.4.7), the four technologies the stack comprises (MongoDB, 2021b).

2.3 User Experience

User experience, or UX design, is the process of understanding how a user interacts with and experiences a system (*User experience* 2021). The concepts and principles we have used in the project regarding user experience are described below.

2.3.1 Usability

According to ISO 9241-11, usability is the *"extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use"* (*ISO 9241-11* 2018).

2.3.2 Design Principles

Design principles are widely applicable laws, guidelines, biases, and design considerations for designing an easy-to-use, intuitive, and enjoyable application. (Foundation, 2021).

2.3.2.1 Don Norman's Design Principles

Don Norman defines six principles of design in “The Design of Everyday Things” (Dahl, 2017):

- Visibility – The principle of how the more visible the functionality is, the more likely it is to understand how to use it.
- Affordance – The principle of giving an indication of what action an element is supposed to serve.
- Constraints – The principle of limiting the number of actions a user can perform.
- Feedback – The principle of giving information to the user about what action has been performed and the result of that action.
- Mapping – The principle of creating a context between layout and interaction.
- Consistency – The principle of having similar elements and similar operations accomplish similar tasks (Rekhi, 2018).

2.4 Frameworks and technologies

This section describes the frameworks and technologies that are relevant to this project.

2.4.1 HTML

HTML (HyperText Markup Language) is a markup language for displaying documents in a web browser. It describes the structure of a webpage, with elements that often contain tags and content (*HTML* 2021).

2.4.2 CSS

CSS (Cascading Style Sheets) is a style sheet language used to describe rules for presenting a document (such as layout, colors, and fonts) written in a markup language such as HTML. Using CSS can improve accessibility, provide flexibility and control of the presentation, enable multiple pages to share formatting, and accessing a web page on a mobile device. (*CSS* 2021).

2.4.3 JavaScript

JavaScript is a lightweight, interpreted, or just-in-time compiled programming language. It is a scripting language for web pages and is used by many non-browser environments, such as Node.js. JavaScript has first-class functions, is prototype-based, multi-paradigm, and single-threaded. It supports object-oriented, imperative, and declarative programming styles. It is a loosely typed and dynamic language. (MDN, 2021b).

2.4.4 SVG

SVG (Scalable Vector Graphics) is an XML-based markup language for describing two-dimensional vector graphics. It has support for interactivity, animation and can be rendered to any size without loss of quality. The files are defined as XML text files and designed to work well with other web standards, such as CSS and JavaScript (MDN, 2021c).

2.4.5 TypeScript

TypeScript is a programming language that is a superset of JavaScript. It is a typed superset, which means that it adds rules about how different values can be used. Since it is a superset, JavaScript code can be written in a TypeScript file. It has static type checking, which means it detects an error based on the kinds of values being operated on before executing the code. TypeScript also preserves the runtime behavior of JavaScript, which means that JavaScript code in a TypeScript file will be guaranteed to run the same, despite type errors (*Documentation* 2021).

2.4.6 Angular

Angular is an open-source development platform based on TypeScript. It includes a component-based framework for building scalable web applications, collecting first-party libraries for additional functionality, and a suite of developer tools to help develop, build, test, and update code (Angular, 2021a).

2.4.6.1 Components

Angular applications are built up by components. Components include a TypeScript class with a component decorator, which specifies a CSS selector that defines how the component is used in a template, an HTML template that declares how the component renders, and optional CSS styles that define the presentation (Angular, 2021a).

2.4.6.2 Templates

Angular templates extend HTML with text interpolation that lets dynamic values from the component be inserted in the template and property binding, event listeners, and directives (such as *ngIf and *ngFor). Dependency injection helps declare dependencies of typescript classes while taking care of their instantiation (Angular, 2021a).

2.4.6.3 CLI and First-Party Libraries

Angular has a CLI that helps generate components, compile, build, serve, run unit tests, and run end-to-end tests. Angular also provides several first-party libraries that add additional functions, such as routing, forms management, client-server communication, and more (Angular, 2021a).

2.4.6.4 Single Page Application

A Single Page Application (SPA) only loads a single HTML page. The application renders its features when it is needed instead of loading a new page. Routes are used to navigate through an Angular SPA (Angular, 2021b).

2.4.7 Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment that runs the V8 JavaScript engine. It allows server-side writing code in addition to client-side code without the need to learn a different language (*Introduction to Node.js* 2021).

Node.js is single-threaded and event-driven and processes incoming requests with an event loop, allowing Node.js to perform non-blocking I/O operations by offloading operations to the system kernel whenever possible (Node.js, 2021).

Node.js has a package manager, called npm, designed to simplify installation, updating, and uninstallation of packages from the npm registry (*Node.js* 2021).

2.4.8 Express.js

Express is a minimal and flexible Node.js web application framework. It provides several middleware modules and utility functions for building web applications and APIs (*Express* 2021).

2.4.9 WebSocket

WebSocket is an application layer communication protocol standardized by the IETF as RFC 6455 (*rfc6455* 2021). It establishes a full-duplex communication channel over a single TCP connection and enables real-time data transfer and interaction between a client and a web server (*WebSocket* 2021).

2.4.9.1 Socket.io

Socket.io is a node.js library that enables real-time, bidirectional, and event-based communication between client and server. It consists of a node.js server and a Javascript, or node.js, client library (Arrachequesne, 2021).

2.4.10 Version Control

"Version control involves keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other." (Sommerville, 2016a)

2.4.10.1 Distributed Version Control Systems

In a distributed version control system, such as Git, clients fully mirror the repository, including its entire history. This means that the repository is stored on each client's computer that has cloned the repository. Each clone is then a full backup of the data in the repository and has the ability to be restored (Chacon and Straub, 2021p. 12-13).

2.4.11 Integrated Development Environment

An integrated development environment (IDE) is a software application that implements numerous resources for software development. They are designed to maximize productivity by providing components with similar user interfaces and presenting a single program in which all development is done. An IDE frequently contains compilers, interpreters, possibilities for refactoring and formatting, test automation, and integration for version control systems (*Integrated development environment* 2021).

2.4.12 REST

Representational State Transfer (REST) is a software architecture style that uses a subset of HTTP to create web services and APIs (*REST* 2021).

2.4.12.1 REST Constraints

Roy Thomas Fielding described the basis of REST by six constraints (Fielding, 2000):

Client-Server

The principle behind the Client-Server constraint is to separate the user interface concerns from the data storage concerns, improving portability of the user interface across multiple platforms and scalability by simplifying server components (Fielding, 2000).

Stateless

Client-server interaction must be stateless. Each request must contain all information necessary to understand the request and cannot take advantage of any stored context on the server (Fielding, 2000).

Cache

The data within a response to a request has to be labeled as cacheable or non-cacheable. If a response is cacheable, a client cache is given the right to reuse that response data for later, equivalent requests. This has the potential to remove interactions, improve efficiency, scalability, and user-perceived performance (Fielding, 2000).

Uniform Interface

Uniform interface is the feature of applying generality to the component interface, which creates a simplified system architecture with improved visibility of interactions (Fielding, 2000).

Layered System

The layered system constraint allows the architecture to be composed of hierarchical layers with restricted knowledge of other layers than the one interacted with (Fielding, 2000).

Code-on-Demand

An optional constraint that allows client functionality to be extended by downloading and executing code in the form of applets or scripts simplifies clients by reducing the number of features required to be pre-implemented (Fielding, 2000).

2.4.12.2 REST API

A REST API is an application programming interface (API) that conforms to the restraint of the REST architectural style, allows for interaction with RESTful web services, and enables CRUD (Create, Read, Update, Delete) operations. An API is a set of definitions and protocols for building and integrating application software, which lets a client communicate with a resource or service (redhat, 2021).

2.5 Database theory

This section contains the theory behind the database used in this project.

2.5.1 MongoDB

MongoDB is a NoSQL (2.5.2) document database that is free to use. It stores data in documents formatted as the JSON-like format BSON. The fields in each document can vary, and the data structure can be changed over time. It is a distributed database with high availability, horizontal scaling, and geographic distribution and supports ad hoc queries, indexing, and real-time data aggregation (MongoDB, 2021b).

2.5.2 NoSQL

NoSQL databases stores data differently from relational databases. The database provides flexible schemas, scales easily, allows related data to be nested within a single data structure and is non-tabular, which means that the data in the database is not formatted in a table. The main types of NoSQL databases are document, key-value, wide-column, and graph (MongoDB, 2021b).

2.6 Security

This section describes the methods and standards used to provide security in the application.

2.6.1 Stateless Authentication

Stateless authentication, or token-based authentication, is an authentication method that stores user session data on the client-side. The token conveys all the required information for authenticating the request and is cryptographically signed (*What is Stateless Authentication ?* 2021).

2.6.1.1 JSON Web Tokens

"JSON Web Tokens (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object"(auth0.com, 2021)

It is digitally signed, either using a secret (HMAC) or a public/private key (RSA or ECDSA), and is used for authentication, authorization, and information exchange. It has structured consists of a header, payload, and a signature (auth0.com, 2021).

2.7 Testing

This section describes the types of testing used in the project.

2.7.1 Unit Testing

Unit testing is a typically automated software testing method where units of source code are tested to see if it meets the intended behavior (*Unit testing* 2021).

2.7.2 Usability Testing

Usability testing is an observational methodology used to uncover problems, identify improvements, and learn about the user's behavior and preferences. During usability testing, a facilitator asks a participant to perform tasks while the facilitator observes the participant's behavior and interaction with the tasks (Moran, 2019).

2.7.3 Integration Testing

Integration testing is a technique for testing functions between several modules in a software system. Integration testing tests large parts of a system at a time, as opposed to, for example, unit testing, where one tests at the lowest level at a time (*ISO/IEC/IEEE 24765* 2021).

Chapter 3

Materials and methods

This chapter will explain how the different tools and technologies were used to solve the project. Here are screenshots of our prototype, colors and design patterns, different IDE's, communication, and version control. This chapter will also be about the project architecture, HTTP Protocols, and its response. At the end of this chapter, the group will discuss security, and how its implemented, and some testing.

3.1 Project Organization

The project group consists of three students studying computer engineering at NTNU in Ålesund, Bjørn-Erik Ervik, Espen Skrede Vad, and Pernille Olsen. It also includes project supervisor Girts Strazdins and Enrico Agostinelli, owner of restaurant Cinque Minuti in Ålesund.

3.2 Project planning

In January, the group worked on and delivered the pre-project report (see appendix C) for the course "IF300114 - Ingeniørfaglig systemteknikk og systemutvikling". The group had the first meeting with the project supervisor and project owner on January 13th. Thoughts on the project and the client's wishes and expectations were discussed, and future meetings were planned. Meetings were planned for every other week. The group also worked out a product specification (see appendix A) and planned out the web application design (see appendix B and section 3.4.2) during this time. It was important for the client that the web application would be intuitive and easy to use, so the group focused a lot on the design in the planning stage.

The group did not have previous experience using full stack solutions, like the MEAN stack, so the essentials had to be learned. The group followed different tutorials, read documentation, and created small projects to test it out. This was important for the project's future since a lot was learned quickly, and the group was more prepared to start the project.

3.3 Project Management

This section explains the methods and materials used for increasing productivity and collaboration in the project.

3.3.1 Agile Software Development Methodology

The group decided to use an agile software development methodology, specifically the framework Scrum (3.3.1.1).

"Agile software development methods are geared to rapid software delivery. The software is developed and delivered in increments, and process documentation and bureaucracy are minimized. The focus of development is on the code itself, rather than supporting documents" (Sommerville, 2016cp. 757).

3.3.1.1 Scrum

Since the group members had used Scrum during projects earlier in the education, it was considered the most suitable project.

Scrum is a simple and easy-to-use framework for agile project organization providing an iterative, incremental approach to development. It is founded on empiricism (asserting that knowledge comes from experience and observation-based decisions) and lean thinking (focusing on the essentials) (Schwaber and Sutherland, 2020).

3.3.1.2 Scrum Team

The scrum team consists of a product owner, a scrum master, and one or more developers. The team is supposed to be self-managing and cross-functional (Schwaber and Sutherland, 2020).

During planning, the group chose a sprint master. He would as well as being a developer, be accountable for the team's effectiveness and make decisions based on the group's input. The rest of the group got assigned as developers. Part of the Scrum team is also the supervisor for the project. The product owner is the client.

Product Owner
The product owner's role is to identify product features and requirements, prioritize and continuously review the product backlog, and ensure the project meets its needs. The product owner can be the customer (Sommerville, 2016b).

Scrum Master
The scrum master's role is to guide the team in using scrum efficiently and ensuring that the scrum process is followed (Sommerville, 2016b).

Developer Team
A developer team is a self-organizing group responsible for developing the project and its necessary documentation (Sommerville, 2016b).

3.3.1.3 Scrum Events

Scrum events consist of sprints of a chosen length, sprint planning, daily scrum, sprint review, and sprint retrospective.

The group decided on a sprint length of two weeks, with one shorter for the exam in for "IF300114 - Ingeniørfaglig systemteknikk og systemutvikling", and one longer for Easter.

At the end of each workday, the group had a daily scrum meeting, where the work of the day and any problems, questions, or opinions were discussed. At the end of each sprint, the group had a meeting (sprint review) with the supervisor and product owner. Before the meeting, a meeting document was created, which summarized what was accomplished during the sprint and some goals for the next sprint. The group showed the result of what had been worked on during the last sprint and discussed what should be worked on for the next, so both the supervisor and the product owner could see the progress.

Sprint

Sprints are events of a certain length, usually two to four weeks. A new sprint starts immediately after the last. All the other scrum events (sprint planning, daily scrums, sprint reviews, and sprint retrospectives) happen within a sprint (Schwaber and Sutherland, 2020).

Sprint Planning

During sprint planning, the whole team creates a plan for what is expected to be achieved during the sprint. The team addresses what can be done during the sprint and how it can be done (Schwaber and Sutherland, 2020).

Daily Scrum

Daily scrum, also called stand-up meetings, are short daily meetings where the team discusses the progress toward the sprint goal, adjusting tasks if necessary. The meeting is held at the same time, every day (Schwaber and Sutherland, 2020).

Sprint Review

During sprint reviews, the team evaluates and presents the sprint result to the customer, determines future adjustments, and decides what to do next. The sprint review is the next to last event of the sprint (Schwaber and Sutherland, 2020).

Sprint Retrospective

Sprint retrospectives are the last event of the sprint, which concludes the sprint. The team evaluates how the sprint went, what went well, problems encountered, and how to solve them (Schwaber and Sutherland, 2020).

3.3.1.4 Scrum Artifacts

Scrum artifacts are information that the team uses to specify the product being developed. The primary artifacts are the product backlog, sprint backlog, and increments (Harris, 2021).

After sprint review and sprint retrospectives, the group planned out the next sprint by adding tasks and user stories to the backlog and decided what tasks and user stories should be in the next sprint.

Product Backlog

The product backlog contains features, improvements, bug fixes, tasks, and requirements needed in development. It is updated when new information is available and added to as new ideas appear (Harris, 2021).

Sprint Backlog

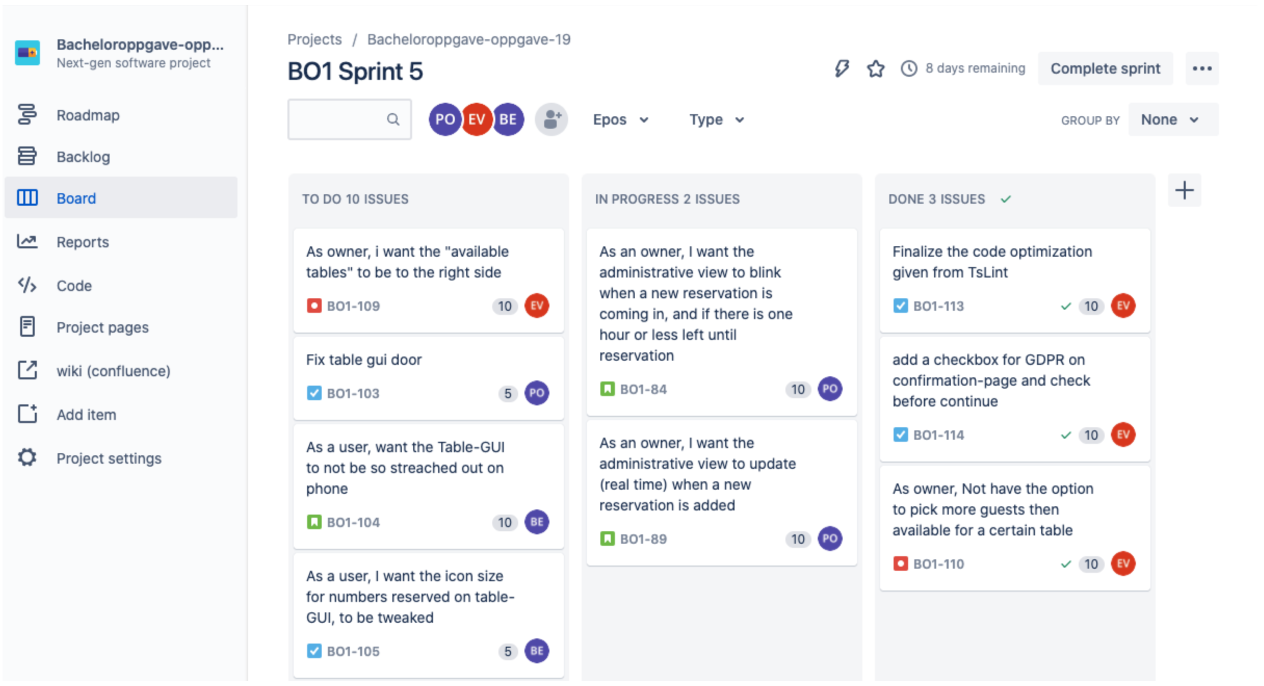
The sprint backlog is a set of product backlog tasks that have been included in a sprint. It is updated during the sprint planning phase (3.3.1.3), and tasks are assigned (Harris, 2021).

User Stories

"A user story is an informal, general explanation of a software feature written from the end user's perspective. Its purpose is to articulate how a software feature will provide value to the customer. A user story can also have a point score called story points. Story points reflect how much a story is "worth" and how difficult it is. Story points are often used to determine "how much" a developer has developed." (Rehkopf, 2021)

3.3.2 Atlassian Jira

The group chose to use Atlassian Jira to plan, track and manage the workflow of the project. It includes a scrum board (Figure 3.1), backlog, and produces reports. When an issue is created in the backlog, the team can discuss which sprint to add the issue too. Only the issues that are in the current sprints will show up on the board. Meanwhile, in the backlog, all the issues are tracked.



BO1 Sprint 5

TO DO 10 ISSUES

- As owner, i want the "available tables" to be to the right side — BO1-109 — 10 — EV
- Fix table gui door — BO1-103 — 5 — PO
- As a user, want the Table-GUI to not be so streached out on phone — BO1-104 — 10 — BE
- As a user, I want the icon size for numbers reserved on table-GUI, to be tweaked — BO1-105 — 5 — BE

IN PROGRESS 2 ISSUES

- As an owner, I want the administrative view to blink when a new reservation is coming in, and if there is one hour or less left until reservation — BO1-84 — 10 — PO
- As an owner, I want the administrative view to update (real time) when a new reservation is added — BO1-89 — 10 — PO

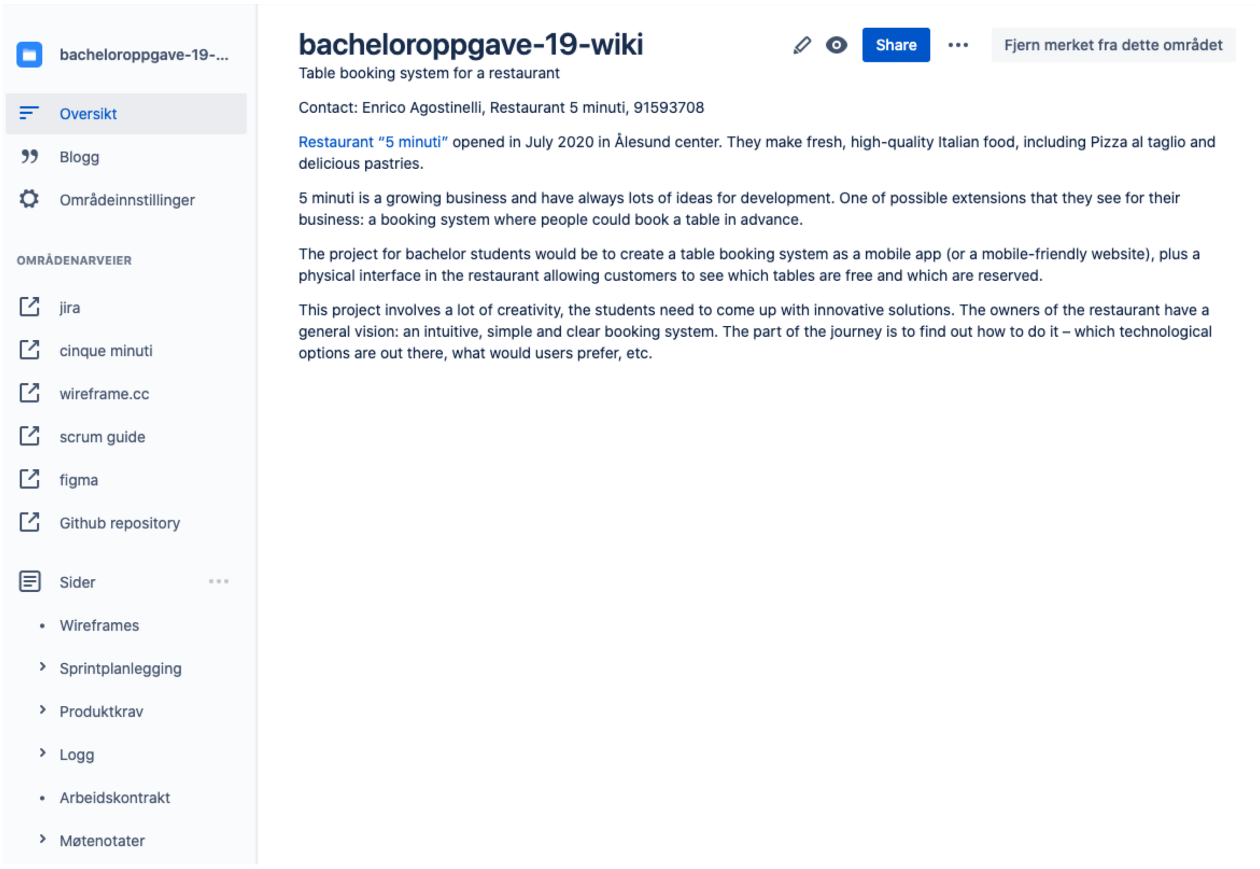
DONE 3 ISSUES

- Finalize the code optimization given from TsLint — BO1-113 — 10 — EV
- add a checkbox for GDPR on confirmation-page and check before continue — BO1-114 — 10 — EV
- As owner, Not have the option to pick more guests then available for a certain table — BO1-110 — 10 — EV

Figure 3.1: Jira - Scrum board

3.3.3 Atlassian Confluence

The group used Confluence (Figure 3.2) to document the process, write logs for each workday, create a product specification(see appendix A), and take meeting notes. Each meeting with the client and supervisor was documented here (see appendix E).



bacheloroppgave-19-wiki

Table booking system for a restaurant

Contact: Enrico Agostinelli, Restaurant 5 minuti, 91593708

Restaurant "5 minuti" opened in July 2020 in Ålesund center. They make fresh, high-quality Italian food, including Pizza al taglio and delicious pastries.

5 minuti is a growing business and have always lots of ideas for development. One of possible extensions that they see for their business: a booking system where people could book a table in advance.

The project for bachelor students would be to create a table booking system as a mobile app (or a mobile-friendly website), plus a physical interface in the restaurant allowing customers to see which tables are free and which are reserved.

This project involves a lot of creativity, the students need to come up with innovative solutions. The owners of the restaurant have a general vision: an intuitive, simple and clear booking system. The part of the journey is to find out how to do it – which technological options are out there, what would users prefer, etc.

Figure 3.2: Confluence - front page

3.3.4 Communication

During the project, the group communicated and cooperated daily over the platform named discord. Discord allows for communication with voice and video calls, screen sharing, and messaging. Communication with the client and supervisor, such as sprint reviews and technical meetings, happened mainly over zoom and email. In the middle of the project, the group's supervisor also joined the discord channel. If the group needed help or answers to questions, he could respond there.

3.4 Graphical User Interface

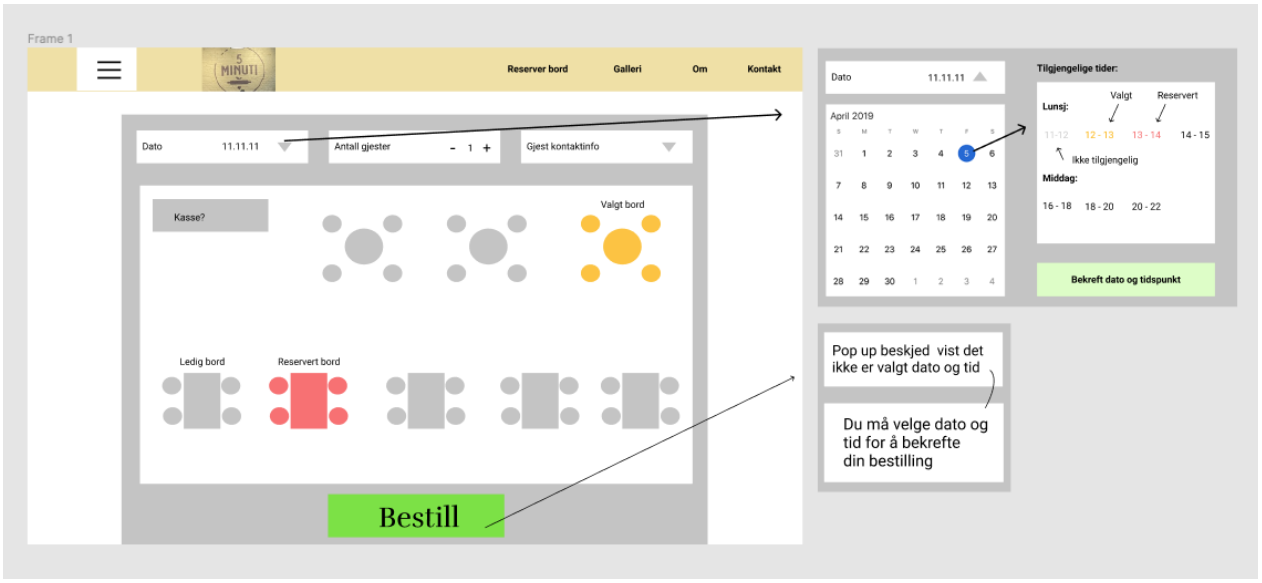
This section explains the methods and tools used to create and ensure an accessible, responsive, user-friendly graphical user interface.

3.4.1 Figma

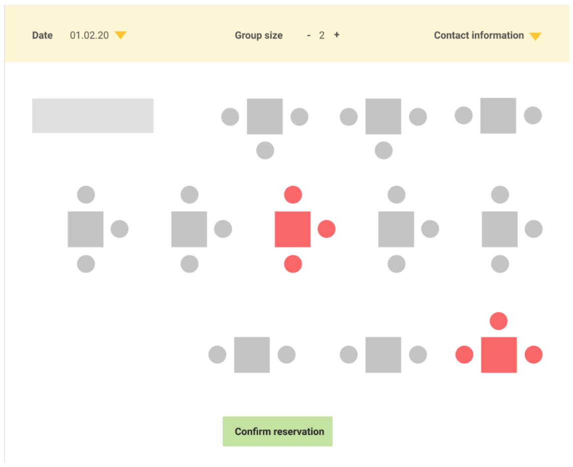
When planning the user interface for the web application, the group used the application Figma. Figma is a web-based, collaborative application for designing UI and UX. It allowed the group to show ideas through wireframes and prototypes and establish the website's functionality. The group used it to create a prototype to visualize the application's result.

3.4.2 Wireframes

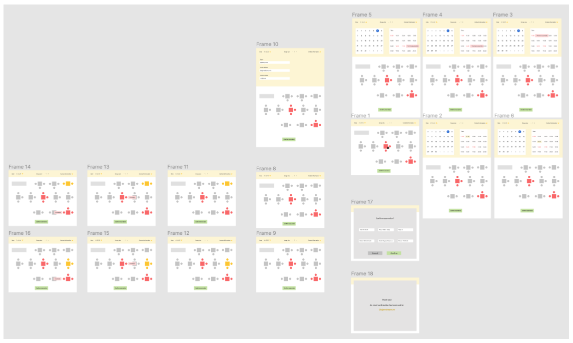
The group started planning the design by creating wireframes. First simple wireframes were created with ideas of functionality the group thought would fit the project (Figure3.3a, or figma project). This included some dropdown functionality and a basic idea of what the application is today. Then the first wireframe was built upon, with a new wireframe with a cleaner design (Figure 3.3b and 3.3c, or figma project). The group then created a prototype (see the section about prototypes 3.4.3).



(a) Wireframe version 1



(b) Wireframe version 2

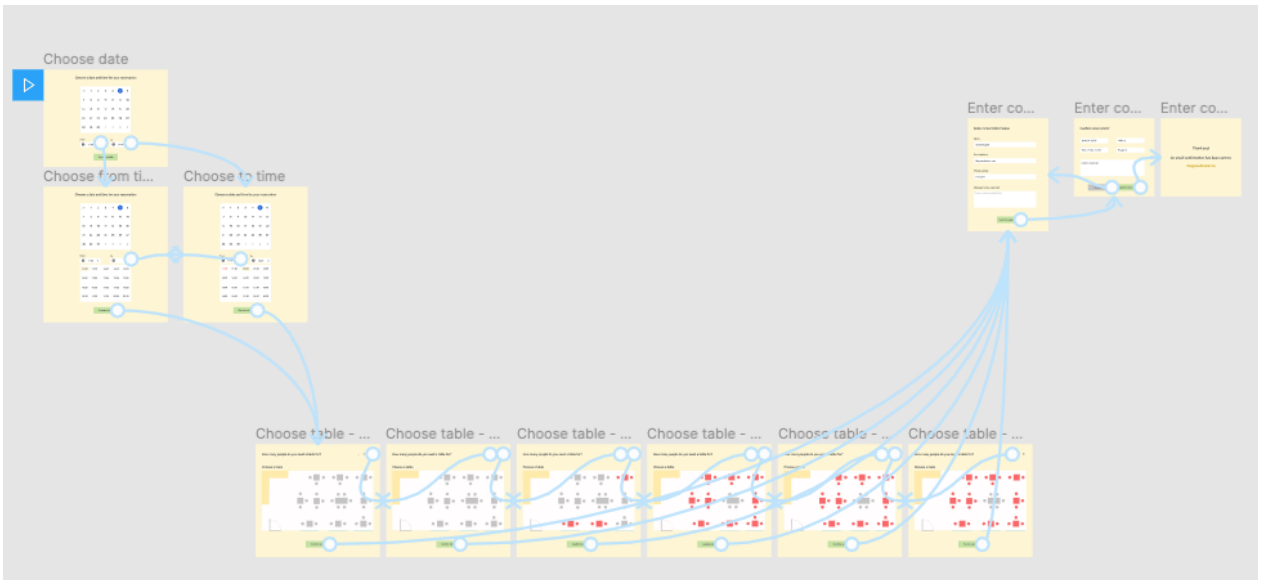


(c) Wireframe version 2

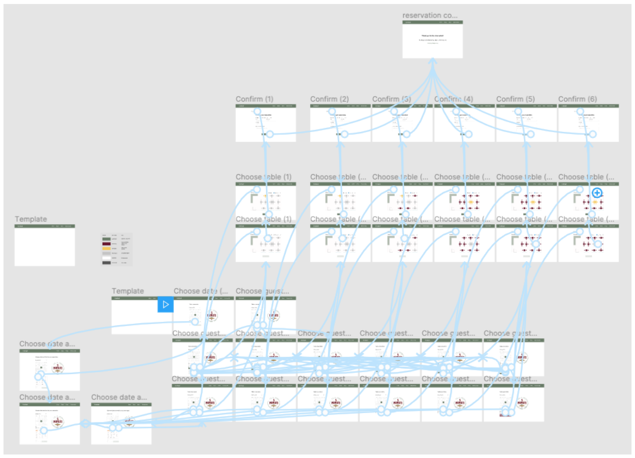
Figure 3.3: Wireframes

3.4.3 Prototypes

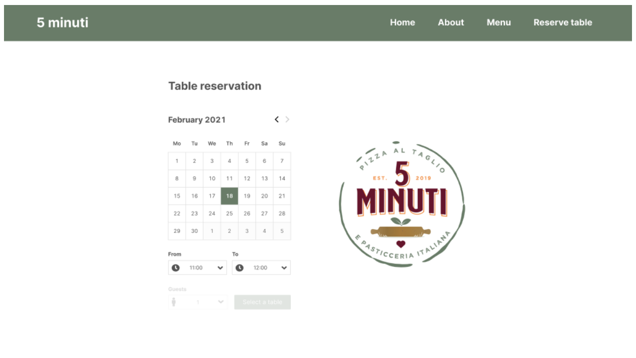
Prototypes are practical for communicating the functionality and interaction of an application to everyone involved in the project. The prototype (Figure 3.4a) was based on the second wireframe, with added interaction such as choosing a table, date and clicking buttons. During the first meeting, the group showed this prototype to the supervisor and product owner. Based on feedback and new ideas, the group created a new prototype (Figur 3.4b and 3.4c) with a new design, including a color scheme reflecting the existing logo. This made the design more coherent, and the group has stuck with this color scheme and the basics of the design since.



(a) Prototype version 1



(b) Prototype version 2



(c) Prototype version 2

Figure 3.4: Prototypes

3.5 User Experience

This section explains the methods used to create an application that considers usability, accessibility, and responsive design. It was essential to the product owner that the application was intuitive and responsive to be used on different devices. Both the customers and the restaurant employees should be able to use the application efficiently and satisfactorily.

3.5.1 Usability

The application was created with usability (2.3.1) considering how both the customers and employees would use it. The method for achieving this was to state what each element was for, so it could not be misunderstood.

3.5.2 Accessibility

To ensure an accessible application, Sim Daltonism (3.5.4) was used to find a balance for colorblindness, Don Norman's Design Principles (3.5.3) were followed, and the group ensured that the application was responsive for different devices (3.5.5).

3.5.3 Don Norman's Design Principles

This section describes how the design of the application follows Don Norman's design principles.

3.5.3.1 Visibility

To keep with the principle of visibility, each critical functionality of the page is visible to the user. On the desktop, all elements are visible and clearly labeled as they go through creating a reservation. There had to be a balance between visibility and a cluttered interface on smaller screens, so a hamburger menu was created to contain all the navigation links. On the admin page, the reservation table is scrollable in the x-direction for visibility on smaller screens. When a new reservation is added, and when a reservation is starting within one hour, the popups have a blinking header that is supposed to be visible for employees.

3.5.3.2 Feedback

When creating the front end of the application, the group used Angular Material, a UI component library for Angular applications. The components have built-in feedback when clicking on an element, like ripple effects and moving labels out of the way when clicking on an input field. On the navbar, the icons move slightly up when hovering over them. Validation of user input was implemented with colors and popups for users to know if and where they have not provided the correct information.

3.5.3.3 Constraints

When using the application, constraints are applied to guide the user by creating a reservation and simplifying it. When filling out contact information, validation of input stops the user from entering too many characters. Users can only select tables available at the chosen time and date, which fits the number of guests chosen.

3.5.3.4 Mapping

Throughout the application, icons are used to signify what an element is representing. For example, when going through the reservation process, the information chosen for the reservation is accompanied by icons that explain the text.

3.5.3.5 Consistency

The color theme was created based on the restaurants existing logo. By combining this with Angular Material, it creates a coherent and consistent theme.

3.5.3.6 Affordance

The application elements are created with consideration of giving the user a clue of how to use them. The inputs either show a label describing that the user should click it (click the date input to open the calendar), are underlined to indicate writing input or with arrows that signify a dropdown menu. The buttons are created to look like familiar buttons found anywhere else online. The navbar icons that link to different pages have a hover label, which tells the user where the link leads.

3.5.4 Sim Daltonism

Sim Daltonism is a color blindness simulation application that creates a real-time filter based on a color blindness algorithm (Fortin, 2021). This application was used to find a balance in contrast and color (3.5.6) that was both accessible and pleasant to look at for both colors blind and non-color-blind people.

3.5.5 Responsive Design

To get a responsive design, media queries were used. Media queries dictates what CSS properties are being used based on display width and height. On the table-GUI page, one team member had a really great idea. This page includes many details that have to be correct. The idea was to create an SVG file of the framed tables and scale the picture for different devices.

3.5.6 Colors

This subsection explains the usage of colors and the color choice itself.

3.5.6.1 Green

The green color used is taken from Cinque Minuti´s logo. It is used on the buttons and the navbar. After the first meeting, the client liked the choice of colors and wanted to keep them.

3.5.6.2 Burgundy

The burgundy color is also from the logo. The usage for this color is only used for the "Covid-19 Precautions"-dropdown.

3.5.6.3 White

The icons in the NAV bar and the text inside the buttons are white.

3.5.6.4 Brown

The brown color used is defined as the standard available table color. When a table is available, the color is set to this brown color.

3.5.6.5 Deep red

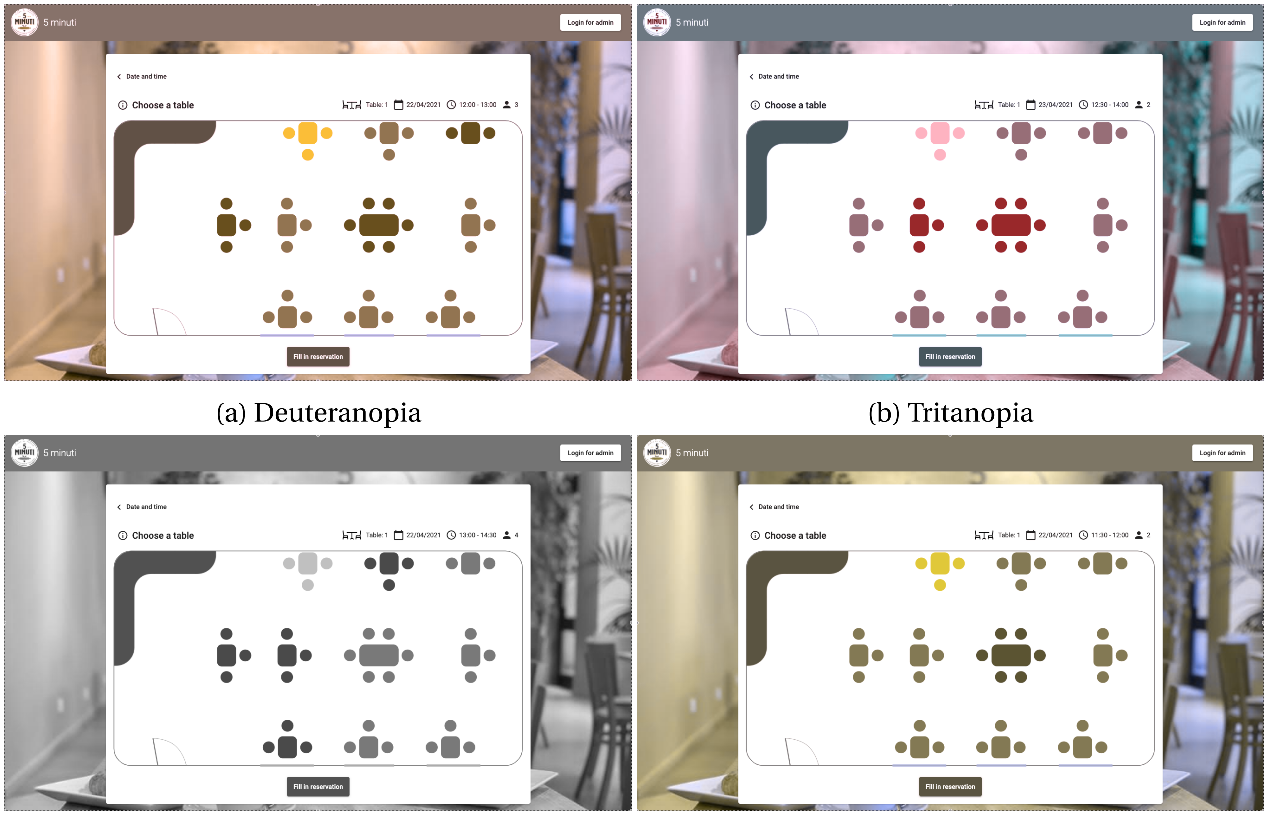
If a table is unavailable, or any of the "table overview"-rules apply, the color changes to a deep red.

3.5.6.6 Yellow

The yellow color is defined as currently selected. When a user clicks on an available table, it changes color from brown to yellow, signifying that a table is selected.

3.5.6.7 Colorblindness friendly

All the colors used in the application are colorblindness friendly (Figure 3.5). That means the difference in available tables, unavailable tables, and currently selected table colors can be seen even with colorblindness.



(a) Deuteranopia (b) Tritanopia

(c) Monochromacy (d) Protanopia

Figure 3.5: Color Blindness Types

3.6 Tools and Technologies

This section describes the tools and technologies used in developing the application.

3.6.1 Development Environments

This subsection describes the environments used in developing the application.

3.6.1.1 Visual Studio Code

Visual Studio Code is a free and open-source code editor. During the first half of the project, it was used but switched to JetBrains WebStorm after finding out that Visual Studio Code did not have language support for TypeScript without any specific plugins.

3.6.1.2 JetBrains WebStorm

WebStorm is an integrated development environment (2.4.11) created for web development. It has support for JavaScript, TypeScript, Angular, Node.js, and more. It includes features and tools that the group found helpful, such as code completion, error detection, Git integration, linters such as TSLint and ESLint, test runners, terminal, HTTP client, and plugins. (JetBrains, 2021)

3.6.2 Version Control

This subsection explains the tools used for version control (2.4.10). Version control was used for collaboration, tracking, and managing the codebase.

3.6.2.1 Git

Since the group developed on the same project, a Version Control System was needed. Git is a distributed version control system (2.4.10.1) and the most used VCS, followed by Subversion (Synopsys, 2021). The group chose to use Git because of their experience with using it. Git is also supported and integrated with IDEs and Git clients like Sourcetree (3.6.2.3).

3.6.2.2 GitHub

GitHub is a web-based platform for hosting and sharing Git repositories. The group used it for hosting and collaborating on the project's source code. It also allowed for the supervisor to get access to the repository.

3.6.2.3 Sourcetree

Sourcetree is a free Git client that offers a visual representation of Git repositories. It is simple and easy to use if lacking experience with the Git CLI and powerful for more experienced users (Atlassian, 2021).

3.6.3 Postman

Postman is often used to check End-to-End points. It is used either before a back end is created or for debugging a project. Then postman can be used to check the requests sent and check the server response. Different HTTP-status codes will also be received.

3.6.4 PuTTY

PuTTY is an open-source terminal (*PuTTY* 2021). The group used PuTTY to connect to the virtual machine (VM) on the university's autodeploy. After setting up the Ubuntu server with Nginx, the application was ready for deployment so the group could clone the project.

After using the git clone to get the project on the server, then the group members can follow the Read me-file that follows the project. Then it is ready for installing the dependencies with the command "npm install," which changed the environment from localhost to HTTPS. In the angular. JSON-file the host is set to "0.0.0.0," - which accepts all IPs. Furthermore, the port is 443 since the application is using HTTPS. After that, the application was to production build, move the distribution files to /var/www/HTML on the server, and the front end was up. Navigated back to the project inside the back end folder and wrote "node app.js" to deploy the back end. All this happened through the terminal window on the VM.

3.7 Project Architecture

This section explains the architecture of the project. The system got two main folders: Front end and back end, and lots of different subfolders underneath them.

There are several ways to have structured the project. One example of this is creating a monolithic web application. A monolithic web application is built as a single unit. The application has three parts: A database, a client-side user interface, and a server-side application. This server-side application will handle HTTP requests, execute logic and read/write data to the database. Because it does everything, it is a monolith, a single logical executable. This sounds practical but has the downside an admin wants to change anything in the system, a developer must build and deploy an updated version. This would make it hard to reuse with other back ends or find a big bug that encapsulates many different components. Another downside is complexity. If the application gets big enough, few developers might know the ins and out of the entire system (Gnatyk, 2021).

For this system, it was decided to go with a microservice solution. The application ended up having three primary services. The three main services are the main application, the authGuard, and the nodemailer. These services can be modified and exported but would need some configuration to be used in other projects.

3.7.1 Front end

The front end is often explained as what is visible to the customer. This section describes the HTML, CSS, TypeScript, and some JavaScript files.

3.7.1.1 Angular

Is a powerful framework that the group used as importing stuff from the Angular Material, such as the calendar. Instead of creating a calendar from northing, the group could just import it was a datepicker. Angular consists of components, routing and modules.

3.7.2 Back end

The back ends it what is not visible for the customer to see. Here are things such as DB setup and connection, authentication, models of a reservation schema, etc. Most of the files in the back end are written in JavaScript.

3.7.2.1 HTTP Protocol

In this subsection, the group will talk about the general basics of the HTTP protocol. HTTP is a protocol for sending an receiving data in the application layer of the OSI Model. The HTTP protocol is designed to be simple. HTTP messages can be read and understood by humans; this makes it easier for the developer and reduces complexity. HTTP is also stateless, which means that it is not guaranteed to respond if a user send a request. If a user want specific information, then a GET request will have to be sent and accepted before communication can begin. Down below are the four most common HTTP requests.

GET
The HTTP GET: Getting data.

POST
The HTTP POST: When creating data.

PUT
The HTTP PUT: To update data.

DELETE
The HTTP DELETE: Deleting data

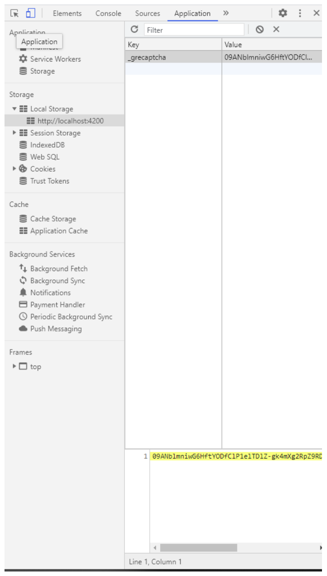
3.7.2.2 Query Parameters

Defines a part of the URL into the browser. Everyone can use query parameters to pass data from one page to another. So in our system, when a user insert how many guests, date, time, and time on the front page, it set the query parameters to move further to the table-GUI page,

or the table selection page. Then when the user select a table, it will also store that data into the URL to bring all the set data to the confirmation page. So the system can get the different parameters the user has already set by getting them from the URL. So when the user are on the confirmation page, the URL would be like
http://localhost:4200/confirm-reservation?date=2021/05/19&fromTime=13:00&toTime=16:00&guests=2&tableSelected=2

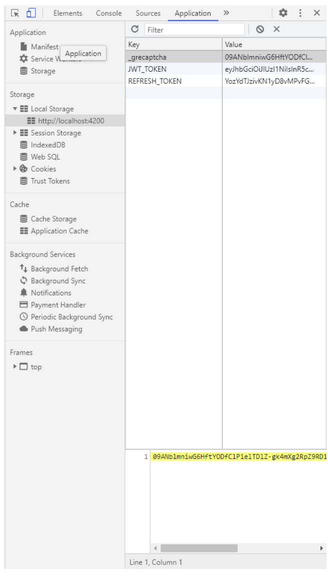
3.7.3 JSON Web Token

As mention earlier in this section, only the server knows the correct password. In that way, the server will generate a token, either its correct or wrong password. However, only the correct token will give access to the route (Figure 3.6). The wrong token will only be for some kind of response as "wrong username and/or password".



Key	Value
_grecaptcha	09ANblmniwG6HftYODfCl...

(a) Without JWT (Before login)



Key	Value
_grecaptcha	09ANblmniwG6HftYODfCl...
JWT_TOKEN	eyJhbGciOiJIUzI1NiIsInR5c...
REFRESH_TOKEN	YozYdTJzivKN1yD8vMPvFG...

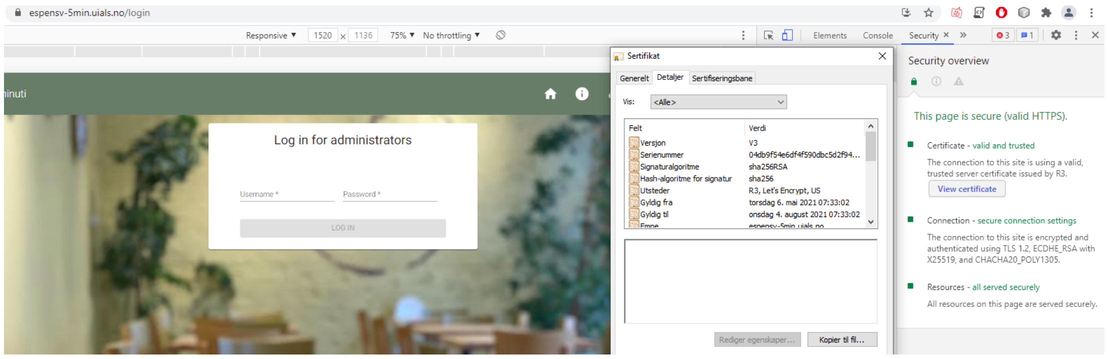
(b) With JWT (After login)

Figure 3.6: JWT

3.7.4 HTTPS

HTTPS makes the communication over the network secure. That is whats the "S" at the end of HTTP means. When adding HTTPS to a server, the protocol is encrypted using Secure Sockets

Layer (SSL). When a website uses SSL, it grants the browser a certificate, and the web page will now be recognized as a trusted web page (Figure 3.7).



Vis: <Alle>

Felt	Verdi
Versjon	V3
Serienummer	04db9f54e6df4f590dbc5d2f94…
Signaturalgoritme	sha256RSA
Hash-algoritme for signatur	sha256
Utsteder	R3, Let's Encrypt, US
Gyldig fra	torsdag 6. mai 2021 07:33:02
Gyldig til	onsdag 4. august 2021 07:33:02
Emne	espensv-5min.uials.no

Security overview

This page is secure (valid HTTPS).

Certificate - valid and trusted

The connection to this site is using a valid, trusted server certificate issued by R3.

Connection - secure connection settings

The connection to this site is encrypted and authenticated using TLS 1.2, ECDHE_RSA with X25519, and CHACHA20_POLY1305.

Resources - all served securely

All resources on this page are served securely.

Figure 3.7: Details certificate for HTTPS

3.7.5 Hash and salt

When hashing a password, it is harder for the "man in the middle" to find a password. The hash of a password happens in the back end on the server-side. When the server hashes the password, it generates a long string that will not be readable for the human eye. If the password were "YouCantReadThis," then the hashing would be
"2868e7b421349a70ded1edafdf3fff86".

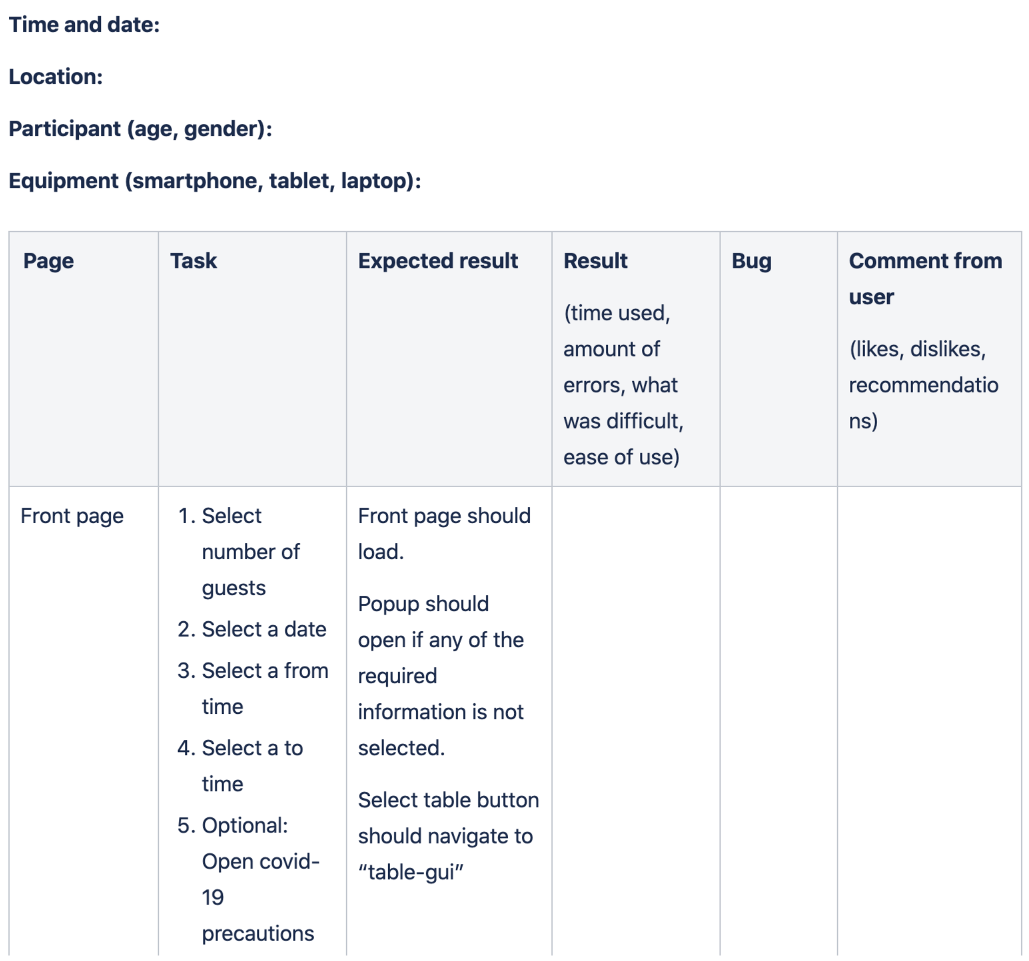
However, with the "just" pure hash of the password, the "man in the middle" can use the MD5 converter to string to unhash the admin or the user's password - if he finds the syntax for where the password is. To make it even harder for the "man in the middle"- attack, salt is added to the password. When salt is added to the hash, it randomly ads bits into the hashed password - so when the guy tries to unhash the password, it will be converted to something different because of the salt added. When salt is added to the password mention earlier, it will get
"e5e61b31b8f1ea41b560d271c782ac7fc352bea0fe975410c8cbaf551a415ce7"
- which again are longer and harder to convert to a plain string again compare to a pure hashed password. If the "man in the middle" are trying to use the MD5 converter to string on a hashed and salted password, the string will still be unreadable because the salt is added will not be converted. This gives a much longer decryption time and ends up making the system more secure.

3.8 Testing

When creating a enterprise project, or any serious project, testing has to be done. Testing for this project consists of two categories, usability testing and automated testing. Usability testing is testing how well the system is received and what feedback the users has to it. Automated testing tries to run the system and checks if certain conditions are met. One example of a automated testing condition is checking if a button is pressable, and what action happens after the button press. Both the usability test and the automated test are crucial for creating robust software.

3.8.1 Usability testing

For usability testing (2.7.3), the group created a template (Figure 3.8) to guide the test and measure results. The results can be found in chapter 4 (4.4.2).



Time and date:

Location:

Participant (age, gender):

Equipment (smartphone, tablet, laptop):

Page	**Task**	**Expected result**	**Result** (time used, amount of errors, what was difficult, ease of use)	**Bug**	**Comment from user** (likes, dislikes, recommendatio ns)
Front page	1. Select number of guests 2. Select a date 3. Select a from time 4. Select a to time 5. Optional: Open covid-19 precautions	Front page should load. Popup should open if any of the required information is not selected. Select table button should navigate to “table-gui”			

Figure 3.8: Usability test template

3.8.2 Ghost Inspector

After testing different solutions for testing, the team chose Ghost Inspector.

"Ghost Inspector is an automated website testing and monitoring service. It carries out operations in a browser, the same way a user would ensure that everything is working properly" (*Ghost Inspector* 2021).

Tests were made for every component and the entire process. The results of these tests can be found in section 4.4 and appendix G.

3.8.3 Lighthouse

"Lighthouse is an open-source, automated tool for improving the quality of web pages. Everyone can run it against any web page, public, or requiring authentication. It has audits for performance, accessibility, progressive web apps, SEO, and more. Lighthouse can run in Chrome DevTools, from the command line or as a Node module. Lighthouse needs a URL to audit, it runs a series of audits against the page, and then it generates a report on how well the page did. From there, use the failing audits as indicators on how to improve the page. Each audit has a reference doc explaining why the audit is essential and how to fix it" (*Lighthouse* 2021)

Chapter 4

Result

This chapter will be the project results and how the group implemented the theory and methodology to solve the problem. This chapter consists of four parts, design, functionality, architecture, and testing. The design section consists of how and why the team has designed the system. The functionality shows how the existing system looks and how it is used. The architecture gives an overview of the group's technologies and how the different software components work together. Lastly, the final sub-chapter will be about the testing, both as in software automated testing and regular user testing, where users try the system to give feedback.

4.1 Design

The Cinque Minuti's logo inspired the design and color scheme of the application. The page background, which changes based on screen size, consists of two images from the restaurant. The user interface was created with Angular Material, a UI component library. These components were styled to fit in with the theme of the web application. The group also used Material Icons, which integrates easily with Angular applications.

4.1.1 Color

As mentioned in the method chapter, the group has chosen the colors based on the Cinque Minuti's logo. The group did not receive instructions on color, so the group decided on the color palette in agreement with the client. The colors were minor adjusted for colorblindness (Section 3.5.4 and Figure 3.5).

4.2 Functionality

This section will show an overview of the possibilities that exist in the complete system. This includes different diagrams, screenshots of every page and subpage while the group is explaining the results.

4.2.1 Overview

The most central part of the system is the reservation mechanic, and it is essential to understand the process of booking a reservation.

4.2.1.1 UML Use Case Diagram

This use case (Figure 4.1) diagram shows what a regular customer can do and what an admin can do. The customer can, of course, visit all the pages in the navbar. The user can make a reservation that includes the date, time, group size, selecting table, confirm the reservation by filling in the name, phone number, and email, and receive an email about the reservation. When it comes to an admin, he can do all the previously mentioned things. He can also log in, log out and see the reservation administration. Inside reservation administration, he can view all reservations, view today's reservations, delete a reservation, or set a reservation as prepared or not prepared. He will receive notifications inside the administration panel when a new reservation comes in or one hour until reservation. So they have time to prepare the table if it has not been done yet.

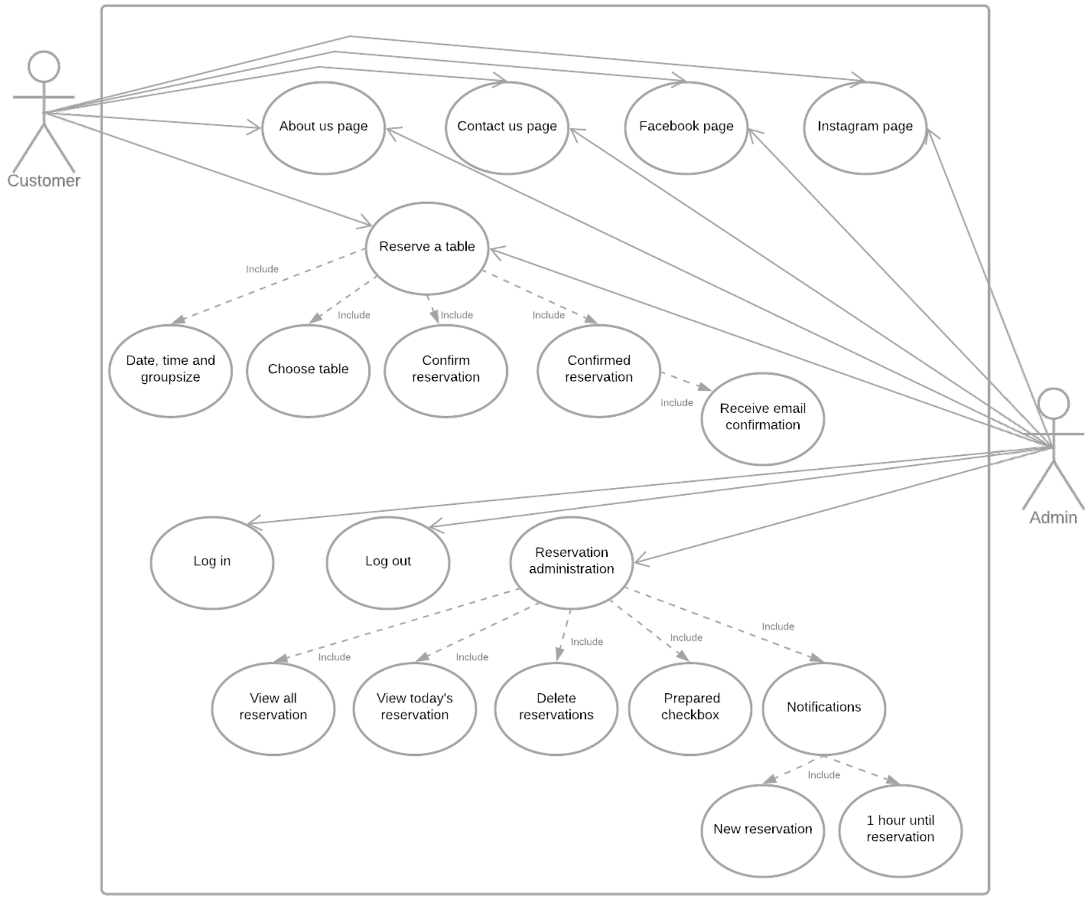


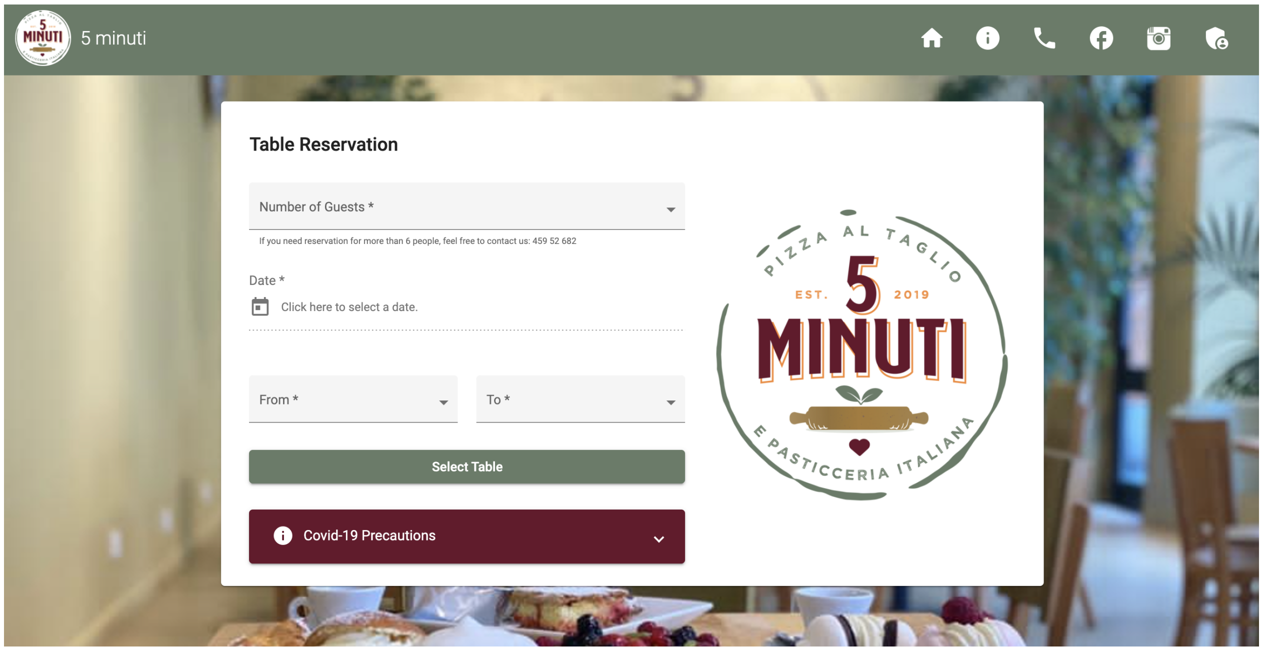
Figure 4.1: UML Use Case Diagram

4.2.2 Regular User Access

This system has two main groups that will use it. These two main groups are regular users and an admin. Regular users can access most features, like creating a reservation or checking the about us and contact us page. This sub-chapter will be about what a normal user can see and use and how the experience is intended for them.

4.2.2.1 Homepage

The homepage is the first part of the system for creating a reservation. The user chooses how many guests they are, the date for the reservation, and when the reservation begins and ends. The homepage (Figure 4.2) content consists of a navbar and the reservation form. The navbar has the restaurant's logo and different icons that the user can use for navigation. The reservation form is where the customer chooses how many people and when they want to reserve a table.



If you need reservation for more than 6 people, feel free to contact us: 459 52 682

Figure 4.2: The Home page

Navbar

The navbar consists of two main parts, a logo with a brand name and navigation buttons. The logo is the Cinque Minuti's logo, where if clicked on (or the text next to it), the user returns to the home page. The second part of the navbar is the navigation buttons. The navigation buttons are the homepage button, the about us button, the contact us button, the go-to Cinque Minuti's Facebook button, the go-to Cinque Minuti's Instagram button, and an admin access button. These have the function that their name implies and redirects to the different parts of the system corresponding to their name.

Select Guests

The select guest is the first part where the user starts to fill in their reservation. In this field, the user chooses how many guests they want for their reservation. This can anywhere from one to six guests. A small caption under selecting guests says, "If you need a reservation for more than six people, feel free to contact us: 459 52 682". This signals to the customer that it is possible to reserve for more than six people, even if it is not an option in this section of the reservation. The standard for six people was set when the requirement specification was written and for practical purposes for how many can sit at a table.

Calendar

There is also a calendar option (Figure 4.3a) where the user can see a calendar and pick a date for the reservation. Mondays are disabled, as the store is closed on those days.

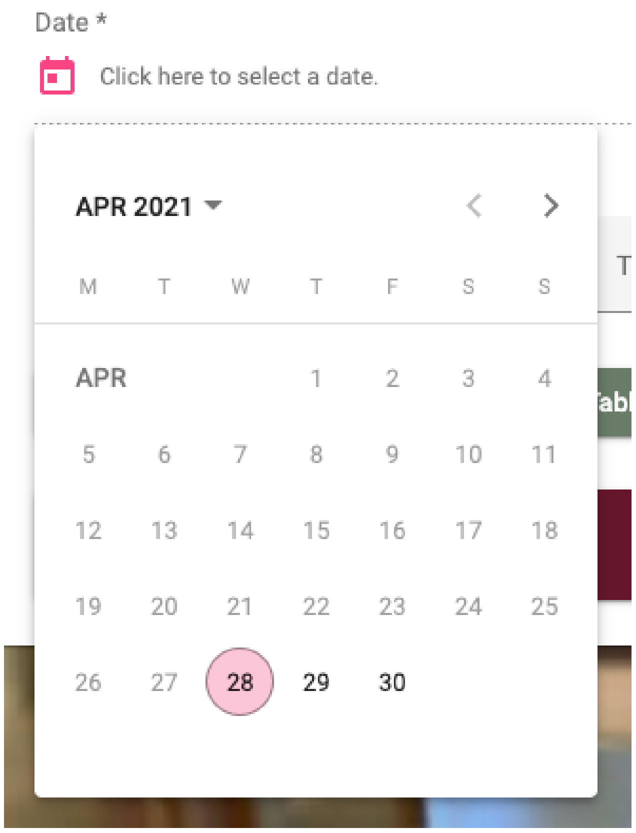
Select Time

There is also a form input the user can choose a from time for the user's reservation, and a to-time input the user can choose an input for the to times. This input is in half-hour intervals and usually starts at 11:00 and closes at 19:00, except on Fridays and Saturdays when it closes at 22:00. When selecting a from and to time (Figure 4.3b), the user can see how many tables are available at the chosen time.

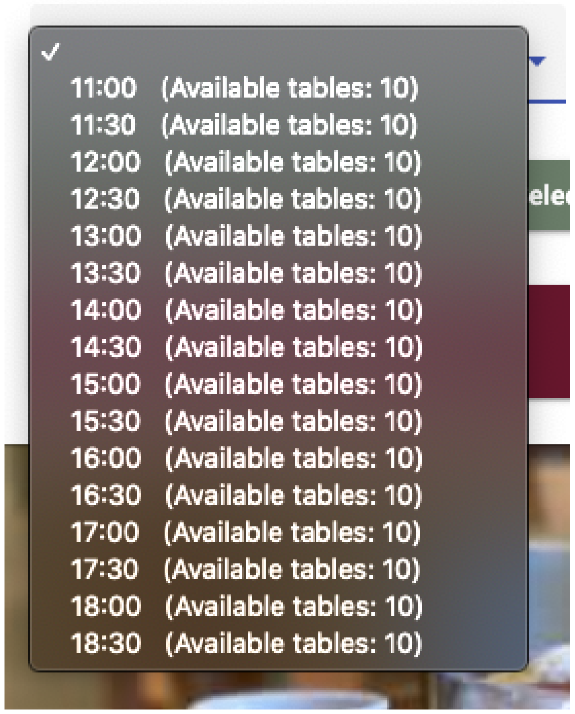
Covid-19 Precautions

Below is a Covid-19 precaution dropdown (Figure 4.3c) where the user can see more information about the COVID-19 rules and restrictions in the restaurant. This was created to show the customers that the restaurant takes Covid-19 guidelines seriously. This button is burgundy, so the users will see that it is crucial and signifies it is dangerous or wary.

Background The background is a shot of the bakery section of the counter and helps give the user a better feel of what the restaurants have to offer.

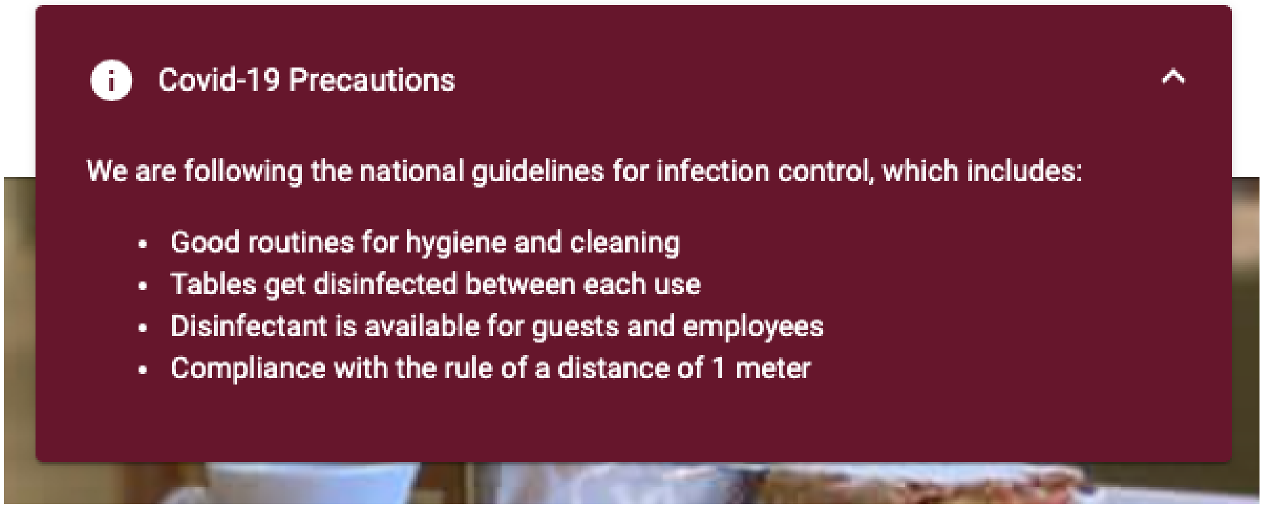


(a) Calendar



```
✓
11:00  (Available tables: 10)
11:30  (Available tables: 10)
12:00  (Available tables: 10)
12:30  (Available tables: 10)
13:00  (Available tables: 10)
13:30  (Available tables: 10)
14:00  (Available tables: 10)
14:30  (Available tables: 10)
15:00  (Available tables: 10)
15:30  (Available tables: 10)
16:00  (Available tables: 10)
16:30  (Available tables: 10)
17:00  (Available tables: 10)
17:30  (Available tables: 10)
18:00  (Available tables: 10)
18:30  (Available tables: 10)
```

(b) From time



Covid-19 Precautions

We are following the national guidelines for infection control, which includes:

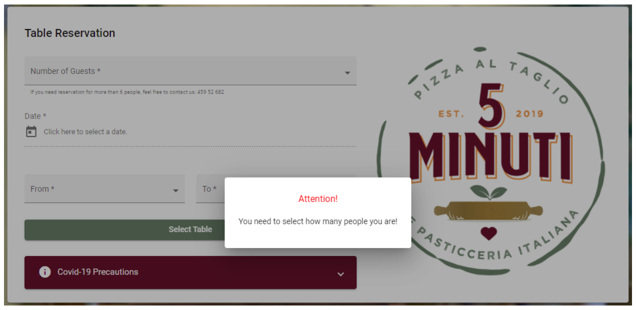
- Good routines for hygiene and cleaning
- Tables get disinfected between each use
- Disinfectant is available for guests and employees
- Compliance with the rule of a distance of 1 meter

(c) Covid precautions

Figure 4.3: Home page elements

Missing requirements

The number of guests, date, from- and to-time are required. Users cannot progress without filling them. When the user tries to progress without several guests, the user will warn about needing to choose several guests (Figure 4.4a). If the user has not filled in the date, the user gets a warning about needing to fill out a date (Figure 4.4b). If the user has just filled in the guest's amount and date, but not the time, the user gets a warning about choosing a from time(Figure 4.4c). Furthermore, if the user has not entered the time, the user gets a warning about choosing a time (Figure 4.4d).

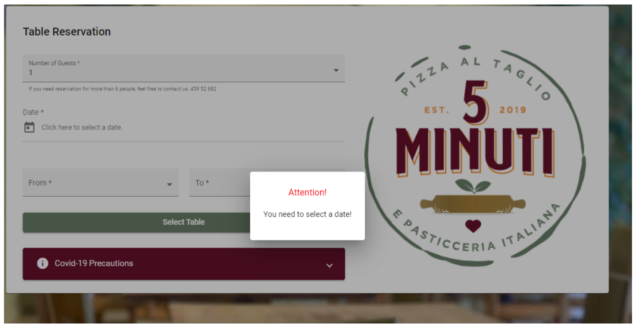


If you need reservation for more than 6 people, feel free to contact us: 459 52 682

Attention!

You need to select how many people you are!

(a) Popup: Guests amount not filled

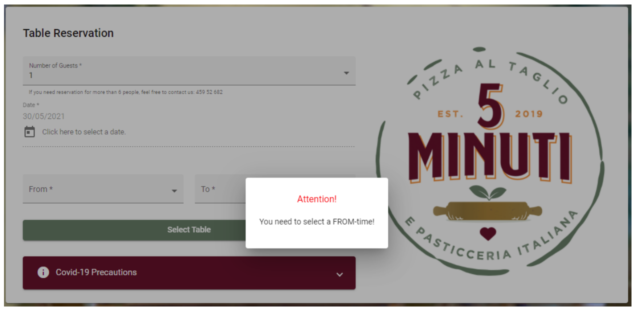


If you need reservation for more than 6 people, feel free to contact us: 459 52 682

Attention!

You need to select a date!

(b) Popup: Date is not filled

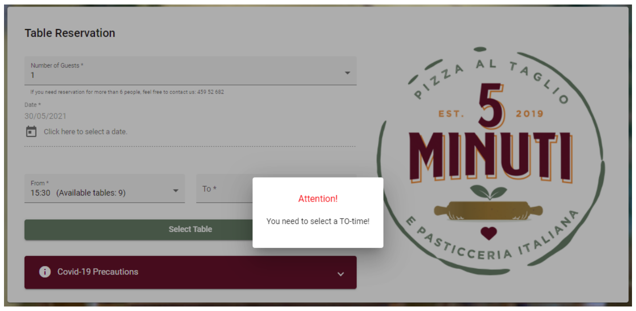


If you need reservation for more than 6 people, feel free to contact us: 459 52 682

Attention!

You need to select a FROM-time!

(c) Popup: From-time is not filled



If you need reservation for more than 6 people, feel free to contact us: 459 52 682

Attention!

You need to select a TO-time!

(d) Popup: To-time is not filled

Figure 4.4: Missing requirements

4.2.2.2 Table Overview

This is the part of the system where the guest gets an overview of the restaurant and can choose where they want to sit. The table overview consists of a navbar, back button, information tab, and several icons such as the table, calendar, clock, and guest icon where the values for the reservations are shown (Figure 4.5). The date and time button takes the user back to the home-page.

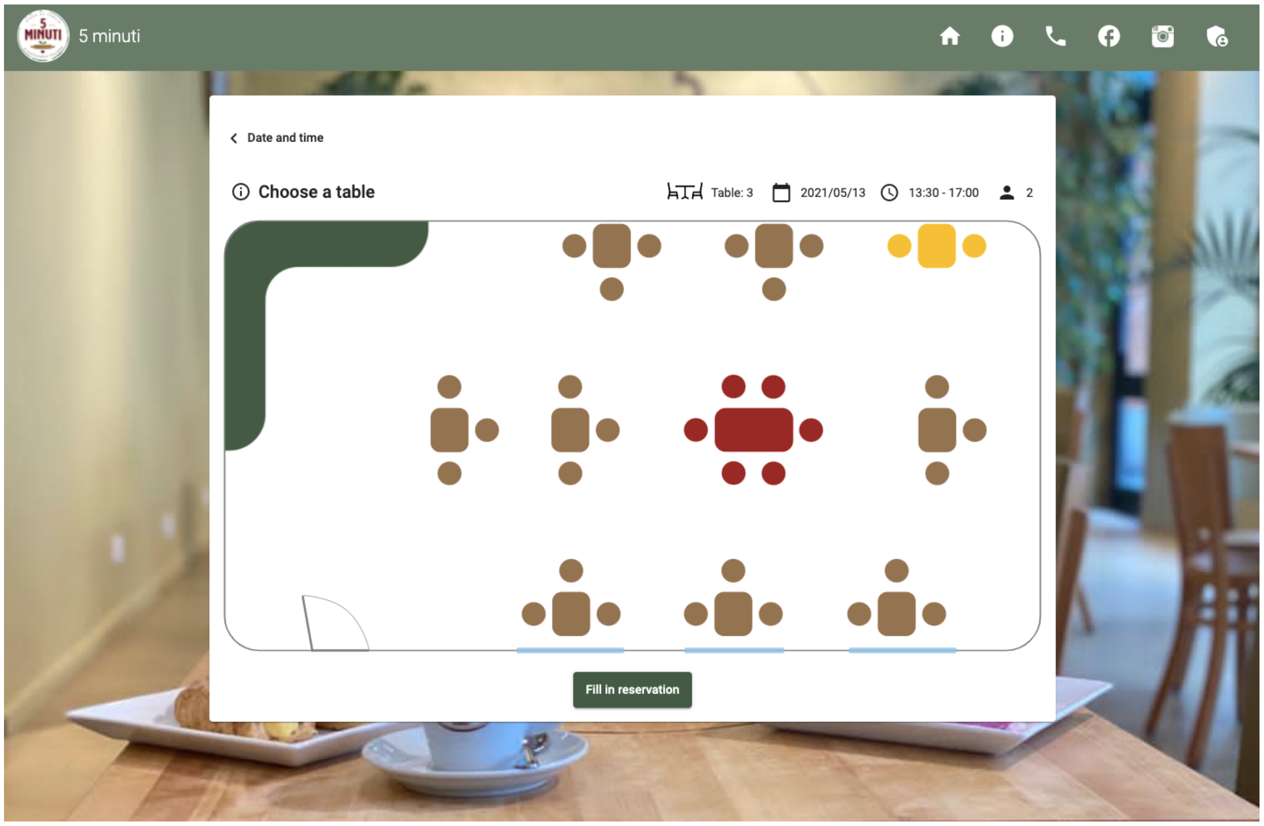
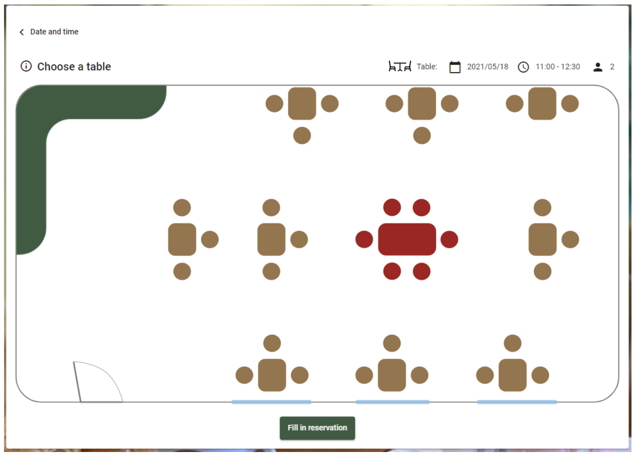
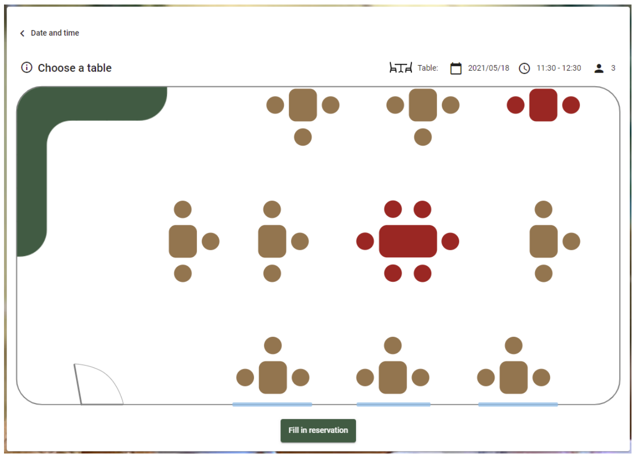


Figure 4.5: Table Overview

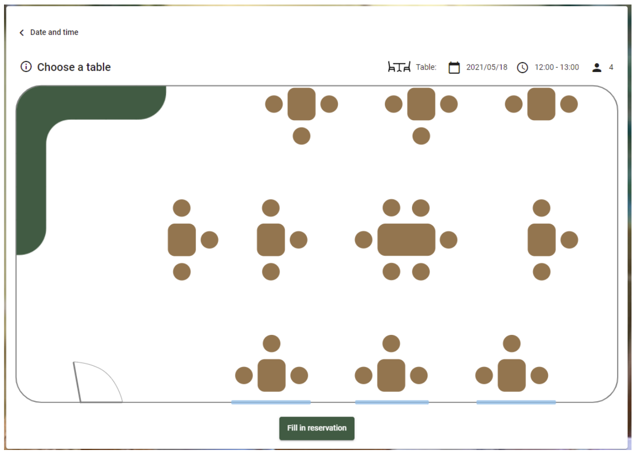
This page has a complex set of rules depending on the input: If a user has chosen a specific date and time, it checks with the database whats the status on the different tables - in other words, if the tables have been reserved or not. If the group contains two people or fewer, they can sit at every table except the big table (Figure 4.6a). If the group contains three people, then the user can choose every table except the two-person table in the corner and the big table (Figure 4.6b). Furthermore, if the group contains four or five people, then the user can sit everywhere. The user can either select the big table, and two regular tables can be selected (Figure 4.6c). Furthermore, the last one, if the group contains six people, the user can sit everywhere except the two-person table in the corner. Because of the table set-up and only one big table, two regular tables are selectable. However, with this amount of guests, a two-person table will be needed (Figure 4.6d). This happens after the DB has checked and made sure the table is available.



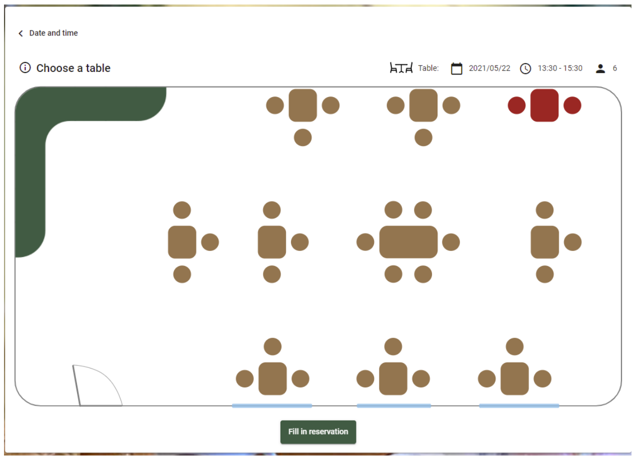
(a) Group contains two people



(b) Group contains three people



(c) Group contains four people



(d) Group contains six people

Figure 4.6: Set rules of selecting table

Reservation Information Line

Over the table overview, a reservation information line includes five informative icons—the first icon in the table overview help icon. The second icon is the table icon. The table icon shows which table the user has chosen. The third icon is the calendar icon. The calendar icon shows which date the user has chosen. The fourth icon is the from and to time icon. The from and to time show what time frame the customers are reserving a table for. Lastly, the fifth icon is the guest icon. The guest icon shows how many guests the user has chosen to reserve a table for.

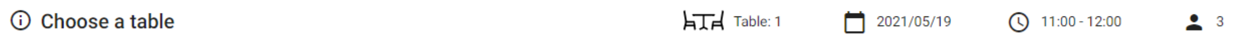


Figure 4.7: Reservation information line

Table Overview Help

An information tab (Figure 4.8) shows the restaurant's layout with the counter, the door where the iPad is standing, where the toilet is, and the fire exit. The tables on this overview are numbered so that customers can understand the numeration of tables. There is also an overview of the color scheme where it says what type of color means what, brown for available, yellow for selected, and red for unavailable.

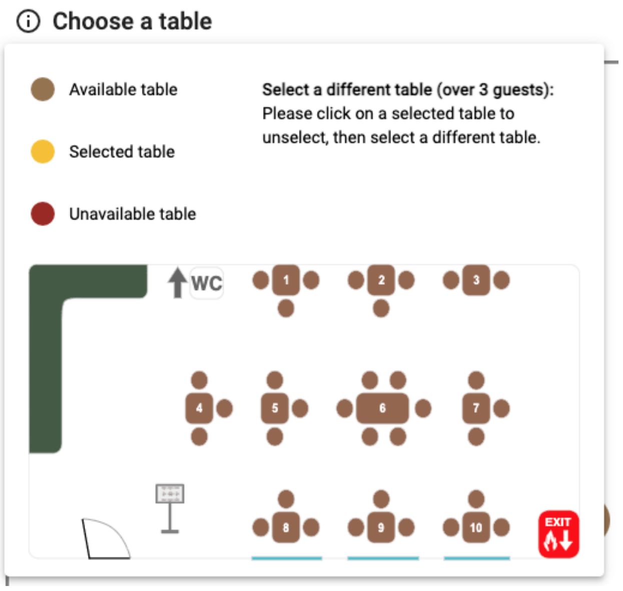

Choose a table
Available table
Selected table
Unavailable table
Select a different table (over 3 guests):
Please click on a selected table to unselect, then select a different table.
WC
1
2
3
4
5
6
7
8
9
10
EXIT

Figure 4.8: Table Overview Help

Table Overview

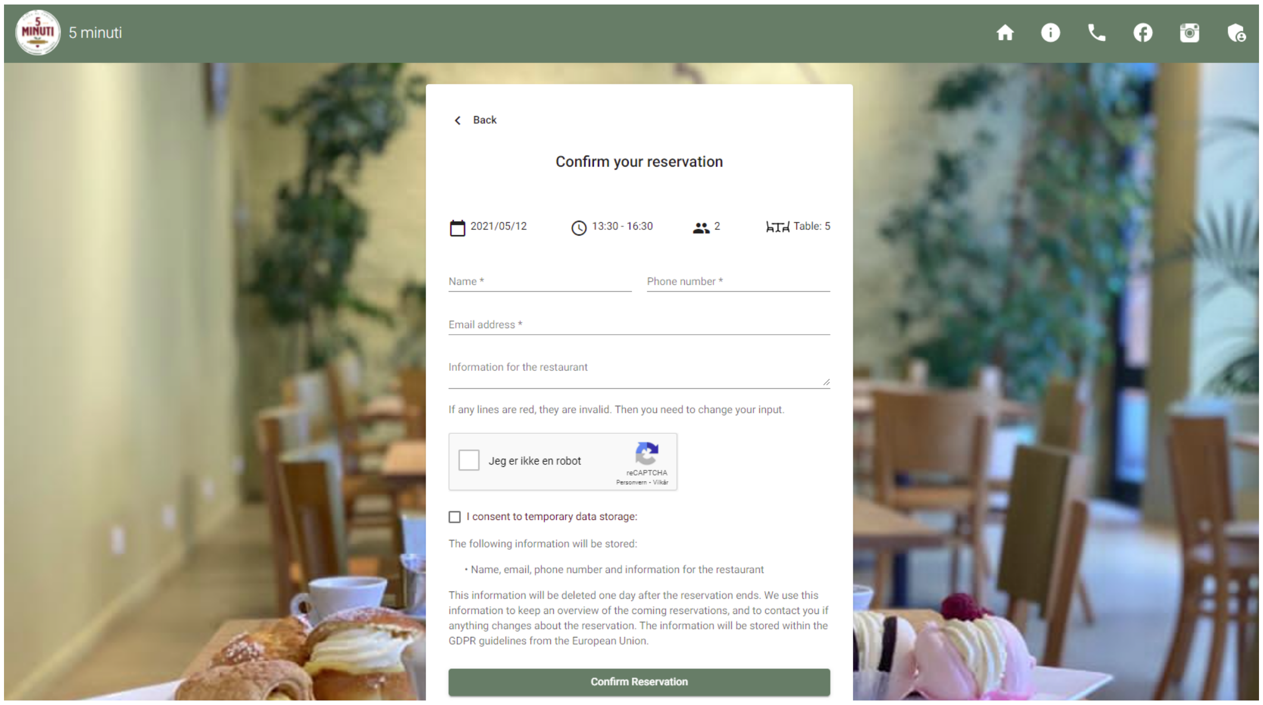
Below the information and the icons are the graphical part of the table booking system (Figure 4.5). It shows the user an overview of the counter and the available tables for the user to reserve. It also shows a door, and windows, so the user knows how the restaurant is formed from someone entering.

If a table is not picked and the user presses an available table, it will be selected. If the user has selected three guests, the user cannot choose table number 3, which has two seats, because that is too few seats for three guests. Likewise, with table number 6, the user cannot pick table number 6 because there are too many seats, and it would be unfair to the other guests.

If the user has selected more than three guests, the user can reserve two tables and then combine them. It will then show that the user has chosen a table, for example, Table 1 and Table 2, for the total reservation. Below the table overview, a fill-in reservation button takes the selected they will and lets them go further into the booking progress.

4.2.2.3 Confirm Reservation

Confirm reservation (Figure 4.9) consists of the navbar, a back button, a confirm reservation header, and four icons at the top of the page's layout.


5 minuti

Back

Confirm your reservation

2021/05/12 13:30 - 16:30 2 Table: 5

Name *

Phone number *

Email address *

Information for the restaurant

If any lines are red, they are invalid. Then you need to change your input.

Jeg er ikke en robot

reCAPTCHA
Personvern - Vilkår

I consent to temporary data storage:

The following information will be stored:

- Name, email, phone number and information for the restaurant

This information will be deleted one day after the reservation ends. We use this information to keep an overview of the coming reservations, and to contact you if anything changes about the reservation. The information will be stored within the GDPR guidelines from the European Union.

Confirm Reservation

Figure 4.9: Confirm Reservation

Icons

These four icons show the user what they have chosen for their reservation. These are described in paragraph4.2.2.2.

User contact information

The confirm reservation then consists of an input field to input the name, phone number, email, and specific information for the restaurant.

reCaptcha

The user is asked to check a reCaptcha to hinder hacking and brute force attacks.

Consent and confirm button

As the last step of the reservation, the user has to consent to temporary data storage. This part contains information about the GDPR, what data is stored, how long, and what the data is used for. Lastly, the confirm reservation button will hold the reservation in the database.

4.2.2.4 Reservation confirmed

The confirmed reservation page (Figure 4.10) ends the reservation process. It informs the user about what email address the email confirmation is sent to and contains an animated SVG that triggers left to right using an SVG border. This gives a feeling of the signature being written, giving the page a little more personal touch.

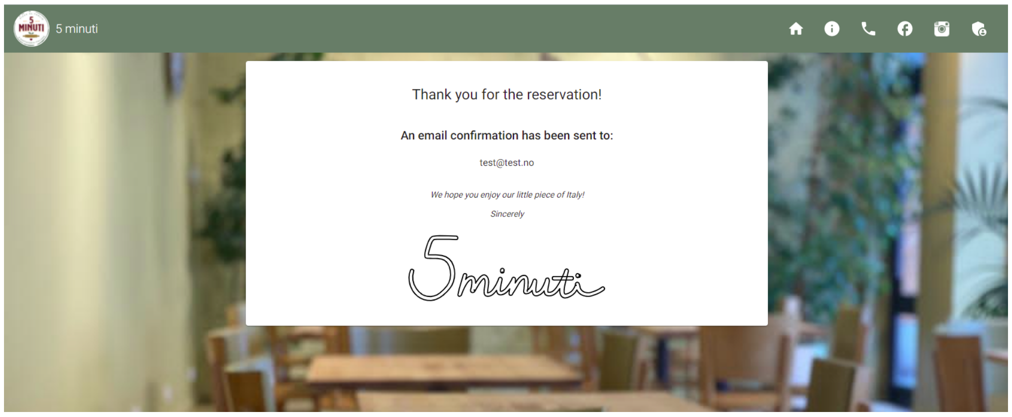
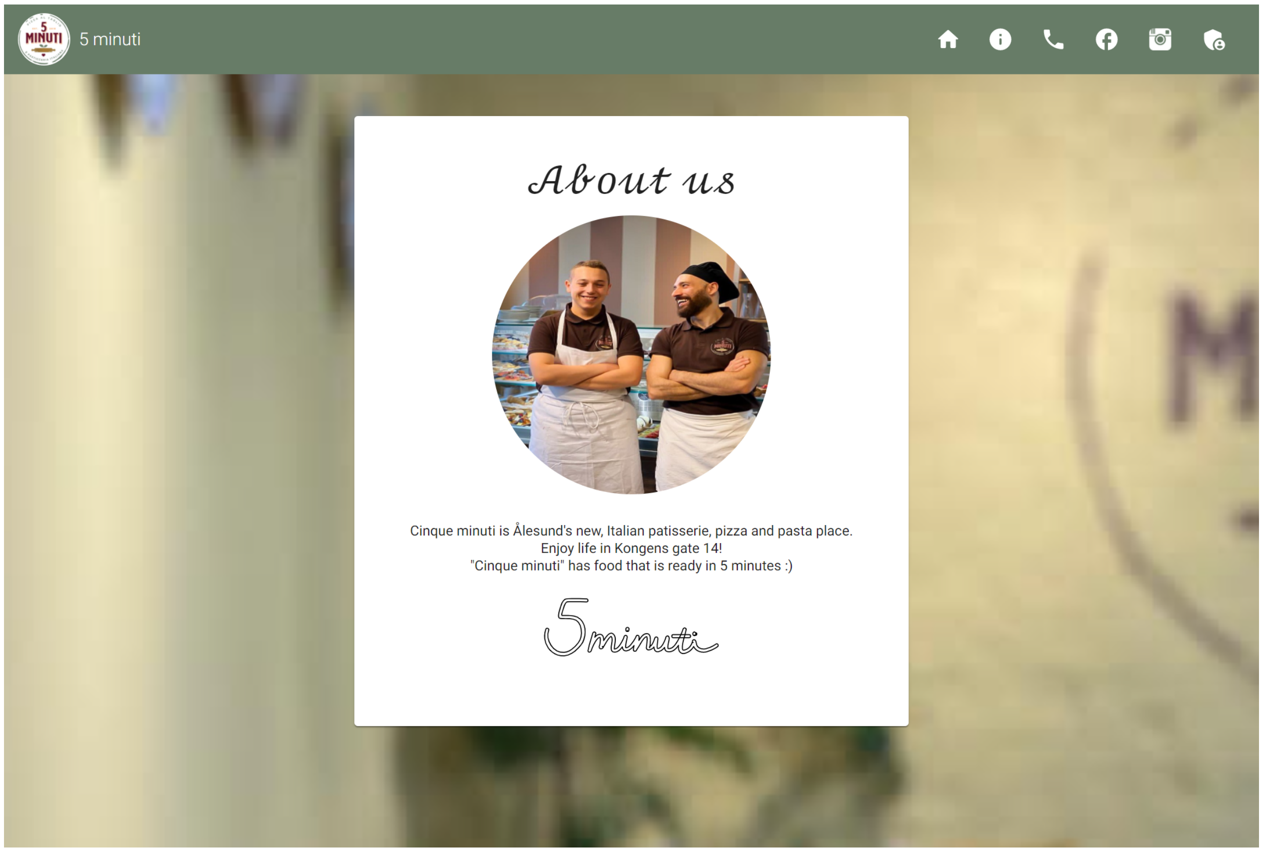

5 minuti
Thank you for the reservation!
An email confirmation has been sent to:
test@test.no
We hope you enjoy our little piece of Italy!
Sincerely
5minuti

Figure 4.10: Reservation Confirmed

4.2.2.5 About us

The about us page (Figure 4.11) tells the customer about the restaurant. The page tells what the restaurant wants to give an impression of, what they stand for, and what services they can provide. The font of the headline was chosen to provide more of a classy Italian look, and the picture was selected to show a sign of comfort and a pleasant atmosphere. There is also text about what the restaurant has to offer to make the customer want to visit. At the end of the about us page, there is an animated SVG signature. This animated SVG signature is the same used on the "confirmed reservation"-page. The signature is designed to give the impression that these are their values and that they can sign them because the restaurant believes in them.

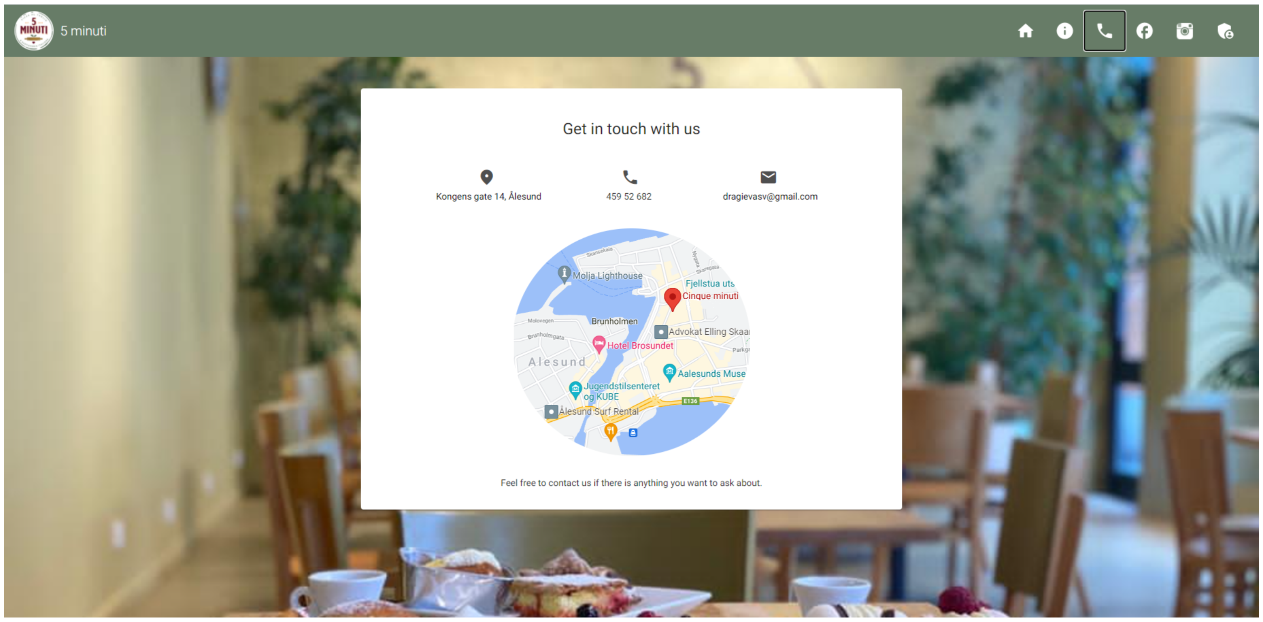


Cinque minuti is Ålesund's new, Italian patisserie, pizza and pasta place.
Enjoy life in Kongens gate 14!
"Cinque minuti" has food that is ready in 5 minutes :)

Figure 4.11: About us page

4.2.2.6 Contact us

The contact us page (Figure 4.12) gives the user information on how to contact the restaurant. It has the restaurant's location, the phone number of the restaurant, and their business email. It also has a circular picture of a map signifying where the restaurant is located. The address and the image have a link to google maps. Lastly, it has a text that says, "Feel free to contact us if you want to ask us anything." This text shows that they are approachable, and contact them if they have any inquiries.



Get in touch with us

Kongens gate 14, Ålesund

459 52 682

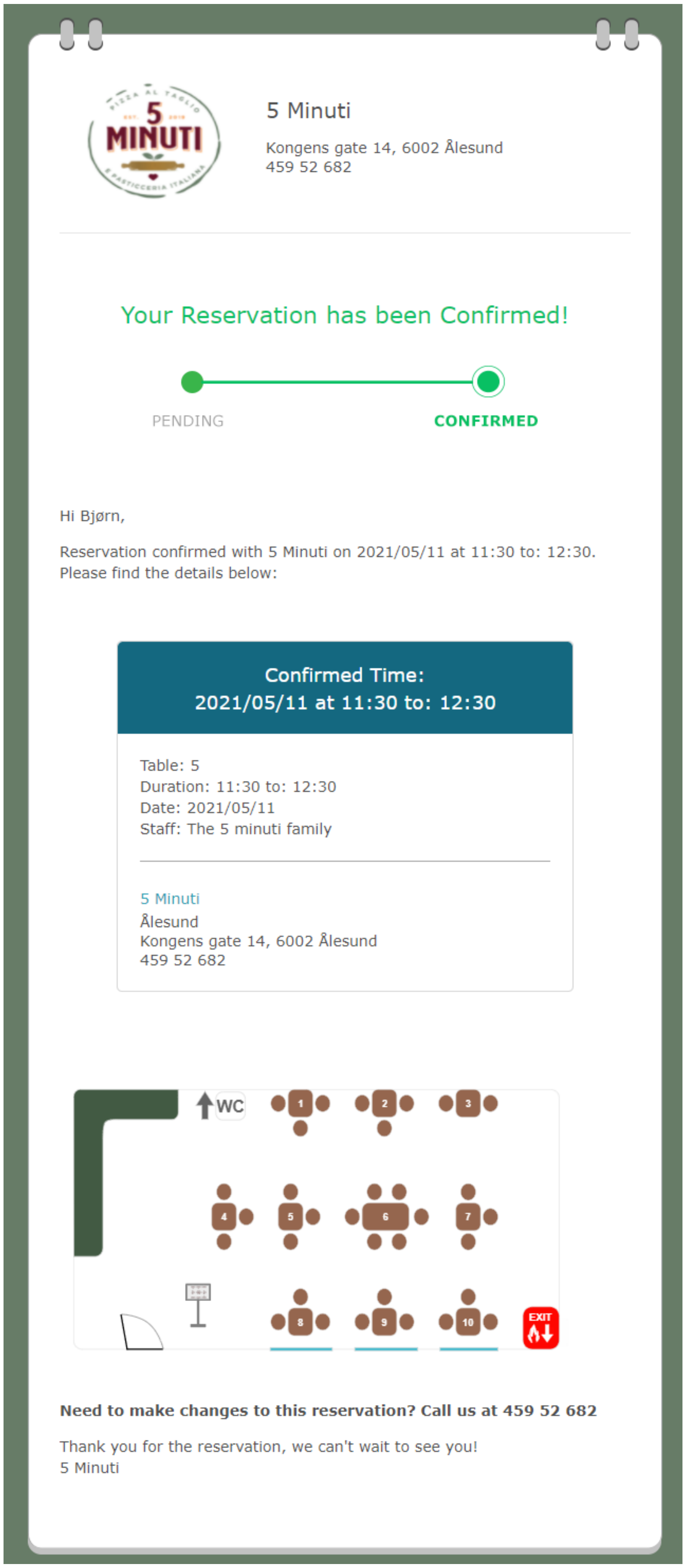
dragievasv@gmail.com

Feel free to contact us if there is anything you want to ask about.

Figure 4.12: Contact us page

4.2.2.7 Email service

The email service (Figure 4.13) is for the customer to get a confirmation of the reservation. It includes the reservation date and time, table chosen, contact information, and restaurant overview.



5 Minuti

Kongens gate 14, 6002 Ålesund
459 52 682

Your Reservation has been Confirmed!

Hi Bjørn,

Reservation confirmed with 5 Minuti on 2021/05/11 at 11:30 to: 12:30.
Please find the details below:

Confirmed Time:
2021/05/11 at 11:30 to: 12:30

Table: 5
Duration: 11:30 to: 12:30
Date: 2021/05/11
Staff: The 5 minuti family

5 Minuti
Ålesund
Kongens gate 14, 6002 Ålesund
459 52 682

Need to make changes to this reservation? Call us at 459 52 682

Thank you for the reservation, we can't wait to see you!
5 Minuti

Figure 4.13: Email confirmation

4.2.3 Administrator User Access

The second group that will use this system is the administrator. The administrator user has access to all features but, most importantly, can access the administrative user resources. These resources are accessed after the admin has logged in with the Admin-view login portal. After successfully logging in, the admin gets access to the All reservations and Today's reservations. These hold all the reservations made in the system (that has not been deleted) and the reservation for that current day. This is necessary for the restaurant to use the system in practice and reserve tables the day they need to be reserved.

4.2.3.1 Admin-view Login

The Admin-view login is what authorizes a user to get access to the administrative user resources. The admin view login (Figure 4.14) consists of a title, input fields, and a button. The login title is login for administrators, and the input field is for the username and password. There is also a hint in the input fields for the user to easier understand where to write the username and password. After the user has put in the values, the login button will be available to click. Then if the username and password are correct, they will be logged into the admin view.

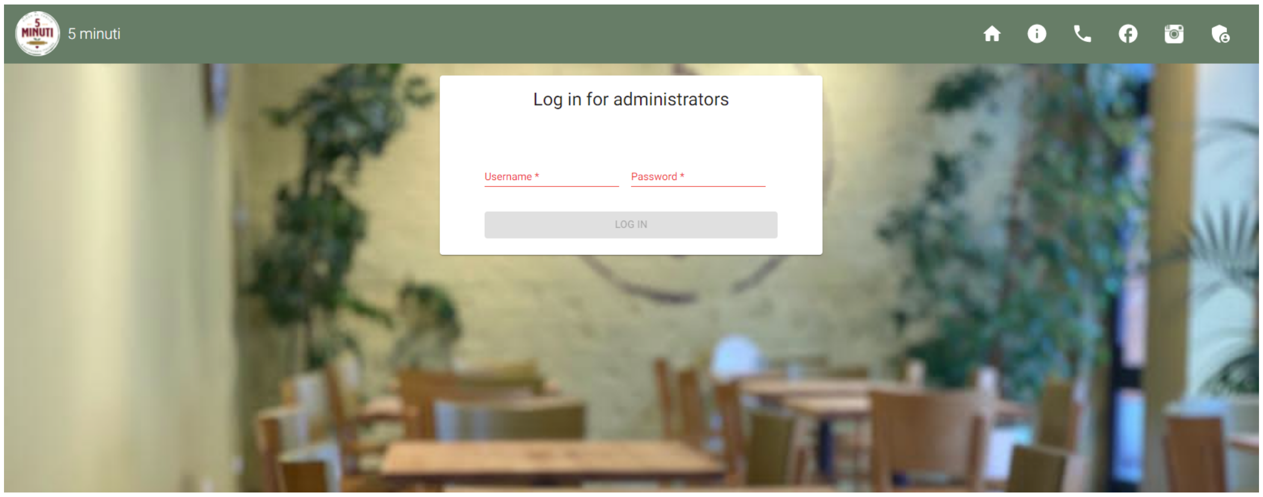
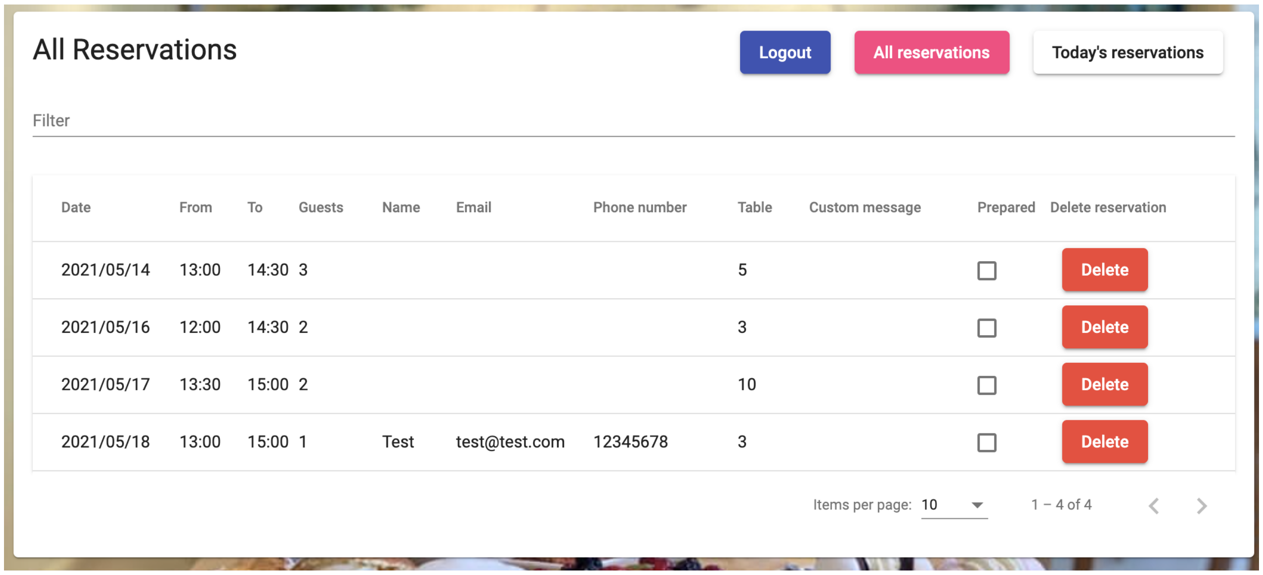


Figure 4.14: Admin Log In

4.2.3.2 Admin-view All Reservations

The Admin-view all reservations is the page that holds the table with all the reservations. The Admin-view all reservations (Figure 4.15) has a title, buttons, a filter, and a table with the reservation information. On the top right, there are three buttons. The "log out"-button logs the user out of the system. The "All reservations"-button takes the user to the all reservation tab, and the "Today's reservations"-button takes the user to the today's reservation tab.



Date	From	To	Guests	Name	Email	Phone number	Table	Custom message	Prepared	Delete reservation
2021/05/14	13:00	14:30	3				5		☐	Delete
2021/05/16	12:00	14:30	2				3		☐	Delete
2021/05/17	13:30	15:00	2				10		☐	Delete
2021/05/18	13:00	15:00	1	Test	test@test.com	12345678	3		☐	Delete

Figure 4.15: All Reservations

Filter

A filter lets the user search through the table. This can search for any value.

Table Parameters

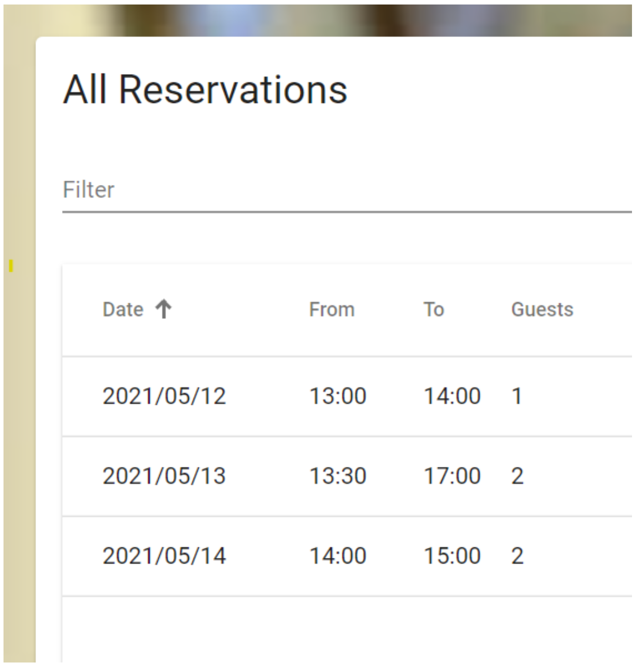
The all reservations table holds all the reservations in the database. It has ten parameters: date, from- and to-time, guests, name, email, phone number, table, custom message, and prepared.

The team argued whether the customer should only be able to reserve tables next to each other. Still, the group ended up going for a solution to select any two tables and then book them for a bigger group. This makes it so that a table entry can look 4,8 (table 4 and 8 reserved).

There is also a custom message where the customer can tell the restaurant about any special needs. Lastly, there is also a prepared checkbox that lets the restaurant employees know whether the reservation is prepared or not.

Sorting

An admin can sort any of the values from lowest to highest to how it is sorted in the database by clicking the header (Figure 4.16). This works for both strings and integers and will sort either by letter or number.



All Reservations

Filter

Date ↑	From	To	Guests
2021/05/12	13:00	14:00	1
2021/05/13	13:30	17:00	2
2021/05/14	14:00	15:00	2

Figure 4.16: Sorting the date parameter by ascending

Deleting

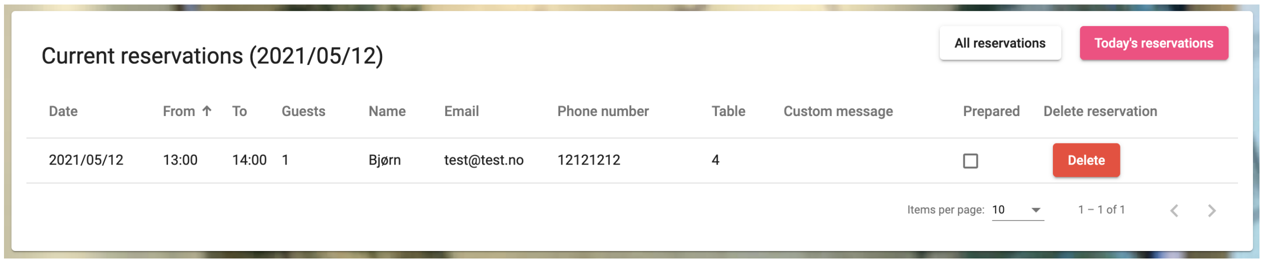
A delete button is placed next to each reservation. When clicking the delete button, a warning will appear with "Are you sure you want to delete this reservation?" since it is easy to click by accident and delete the wrong reservation. The warning lets the user have a second chance at deleting the reservation. If the user deletes the reservation, it will be deleted from the table view and permanently deleted from the database.

Items per page

At the bottom of the table, the user can select how many items per page the table should show, and the user can also select which page the user wants to be on. The user selects this by using the arrows at the bottom of the table (Figure 4.15). The default for items per page is set to 10 but can be set to 5, 25, and 100.

4.2.3.3 Admin-view Today's reservations

Today's reservations view (Figure 4.17) is practically the same as the all reservations view, except that it only shows the reservations for that current day and does not contain a filter.



Current reservations (2021/05/12)

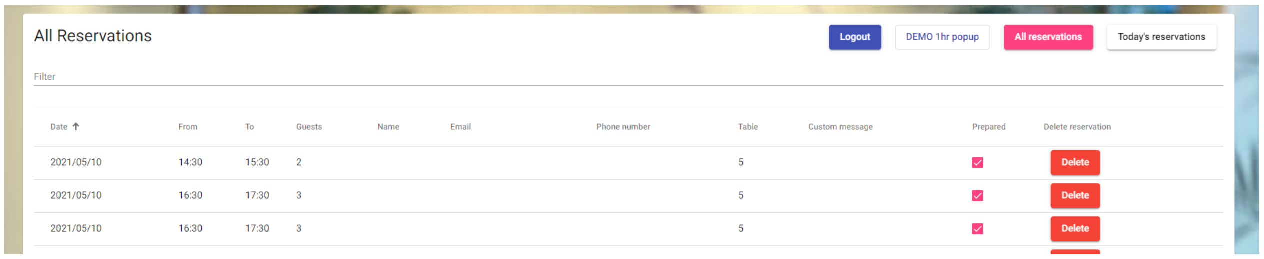
Date	From ↑	To	Guests	Name	Email	Phone number	Table	Custom message	Prepared	Delete reservation
2021/05/12	13:00	14:00	1	Bjørn	test@test.no	12121212	4		☐	Delete

Items per page: 10 1 – 1 of 1

Figure 4.17: Today's Reservations

4.2.3.4 Admin-view Personal information deleted 1 day after reservation end

According to the GDPR rules, a business has to delete personal information if it is not used for a specific purpose other than just storing it. The company not being able to store the reservation is, generally speaking, their issue. Still, it would be appreciated with statistics on when and where users would create reservations for future analysis of the shop's customer flow. Therefore all personal information is deleted one day after the reservation ends (Figure 4.18). Deleting the personal information one day after means that the date and time from when the reservation happened can still be seen, but not anything that can be traced back to anyone.



All Reservations

Filter

Date ↑	From	To	Guests	Name	Email	Phone number	Table	Custom message	Prepared	Delete reservation
2021/05/10	14:30	15:30	2				5		✓	Delete
2021/05/10	16:30	17:30	3				5		✓	Delete
2021/05/10	16:30	17:30	3				5		✓	Delete

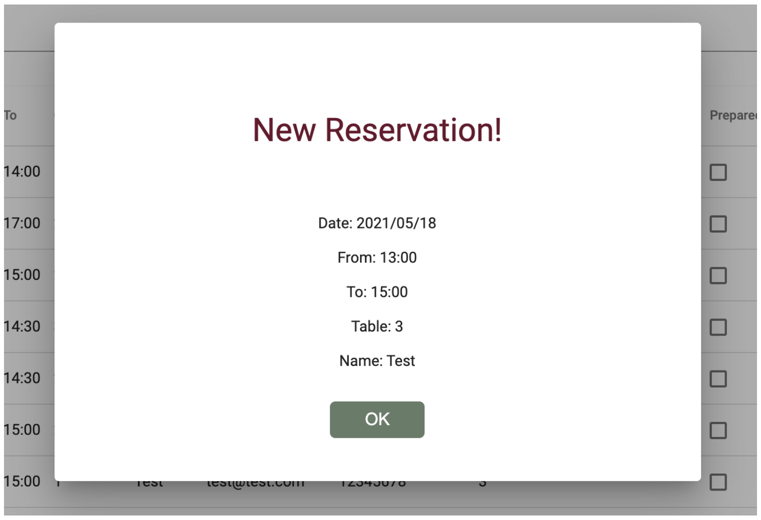
Figure 4.18: Deleting personal information 1 day after reservation is done3

4.2.3.5 Notifications

When a user is logged in as an administrator, notifications are available. The notifications are created with socket.io (2.4.9.1) and the Angular Material MatDialog service.

New Reservation

When a new reservation is added, a popup (Figure 4.19) appears with information about the reservation. The popup has information about the date, time, table, and name chosen for the reservation.



New Reservation!

Date: 2021/05/18

From: 13:00

To: 15:00

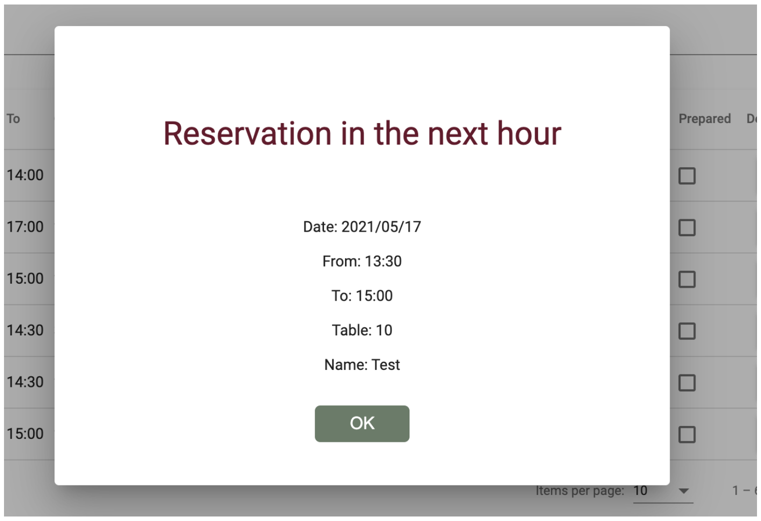
Table: 3

Name: Test

Figure 4.19: New reservation popup

Reservation within one hour

When there is one hour or less until a reservation, a popup (Figure 4.20) appears to notify the restaurant's employees. The popup has information about the date, time, table, and name chosen for the reservation.



Reservation in the next hour

Date: 2021/05/17

From: 13:30

To: 15:00

Table: 10

Name: Test

OK

Figure 4.20: Reservation within one hour popup

4.3 Architecture

The MEAN-Stack and nodemailer make up the basis of the architecture. This consists of a front end and a back end that communicates with the cloud.

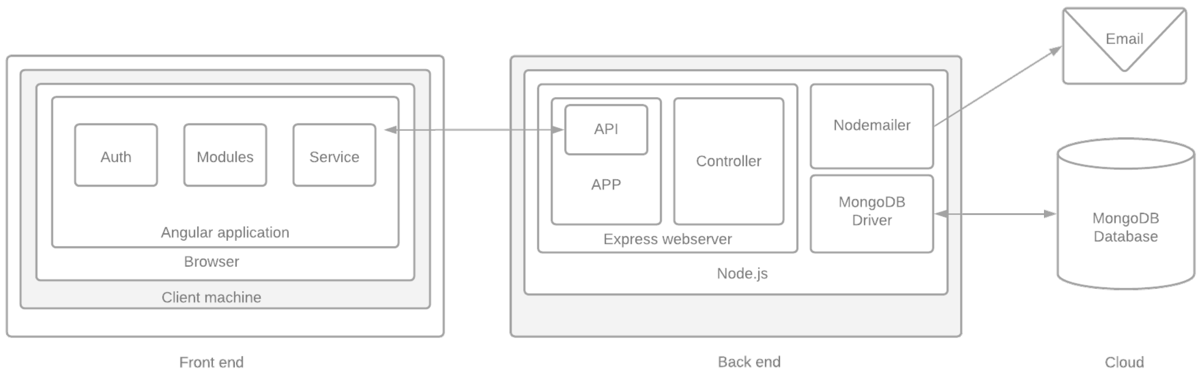


Figure 4.21: The projects architecture (MongoDB, 2021a)

4.3.1 Front End Architecture

In this section, the group will explain how the application is built and its routing. Angular is a powerful tool combined with TypeScript, HTML, and CSS and seemed to have many compact features that could easily be applied to the project.

4.3.1.1 Angular

As mentioned, Angular is a robust framework, including components, routing, and modules (Figure 4.22). With Angular, a developer can import the core and a set of TypeScript libraries (Angular, 2021b).

Modules

Angular applications have their modules system called NgModules. NgModules are containers for a different type of code dedicated for specific tasks. One of these tasks is the date picker module. This lets the developer import a calendar with relative ease and imports it into the developer's project. This way of structuring the system makes it easier to create things faster since features can be imported and easily applied to a project. Every Angular application has at least one NgModule class, the root module; this module is called the AppModule has to file name app.module.ts. When creating a new project, the project only has one Root NgModule, but it can include child modules in a depth hierarchy. This is seen in the authentication service, where there are separate modules. This can be very beneficial in terms of having clarity, where different modules have different imports (Angular, 2021b).

Components

Angular Components are built on a hierarchy. Inside the components folder, TypeScript-files(TS), HTML files, or the CSS files for the specific page can be found. Each component defines a class that contains application data and logic, which is associated with HTML. Inside the component-TS file, the HTML is referred to as a template. (Angular, 2021b).

Routing

Angular has its way of routing components and paths. One can create different routes in the angular routing and bind them to different components and their designated URL. Angular routing also makes it easier with router links, where users can navigate throughout the site, and it goes through the angular system and not an URL.

To secure a route and different components, the canActivate, and the canLoad feature can be used. The canActive condition can only activate a particular path or load a specific path if a condition was met. The canActivate and canLoad are what are used by AuthGuard to secure the routes to the project. The authguard component made it so to access All reservations and current reservations; a JWT token is needed. This would be given from the back end and, after getting it, would give the user access to the Admin resources.

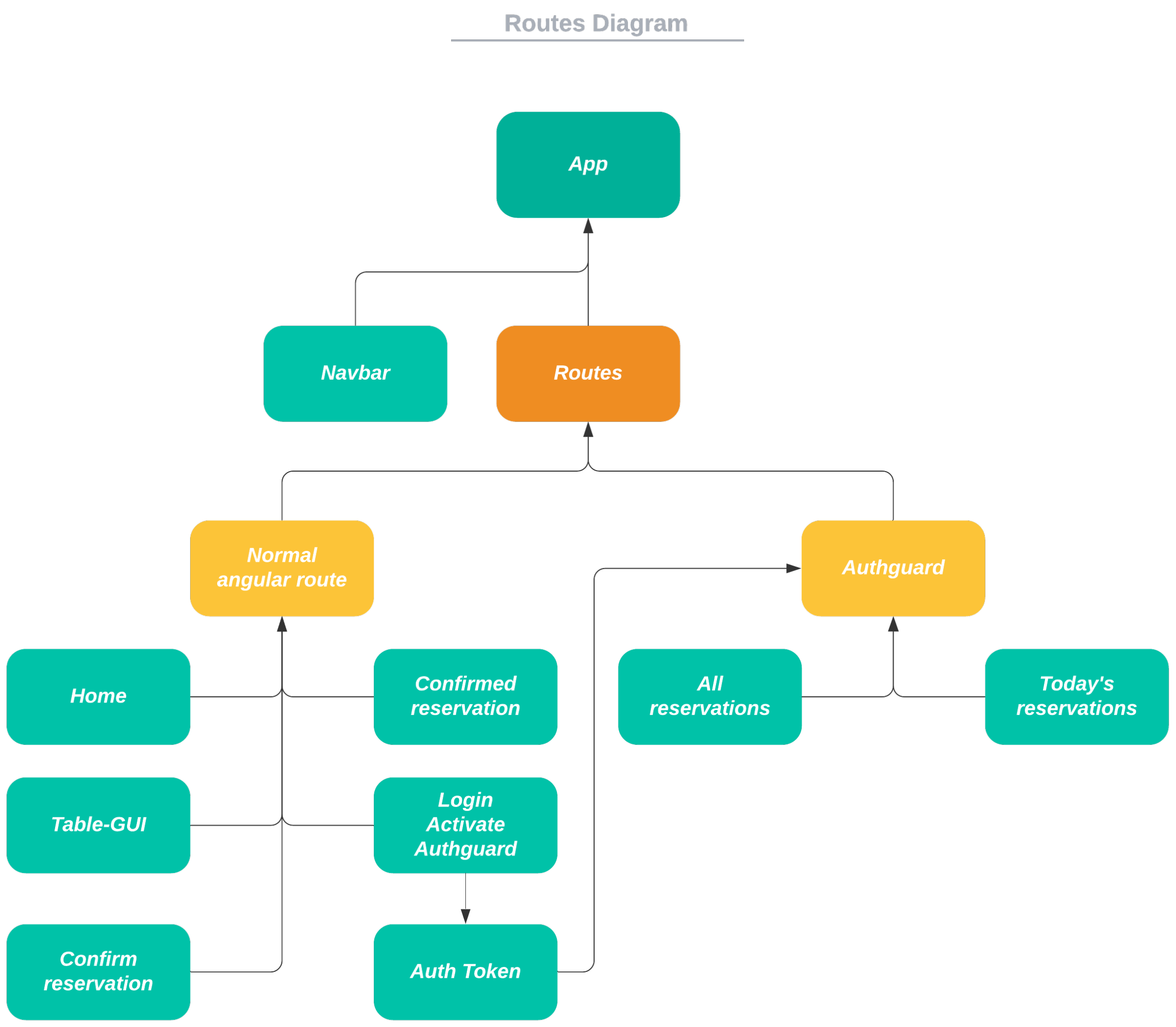


Figure 4.22: Routing diagram

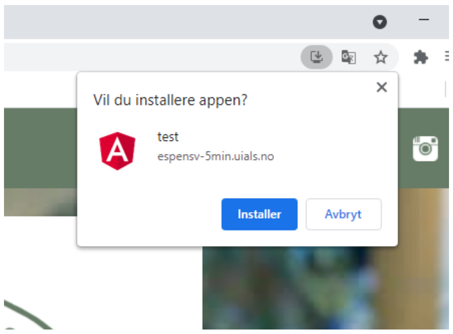
4.3.1.2 Optimization

One thing that helped optimize the app was to build to production. This took the total build size of 160 MB to 1.5MB. The group also had many optimization tips from lighthouse, which gave us help on creating a website that google promotes and is up to "Web development" standards.

One example of this is using a robots.txt file so that web crawlers can index the website. Thus giving google insight into the content of the website.

4.3.1.3 Progressive Web Application

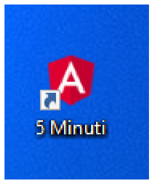
The Angular framework can convert an application to a Progressive Web Application (PWA). This allows us to export the system as an app on most platforms, like iOS, Android, or desktop applications (Figure 4.23). This came very handily when one of the requirements for the system was that it either was a mobile application or mobile-friendly.



Vil du installere appen?

test
espensv-5min.uials.no

(a) PWA install



(b) PWA app icon

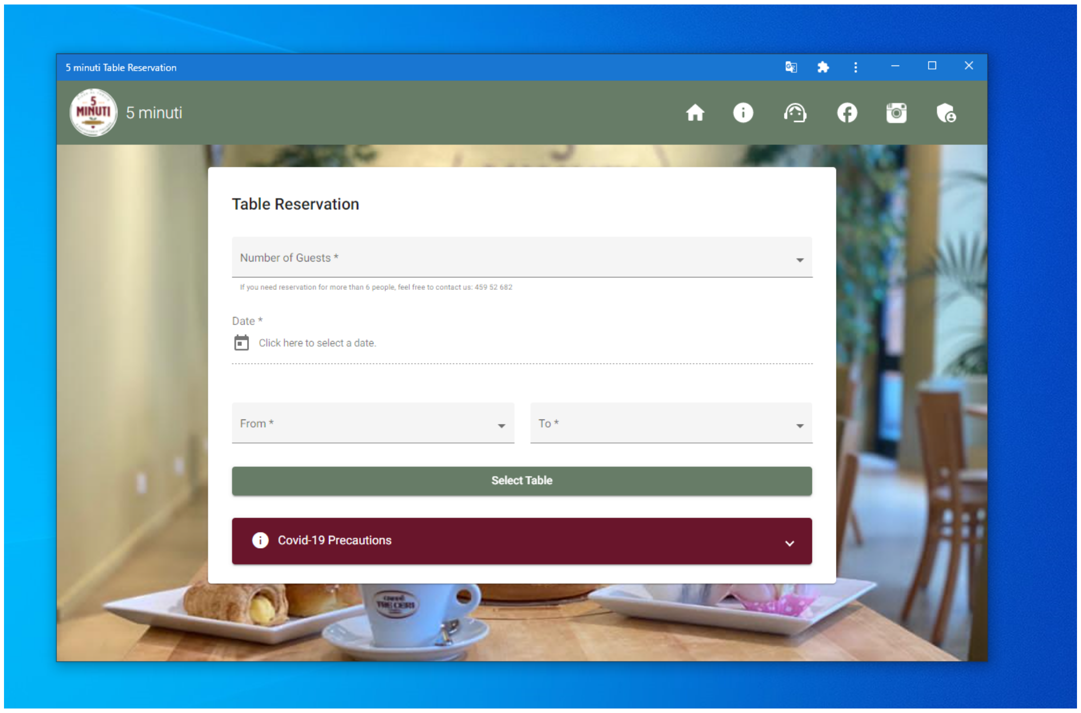


Table Reservation

Number of Guests *

If you need reservation for more than 6 people, feel free to contact us: 459 52 682

Date *
Click here to select a date.

From *
To *

(c) PWA desktop app running

Figure 4.23: PWA install windows desktop

4.3.2 Backend-architecture

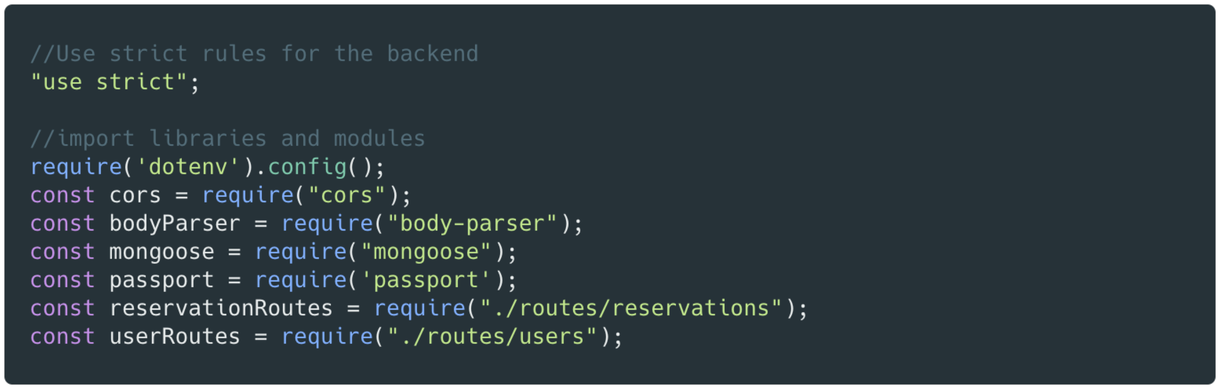
This chapter will be about the back end, for instance, how the Express.js web server is configured. The code here is not something the regular user can see. The files here are primarily based on the setup and configurations on the system's web server, database, and controller. Security is also defined here. The system has security in the back end as well and not only in the front end because everyone has the opportunity to create their front end. It will also be about the most important classes and their functions.

4.3.2.1 app.js

This subchapter will explain how the backend is built up. With the main app.js backend server, its controllers, its models, and its database configuration.

app.js Server

Here (Figure 4.24) are the libraries used for the app.js server.



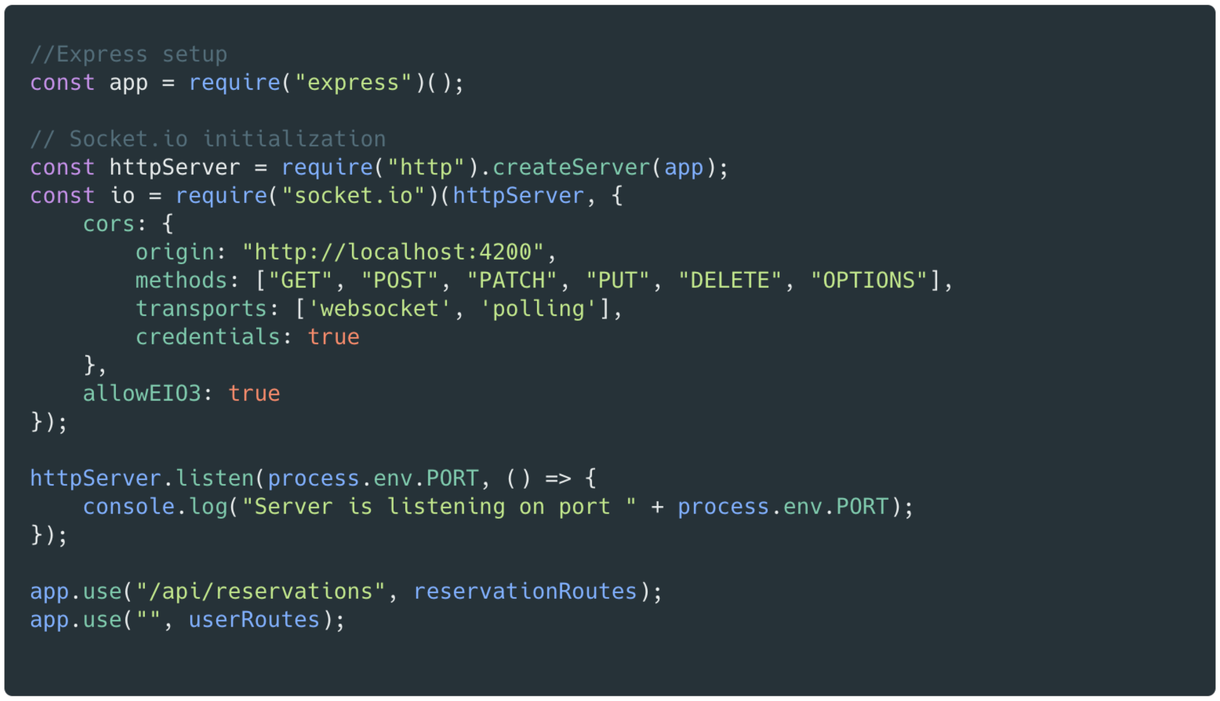
```
//Use strict rules for the backend
"use strict";

//import libraries and modules
require('dotenv').config();
const cors = require("cors");
const bodyParser = require("body-parser");
const mongoose = require("mongoose");
const passport = require('passport');
const reservationRoutes = require("./routes/reservations");
const userRoutes = require("./routes/users");
```

Figure 4.24: App.js libraries

Express and Socket.io Setup

This is the express (2.4.8) server and socket.io (2.4.9.1) setup (4.25).

First, the express library is imported. Then, using express, an HTTP server is created. Socket.io is imported and initialized using that HTTP server. The HTTP server starts and listens to the chosen port.



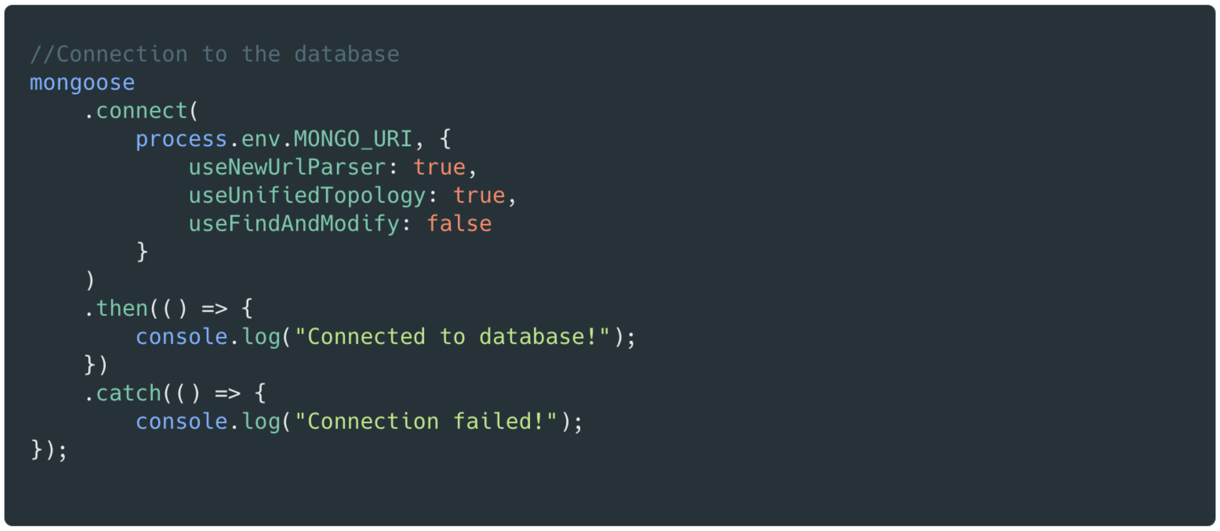
```
//Express setup
const app = require("express")();

// Socket.io initialization
const httpServer = require("http").createServer(app);
const io = require("socket.io")(httpServer, {
    cors: {
        origin: "http://localhost:4200",
        methods: ["GET", "POST", "PATCH", "PUT", "DELETE", "OPTIONS"],
        transports: ['websocket', 'polling'],
        credentials: true
    },
    allowEIO3: true
});

httpServer.listen(process.env.PORT, () => {
    console.log("Server is listening on port " + process.env.PORT);
});

app.use("/api/reservations", reservationRoutes);
app.use("", userRoutes);
```

Figure 4.25: Express and socket.io server setup

Mongoose setup

"Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB" (Karnik, 2018).

This configuration (Figure 4.26) uses Mongoose to connect to the MongoDB database.

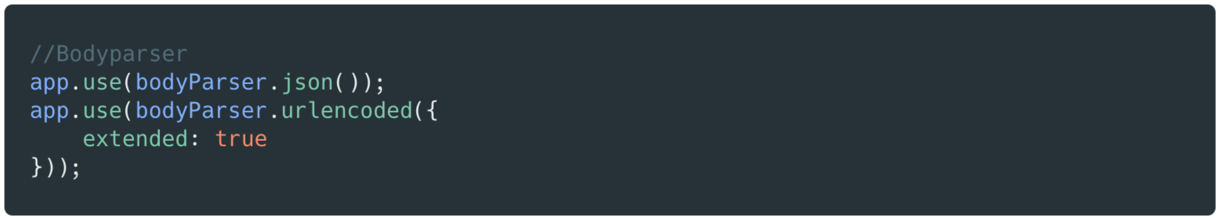


```
//Connection to the database
mongoose
    .connect(
        process.env.MONGO_URI, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useFindAndModify: false
        }
    )
    .then(() => {
        console.log("Connected to database!");
    })
    .catch(() => {
        console.log("Connection failed!");
});
```

Figure 4.26: Mongoose setup

Body-parser initializing

Body-parser (Figure 4.27) is the Node.JS body parsing middleware. It is responsible for parsing the incoming request bodies in a middleware before the system takes it to use.

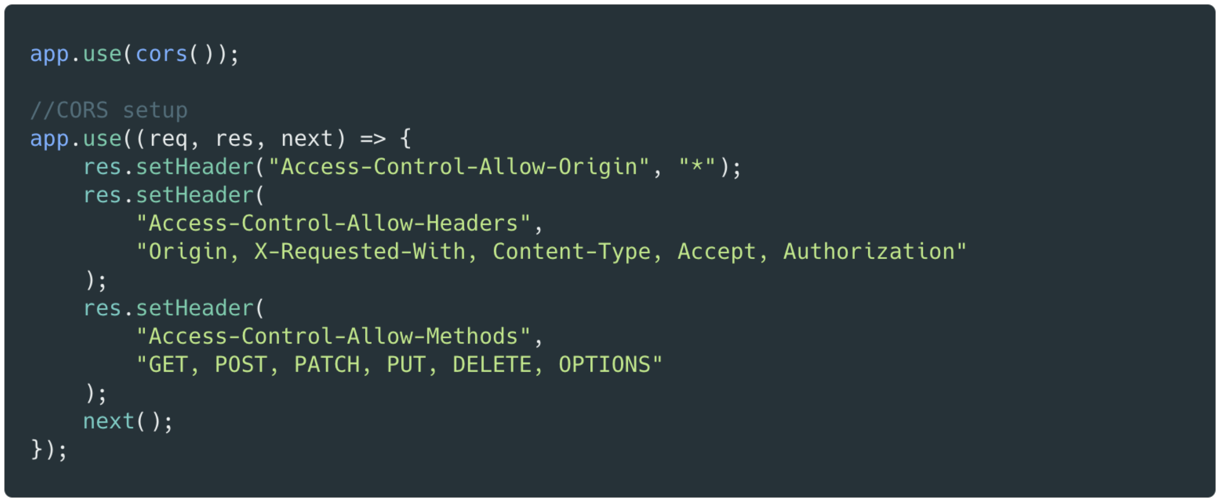


```
//Bodyparser
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
    extended: true
}));
```

Figure 4.27: Body-parser

CORS

Here (Figure 4.28), the CORS middleware for express is used to *"allow restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served"*(*CORS* 2021).



```
app.use(cors());

//CORS setup
app.use((req, res, next) => {
    res.setHeader("Access-Control-Allow-Origin", "*");
    res.setHeader(
        "Access-Control-Allow-Headers",
        "Origin, X-Requested-With, Content-Type, Accept, Authorization"
    );
    res.setHeader(
        "Access-Control-Allow-Methods",
        "GET, POST, PATCH, PUT, DELETE, OPTIONS"
    );
    next();
});
```

Figure 4.28: CORS handling

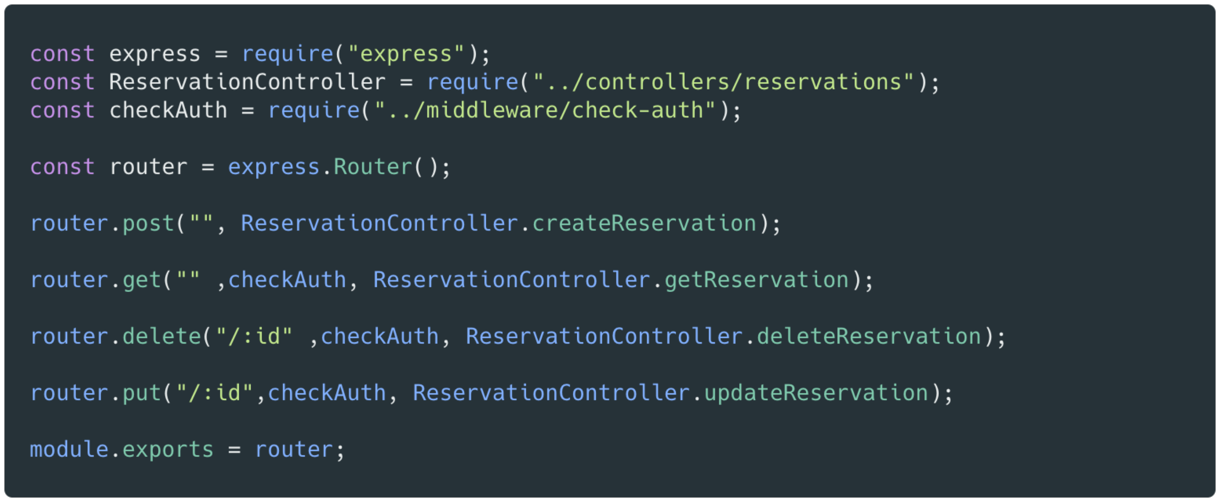
4.3.3 Routes and Controllers

A route is a section of Express code that associates an HTTP verb, a URL path or pattern, and a function called to handle that pattern. The routes are created with the express Router middleware. The routes forward the request to a controller, which reads and writes data to the models/database and sends an HTTP response (MDN, 2021a).

4.3.3.1 Reservation Routes and Controller

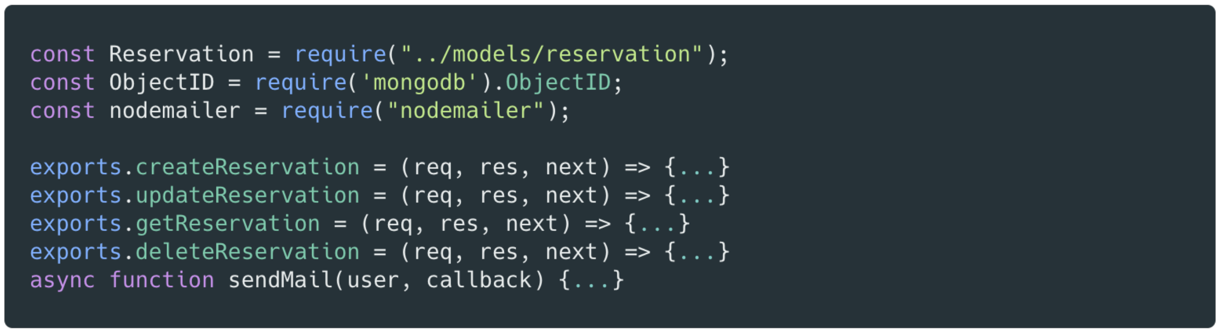
The reservation routes (Figure 4.29a) express Router middleware to forward requests to its controller (Figure 4.29b). The router also uses the checkAuth middleware to protect the get, delete and put routes.

createReservation posts a new reservation to the database. *getReservation* queries for new documents (reservations) from the database. *updateReservation* updates a document in the collection by id and is explicitly used for updating a document when the prepared checkbox is checked or unchecked. *deleteReservation* deletes a document by id from the collection. *sendMail* sends an email confirmation when the reservation process is finished.



```
const express = require("express");
const ReservationController = require("../controllers/reservations");
const checkAuth = require("../middleware/check-auth");

const router = express.Router();

router.post("", ReservationController.createReservation);

router.get("" ,checkAuth, ReservationController.getReservation);

router.delete("/:id" ,checkAuth, ReservationController.deleteReservation);

router.put("/:id",checkAuth, ReservationController.updateReservation);

module.exports = router;
```

(a) Reservation Routes



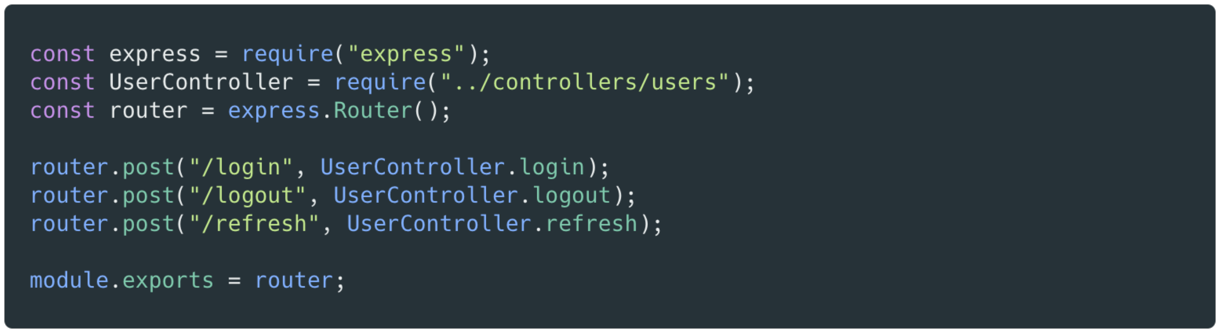
```
const Reservation = require("../models/reservation");
const ObjectID = require('mongodb').ObjectID;
const nodemailer = require("nodemailer");

exports.createReservation = (req, res, next) => {...}
exports.updateReservation = (req, res, next) => {...}
exports.getReservation = (req, res, next) => {...}
exports.deleteReservation = (req, res, next) => {...}
async function sendMail(user, callback) {...}
```

(b) Reservation Controller

Figure 4.29: Reservation Routes and Controller

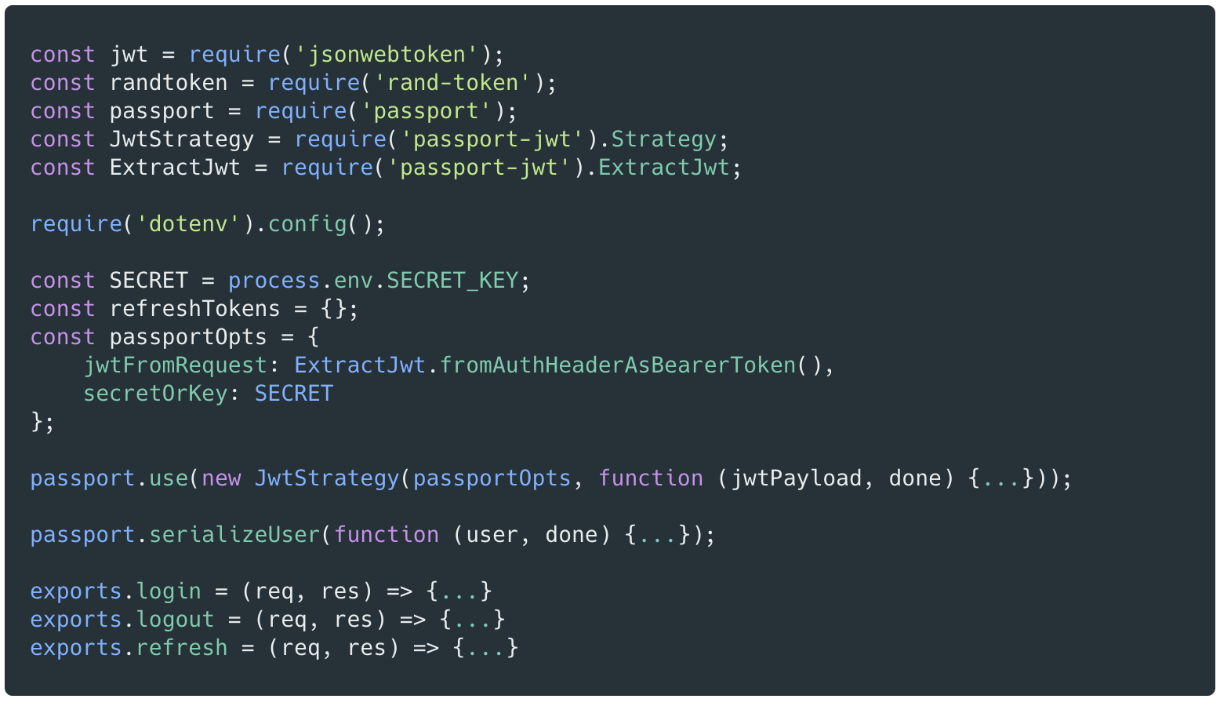
4.3.3.2 User Routes and Controller

The user routes (Figure 4.30a) and the user controller (Figure 4.30b) are similar to the reservation routes and controller. They handle the login, logout, and refresh of the administrator user. The controller uses JWT tokens (2.6.1.1) and passport.js as authentication middleware. After the administrator has logged in, they get a JWT token. The refresh token is so that the user does not have to log in every time they return to the system if they are already logged in. This makes it easier for the end-user.



```
const express = require("express");
const UserController = require("../controllers/users");
const router = express.Router();

router.post("/login", UserController.login);
router.post("/logout", UserController.logout);
router.post("/refresh", UserController.refresh);

module.exports = router;
```

(a) User Routes



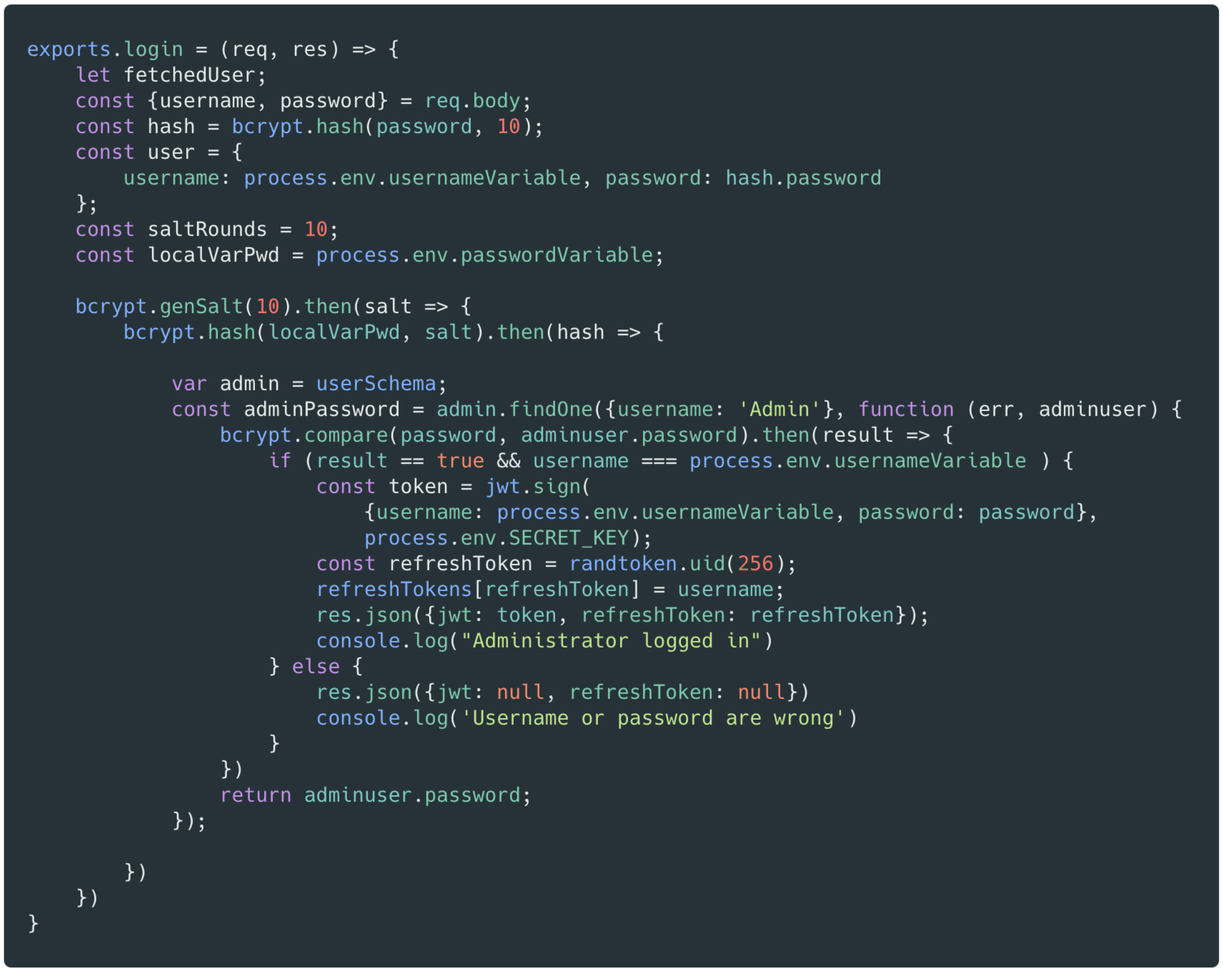
```
const jwt = require('jsonwebtoken');
const randtoken = require('rand-token');
const passport = require('passport');
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;

require('dotenv').config();

const SECRET = process.env.SECRET_KEY;
const refreshTokens = {};
const passportOpts = {
    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
    secretOrKey: SECRET
};

passport.use(new JwtStrategy(passportOpts, function (jwtPayload, done) {...}));

passport.serializeUser(function (user, done) {...});

exports.login = (req, res) => {...}
exports.logout = (req, res) => {...}
exports.refresh = (req, res) => {...}
```

(b) User Controller

Figure 4.30: User Routes and Controller

Controller

In the back end, under controllers, the developer can find the user's JavaScript file (Figure 4.31). The controller is an essential part of the back end because it handles the security. First, the system takes the input from the user, then hashing it and adds salt to it, for then comparing the password with the password in the database - which is also hashed and salted. Then if the result is true and the username is correct, it will provide the guy who tries to log in with a successfully token. Otherwise, if the password is wrong or the username is wrong, a JWT token will be provided as empty. Without a JWT token, a user cannot log in or access administrative resources. If they attempt to log in without a JWT token, they will get feedback from the server that the input was wrong.



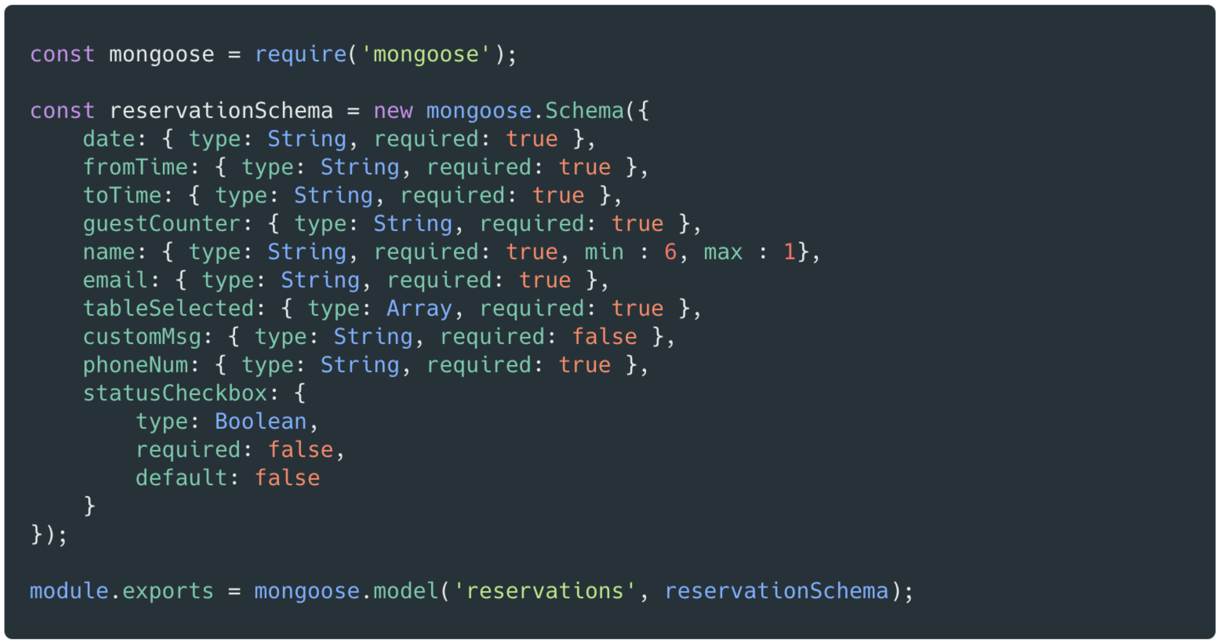
```
exports.login = (req, res) => {
    let fetchedUser;
    const {username, password} = req.body;
    const hash = bcrypt.hash(password, 10);
    const user = {
        username: process.env.usernameVariable, password: hash.password
    };
    const saltRounds = 10;
    const localVarPwd = process.env.passwordVariable;

    bcrypt.genSalt(10).then(salt => {
        bcrypt.hash(localVarPwd, salt).then(hash => {

            var admin = userSchema;
            const adminPassword = admin.findOne({username: 'Admin'}, function (err, adminuser) {
                bcrypt.compare(password, adminuser.password).then(result => {
                    if (result == true && username === process.env.usernameVariable ) {
                        const token = jwt.sign(
                            {username: process.env.usernameVariable, password: password},
                            process.env.SECRET_KEY);
                        const refreshToken = randtoken.uid(256);
                        refreshTokens[refreshToken] = username;
                        res.json({jwt: token, refreshToken: refreshToken});
                        console.log("Administrator logged in")
                    } else {
                        res.json({jwt: null, refreshToken: null})
                        console.log('Username or password are wrong')
                    }
                })
                return adminuser.password;
            });

        })
    })
}
```

Figure 4.31: Bcrypt

4.3.4 Models

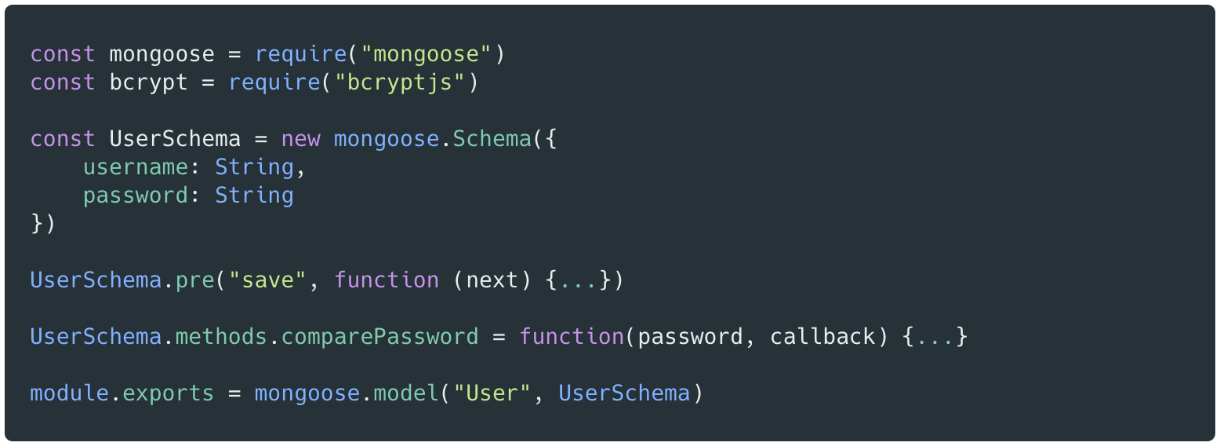
Models are constructors compiled from schema definitions. An instance of a model is called a document. Models are responsible for creating and reading documents from the underlying MongoDB database (Mongoose, 2021).

The reservation model (Figure 4.32a) creates a schema that compiles to a model. It contains all of the properties of a reservation. The user model (Figure 4.32b) also consists of a schema containing properties for a user. The user model also uses bcrypt to hash and salt and then compare the password in the database.



```
const mongoose = require('mongoose');

const reservationSchema = new mongoose.Schema({
    date: { type: String, required: true },
    fromTime: { type: String, required: true },
    toTime: { type: String, required: true },
    guestCounter: { type: String, required: true },
    name: { type: String, required: true, min : 6, max : 1},
    email: { type: String, required: true },
    tableSelected: { type: Array, required: true },
    customMsg: { type: String, required: false },
    phoneNum: { type: String, required: true },
    statusCheckbox: {
        type: Boolean,
        required: false,
        default: false
    }
});

module.exports = mongoose.model('reservations', reservationSchema);
```

(a) Reservation Model



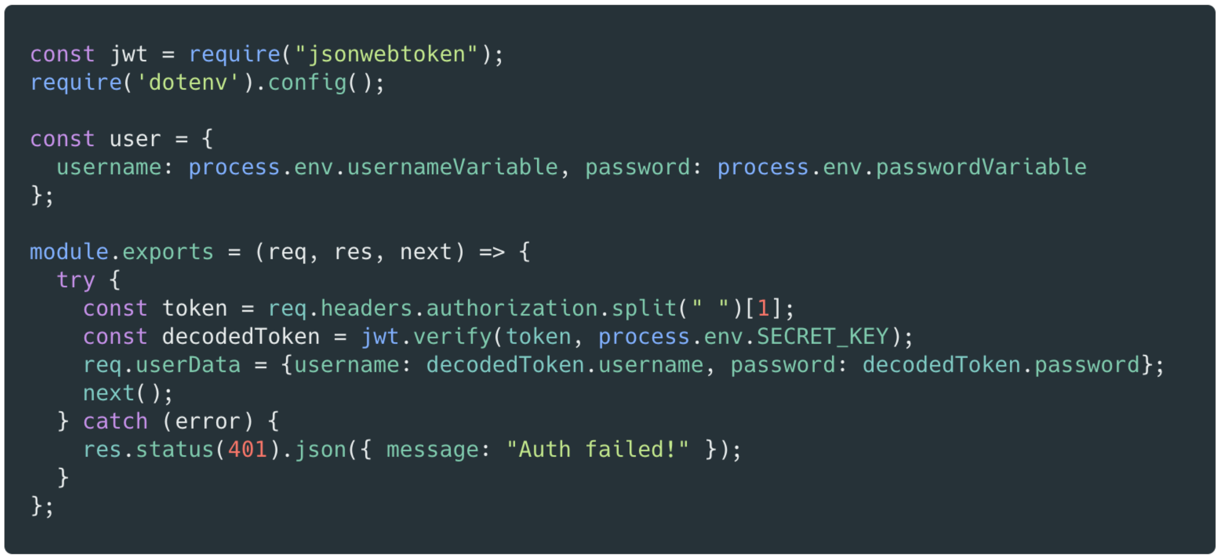
```
const mongoose = require("mongoose")
const bcrypt = require("bcryptjs")

const UserSchema = new mongoose.Schema({
    username: String,
    password: String
})

UserSchema.pre("save", function (next) {...})

UserSchema.methods.comparePassword = function(password, callback) {...}

module.exports = mongoose.model("User", UserSchema)
```

(b) User Model

Figure 4.32: Reservation and User Models

4.3.5 Auth

There is the middleware (Figure 4.33) to check if the user has a valid JWT token, then giving access to deleting or accessing the entire database, and not just tables available at a certain time.



```
const jwt = require("jsonwebtoken");
require('dotenv').config();

const user = {
  username: process.env.usernameVariable, password: process.env.passwordVariable
};

module.exports = (req, res, next) => {
  try {
    const token = req.headers.authorization.split(" ")[1];
    const decodedToken = jwt.verify(token, process.env.SECRET_KEY);
    req.userData = {username: decodedToken.username, password: decodedToken.password};
    next();
  } catch (error) {
    res.status(401).json({ message: "Auth failed!" });
  }
};
```

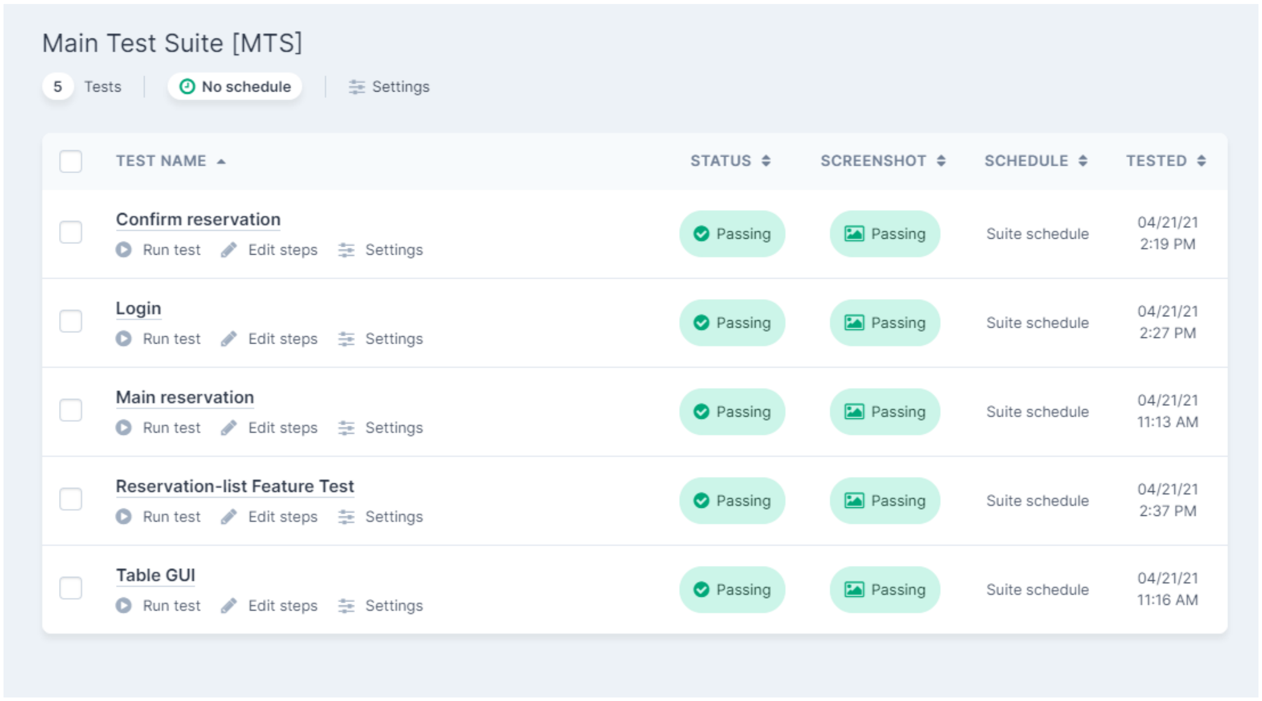
Figure 4.33: Authentication check

4.4 Testing

In this section, the team will talk about testing and its results. Automated testing, usability testing, and physical testing are the topics the team will talk about here.

4.4.1 Automated Testing

Different solutions for testing were tried out, but the group chose ghost inspector (Figure 4.34). This gives a more accurate test to what a user would do and the process in its entirety than string and int testing.



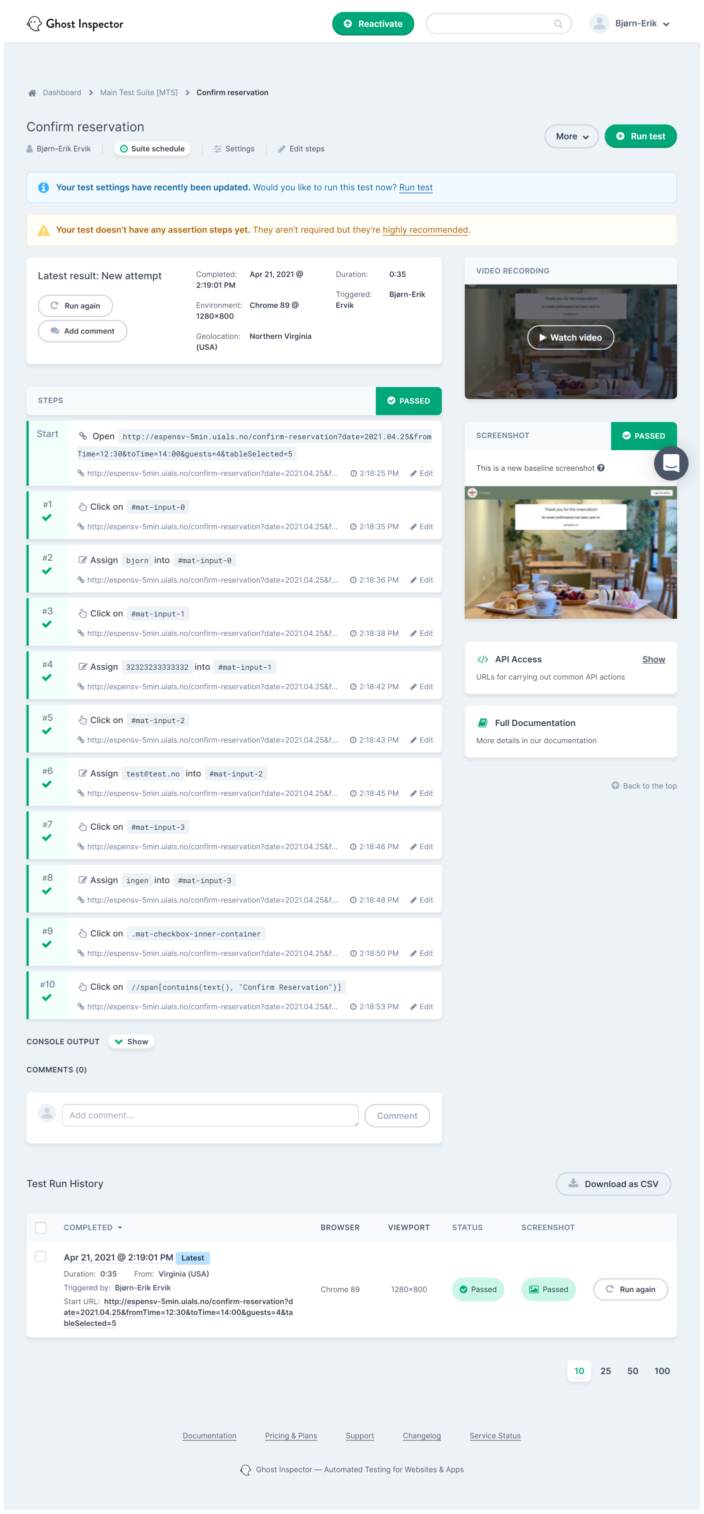
Main Test Suite [MTS]

5 Tests | No schedule | Settings

TEST NAME	STATUS	SCREENSHOT	SCHEDULE	TESTED
Confirm reservation Run test Edit steps Settings	Passing	Passing	Suite schedule	04/21/21 2:19 PM
Login Run test Edit steps Settings	Passing	Passing	Suite schedule	04/21/21 2:27 PM
Main reservation Run test Edit steps Settings	Passing	Passing	Suite schedule	04/21/21 11:13 AM
Reservation-list Feature Test Run test Edit steps Settings	Passing	Passing	Suite schedule	04/21/21 2:37 PM
Table GUI Run test Edit steps Settings	Passing	Passing	Suite schedule	04/21/21 11:16 AM

Figure 4.34: A complete list of pages the tests were run on

Tests were done for every component and the process in its entirety. Here (Figure 4.35) is the final test for the last component with input values, where the values are tested as shown and what values are expected.


Ghost Inspector
Reactivate
Bjørn-Erik
Dashboard > Main Test Suite [MTS] > Confirm reservation

Confirm reservation

Bjørn-Erik Ervik | Suite schedule | Settings | Edit steps
More
Run test

Your test settings have recently been updated. Would you like to run this test now? Run test

Your test doesn't have any assertion steps yet. They aren't required but they're highly recommended.

Latest result: New attempt

Completed: Apr 21, 2021 @ 2:19:01 PM
Environment: Chrome 89 @ 1280×800
Geolocation: Northern Virginia (USA)
Duration: 0:35
Triggered: Bjørn-Erik Ervik

STEPS — PASSED

Start — Open `http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&fromTime=12:30&toTime=14:00&guests=4&tableSelected=5`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:25 PM Edit

#1 ✓ Click on `#mat-input-0`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:35 PM Edit

#2 ✓ Assign `bjorn` into `#mat-input-0`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:36 PM Edit

#3 ✓ Click on `#mat-input-1`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:38 PM Edit

#4 ✓ Assign `32323233333332` into `#mat-input-1`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:42 PM Edit

#5 ✓ Click on `#mat-input-2`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:43 PM Edit

#6 ✓ Assign `test@test.no` into `#mat-input-2`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:45 PM Edit

#7 ✓ Click on `#mat-input-3`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:46 PM Edit

#8 ✓ Assign `ingen` into `#mat-input-3`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:48 PM Edit

#9 ✓ Click on `.mat-checkbox-inner-container`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:50 PM Edit

#10 ✓ Click on `//span[contains(text(), "Confirm Reservation")]`
http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&f... 2:18:53 PM Edit

SCREENSHOT — PASSED

This is a new baseline screenshot

Thank you for the reservation!
An email confirmation has been sent to:

API Access — Show
URLs for carrying out common API actions

Full Documentation
More details in our documentation

COMMENTS (0)

Test Run History

COMPLETED	BROWSER	VIEWPORT	STATUS	SCREENSHOT	
Apr 21, 2021 @ 2:19:01 PM Latest Duration: 0:35 From: Virginia (USA) Triggered by: Bjørn-Erik Ervik Start URL: http://espensv-5min.uials.no/confirm-reservation?date=2021.04.25&fromTime=12:30&toTime=14:00&guests=4&tableSelected=5	Chrome 89	1280×800	Passed	Passed	Run again

Figure 4.35: The testing for Confirm Reservation

4.4.2 Usability Testing

4.4.2.1 User tests

The usability test was done with ten people. The group created and used a template (See section 3.8.1), where the amount of time used, bugs, complicated actions, and errors of the different pages were measured. On average, the time used was between 1 to 3 minutes. The application got great feedback on the ease of use and the application's design, and not many difficulties were observed.

Observations:

- Problem with finding the information view on the table overview when asked to find it.
- Some problems getting past the captcha on confirm reservation
- Invalid date on iOS
- Not possible to use other characters than letters on confirm reservation message
- Difficult to see the message about turning the phone to see the navbar links

Feedback:

- Make the home page, choose date button press-able on the entire line where it is grayed out, and not only the calendar icon.
- Chronological order on the popups at the front page.
- Validation of the custom message to the restaurant at confirmation-page
- Maybe for future planning, adding order food to be ready on arrival
- Making the signature on the confirmed reservation page smaller
- Centering the content of the confirmed reservation page
- Mobile view, navbar only available when rotating device

- Mobile view, Remove the message: "Please rotate your device" on both the home page and table selection page.

After the tests, the group agreed on the observations, and the feedback needs to be taken seriously. The group used the feedback and fixed what were given responses. An example of this is making the signature smaller on the confirmed page. This means that many of these will be irrelevant to the current project since they are already fixed.

4.4.2.2 Implementation Testing

The group went to the restaurant to do an implementation test to see how the system would be used in practice. When the group was there, the group let Svetlana try out the system while explaining how the different parts of the system worked (Figure 4.36). After Svetlana tried to make a reservation with her MAC and phone, she checked out the system and how it would be used in practice. As seen in 4.39) after the reservation is filled in, Svetlana as an admin gets a notification. On this screen, all the employees can see all the reservations as well. At this meeting, Svetlana gave us the username and password for their web hotel. Unfortunately, their current web package only includes forwarding of Domain Name Server (DNS). Because of that web package, the web page was forwarded to the auto-deploy server. The result here is: www.5minuti.no

Figure 4.36: Bjørn-Erik and Espen explaining the system to Svetlana

In the project assignment, the group was supposed to give the customer the option to see what tables are available in the restaurant. This was going to be done with an iPad that stood next to the door inside the restaurant. However, because of Covid-19, the restaurant cannot have an iPad that is going to be used by multiple people. Therefore the group brought a laptop that simulates an iPad for potential future use (Figure 4.37).

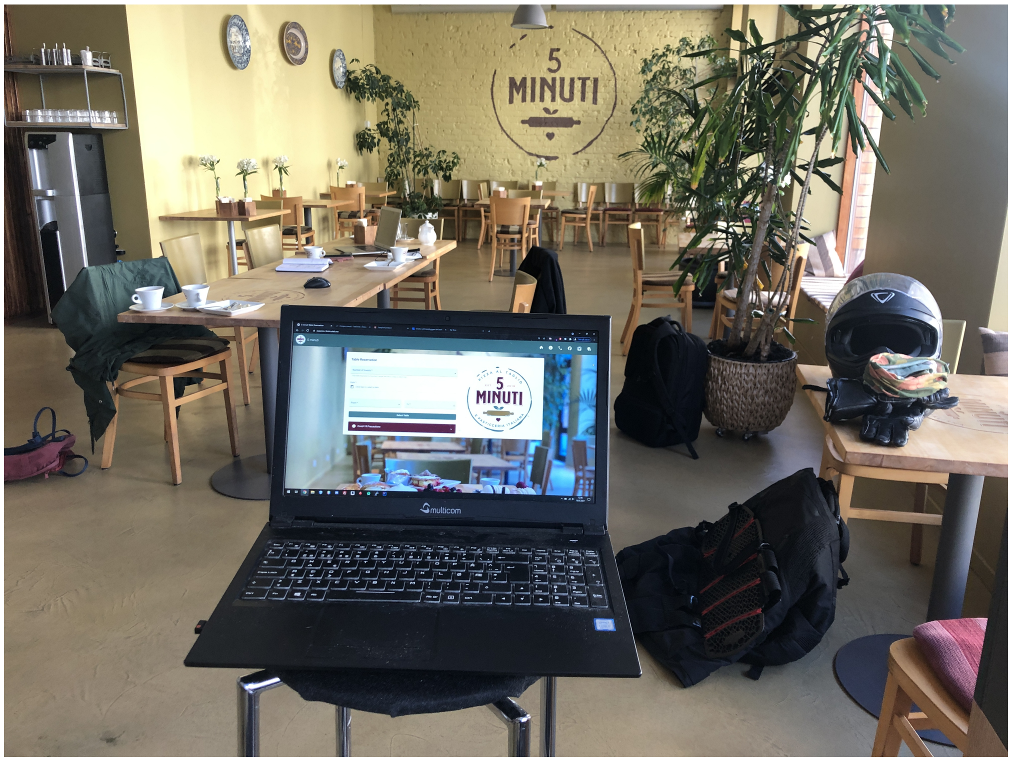


Figure 4.37: Laptop simulating future Ipad

The restaurant will, in the future, use a monitor hooked up to a laptop to see the future reservations. Here a laptop was used to simulate that (Figure 4.38).

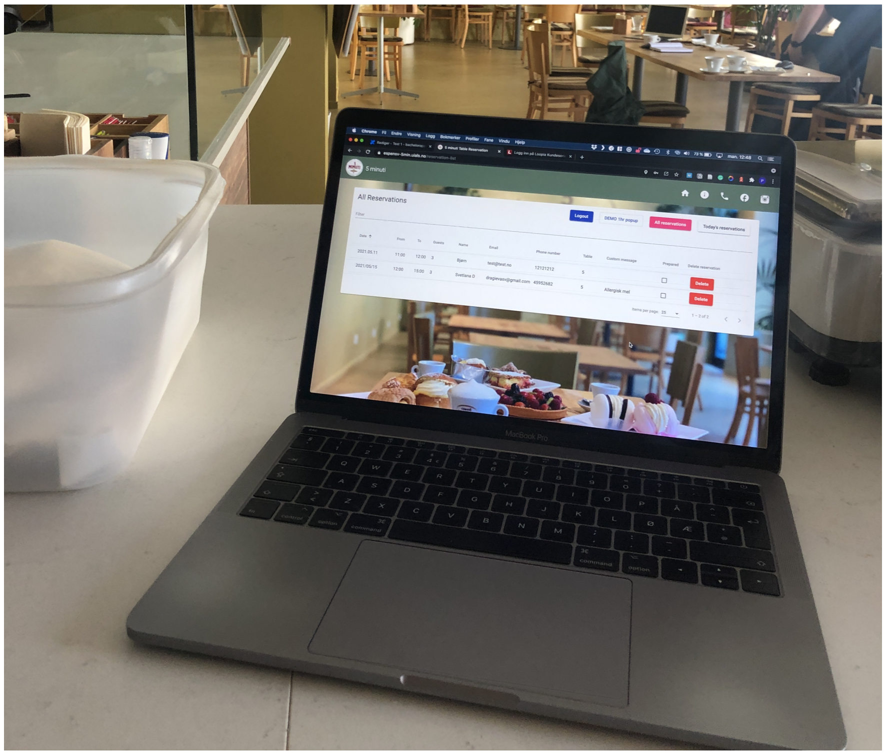


Figure 4.38: Laptop simulating monitor the restaurant will use for reservations

In the future, when the system is used, and a reservation is made, a "new reservation" pop-up will show. Here a laptop was used to simulate what that would look like (Figure 4.39).

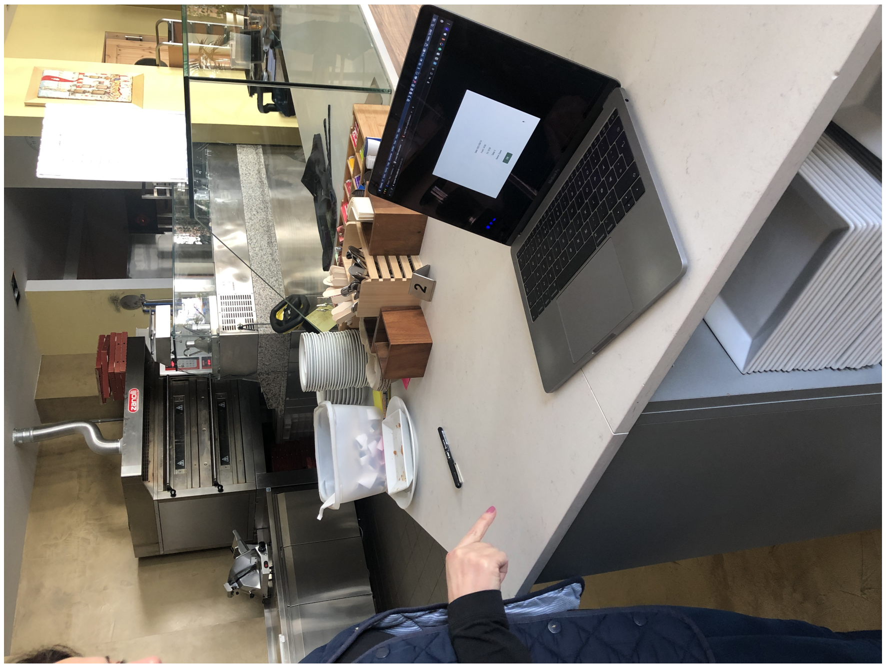


Figure 4.39: Laptop simulating a new reservation pop-up

Chapter 5

Discussion

This chapter will be about the thoughts and the reasoning around the developed system and the things that worked well or things that could have been done differently.

5.1 The main reservation system

The final table booking system fulfilled all the client's requirements, and in general, the group and the customer are happy with the delivered product. We also added quite a few extra features that they did not ask for, but that improved the system's quality. We got much creative freedom as the client-focused more on what the end product was supposed to provide. This went well for us as a group since it gave us more freedom. With this freedom, we could design the system as we wanted. This gave us the option to make the color palette fit the system. In retrospect, we were happy with the amount of creativity and the solution we created.

5.1.1 Home page

The home page ended up being suitable for its intended purposes. It makes it easy for the average user to pick a time and date for their reservation. The home page was built up by Angular Material Components, which made styling the website and give it features more accessible. This was true except for the calendar, also known as the date picker. The datepicker was hard to implement at first and took some time to get correct.

One of the difficult things with the calendar was having closing hours depending on which day it is, as well they are closed every Monday. So we have to fetch the input of the datepicker to see what date and day it is to give a return of the from time and to time.
In the middle of the project, our supervisor came up with an idea to implement available tables on the front page when you choose from time. The from the time are built on a for-loop. With the tables set rules, we had some issues to make available tables respond correctly. The logic behind it is complex. First, you need to make sure there are currently no reservations, and then you are from time has to be bigger than the time in the reservation. And then, you can add the table set rules to make the available tables respond correctly.

Angular also made it easier to form control and form groups. This was made more accessible since it had built-in functions and relatively easy-to-understand mechanics.

We also had to implement a COVID-19 precautions dropdown which informs the customer about the COVID-19 regulations the restaurant is following. Luckily with the red, light green, and white, we were still able to create a color palette that fits with the Cinque Minuti's logo.

5.1.2 The table overview

The development of the table GUI was complex. The first problem was how we would create the core components of the system. How would we create objects representing tables and chairs with HTML, CSS, and JavaScript. There are two parts to creating the table overview components. These two components are design and how the different objects would be placed in the browser. How are we suppose to create circles and squares to represent the different physical objects in the restaurant. Moreover, we should have them perform advanced placement and logic when the different box models are stacked inside each other. Behavior like this complicates using the DOM.

With Angular Material grid list, we could make grids with specific sizes and particular properties. This was our first solution. We made grids that would create little squares, and we were happy with the result that that technology gave us. It would be tremendously challenging to make this otherwise, if not wholly impossible, to make this out of HTML elements. One tricky thing was the size of the table overview and its effect on devices of different sizes. First, we tried to make HTML that used vh (view height) and vw (view width), which worked for a while.

Nevertheless, the tables and chairs were too small for smaller screens, and we had to use a lot of CSS (8.8 MB), which is a lot; for reference, the standard "Max limit" set by angular is 2MB. Luckily, we had one on our team that was good with SVGs who could create the SVGs instead. With this technology and options, the SVGs looked great on mobile and desktop. We also had an issue where the table overview would stretch very long and not look good on specific screens—setting a max-width fixed this by putting a max-width on the container.

Nevertheless, with SVG, they were set to a specific width and height, and SVG's with a specific width and height solved the problem. There were also issues with fixing the door and its sizing when using grid. The door sizing issue with the table overview remade was fixed when using the SVG. We are happy that we started very early with this process and had so much time refining and re-imagining the product.

Another challenge we had was implementing this graphical table booking system into actual numbers. We had to find out how the tables would be stored in a database and know how many tables are available in a particular situation. Moreover, this was hard to imagine and a significant technical problem for us to solve. We needed much time and a decent amount of abstraction to turn this into programmable logic. We solved it by having a system where from one to ten, the tables were named and had a from and to times where they were available to add, then stored in the database and unchecked if it was available from a specific time. If the tables were available, then even more rules were applied to the table, either if the guests were too few or too many to sit at the currently selected table. Developing this system and initialize how we would create a solution for this problem was tricky. Luckily we have great technology nowadays like the MEAN-stack to help us.

5.1.3 The confirm reservation page

. Very much with the same with the confirm reservation page. The angular framework made this more accessible than it presently could have been with the form fields and inputting name, phone, or email. You could quickly put in the information that looked good, and that could have specific requirements such as the name cannot have numbers, or the email must have an at the sign. It also made it easier to style the project so that it all had the same form factor. First, we applied RegEx to handle the fields for length, characters, and symbols, but it did not work out well since we need to add "ÆØÅ" in some fields, and in other fields, we need to handle symbols, for instance, "@ or quote marks." So we used HTML to handle the field's length and just used RegEx a few places where it needs to be handled. The form-structure angular offers, combined with using angular material, gave a styled and easy-to-use confirm reservation. With the Angular Material framework, it was easier to take the information from confirmed reservation into the MongoDB database as there are many tutorials on it and documentation to help.

5.1.4 The Admin-view

The administrative system has been OK to develop. We started early by implementing some of the data into a table, which was the earliest iteration of our current admin dashboard. With our current version, we have sorting, deleting, a prepared status, and the option to log in and log out. Today's implementation of the sorting, filtering, and showing only current reservations were challenging to develop. These features would take a long time to develop if they were not for Angular and Angular Material.

5.1.5 The Email Service

From a technical standpoint, the email service was not too hard to implement, and the only problem we had by creating the confirmation email is that separate CSS and HTML files could not be used. We had to use inline HTML inside the class to make it work. It took a little more time because of this. The outcome was well, and if the group creates a system like this again, it would be easier to do. Instead of using inline HTML, we could have used express to send a GET request with the HTML file response (Neswine, 2021).

5.1.6 Front end

We created the front end with a simple user interface in mind. If we had more time, we would have included a user log-in. This would let the customer skip the confirmation page where they fill out their contact information. This is the only place where the customer needs to type by himself. All the other fields are drop-down menus or click-to-select menus. We discussed having a login system for customers but found that it was not a priority for Enrico.

5.1.6.1 Angular

Angular has been a compelling and very time-efficient library to work with. It saved us a lot of time and helped us create many features. However, it had a very steep learning curve.

The architecture behind Angular and how it was built up was hard to understand initially. This was our first time using Angular, which resulted in initial concepts like components, modules, and routing taking some time to understand. After getting some experience learning how to use the platform, development became easier.

Angular is one of the most popular front-end frameworks, which means that the community around it is big. There are also many compelling features for routing and protection, such as authentication guarding and all services that can easily talk to the back end.

5.1.6.2 Design and user experience

Since we planned out the application's design using wireframes and prototypes, we had a clear idea of what we wanted the application to look like. We were asked to create a simple and intuitive application, which the planning resulted in. We have also tested it with friends and family of different ages, which seem to understand the system and think it has a pleasing aesthetic. We are generally speaking very pleased with the result.

5.1.7 Back end

We are happy with the results of the back end. None of us had too much experience with writing server-side code. When we look back at how we built the back end now, it could have been built differently. We could have used more "business rules" in the back end. In this case, we mean more of the table selection logic could have been in the back end (Figure 4.6).

5.1.7.1 REST API

Creating the REST API was not easy in the early stages of the project, where we had little knowledge about how to use it properly and set it up with the database correctly.

5.1.7.2 Architecture

We formed the back end after the project's needs during its development. Focusing on the project's needs leads us to find features we want to implement, implement, and refine. This process worked out for the group decently, and the group members are happy with the outcome and quality.

However, the group could have planned the architecture of the back end with more theoretical concepts. In the earlier stages of the project, the group focused more on creating something than architectural design theory. In hindsight, maybe there should have been more focus on theory and plan to give the back end even better quality. This does not mean the back end has any significant flaws and does not do its job. However, there are many concepts out there that might have been smart to apply.

5.1.7.3 Database

A NoSQL database such as MongoDB has good integration with TypeScript and Angular. The documentation was OK for MongoDB, and the group struggled a little bit in the early phases of implementing the database, pushing and pulling data into the database. These issues were more on having a correct setup on service and our controller to fetch up the information. After some tutorials using MEAN like we were able to do, successfully one of the earliest things we had to do, pushed the table entries that being which tables are supposed to go into the database. Then have the database store and retrieve that data. This was crucial for us in the early stages of this project. The reason for that is we needed the tables to be stored, and we needed them to read it. This is hard to visualize as a process, but MongoDB as a database structure made it OK. The group also used mongoose to keep the server up and not exiting after one command.

NoSQL vs SQL
We went with a NoSQL because it was what was used in the MEAN stack. It could have been easier with a SQL database since the different fields for values would be easier to distinguish since they have to be defined. However, it did not bring any problems using MongoDB, and it worked fine for our intended use. In general, MongoDB was fast and reliable and did not come with any significant drawbacks.

5.1.7.4 Security

Creating a system with robust security was challenging. The first challenge in this endeavor was creating a secure connection between the reservation list and the administrative view. Creating adequate protection was proven more challenging to do than we first anticipated. The authentication service authGuard would mainly solve the security problem, and the backend JWT and Bcrypt compare logic.
A user will not have access to the administrative resources unless they are verified. A user can get verified if they log in and authenticate themselves. If a user is verified, a Jason web token will be sent. If the user has a JWT token, the user will be sent to the admin page. The second part about our security problems was that we had an open API where anyone could access the reservations. Finding the right path to accessing this was not easy, but a person with web development experience could find the route. The group patched the security bug shortly after and confirmed that the bug no longer existed. The system ended up having a solid security profile for the system, and the group was pleased with the protection of data and routes.

5.1.8 Testing

One of the things that went well with the testing aspects of our angular application is the ghost inspector software. With ghost inspector, tests are being used as they were a user. In different places of the world with different applications and devices, this gives us a benefit to see how it will be for the end-user to use our product and test everything before running. We tried out native Angular Testing but chose to go for the ghost inspector route. Using the software was effortless and spotless to set up and has been a breeze to use and have.

5.1.9 Existing solutions

When it came to existing solutions for the system we have already made, there were some out there were some graphical table booking available systems, and it has been done before. However, the source code was not available either that or it was already made in PHP or other systems that were either too complex or unable to withdraw the logic separately to be used again in JavaScript an example of this is WordPress, where a system has been made but as a plugin but that would not be sufficient enough for us to copy and or use code as inspiration for the project at hand this gave us the option to look at different solutions that have already been deployed. However, there were not too many of them in a graphical sense the furthest thing we have had were the cinemas where you could choose a row and seat, but this is practically a leap from the booking system we were supposed to make that was way more than just rows where people could sit there are not too many examples.

5.1.10 MEAN versus other stacks

Why did we choose the MEAN stack to compare to MERN or MEVN? It is the same stacks, except React or Vue instead of Angular. Or in this case, why not any other stacks? None, any of us, really had any experience with MEAN or MERN. Moreover, when we started researching different stacks, the MEAN stack seemed like a popular and strong stack. The MERN also seemed popular and strong, but we went with the MEAN. Some of the reasons for that are because MEAN sounded interesting, and we all wanted to try something new. One of the other reasons was that with the Angular framework came mobile-friendliness and exporting it as a PWA natively. This is also a feature with React and Vue.

5.2 Agile development

In general, using agile development tools has made the project more accessible and more structured. We used the tools from the beginning, and they gave us a good overview of the project and good project planning.

5.2.1 Sprint meetings

For each sprint, we needed several meetings to plan and talk about the project. These meetings were primarily reviews of what has been done and what needed to be done.

5.2.1.1 Sprint review

It was practical to have these reviews where we discussed the progress of the last sprint. The client and supervisor were mainly happy with the progress after every sprint. We did not hear any significant complaints, mostly suggestions and changes. During sprint reviews, notes were taken on the various topics discussed. These can be found in the appendixD. During the last half of the project, we planned out the sprint reviews. This planning would mainly consist of what we had done and what we would do in the next sprint. We could have planned the sprint reviews from the start of the project to better prepare for the meetings.

5.2.1.2 Daily scrum

We had daily scrum meetings after each workday. These were short meetings that summarized the work of the day. This worked well to keep everyone aware of the progress and current issues.

5.2.1.3 Sprint retrospect

In the sprint meetings, we talked about what we had completed from the last sprint. These were valuable and gave us good insight into the project and how our goals were coming along. This was often done after the sprint review and mostly talked about. In retrospect, it could have been beneficial to have taken notes. Some notes were taken and written in the messaging application Discord. However, it could have been a better solution to have a separate document for the notes.

5.2.2 Scrum board

This was a convenient tool for us to use, and in general, did not have too many negatives. It gave an excellent overlay about what needed to be done and what had been done. It also gave insight into what the other group members were working with.

Chapter 6

Conclusions

Enrico Agostinelli felt the time has come to expand the systems for his restaurant. Enrico wants a table booking system so his customers could book a table through the internet. Before the first meeting, the group discussed different possibilities, how the project can be solved, and different scenarios that may appear.

The group created a web application that made it efficient to book a table reservation online, where you can see the physical layout of the restaurant. All this is packed in a fast, reactive, and modern user interface.

The team worked with both front end and back end, so all have understood full-stack development and how it was built. The team has gotten more experience from the application and service into the front end to the API and server in the back end. With that being said, all the team members participated equally when creating the application, writing in TypeScript, HTML, and CSS.

6.1 Further work

Alternatives for further upgrades of this web application could be to add an application to the device store, create a seasonable interface for selecting a table, more functions to the admin view.

6.1.1 Add the application on PlayStore and AppStore

Now we have a progressive web application, but it would be nice to add the application on the different stores regarding their operating system, both for both Android and iOS.

6.1.2 Seasonable interface

In some of the later meetings with Enrico, he mentioned they got a smaller area right outside of the restaurant to place some tables during the summertime. It would be nice to have a different interface for selecting a table deepening on what season it is. So when it is summer, he could switch the interface so customers could reserve a table outside as well, and when winter arrives, he could change back to the standard interface.

6.1.3 Admin view

The admin view can only see all the reservations, including today's reservations and deleting a reservation. Further work could be to implement an edit button to log in and change a reservation. For instance, change either from or to time, date, the number of guests, or even the selected table. A scenario may appear where a customer creates a reservation and want to change it later. The new reservation popup appears several times in a row when a new reservation is incoming, which is a bug that would be great to have resolved for the employees in the restaurant.

6.2 Cooperation

Once the team has gotten the task, we will discuss some ground rules before we progress. Everyone agreed on the working hours. Next was to set up Jira and Confluence, where everyone got their role to create issues and user stories. The team always had the opportunity to see what and which issue the other ones were working with. In addition, we had stand-up meetings as well.

6.3 Results

Good cooperation and planning gave a good result that we could deliver to the product owner proudly. If we did not establish the ground rules initially, we would not have gotten the same result as we got today. The team cared about the issues the others were working with and helped each other. In that sense, the team carried each other when help was needed.

We finished the project with the given problem description and had some time to improve and implement a few extra features. The last part of the project was not focused on creating more features but rather a robust solution. Doing this gave us software that we are proud of that can be used in the modern world.

Appendices

Bibliography

Angular (Mar. 2021a). *Angular - Introduction to the Angular Docs*. URL: https://angular.io/docs (visited on 03/15/2021).

— (2021b). *Angular - Introduction to Angular concepts*. URL: https://angular.io/guide/architecture (visited on 05/18/2021).

Arrachequesne, Damien (May 2021). *Introduction*. URL: https://socket.io/docs/v4/index.html (visited on 05/12/2021).

Atlassian (2021). *Sourcetree*. en. URL: https://www.sourcetreeapp.com (visited on 04/28/2021).

auth0.com (2021). *JWT.IO - JSON Web Tokens Introduction*. en. URL: http://jwt.io/ (visited on 05/07/2021).

beredskapsdepartementet, Justis-og (July 2018). *Ny personopplysningslov og EUs personvern-forordning*. no. Redaksjonellartikkel. Publisher: regjeringen.no. URL: https://www.regjeringen.no/no/tema/lov-og-rett/innsikt/ny-personopplysningslov/id2592984/ (visited on 05/06/2021).

Chacon, Scott and Ben Straub (Apr. 2021). *Pro Git*. 2nd edition. Apress.

CORS (May 2021). en. Page Version ID: 1020793000. URL: https://en.wikipedia.org/w/index.php?title=Cross-origin_resource_sharing&oldid=1020793000 (visited on 05/17/2021).

CSS (May 2021). en. Page Version ID: 1021443159. URL: https://en.wikipedia.org/w/index.php?title=CSS&oldid=1021443159 (visited on 05/05/2021).

Dahl, Yngve (2017). *Designprinsipper*. no. URL: https://folk.ntnu.no/baldurk/skolearbeid/MMI/Forelesninger%20MMI/30-Designprinsipper.pdf.

Datatilsynet (2021). *About us*. en. URL: https://www.datatilsynet.no/en/about-us/ (visited on 05/06/2021).

Documentation (2021). en. URL: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html (visited on 05/05/2021).

Express (2021). en. URL: https://expressjs.com/ (visited on 05/05/2021).

Fielding, Roy Thomas (2000). *REST*. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (visited on 05/06/2021).

Fortin, Michel (Apr. 2021). *michelf/sim-daltonism*. original-date: 2016-01-28T12:28:11Z. URL: https://github.com/michelf/sim-daltonism (visited on 05/07/2021).

Foundation, The Interaction Design (2021). *What are Design Principles?* en. URL: https://www.interaction-design.org/literature/topics/design-principles (visited on 05/06/2021).

GDPR (2021). en-US. URL: https://gdpr-info.eu/ (visited on 05/06/2021).

General Data Protection Regulation (May 2021). en. Page Version ID: 1021299333. URL: https://en.wikipedia.org/w/index.php?title=General_Data_Protection_Regulation&oldid=1021299333 (visited on 05/05/2021).

Ghost Inspector (2021). en. URL: https://ghostinspector.com/docs/getting-started/ (visited on 05/07/2021).

Gnatyk, Romana (2021). *Microservices vs Monolith: which architecture is the best choice?* URL: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/ (visited on 05/18/2021).

Harris, Chandler (2021). *Agile scrum artifacts*. en. URL: https://www.atlassian.com/agile/scrum/artifacts (visited on 05/04/2021).

HTML (Apr. 2021). en. Page Version ID: 1016919858. URL: https://en.wikipedia.org/w/index.php?title=HTML&oldid=1016919858 (visited on 05/05/2021).

Integrated development environment (Mar. 2021). en. Page Version ID: 1015043038. URL: https://en.wikipedia.org/w/index.php?title=Integrated_development_environment&oldid=1015043038 (visited on 05/04/2021).

Introduction to Node.js (2021). en. Section: Node.js. URL: https://nodejs.dev/learn/introduction-to-nodejs (visited on 05/05/2021).

ISO 9241-11 (2018). URL: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en (visited on 05/06/2021).

ISO/IEC/IEEE 24765 (2021). *ISO/IEC/IEEE 24765:2010*. en. URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/05/50518.html (visited on 05/11/2021).

JetBrains (May 2021). *Meet WebStorm*. en-US. URL: https://www.jetbrains.com/help/webstorm/2021.1/meet-webstorm.html (visited on 04/27/2021).

Karnik, Nick (Feb. 2018). *Introduction to Mongoose for MongoDB*. en. URL: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/ (visited on 05/17/2021).

Lighthouse (2021). en. URL: https://developers.google.com/web/tools/lighthouse (visited on 05/14/2021).

MDN (2021a). *Express Tutorial Part 4: Routes and controllers - Learn web development | MDN*. en-US. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes (visited on 05/17/2021).

— (2021b). *JavaScript*. en-US. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (visited on 05/05/2021).

— (2021c). *SVG: Scalable Vector Graphics*. en-US. URL: https://developer.mozilla.org/en-US/docs/Web/SVG (visited on 05/05/2021).

MongoDB (2021a). *The Modern Application Stack – Part 1: Introducing The MEAN Stack | MongoDB Blog*. en-us. URL: https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack (visited on 05/12/2021).

— (2021b). *What is NoSQL? NoSQL Databases Explained*. en-us. URL: https://www.mongodb.com/nosql-explained (visited on 05/06/2021).

Mongoose (2021). *Mongoose*. URL: https://mongoosejs.com/ (visited on 05/17/2021).

Moran, Kate (Dec. 2019). *Usability Testing 101*. en. URL: https://www.nngroup.com/articles/usability-testing-101/ (visited on 05/07/2021).

Neswine, Muhammed (2021). *Loading basic HTML in Node.js*. URL: https://stackoverflow.com/questions/4720343/loading-basic-html-in-node-js (visited on 05/19/2021).

Node.js (May 2021). en. Page Version ID: 1021215941. URL: https://en.wikipedia.org/w/index.php?title=Node.js&oldid=1021215941 (visited on 05/05/2021).

Node.js (2021). *The Node.js Event Loop, Timers, and process.nextTick().* en. URL: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/ (visited on 05/05/2021).

Personopplysningsloven (2021). nb. URL: https://jusleksikon.no/wiki/Personopplysningsloven (visited on 05/05/2021).

PuTTY (May 2021). en. Page Version ID: 1022384362. URL: https://en.wikipedia.org/w/index.php?title=PuTTY&oldid=1022384362 (visited on 05/12/2021).

redhat (2021). *What is a REST API?* en. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (visited on 05/07/2021).

Rehkopf, Max (2021). *User Stories.* en. URL: https://www.atlassian.com/agile/project-management/user-stories (visited on 05/04/2021).

Rekhi, Sachin (Feb. 2018). *Don Norman's Principles of Interaction Design.* en. URL: https://medium.com/@sachinrekhi/don-normans-principles-of-interaction-design-51025a2c0f33 (visited on 05/06/2021).

REST (May 2021). en. Page Version ID: 1020836910. URL: https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1020836910 (visited on 05/06/2021).

rfc6455 (2021). URL: https://datatracker.ietf.org/doc/html/rfc6455 (visited on 05/12/2021).

Schwaber, Ken and Jeff Sutherland (2020). *Scrum Guide.* URL: https://scrumguides.org/scrum-guide.html.

Sommerville, Ian (2016a). "25 - Configuration Management". In: *Software Engineering.* 10th edition. Pearson, pp. 730–755. ISBN: 1-292-09613-6.

— (2016b). "3.3 Agile project management". In: *Software Engineering.* 10th edition. Pearson, pp. 84–88. ISBN: 1-292-09613-6.

— (2016c). "Glossary". In: *Software Engineering.* 10th edition. Pearson. ISBN: 1-292-09613-6.

What is Stateless Authentication ? (2021). *Stateless Authentication.* URL: https://doubleoctopus.com/security-wiki/network-architecture/stateless-authentication/ (visited on 05/07/2021).

Synopsys (2021). *Compare Repositories - Open Hub.* URL: https://www.openhub.net/repositories/compare (visited on 04/28/2021).

Unit testing (Apr. 2021). en. Page Version ID: 1017740600. URL: https://en.wikipedia.org/w/index.php?title=Unit_testing&oldid=1017740600 (visited on 05/07/2021).

User experience (Apr. 2021). en. Page Version ID: 1020620068. URL: https://en.wikipedia.org/w/index.php?title=User_experience&oldid=1020620068 (visited on 05/12/2021).

WebSocket (May 2021). en. Page Version ID: 1021899806. URL: https://en.wikipedia.org/w/index.php?title=WebSocket&oldid=1021899806 (visited on 05/12/2021).