NTNU
Norwegian University of
Science and Technology

The Vent Learning Analytics Dashboard

and VSON container format for visualization data

Nils Herde

Master of Science in Informatics
Submission date: June 2018
Supervisor: Michail Giannakos, IDI
Co-supervisor: Boban Vesin, IDI
Katerina Mangaroska, IDI

Norwegian University of Science and Technology
Department of Computer Science

Summary

Education is increasingly more digitized, and dispersed on several platforms, devices and applications. Inspired by domains like business intelligence; learning analytics is becoming increasingly prevalent as a means to improve and adapt the educational field. However, challenges arise when trying to collect and analyze data originating from various sources, as well as when trying to extract meaningful insights from large amounts of data. This thesis documents the development of Vent, an educational dashboard meant to visualize data in an attempt to aid educators in their work, and highlights some of the challenges tied to integrating highly divergent data sources in a coherent manner. The thesis also introduces the VSON data container, conceptualized explicitly to carry data meant for visualization purposes.

Sammendrag

Utdanning blir stadig mer digitalisert, samt spredt utover flere plattformer, enheter og applikasjoner. Inspirert av domener som virksomhetsetterretning har læringsanalyse blitt et stadig viktigere verktøy undervisere kan ta i bruk for å forbedre og tilpasse undervisningen. Det oppstår likevel utfordringer i forbindelse med innsamling og analyse av data fra disse mange kildene, samt i prosessen med forstå essensen i potensielt svært store datamengder. Denne oppgaven dokumenterer utviklingen av Vent, et verktøy for visualisering av undervisningsdata som forsøker å bistå undervisere i deres arbeid, i tillegg til å fremheve en del utfordringer i forbindelse med integrering av meget varierte data på en enhetlig måte. Oppgaven introduserer også det nye VSON-dataformatet, som er spesifikt utviklet for å bære data ment for bruk i visualiseringer.

Preface

This thesis was written during the school year of 2017/2018 for the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). It is the final stretch of a six year journey I have traveled with many fine people without whom this would not have been possible. Not least, my family that has been of immense support, especially these last couple of months.

I would like to thank my supervisor Michail Giannakos as well as my co-supervisors, Boban Vesin and Katerina Mangaroska for their aid and support during my work with this thesis. I hope that the thesis will prove useful for them in their continued quest of improving education for all students.

A special thanks to Annie Aasen for support, proofreading, and for simply being quite awesome.

2018-06-01

Nils Herdé

Table of Contents

List of Figures

Chapter 1

Introduction

This thesis chronicles the development of the Vent[1] educational dashboard from initial conception, through the changed focus triggered by development challenges, all the way to its current state as of spring 2018. In this chapter I will present the background for the development project, the initial objectives of the development process as well as the additional objectives redefined during the development. Finally I will present the outline of this thesis.

1.1 Background and motivation

The origins for this thesis was proposed by my supervisor Michail Giannakos and his team consisting of Boban Vesin and Katerina Mangaroska. Having researched educational dashboards they identified that the way forward in understanding education is in looking at the complete picture instead of individual aspects of a complex educational picture (Mangaroska and Giannakos (2017)). In other words, they wanted to develop software that could gather data from several data sources and combine them into holistic visualizations.

Giannakos is the course supervisor of IT2805 - Web Technologies at the Norwegian University of Science and Technology (NTNU). In this subject they use software by the name of ProTuS. ProTuS is by its own definition "a tutoring system designed to help learners in learning essentials of programming languages. The environment is designed for learners with no programming experience " (Vesin (2018)). ProTuS is explicitly crafted to track and monitor user behaviour on the site as to generate data for later analysis. ProTuS also incorporates Mastery Grids developed at the university of Pittsburgh for some of its features, which also tracks and analyses data as to allow for adaptive learning (Sahebi (2018)).

Data from ProTuS and Mastery Grids together with NTNUs learning management system Blackboard (Blackboard (2018)) were to form the basis for such a multi-data educational dashboard.

[1] Visualized Education NTNU

1.2 Contributions and objectives

Based on the context described above, the aim of this thesis seemed somewhat straight forward. Create a system that could collect data from the mentioned data sources and convey this information through useful and informative visualizations.

The exact visualizations to create would be elicited through dialog with professors responsible for the courses from which data would be gathered from. The resulting list which can be seen in appendix A.

1.2.1 Initial goals

The initial goals for this thesis can therefore be summed up with the following steps:

1. Decide on a set of technologies flexible enough for interaction with multiple data sources
2. Find sensible and smart ways of pulling data from said data sources
3. Elicit desirable visualizations from course professors
4. Extract data needed to create visualizations
5. Display visualizations in a sensible way

The end result of this would be a made-to-measure dashboard specifically tailored to the needs of Giannakos and his team.

1.2.2 Changing focus

As mentioned, in addition to recounting the development of this educational dashboard, this thesis is partly responsible for chronicling why and how my focus shifted somewhat during this process. More on that in chapters 3 and 4. In short the various challenges associated with having to work with data sets that had no particular standard, missing documentation and generally were not at all generated with visualizations in mind, leads me to propose some solutions to help facilitate visualizations of educational data in a more general way. This in turn resulted in the addition of a few goals:

1. Extract common denominators for data needed to create the requested visualizations mentioned in appendix A
2. Find a suitable way of containing this data
3. Express this container format in a general and reproducible way
4. Create a visualization dashboard built around this new data format

The results of the first three of these steps are what will be referred to as the VSON data format, and is explained in detail in chapter 4. Note that this chapter has been written as to be understandable on its own for those that should not be interested in the rest of the thesis.

1.3 Thesis structure

Chapter/Appendix	Description
1. Introduction	A short introduction to the background and goals for the project
2. Background Theory	Some background theory and related work
3. Implementation	The implementation of the Vent educational dashboard
4. VSON	An introduction and explanation of the VSON data format developed for this thesis
5. Results	The results of the development process in terms of visualizations and goals achieved
6. Discussion & future work	Discussion of the results achieved and the way forward
A. Elicited visualizations	A list of the visualizations requested by professors Michail Giannakos and Hallvard Trætteberg
B. Vent Readmes	The readme files created for the Vent educational dashboard
C. VSON Schema	The schematic definition of the VSON data format
D. Hoov	The Hoov application for data gathering that was developed but never used during the project

Table 1.1: Overview of the thesis structure

Chapter 2

Background and related work

This chapter will briefly explain some of the background for this project, as well as look into some related work. Initially this thesis was intended as a made-to-measure project with a targeted focus strictly accommodating the inceptive specifications. As the project progressed, the scope shifted slightly as I will explain later in the thesis. Not aiming to be a complete literature review the initial scope of this thesis was limited to the work performed by Giannakos and his team. The experiences made during the project did however require some re-alignment with the *status quo* of data sharing in educational research, and hopefully the end result will end up having a positive effect on the field of visualizing learning analytics.

The chapter is divided into sections discussing the current state of (digitized) education and learning analytics, educational dashboards and multimodal learning analytics, before introducing The Experience Api (xAPI) (Rustici (2018)).

2.1 Digitized education

Not that long ago, education mostly circled around the textbook, the classroom and the occasional quiz or exam. Of course there have been elements of television, excursions, experiments and group work for decades or even centuries. However the digitization of all things has also included the field of education (Schroeder (2009)), and what is certain is that the educational circle no longer revolves around a few select entities (Muñoz-Cristóbal et al. (2016)). A modern educational environment involves a multitude of devices and applications spanning everything from personal mobile phones to powerful work stations, and simple text editors to fully fledged LMSs (Schroeder (2009)).

2.1.1 Learning Analytics

Common for these devices and applications is that they allow for unprecedented logging and data collection. Registering time spent on individual resources or how a student types his or her essay is not only feasible, but often quite trivial to accomplish. Experiences from

other fields where the digitization has opened the gates on big data analysis and process optimization suggest that ways of improving education might be found in these large sets of educational data (Muñoz-Cristóbal et al. (2016)). The challenge is not only in analyzing and understanding the data, but also in the process of gathering it in a sensible form from its multiple sources. The large variety of applications and ecosystems introduces challenges with regards to this process. An understandable egocentric perspective fueled by deadlines, lack of information, or even sometimes malicious intents of purposeful lock-in can make data-sharing an afterthought in the development process.

Integrating, analyzing and understanding the vast amount of information available with regards to education will not only be important to individual educators and students, but also to policy makers and institutions on various levels. But still there are many challenges associated with this, like data privacy issues, scoping and determining what data is pedagogically meaningful and to what people or instances (Johnson et al. (2011); Elias (2011)).

2.2 Dashboards

Many definitions exist for (educational) dashboards. According to Few (2006) an information dashboard is "a visual display of the most important information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance". Yoo et al. (2015) in turn, defined a learning dashboard as "a display which visualizes the results of educational data mining in a useful way". A summary of more definitions can be found in the literature review by Schwendimann et al. (2017), together with their own definition that defines a learning dashboard to be "a single display that aggregates different indicators about learner(s), learning process(es) and/or learning context(s) into one or multiple visualizations."

However, the usefulness of displayed information on a dashboard varies from use case to use case. An altimeter is probably more essential on the airplane dashboard than in the car. To put this in context with Vent, in our use case the dashboard will not first and foremost be a so called 'always on', glanceable display. Rather Vent will cluster multiple sets of information into manageable sub-views as to not overwhelm the viewer with too much information. Therefore it could be argued that Vent oversteps the dashboard definition by offering multiple views, however I see few reasons to get caught up in semantic nit-picking. For all intents and purposes Vent is a learning dashboard.

2.2.1 Dashboard for multimodal learning analytics

Where Vent initially would differentiate itself is in the way it combines multiple data sources to give useful information to the educator. Schwendimann et al. (2017) found that more than half of the researched dashboards only relies on one data source. And according to Mangaroska and Giannakos (2017) "learning is becoming more blended and distributed across different learning environments and contexts, making it impossible to holistically understand the process of learning if integration is neglected". So much so that they "highlight the importance of learning analytics integration and aggregation of learning-related data across multiple sources for designing informed and optimal learning strategies". In

other words, given the multitude of software and hardware found in education as described in 2.1, looking at isolated sources of data will yield very few true insights into the overall state of an educational context. The integration of multiple data sources is also one of the challenges highlighted by Dyckhoff et al. (2012), and it is here that Vent set out to differentiate itself.

2.3 The Experience API

"The Experience API (or xAPI) is a new specification for learning technology that makes it possible to collect data about the wide range of experiences a person has (online and offline)." Rustici (2018)

During the development of Vent, it became obvious that a standard vocabulary to talk about distributed educational data was needed. The details will be discussed in section 3.4.2 and chapter 4. xAPI aspires to be that vocabulary by using a student-centered approach build on current web technologies (Kevan and Ryan (2016)). xAPI describes learner action in so called "Activity Statements" containing a minimum of three properties: "Actor", "Verb" and "Object". Not unlike the Resource Description Framework[1] model known from semantic web technologies which uses subject, predicate, object (Lassila et al. (1998)) An example would be Annie (actor) read (verb) The Hitchhiker's Guide to the Galaxy (object), or Stephen (actor) completed (verb) the midterm (object). Presumably using unique identifiers for actors and objects. xAPI predefines certain additional properties containing the likes of contextual or assessment data. However it is inherently extensible as to accommodate unforeseen data collection needs (Kevan and Ryan (2016)). These activity statements are generated by each individual educational application and transmitted to one or more central learning record stores (LRS). In an effort to promote flexibility, the xAPI maintainers removed its core verbs and left it up to the community to figure out what verbs are best suited in the move from version 0.9 to version 0.95.

Together with its very broad actor, verb, object model, this makes xAPI a very comprehensive and flexible language well suited for most use cases regarding transfer and storage of educational data. The lack of specificity does however allow for miscommunication like two data sources using different verbs to mean the same thing. For example using finished, ended and completed interchangeably, essentially segregating similar data points into different verb-actions. This could be mitigated somewhat using common techniques for natural language processing. Surrounding xAPI are also guidelines for secure communication and a few other (helpful) elements that to some extent adds to its complexity.

In my oppinion, xAPI seems like an overall well thought out standard that should fit all but a very select few use cases for talking about educational data. It is however worth noting that xAPI spans educational data in general and is not specifically tied to the visualization of educational data. More on that in section 4.4.1.

[1] https://www.w3.org/RDF/

Chapter 3

Implementation

This chapter will present the implementation details of Vent. Vent stands for 'Visualized Education NTNU' and consists of two main parts. The data-processing application built using Spring Boot[1] serving as the compatibility layer between data source and presentation layer, and the web application based on Vue.js[2] and Highcharts[3] serving up the visualizations to the end user. I will present the architecture and implementation details, as well as the reasoning behind each component in the technology stack. Lastly, I will discuss challenges encountered during development.

3.1 Requirements and reasoning

The main requirement that influenced the decision process with regards to the technology was the need to easily integrate with multiple data sources. Choosing the correct framework when integrating with existing systems can be the difference between a day's or a month's worth of work. Knowing that the data sources would be accessed through vastly different means, the emphasis was put on finding a flexible framework that could handle that. At that time the two main data sources that would be integrated was to my knowledge:

- ProTuS - by direct access to the MySQL database
- Blackboard[4] - presumably via some sort of HTTP based API like REST or SOAP

Knowing this, and based upon previous experience, the following main points were laid down as requirements for a back end framework:

[1] https://spring.io/projects/spring-boot
[2] https://vuejs.org/
[3] https://www.highcharts.com/
[4] NTNUs new LMS

- A powerful ORM tool[5]
  - Avoids having to write manual SQL-queries for database interaction
  - Very flexible when working with multiple sources of data
- Modern HTTP libraries
- Easy exposure of data as JSON representations
- Good integration with an IDE[6]
  - Can save a lot of time when developing and refactoring an application

For the graphical interface the choice of creating a web based application was a fairly easy one. Barring the need for higher than average performance or the need to access specific low-level APIs like virtualization support or similar, the advantages of a web application are numerous. Some of these are:

- Total platform and device independence
- Unparalleled accessibility through all connected devices
- A vast amount of libraries and tools for both code and graphical elements
- Great IDE support

Both with regards to back and front end the amount of available frameworks, libraries and tool sets are overwhelming. A detailed analysis of all options is not within the scope of this thesis, and the decisions were made based upon conversations with the supervisor as well as personal preference. The technology stack will be presented in the next section.

3.2 The back end

3.2.1 Spring Boot

Spring Boot is a ”convention over configuration” approach to minimize configuration and fuzz as to just get up and running with the Spring framework as efficiently as possible. The Spring framework is a popular and widely used Java-based application framework that has been around for more than 15 years. One of its key features is its support for all Java-based data access frameworks like JDBC[7], Hibernate[8] and JPA[9]. This makes Spring a superbly flexible framework when it comes to working with different data sources. When the data sources are configured as desired, the data is exposed as POJO (Plain Old Java Objects, or in this case POKO as Kotlin is used) to the application and any operations on

[5] https://en.wikipedia.org/wiki/Object-relational_mapping
[6] Integrated Development Environment
[7] http://www.oracle.com/technetwork/java/javase/jdbc/index.html
[8] http://hibernate.org/
[9] https://en.wikipedia.org/wiki/Java_Persistence_API

the objects are mirrored to the database or databases. This avoids manual writing of SQL-queries which is both time consuming and prone to breaking when handling edge-cases. Especially when modifying or refactoring the application as most IDEs do not work too well with what is often plain text queries.

Other benefits of using Spring Boot is that through its widespread use there are large amounts of documentation, blog posts and support tickets with valuable information, especially useful for a solo developer. Being Java-based also opens the door to a plethora of libraries of all kinds. More on this in the Kotlin section (3.2.2).

MySQL

Seeing as Spring Boot abstracts operations on the database system itself, the decision to pick a database system for Vent was not terribly impactful. Given that one of the data sources (ProTuS) itself uses MySQL and that it is the default in a vanilla Spring Boot configuration, I decided to use that. However replacing the database layer with PostgreSQL, MSSQL, or even NoSQL databases like MongoDB or Neo4J, should be fairly trivial as all of these are supported in conjunction with Spring Boot.

3.2.2 Kotlin

As previously mentioned Spring Boot, is a Java-based framework with all the pros and cons associated. Among these cons are critiques[10] of excessively verbose code that can negatively impact readability, and the widespread problem of Null Pointer Exceptions[11].

Kotlin is a programming language that explicitly addresses these shortcomings including others. While not being the only programming language trying to achieve this, what separates Kotlin from most of the others is that it can be compiled to run in the Java Virtual Machine. This allows for 100% interoperability with Java-based frameworks and systems. Kotlin version 1.0 was released in February 2016, however the language has been around since 2011 and is therefore not as young as one might believe. In 2017 Google announced first-class support for Kotlin on the Android platform, which is a testament to its maturity. Tempted by these advantages and the relatively low risk, I decided to give Kotlin a try in the preliminary phase of the development. I was happy enough with the results that I decided to continue writing in Kotlin.

3.3 The front end

The assumption made concerning the front end part of the application was that this would be a fairly thin layer without too much complexity. However Vent is hopefully going to see continued development in the future and it seemed a lack of foresight to not plan for future growth. The goal was therefore to find tools with low overhead to allow me to quickly get started, but that would still be powerful enough to scale as needed without having to be replaced at a later point.

[10] https://softwareengineering.stackexchange.com/questions/141175/why-is-verbosity-bad-for-a-programming-language

[11] https://en.wikipedia.org/wiki/Criticism_of_Java

As the main goal of the front end is to display visualizations of educational data, finding a good tool to facilitate this was therefore key. This was a field wherein I did not have a lot of experience, and it was thus the single item in the technology stack where most of the research time was spent.

3.3.1 Vue.js

The nature of the project having many charts and visualizations with different values seemed like a perfect fit for a component based approach. In short, a component is a small, self-contained part of ones code that handles logic, templating and styling for itself. Somewhat comparable to an object in the object-oriented paradigm, a component is instantiated by a parent component with parameters creating a tree-structure. See figure 3.1 for illustration.

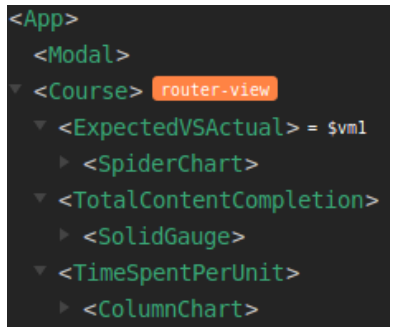

Figure 3.1: Vue components in nested tree structure

Several frameworks offers this kind of functionality, with the most popular right now being React[12]. However personal preference made me look for an alternative and after some research I landed on Vue.js. According to their documentation[13] one of its key features is that Vue scales down just as well as up. This means that the threshold to get started with Vue.js is low, and that the framework should be able to fit a growing application size with ease.

3.3.2 Highcharts

Many JavaScript charting libraries were researched and considered for this project. Some of these were Google charts, Chartlist.js, D3, and Charts.js. In the end the choice fell on Highcharts. Highcharts excelled in particular in two categories. In the variation of charts offered, which was important at a point were I knew little about what visualizations were going to be used. And the substantial amount of exhaustive documentation provided for the library, adding obvious value to its use. In addition, Highcharts seemed trivial to integrate with Vue.js. These factors made Highcharts a very good match for my use case.

[12]https://www.npmjs.com/npm/state-of-javascript-frameworks-2017-part-1
[13]https://vuejs.org/v2/guide/comparison.html#Scale

Note that replacing the charting library should not only be possible, but the very modular nature of the project should facilitate such a change.

3.4 Architecture

In this section I will have a look at the architecture and what considerations were made to shape the final design of the application. It is in this section that I discuss the challenges that spurred the need for a standard format expressing educational data for visualization purposes discussed in chapter 4.

3.4.1 Reasoning and early iterations

The initial flow of the Vent application was quite simple. Data from various sources would be collected, and processed in some fashion before finally being visualized in a graphical manner. This description coincides very well with the separation of layers associated with a Model-View-Presenter[14] architectural pattern. MVP is a derivation of the more known Model-View-Controller pattern, where the view and model layers are strictly separated by the presenter (controller) layer. In the initial architecture design I therefore sketched out a fairly straight-forward implementation of a layered MVP architecture, where different data sources could also be horizontally separated, as shown in figure 3.2

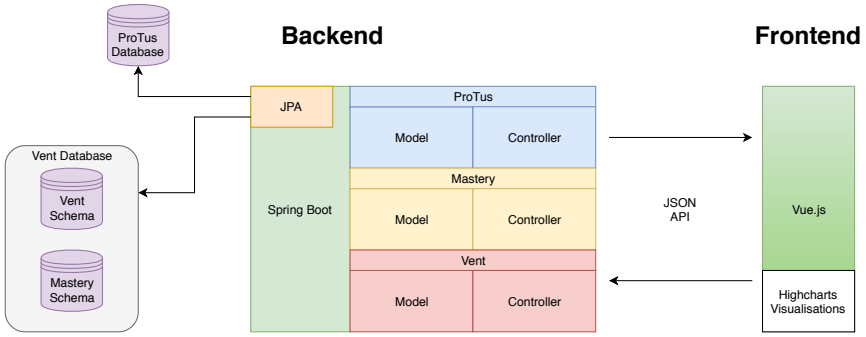

Figure 3.2: Early draft of the architecture.

This turned out to be a robust base to build on, with a lot of reasonable assumptions and decisions that later fascilitaed my development work. However, further work uncovered some challenges that required me to rethink parts of the design. Or rather, I discovered advantages to exploring a more general approach.

3.4.2 Factors and challenges affecting the architecture

During the development of Vent, there has been a total of four different data sources discussed for implementation. To illustrate the effect they had on the architecture I will look at each one separately.

[14] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter

Blackboard

Blackboard Learn is the learning management system used at NTNU starting from 2017[15] and was initially intended to be one of the data sources used for the visualizations. However in late November 2017 the team (myself and my supervisors) were finally denied access to the data in a time frame suitable for this thesis. Although this meant that Blackboard would not be integrated into the initial version of Vent, the goal has always been to allow for its integration at a later point in time. Blackboard exposes its data as a JSON API[16], which in turn implied that the system would need to be able to pull from, and presumably cache, data from such an API.

ProTuS

ProTuS is a programming tutoring system developed at NTNU, and is currently actively used in a few different classes at NTNU. ProTuS is explicitly designed to collect data and is thus a great source of information for the kinds of visualizations intended to be found in Vent. ProTuS does not currently have a dedicated API, but access was granted to the entire MySQL database. This makes for easy integration with Vent, as Spring Boot handles all connectivity between data store and POJO. However, despite easy connection there were a few caveats.

ProTuS is under active development and its database design shows signs of having been built on bit by bit for years, and a thorough refactoring has not been performed in some time. Thus the database structure is not necessarily ideal or very intuitive to understand for someone not actively maintaining it. Another challenge that I faced was that the database structure of ProTuS was not stable throughout the development of Vent. In figure 3.3, 3.4 and 3.5 you can see a few snapshots of how the database differed between early fall 2017 all the way to spring 2018. Most notably is the removal of many foreign keys working as semantic links that complicates flow and navigation when trying to extract data from the database. Every database change on the ProTuS side implied reworking the ProTuS implementation in Vent. This constant need for adjustment to a changing data source heavily affected the realizations mentioned later on.

[15] https://www.ntnu.no/aktuelt/2016/els

[16] https://developer.blackboard.com/portal/displayApi

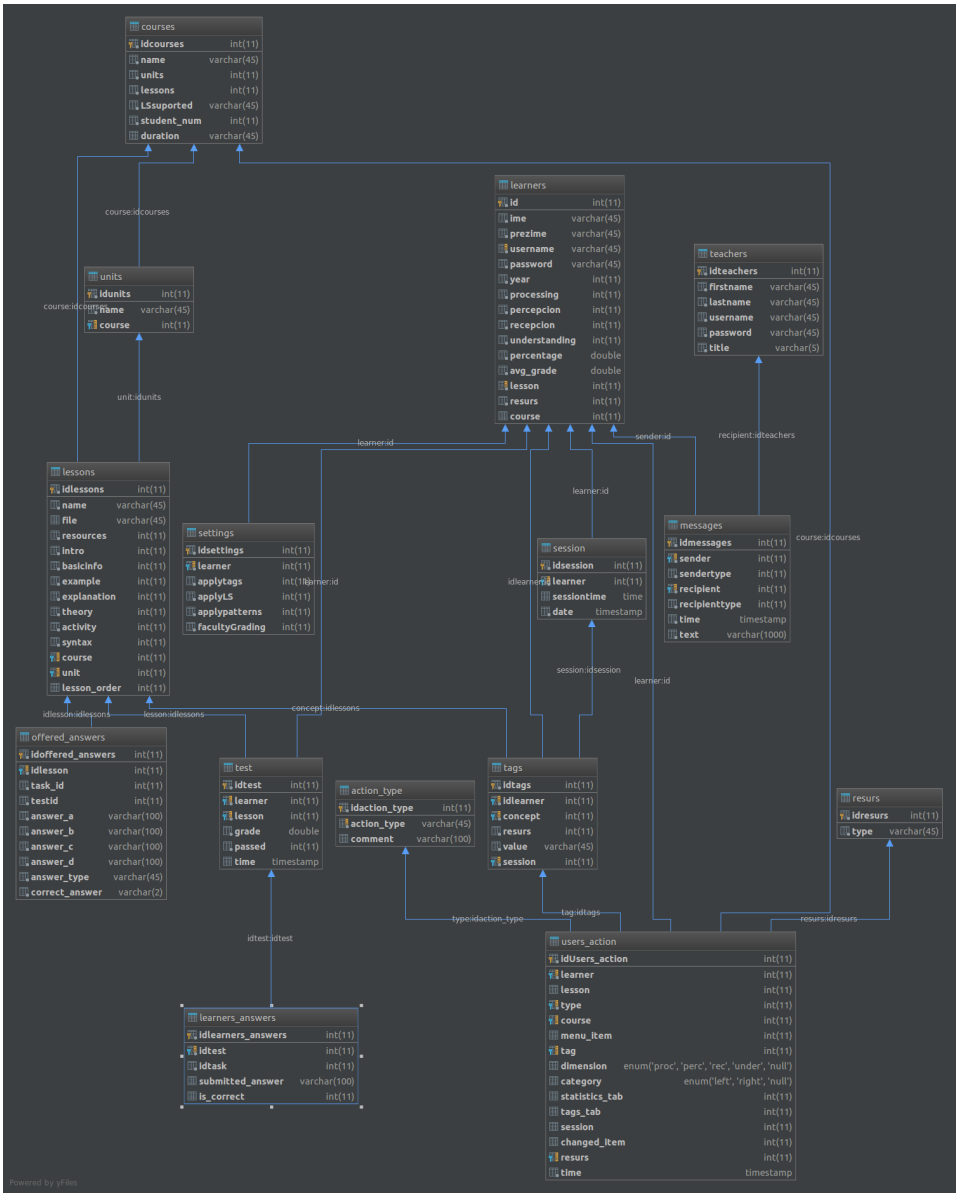

Figure 3.3: ProTuS database design late September 2017

Figure 3.4: ProTuS database in mid spring 2018

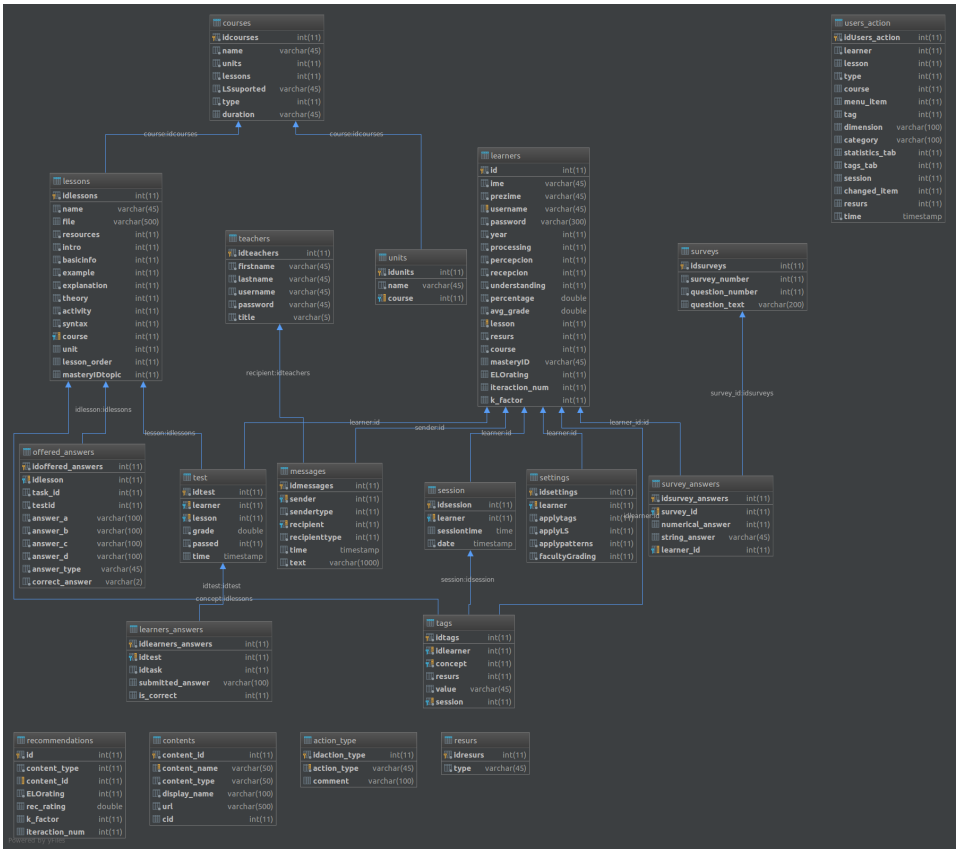

Figure 3.5: ProTuS database in late spring 2018

Mastery Grids

Mastery Grids (Sahebi (2018)) is integrated into ProTuS and supplies a number of tasks and assignment presented to the users of ProTuS. In most cases the user will get a pop-up leading to the Mastery Grids site where they will complete the proposed unit. Mastery Grids mine their own set of data from the user interactions and together with ProTuS were the fundamental data sources planned for Vent when Blackboard integration was delayed.

Mastery Grids provides a public API for their recorded data. The format is JSON-esque in that it is clearly meant to be JSON, but the API returns invalid JSON that breaks most parsers of the format. Another problem was the lack of documentation for this API as properties had nondescript names like ”k” or ”p”. Upon request even the Mastery Grids developers had problems defining what the different data actually is, since a lot of it was implemented by people who have since left the project. Finally, a divergence from convention made the format much harder to parse than one would expect. Normally an array of topics in a JSON-format would look like listing 3.2. However, we see that the data presented in 3.1 has variable data as property keys instead of values, which in the end

resulted in having to write a fully custom parser and test classes for this data instead of being able to plug in an existing one from a parsing library.

```
topics: {
    "Variables and Operations": {
        values: {
            "Examples":{
                "k": 0,
                "p": 0
            },
            "Challenges": {
                "k": 0,
                "p":0
            },
            "Coding": {
                "k":0,
                "p": 0
            }
        }
    }
}
```

Listing 3.1: Actual data received from the Mastery Grids API

```
"topics": [
    {
        "name": "Variables and Operations",
        "values": [
            {
                "category": "Examples",
                "k": 0,
                "p": 0
            },
            {
                "category": "Challenges",
                "k": 0,
                "p":0
            },
            {
                "category": "Coding",
                "k":0,
                "p": 0
            }
        ]
    }
]
```

Listing 3.2: Proper formatting of data according to JSON convention

JExercise

The fourth data source explored for implementation into Vent was JExercise. JExercise is a plugin to the IDE Eclipse, developed by NTNU professor Hallvard Trætteberg, that guides students through their programming exercises. In addition, it tracks a number of metrics regarding the students' progress and programming habits which are to be visualized in Vent.

Initially, useful visualization data from JExercise was to be pushed to a server from each user installation and from there be integrated into Vent. The server was implemented as a simple Node[17] app as per the specifications of Trætteberg for maximum flexibility. The app, named Hoov(er), can be seen in appendix D. However for various reasons the server was never used and data ended up being provided as exported files from JExercise.

The data exported directly from JExercise is in the form of files with a `.ex` extension. These files are in the XMI (XML Metadata Interchange) format specifically intended to be read and processed by the teacher's software. This is a highly specialized format that requires numerous libraries and custom packages related to JExercise to process, which is not suitable for parsing, indexing or browsing by general learning analytic tools (Dodero et al. (2017)). It was never my intention to integrate this fully into Vent as to avoid unnecessary complexity, but rather have a separate service that could receive relevant data from JExercise. Even if Hoov did not end up being used, the idea was still to have a separate service that could do conversion and extract relevant data. This service proved a lot of work, and was never prioritized. Especially as such a service was under active development by another masters student Boye Borg Nygård for his thesis[18].

In other words JExercise did not end up integrated in the initial version of Vent, but should be easier to implement once Nygård's extraction service is finalized. I did however see the challenges posed by highly specialized and somewhat obscure formats.

Summary

In summary we can see how even a modest number of different data sources can exponentially complicate this kind of project development. The unstable nature of some data repositories, the lack of documentation and adherence to standards, as well as the large amount of noise in general data sets that complicate extraction of data relevant for visualizations. These were the recurring challenges that spurred the idea I will take a closer look at in chapter 4.

3.4.3 Final implementation

In this section I will have a look at the final architecture of Vent, as well as go into some details of each layer. Excerpts of code will be presented to illustrate sections, but for complete reference all code is available in the project repository[19].

The architecture closely resembles the one showed in figure 3.2, except that the Vent-specific section has been transformed into an additional layer on top of the modules of each data source. The data source controllers have also been changed to act as conversion layers from source model to VSON-format which is then exposed by the Vent controller. This is a result of the lessons learned from working with various data sources as explained in 3.4.2 and elaborated on in chapter 4.

[17] https://nodejs.org/en/
[18] https://github.com/boyeborg/work-in-progress
[19] https://github.com/hernil/vent

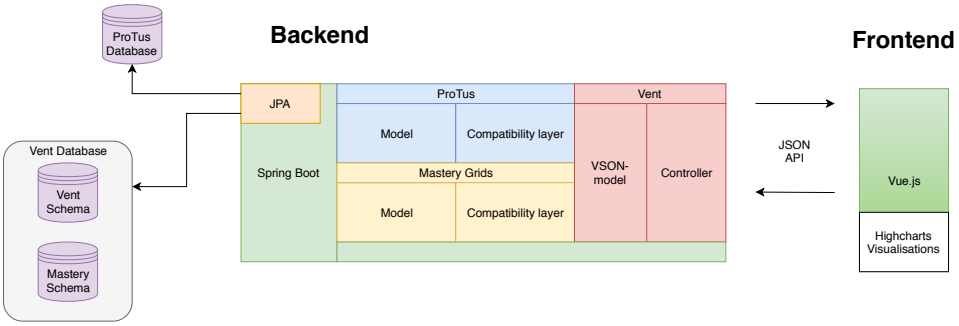

Figure 3.6: Final design of the architecture.

Models

The models consist of Kotlin data-classes that are defined as Entities managed by Spring Boot. Examples of such classes can be seen in listings 3.3 and 3.4. Note that in these examples Kotlin provides `get()` and `set()` as well as `equals()`, `hashcode()` and `toString()` methods out of the box unless explicitly implemented in the classes themselves. In Java, a similar class would probably be upwards of 100 lines long. The different annotations have the following functionality:

- `@Entity` - Defines the class as a JPA entity
- `@Id` - Defines the `did` property as the entity's primary key
- `@JsonIgnore` - Informs JPA to not include this field when exposing the data as JSON
- `@GeneratedValue` - Informs JPA that this is a managed value that it should generate
- `@OneToMany` - Defines a `one-to-many` relation to other classes that will be reflected in the database

Other things that should be noted is the `override` keyword being used because the `CourseUnit` class implements the `Unit` interface as seen in 3.5. Also, all properties are given default values as this is a prerequisite for JPA.

```
1  @Entity
2  data class CourseUnit(
3          @Id
4          @JsonIgnore
5          @GeneratedValue(strategy = GenerationType.AUTO)
6          val did: Long = 0,
7          override val name: String = "",
8          override val type: String = "",
9          override val topic: String = "",
10         override val difficulty: Int = 0,
11         val opened: Double = 0.0,
12         val completed: Double = 0.0,
```

```
        val recommended: Double = 0.0,
        override val timeSpent: Int = 0,
        override val performance: Double = 0.0,
        val expectedPerformance: Double = 0.0
        ) : Unit {

}
```

Listing 3.3: A Kotlin data class complete with getters and setters

```
@Entity
data class Learner(
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private val dId: Long = 0,
        val id: String = "",
        val name: String = "",
        @OneToMany(cascade = [CascadeType.ALL])
        var topics: List<Topic> = mutableListOf(),
        @OneToMany(cascade = [CascadeType.ALL])
        var activityTopic: List<ActivityTopic> = mutableListOf()
)
```

Listing 3.4: A Kotlin data class complete with getters and setters

```
interface Unit {
    val name: String
    val type: String
    val topic: String
    val difficulty: Int
    val timeSpent: Int
    val performance: Double
}
```

Listing 3.5: A Kotlin interface

Compatibility layer

In the compatibility layer, the raw data from a given data source is processed and converted into a data format (VSON as described in chapter 4) that is comprehensible to the Vent front end performing the actual visualizations. The complexity of implementing this layer is directly proportional to the simplicity and clarity of the source data.

Vent layer

Data converted in the compatibility layer is stored as Models in the same way that the source data, except that they are in processed form. This is in some ways a duplication of the data, but can be seen as a form of cache to avoid having to perform the computations of the compatibility layer for each request. If the amount of data should become unmanageable, it is easy to envision a simple purging mechanism for source data caching where the data represented in the Vent layer would become the main data repository. However, depending on the ease of access to the actual source data, keeping the cached data would significantly simplify detailed queries at the relatively low cost of additional storage requirements.

Controllers

The controller in Spring Boot is the part of the application that handles HTTP-requests and from there serves data, performs operations or in other ways react to these requests. For my use case the controllers only serve up data upon request, and thanks to Spring Boot the end result is quite simple. The code seen in listing 3.6 is all that is needed to serve the data for a given course.

The `@CrossOrigin` annotation defines from what origins the requests will be considered valid. In this example I only allow them to come from a local deployment of the Vent front end. In production one would add the server on which the solution has been deployed.

Consider the HTTP Get-request to the URL:
`http://localhost:8080/course/courseId`
`@RequestMapping` defines what URLs should trigger this controller class, in this case `/course` and the `@GetMapping` annotation will trigger the `getDataById` function when appending the url with `courseId`. `@PathVariable` will map the subsequent `courseId` to the `id` variable.

```
@RestController
@CrossOrigin(origins = arrayOf("http://localhost:3000"))
@RequestMapping("/course")
class CourseResource(val repository: CourseRepository) {
    @GetMapping(value = "/{id}")
    fun getDataById(@PathVariable id: String) = repository.findById(id)
}
```

Listing 3.6: A Spring Boot controller written in Kotlin

Front end

The front end is written in JavaScript using the Vue.js framework and Highcharts library. Vue advocates the use of a Component-based development model where one creates atomic self-contained modules that can be instantiated with parameters. Not unlike the core principle of object-oriented programming. Each component handles its own templating, logic and styling based on the data it is instantiated with. In its simplest form a Vue component looks like listing 3.7. In 3.7 the component simply takes in a property called text and displays that text in red. Using this component would look something like listing 3.8.

```
<template>
  <div class="simple-component">
    {{ this.text }}
  </div>
</template>

<script>
  export default {
    name: 'SimpleComponent',
    props: {
      text: {
        type: String,
        default: "Hello World",
```

```
14          required: true,
15        },
16      },
17    };
18 </script>
19
20 <style scoped>
21   .simple-component {
22     color: red;
23   }
24 </style>
```

Listing 3.7: A very basic Vue.js component

```
1 <simple-component :text="Good morning World"></simple-component>
```

Listing 3.8: Using a simple Vue.js component

Each chart type is thus created as a component and can be instantiated with the appropriate data according to the view. A view is simply a collection of components instantiated with some data. Several instances of the same chart type can of course be used in the same view. This approach is not only a pleasant way of organizing code, but forces a very modular approach that makes changing or modifying parts of the application very simple. Creating a new type of visualization and plugin it into the application is simple, and one could also imagine pulling in other charting libraries than Highcharts if something should be missing there. Creating new views that are student-centric, or perhaps for student assistants is a matter of combining existing views and components in the desired way.

3.5 Challenges in development

3.5.1 Technical

Other than data source related problems, there were some purely technical challenges.

Although Spring Boot vastly simplifies the process of getting started with a Spring based application, Spring is still a large and rather complex ecosystem that offers almost infinite configuration options. Getting familiar enough with it to understand the process of, for instance, creating multiple data stores, required a certain amount of time consuming trial and error before succeeding. Although this is expected when exploring new technologies, there is no doubt that having someone familiar with the framework to consult would have been an advantage. All in all, using Spring Boot did still result in time saved by facilitating processes like data management and HTTP control flow.

Choosing Kotlin as a back end language also implied some challenges. This was expected, considering it was a new language that had to be learned. Especially using Kotlin in conjunction with Spring entailed some challenges, as documentation for this combination is somewhat sparser than with traditional Java. All in all, this gave no more trouble than what was expected, and inherent properties of Kotlin actually ended up being very pleasant to work with. For example its much condensed syntax, and its very modern URL library that made working with remote APIs a much more enjoyable process than what I have previously experienced with Java.

3.5.2 Data sources

The challenges related to the data sources have already been pointed out on section 3.4.2. Suffice to say that the various data sources have not been all too pleasant to work with, and frankly quite challenging to integrate. Importing data has spanned from being very simple with the ProTuS database integration, to plain frustrating with the JExercise data that requires numerous custom packages and libraries even to open and process. The lack of documentation for the various properties in the data sets made the creation of the compatibility layer very challenging. Because of the aforementioned challenges, and given that the focus was shifted somewhat towards trying to find more clever ways to work with educational data in visualization context, all of these integrations have not been fully completed.

Chapter 4

Vent System Object Notation

This chapter will present and explain the `Vent System Object Notation` (VSON for short). It will be written as to be understandable on its own so that readers may learn about VSON without needing to read the entire thesis. First I will reiterate some of the challenges that lead to the conception of VSON, then I will describe what VSON is and what problems it does, and does not, try to solve. Lastly I will present the actual schematic definition of VSON before briefly mentioning related work in the form of The Experience API.

4.1 Conception

The conception for VSON was two-fold. For one, any graphical visualization of data would need to process a somewhat stable data format. In the case of Vent that would, at the very least, mean doing a processing of various data sources in the front end in charge of actual visualizations, or rather that the Vent back end presented preprocessed data to the front end. In any case some data carrier format would need to be designed.

In addition, as explained in more detail in section 3.4.2, some challenges in working with multiple data sources were uncovered:

- Unstable - Unstable data sources means that when the source data format changes, the visualization application has to be modified to account for said changes.
- Undocumented - Most data sources had little, or no, documentation. This made it hard to understand what properties map to what behaviour.
- Noise - The data sources contain a lot of data not necessarily linked to actual behaviour needing visualization. This makes it harder to process, ads overhead and can also be a source of leakage of unwanted information.

From these experiences came the idea of creating a unifying format that contains the data needed for basic educational visualizations. (For other efforts in this area, refer to section 4.4)

4.2 What is VSON

VSON is a very concrete solution to a very specific problem. It is *a minimal standard for representing data specifically intended for graphical visualizations of educational data*. Nothing more, nothing less. Underlying this is the idea that a small, highly specialized and targeted set of tools is better than a large, monolithic one that tries to do everything at once. That is why *VSON is **not** a general format for representing educational data*, but rather a higher order abstraction of such data meant for visualization purposes. The name VSON is a nod to JSON (JavaScript Object Notation) and stands for Vent System Object Notation. Vent (Visualized Education NTNU) is the visualization tool that VSON was initially developed for.

4.2.1 Simple and minimal

VSON is meant to do one thing, and to do it well. Keeping VSON minimal is in part to make it easy to understand and to use. Having to read hundreds of pages of documentation not only deters many from using something, it also inevitably leads to some confusion. That is why there, by design, is a very limited set of properties defined by VSON specifically meant to be used in the context of visualizations of educational data. For example, there is a `timeSpent` property because time spent on a task or assignment has been determined to be relevant for visualization. There is no `phoneNumber` property as VSON is not meant to power a phone book.

4.2.2 Standardized and self documenting

When creating a virtual learning environment[1] or educational system, there will probably come a point when visualizing user interaction with the learning material might become useful or even necessary. Instead of trying to figure out what data is useful for such visualizations, VSON offers a standard format that covers a number of visualizations. Data represented as VSON can even be plugged right into Vent for visualization.

Knowing exactly what data is needed, and how to represent it saves a lot of work. Each property defined in VSON is typed, exemplified and described so as to act as documentation for a developer. VSON-data can even be validated as discussed in 4.2.4.

4.2.3 Stable and extensible

Using a stable format for data delivery has advantages both for the visualization tool and for the data provider. The developers of the visualization tool can focus on developing features and new ways to visualize information, knowing that the data being processed will be recieved in a stable form. The data provider can do all sorts of refactoring and changes to the data store knowing that as long as data is still available through stable API endpoints serving VSON-formatted data, the visualization tools will not break.

VSON is meant to be stable, but not rigid or inflexible. VSON does not prohibit additional properties, and only a very select few properties are required. Omitting data

[1] https://en.wikipedia.org/wiki/Virtual_learning_environment

will naturally result in sparser visualizations. So, if your visualizations would benefit from additional information feel free to add the proper properties. However, do consider whether your additions might be generalized and potentially add value for others. In which case do consider extending VSON itself by contacting the maintainers[2].

4.2.4 JSON-Schema

VSON is defined through a JSON-schema[3] as seen in appendix C. JSON-schema is a proposed IETF[4] specification[5] to describe, annotate and validate JSON documents. It is to JSON what XSD (XML Schema Definition) is to XML and tries to bring the advantages of strict validation and annotation to the more human-readable world of JSON. The VSON schema is used as documentation of the VSON data format, and also functions as a validator of VSON data.

JSON was used for VSON as it has already established itself as a standard for data transfer on the web, and because it, in my biased opinion, strikes a good balance between human and machine readability.

4.3 Schema

```
{
  "course": {
    "id": "TDT4100",
    "name": "Object-oriented programming ",
    "units":
    [
      {
        "name": "testassignment",
        "type": "excercise",
        "topic": "some_topic",
        "difficulty": 5,
        "opened": 0.5,
        "completed": 0.5,
        "recommended": 0.5,
        "timeSpent": 3600,
        "performance": 0.6,
        "expectedPerformance": 0.85
      }
    ],
    "students":
    [
      {
        "id": "sha256",
        "performance": 0.5,
        "units":
        [
          {
```

[2] https://github.com/hernil/vson
[3] http://json-schema.org/
[4] Internet Engineering Task Force: https://www.ietf.org/
[5] http://tools.ietf.org/html/draft-handrews-json-schema-01

```
28              "name": "testresource",
29              "type": "task",
30              "topic": "some_topic",
31              "difficulty": 4,
32              "opened": true,
33              "completed": false,
34              "recommended": false,
35              "timeSpent": 3500,
36              "performance": 0.7
37            }
38          ]
39        }
40      ]
41    }
42  }
```

Listing 4.1: Example VSON-file

For the actual schema definition see appendix C or the project repository[6].

4.3.1 Course

The schema describes a data container of type course. A course is the top level of a VSON file. A course has an id which should be unique. The subject identifier is probably a safe choice. In addition it has a human readable name, a list of units and a list of students.

4.3.2 Units

A unit is defined as a part of the course that the student interacts with, and that can have metrics associated with it. Examples of these are,

- A written resource a student should read, or a video to watch
- A small quiz or task to be performed
- An exercise, paper or midterm

For flexibility, I have not tried to define all possible types of units one could imagine interacting with. This also lets the educator decide on what resolution to work with. For example whether to view a midterm as a whole, or wanting to visualize each question or section separately. Also whether it is interesting to distinguish between a written and a video resource. Other properties a unit has should be well defined in the schema itself. There are very few properties that are required as not all metrics can be tracked for all types of units, however omitting data will of course affect the visualizations.

Note also that there is a difference between course units and student units. Course units are *an average* of all students' interaction with a unit. So while the `opened` property of student unit will be a boolean, it will be a numeric value between 0 and 1 for the same course unit. The reason for the duplication of data is two-fold. It adds flexibility in that general visualizations on the data are easily implemented, while advanced filtering on the more precise data is still available. Most use cases of VSON probably will not even use the `students` property in the beginning as skipping it greatly simplifies the implementation.

[6] https://github.com/hernil/vson

It also helps human readability immensely by acting as a summary of the complete data set, which has been determined to be desirable in and of itself.

4.3.3 Other notes

The student ID

The student object has an id. It is a good practice to hash or otherwise anonymize that id as to avoid information leakage from a public API. For example it is a bad idea to expose a student's performance metrics identified by their school email.

Avarages

As with the averages calculated in the course units from the total student units, the student object has a calculated average for the `performance` property. The same reasoning applies for that as for the course units. Performance was chosen as it is perhaps the single most interesting metric. In addition, averaging time spent and other metrics of units as diverse as small reading assignments and midterms make little sense. Exactly how the performance metric is calculated is up to each data provider, however the data should be normalized to a percentage (i.e. a value between 0 and 1).

The dimension of time

VSON data has no concept of time (except the `timeSpent` property of course). That means that if one should desire some sort of time line for the development of the data sets, one would have to provide snapshots of VSON data at the desired resolution. Thus allowing for comparisons between snapshot X and Y. This decision was made for simplicity and clarity. The more features are built straight into the container format, the harder it is to understand and implement.

4.4 Related Work

This short sections presents how The Experience API (xAPI) relates to VSON. For a more comprehensive look at related work with regards to Vent as a whole, see chapter 2.

4.4.1 The Experience API

"The Experience API (or xAPI) is a new specification for learning technology that makes it possible to collect data about the wide range of experiences a person has (online and offline)." Rustici (2018)

xAPI is a complete specification that aims to standardize the way educational data is generated, transferred and stored. As discussed in section 2.3 it is flexible and well thought out to accommodate a vast amount of current and future educational scenarios with its simple syntax of Actor, Verb, Object triplets. However, being a general tool for educational data, xAPI has complexities and overhead that complicates its use as a carrier of visualization data. Therefore xAPI is not strictly an alternative to VSON, but rather a potential

foundation on which VSON could be applied on top to allow for easy visualizations of xAPI data. This is, in my opinion, the obvious next step in allowing VSON to quickly cover a substantial amount of educational data sources and should be a priority in the next phase of its evolution.

Chapter 5

Results

Most results from this thesis are grounded in actual code produced when creating the Vent back and front end, as well as the specification for the VSON data format described in chapter 4. However I have created some proof-of-concept visualizations using components I have developed. These are presented in the next section. Lastly some notes about the results with regards to the goals mentioned in section 1.2.

5.1 Vent

In the initial version of Vent I have created components for a few chart types that seemed the most fitting starting point for the visualizations requested by the professor(s) upon query (see appendix A). They are presented in the following section. First are the components currently used in views, then comes components that are finished and could easily be included in future views but that have not yet been used as of writing. So far, the logic and configuration options of the Vent front end have been kept to the bare minimum as to not overly complicate things. Considering the loose coupling to the back end, and the fact that VSON data by design should not be sensitive there are no concepts of users, login or preferences. In the future it could be conceivable to imagine these features to let individual professors configure their preferred views for a course, but for now that has not been a priority.

Figure 5.1 shows the initial, and only, prompt displayed to the user when first navigating to the Vent dashboard. Vent is a data visualization tool and therefore needs data to visualize. This fetching of data is not performed automatically as the loose coupling from the back end suggests that you should be able to load data from any source. However there is a suggested default that can easily be changed by modifying the `defaultVSONUrl` variable so as to make loading data a one-click task most of the time.

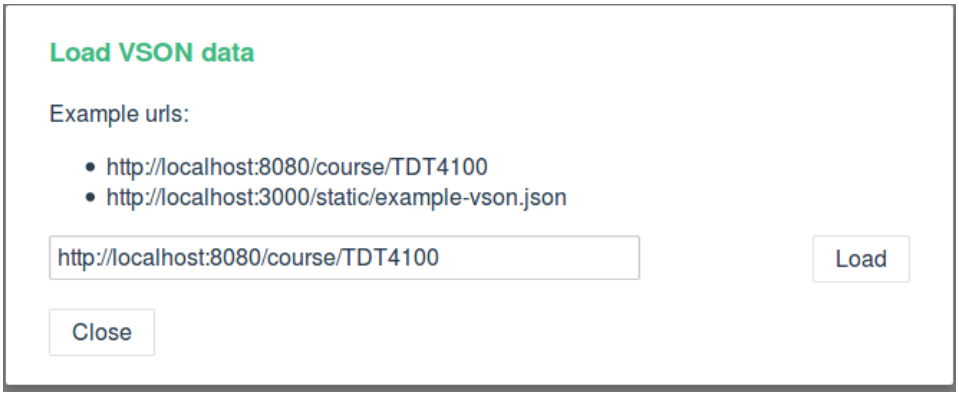

Figure 5.1: The initial prompt when opening Vent

Used components

These components are in active use in views existing in the initial version of Vent.

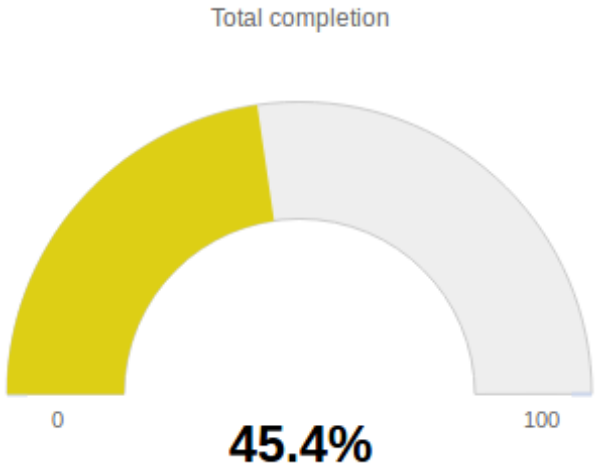

Figure 5.2: A component of the *solidgauge* chart type

The *solidgauge* chart type has been implemented as a component showing a progress from 0 to 100%. Here it is used to illustrate the total completion rate of all course content. Solidgauge is the most prominent graph type in the initial Vent because if its very simple nature.

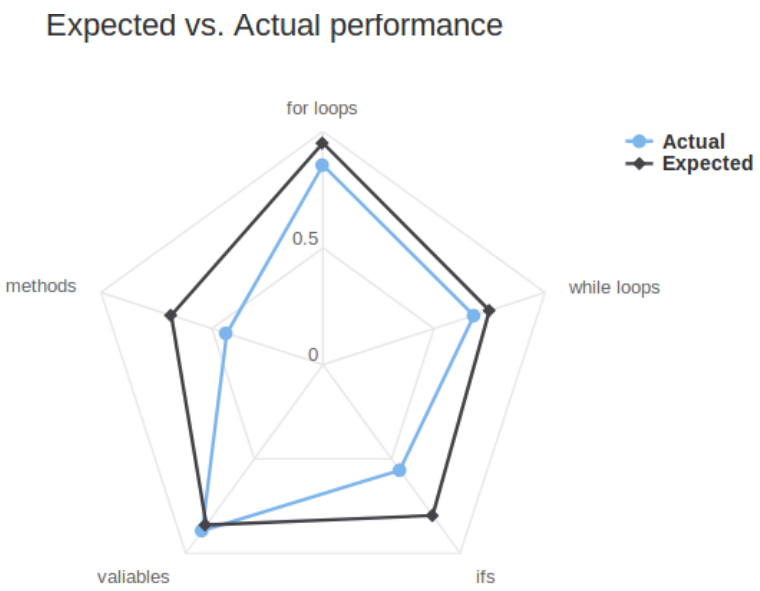

Figure 5.3: A component of the *spider* chart type

The spider, or radar as it is sometimes known, chart type was one that was specifically suggested when brainstorming how to illustrate various data sets. In this example it is used to compare actual performance to the expected performance goals set for each topic.

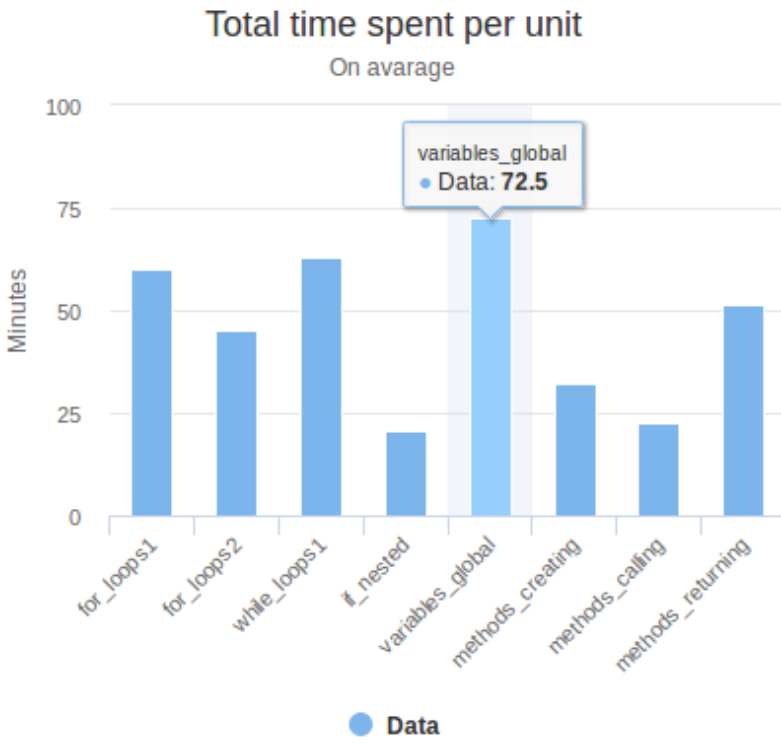

Figure 5.4: A component of the *column* chart type

This simple column chart illustrates the average time spent on each task (each `unit` according the the VSON specification) in the course. Notice the additional information available when hovering over the bar using the mouse cursor.

Implemented components

These chart components are implemented but not currently used to visualize actual VSON data. This is either due to not having found the appropriate use case for them, or that no time could be found to implement the visualizations they could be useful for. Therefore they are illustrated here with dummy data. Note that the entire library of Highchart charts[1] are available and could be made into Vent components when needed.

[1] https://www.highcharts.com/demo

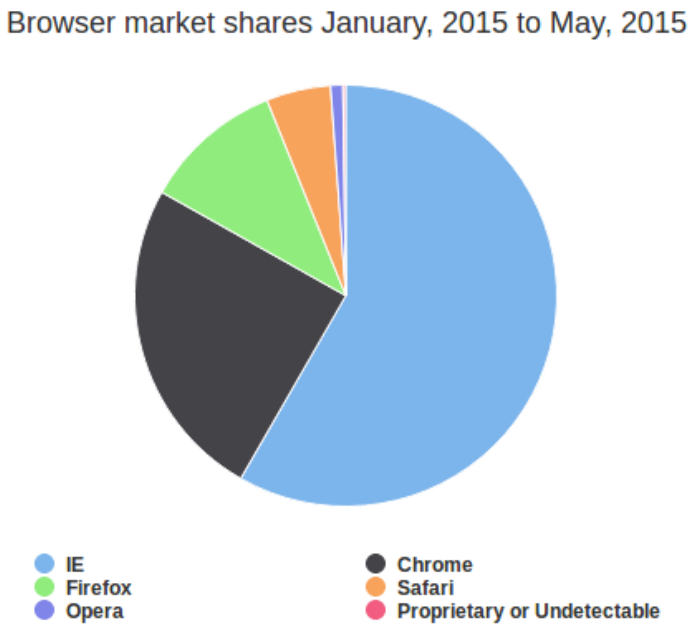

Figure 5.5: A component of the *pie* chart type

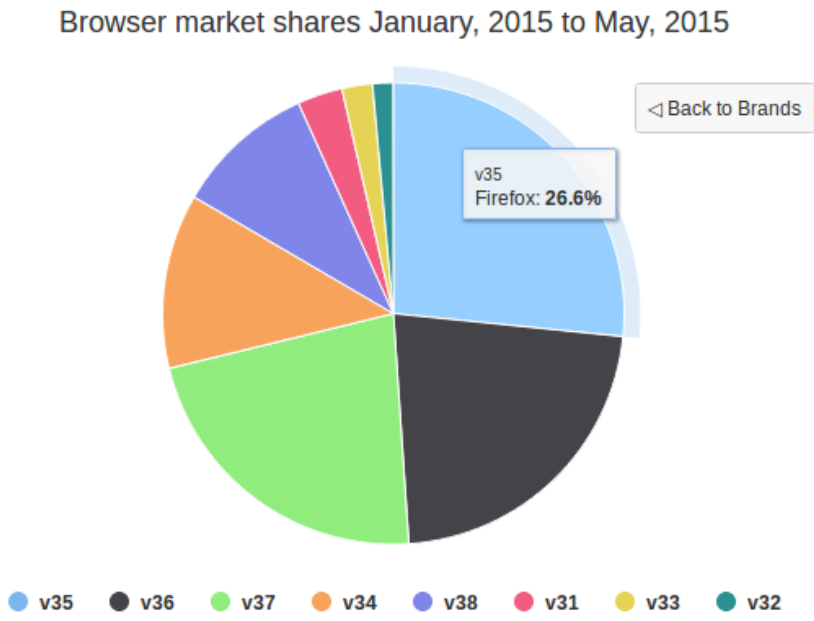

Figure 5.6: A component of the *pie* chart type with drill down

Figure 5.5 shows a pie chart populated with dummy data about browser market shares. The piechart component supports a mode called drill down, where a data point can be clicked for a finer break down of the data as seen in 5.6. A pie chart visualization with drill down can be instantiated by passing in the `series` and `drilldownSeries` properties.

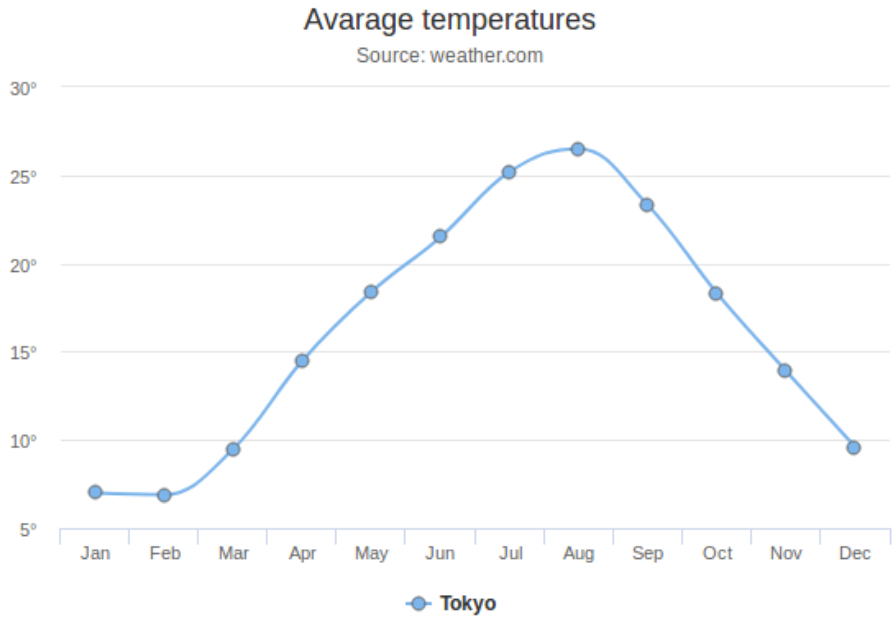

Figure 5.7: A component of the *line* chart type

A simple line chart component creating visualization as seen in figure 5.7 has also been implemented, but has found no particular use case as of writing. The screenshot is therefore populated with dummy data.

5.1.1 Other visualizations

The dashboard has been developed for larger screens and thus does not adapt well to being displayed in a document of this size. However, it is simply a combination of the different components shown, populated with various data. In addition to the visualizations shown in the examples, the following visualizations have been implemented:

- Content usage by topic
- Content usage by content type
- Content completion by topic
- Content completion by content type
- Content recommendation rate by topic

- Content recommendation rate by content type

All implemented visualizations are based off of the desired visualizations as found in appendix A. There are still quite a few of these missing, which should be part of the next development phase of the Vent dashboard.

5.2 Fulfilled goals and objectives

As presented in section 1.2 these were the initial, and additional goals defined during the project. All these are touched upon indirectly in previous sections, and some will be discussed further in chapter 6. Here I will briefly give the status for each one in turn.

Initial goals:

1. Decide on a set of technologies flexible enough for interaction with multiple data sources
2. Find sensible and smart ways of pulling data from said data sources
3. Elicit desirable visualizations from course professors
4. Extract data needed to create visualizations
5. Display visualizations in a sensible way

Additional goals:

1. Extract common denominators for data needed to create the requested visualizations mentioned in appendix A
2. Find a suitable way of containing this data
3. Express this container format in a general and reproducible way
4. Create a visualization dashboard built around this new data format

5.2.1 Status of initial goals

1. Completed with the Vent backend
2. Implemented for ProTuS and Mastery Grids. Denied access for Blackboard data. Awaiting dependence on Boye Borg Nygårds project Nygård (2018) for JExercise implementation
3. Completed and available in appendix A.
4. Needs further work
5. Some working visualizations available in the Vent front end

5.2.2 Status of additional goals

1. Completed as described in appendix C
2. Completed as described in chapter 4
3. Completed with the schematic definition found in appendix C
4. The initial version of the dashboard is available as described earlier in this chapter

Chapter 6

Discussion and future work

In this chapter I will reiterate what this thesis set out to do, what challenges were encountered, what consequences they had for the development and finally what actually ended up being created. Lastly I will recommend a way forward for Vent, VSON and the various data sources encountered during the development.

6.1 Project development summary and final product

Initially this thesis was supposed to chronicle the development of a made-to-measure visualization system specifically for a few select data sources originating from a course taught at NTNU. Several challenges with various aspects of the data sources were encountered. First, the temporary denial of access to data from NTNUs Blackboard LMS platform. Later it was various challenges as explored in section 3.4.2, including challenges regarding constantly changing source format and badly documented APIs. This in turn made me reconsider the project's priorities and long-term viability. The overall value of the project would not be very high if shortly upon completion everything would break after a source change.

From there started the conception of a standard way of representing educational data for visualizations. This could be a standard way to expose data from various sources that could instantly be plugged into a visualization dashboard. Aiming to solve the challenges encountered during the development of Vent, VSON was designed to be simple, stable and well documented. Vent would still be needed as a middle layer, and still be susceptible to the same breaking, however the formalization of the VSON data format would hopefully encourage the use of it as a stable data format at a later point.

Vent as it stands consists of two main parts. The front end acting as a visualization dashboard for VSON-data and the back end that is responsible for converting data from various sources to VSON format. The back end is, as it stands, not fully completed. The foundations are laid and its completion would not be far off provided close work with the data source providers to overcome challenges like bad or nonexistent documentation.

All in all I find Vent as a whole to be much better suited for future expansion than it would have been by simply rushing through the initial integrations without regard to the over all consequences posed by integrating various data sources. Not all aspects are, at the time of writing, fully completed. However, the ground work has been done to greatly facilitate future development in a robust way.

6.2 The way forward

Going forward there are a few different aspects of this thesis that should be further developed. From the bottom up (source data to visualization) these are the following: The source data aligning with standard representations and formats. A compatibility layer between xAPI and VSON. The future development of VSON as a standard. And the continued development of Vent as a visualization tool for educational data. This separation is not the suggested chronological order in which to tackle further development.

6.2.1 Suggested evolution of data sources

Whilst the Vent back end will act as a compatibility layer between the source data provided by ProTuS, Mastery Grids and JExercise this is a rather fragile situation due to the unstable nature of these data formats. For easy compatibility with Vent visualizations exposing their data as VSON would provide a predictable way to communicate the visualization data. However, in my opinion, an even better approach would be to implement the Experience API as the default way to gather and communicate educational data. Following the xAPI standard will yield advantages such as being more open, easier to integrate and also compatibility with a whole ecosystem of educational applications. It is also worth mentioning that Blackboard, NTNUs LMS, already incorporates xAPI. Thus, this move would advance the interoperability between all systems, facilitating a holistic overview that could help achieve insights and improve education. My experiences from this project is that no one is best served by only adhering to their own locked in models. Rallying behind an open standard, that already has shown some merit, will only have positive consequences and accelerate the whole field of educational data mining and learning analytics. This is the same conclusion drawn by Dodero et al. (2017).

6.2.2 Vent back end

The Vent back end as a compatibility layer will hopefully be marginalized to only be needed for the conversion from xAPI to VSON. This might still be a little bit down the road, and completing the conversion from ProTuS, Mastery grids and JExercise to VSON might be a desirable starting point. xAPI to VSON conversion should still be a priority as it would allow for instant integration of Blackboard as soon as access to its data is granted.

6.2.3 Future development of VSON

VSON as it has been presented in this thesis is an initial draft. The only way it will provide any value in the future is by being adopted, adapted and refined in actual use by a

community. Without that, it will at most serve as an influence or starting point for others interested in visualizing educational data. Few things are created right from the beginning, and even if I hope that the initial proposal is a solid one, I fully expect that it will be modified and improved upon adoption in other projects.

6.2.4 Continued development of Vent for visualizations

The Vent front end is perhaps the most straight forward component to develop further. All the foundations are laid down for creating new components and visualizations in a very simple and straight forward way. The list of desired visualizations found in appendix A is a great starting point. The Vent front end is independent of integration with third party data sources as it only relies on being served VSON as its data container. If more data is needed for the visualizations than what VSON currently provides, one can simply add the desired properties at will. If they are found useful they can then trickle down and become a part of the main VSON specification. Hopefully the very loose coupling with the Vent back end makes it even easier to use as any data served as VSON can seamlessly be visualized without the need if a complicated application setup.

Another future development that could be envisioned is the addition of time sensitive views. As of now the dashboard visualizations are a snapshot of a point in time, as VSON data in and on itself does not have any sort of time line property (see section 4.3.3). However, VSON data could easily be time stamped and exposed in weekly, daily or even hourly snapshots and the Vent front extended with views that handle sets of VSON data.

Bibliography

Blackboard, I., 2018. Learning Management System Innovation. http://www.blackboard.com/learning-management-system/index.html, accessed: 2018-03-22.

Dodero, J. M., González-Conejero, E. J., Gutiérrez-Herrera, G., Peinado, S., Tocino, J. T., Ruiz-Rube, I., 2017. Trade-off between interoperability and data collection performance when designing an architecture for learning analytics. Future Generation Computer Systems 68, 31–37.

Dyckhoff, A. L., Zielke, D., Bültmann, M., Chatti, M. A., Schroeder, U., 2012. Design and implementation of a learning analytics toolkit for teachers. Journal of Educational Technology & Society 15 (3), 58.

Elias, T., 2011. Learning analytics. Learning.

Few, S., 2006. Information dashboard design.

Johnson, L., Smith, R., Willis, H., Levine, A., Haywood, K., 2011. The 2011 horizon report. austin, tx: The new media consortium. In: International Conference on Educational Data Mining. Montréal, Québec, Canada. pp. 38–70.

Kevan, J. M., Ryan, P. R., 2016. Experience api: Flexible, decentralized and activity-centric data collection. Technology, knowledge and learning 21 (1), 143–149.

Lassila, O., Swick, R. R., et al., 1998. Resource description framework (rdf) model and syntax specification.

Mangaroska, K., Giannakos, M., 2017. Learning analytics for learning design: Towards evidence-driven decisions to enhance learning. In: European Conference on Technology Enhanced Learning. Springer, pp. 428–433.

Muñoz-Cristóbal, J. A., Rodríguez-Triana, M. J., Gallego-Lema, V., Arribas-Cubero, H. F., Asensio-Pérez, J. I., Martínez-Monés, A., 2016. Toward the integration of monitoring in the orchestration of across-spaces learning situations. In: CrossLAK. pp. 15–21.

Nygård, B. B., 2018. Work in progress. https://github.com/boyeborg/work-in-progress, accessed: 2018-05-26.

Rustici, S., 2018. What is the Experience API? https://xapi.com/overview/, accessed: 2018-03-22.

Sahebi, S., 2018. Adaptive Navigation Support and Open Social Learner Modeling for PAL. http://adapt2.sis.pitt.edu/wiki/Adaptive_Navigation_Support_and_Open_Social_Learner_Modeling_for_PAL, accessed: 2018-03-14.

Schroeder, U., 2009. Web-based learning–yes we can! In: International Conference on Web-Based Learning. Springer, pp. 25–33.

Schwendimann, B. A., Rodriguez-Triana, M. J., Vozniuk, A., Prieto, L. P., Boroujeni, M. S., Holzer, A., Gillet, D., Dillenbourg, P., 2017. Perceiving learning at a glance: A systematic literature review of learning dashboard research. IEEE Transactions on Learning Technologies 10 (1), 30–41.

Vesin, B., 2018. Programming tutoring system. https://xapi.com/overview/, accessed: 2018-03-17.

Yoo, Y., Lee, H., Jo, I.-H., Park, Y., 2015. Educational dashboards for smart learning: Review of case studies. In: Emerging issues in smart learning. Springer, pp. 145–155.

Appendix

A Desired visualizations

This appendix shows a list of visualizations gathered by Katerina Mangaroska in discussion with Michail Giannakos and Hallvard Trætteberg. The list suggests a number of visualizations that would add value to an educational dashboard. The list is in no particular order. It was used to decide what initial visualizations should be available in Vent, as well as formed the basis for what data was needed when developing the VSON data format. The list was sent to me the 24th of February.

The visualizations can be grouped in three categories:

1. Course content

The teacher needs to know what type of content student use in order to make adequate intervention and keep the course quality on a high level.

Metrics the teacher is interested:

- content usage on a weekly level (e.g. challenges, coding exercises, etc)
- content usage per topic
- total content usage (aggregated till the date the teacher wants to check)
- % of students who use recommended content
- % of students who use recommended content by type
- usage of recommended content per week
- usage of recommended content per week per topic
- total usage of recommended content
- % of students going back to already visited content
- time students spend using the explanation tab
- number of online sessions
- total time spend online

2. Student progress

Metrics the teacher is interested:

- % of students completing assignments weekly
- % of students completing assignments weekly by topic
- % of students submitting assignments only once
- % of students submitting correct assignments weekly
- time used to finish an exercise by content
- student progress submitting recommended content
- student progress submitting recommended content by topic
- student progress submitting content by topic
- student performance by topic weekly
- total student performance
- total student performance by topic
- total student performance recommended content vs content
- student performance using exercises
- comparison: students that have done exercises perform better than students that have not

3. Protus vs Eclipse vs Mastery Grid

What type of content according to the level of difficulty students use in Protus (challenges vs. coding exercises) and how their performance in Protus affect their performance in Eclipse (Hallvard assignments score).

The level of difficult students choose in Protus does correlates with the level of difficult students choose when do assignments in Eclipse?

Protus difficulty level: novice, skillful, confident, proficient, expert
Eclipse: easy, medium, difficult

B README files for Vent

Vent

> Visualized Education NTNU

Vent is a visualization tool for educational data based around the VSON format. The two main components, its front and back end are loosely coupleded.

The front end can easily perform visualizations from any source prividing VSON data.

The back end is a framework on which to build integrations between existing data stores and the front end by connecting with source data and converting it to VSON.

Vent and VSON was developed as my master thesis in the spring of 2018 at the Norwgian University of Science and Technology (NTNU).

It is released under the MIT license. Note however that the front end leverages Highcharts which comes with its own license.

Future development or maintanance is left up to the community and the team of Michail Giannakos at NTNU.

B.1 README for Vent backend

vent-backend

> Visualized Education NTNU

Vent is a visualization tool for educational data. This is the backend that can be configured to integrate data from multiple sources, before converting them to VSON and exposing them through an API.

Build Setup

```
# install dependencies
mvn install

# package and run jar
mvn package &&java -jar target/vent-0.0.1-SNAPSHOT.jar
```

Please note:

- Application settings are found in application.yaml and need to be configured according to ones data stores.
- The default data sources are ProTuS and Mastery Grids, but Vent is designed to be extensible

This application is written in Kotlin using the Spring Boot framework. Please refer to their respective documentation for troubleshooting.

B.2 README for Vent frontend

vent-frontend

> Visualized Education NTNU

Vent is a visualization tool for educational data. This is the frontend that handles actual visualization. If you have an API endpoint serving VSON data you can simply spin up the development environment and navigate your browser to http://localhost:3000

Build Setup

```
# install dependencies
yarn install

# serve with hot reload at localhost:3000
yarn run dev

# build for production with minification
yarn run build

# build for production and view the bundle analyzer report
yarn run build --report
```

For a detailed explanation on how things work, check out the guide and docs for vue-loader.

The Vent frontend is a Vue.js app working with Highcharts to create beautiful visualizations. Looking at the code it should be somewhat straight forward to understand its structure.

C VSON json-schema definition

```
{
  "$id": "http://hernil.com/schema.json",
  "$schema": "http://json-schema.org/draft-06/schema#",
  "type": "object",
  "definitions": {
    "unit": {
      "type": "object",
      "required": ["name", "type", "topic"],
      "properties": {
        "name": {
          "$id": "/properties/name",
          "type": "string",
          "description": "Unique human-readable name within the scope of this subject",
          "examples": ["days_to_week_conversion", "inheritance.animals"]
        },
        "type": {
          "$id": "/properties/type",
          "type": "string",
          "description": "Describes what type of unit this is",
          "examples": ["resource", "task", "exercise"]
        },
        "topic": {
          "$id": "/properties/topic",
          "type": "string",
          "description": "Describes what topic this belongs to ",
          "examples": ["For loops", "Inheritance"]
        },
        "difficulty": {
          "$id": "/properties/difficulty",
          "type": "number",
          "description": "The difficulty of the given unit",
          "default": 0
        },
        "opened": {
          "$id": "/properties/opened",
          "type": ["boolean", "number"]
        },
        "completed": {
          "$id": "/properties/completed",
          "type": ["boolean", "number"]
        },
        "recommended": {
          "$id": "/properties/recommended",
          "type": ["boolean", "number"]
        },
        "timeSpent": {
          "$id": "/properties/timeSpent",
          "type": "integer",
          "default": 0,
          "examples": [
            3500
          ]
        },
        "performance": {
```

```
        "$id": "/properties/performance",
        "type": "number",
        "default": 0,
        "examples": [
          0.6
        ]
      }
    }
  }
},
"properties": {
  "id": {
    "type": "string",
    "description": "Unique identifier for a course. For example course code.",
    "examples": ["TDT4100", "IT3010"]
  },
  "name": {
    "type": "string",
    "description": "Human-readable name of the course.",
    "examples": ["Object-oriented programming", "Web development"]
  },
  "units": {
    "type": "array",
    "items": {
      "type": "object",
      "allOf": [
        {"$ref": "#/definitions/unit"},
        {
          "properties": {
            "opened": {
              "type": "number",
              "description": "Avarage student opened rate for this unit. Value between 0 and 1"
            },
            "completed": {
              "type": "number",
              "description": "Avarage student completion rate for this unit. Value between 0 and 1"
            },
            "recommended": {
              "type": "number",
              "description": "Rate at which this unit has been recommended to students. Value between 0 and1"
            },
            "timeSpent": {
              "description": "Avarage time spent on this unit. Value in seconds"
            },
            "performance": {
              "type": "number",
              "description": "Avarage student performance for this unit. Value between 0 and 1"
            },
            "expectedPerformance": {
              "$id": "/properties/expectedPerformance",
              "type": "number",
```

```
106                 "description": "Defines the units expected performance
    measure.",
107                 "default": 1,
108                 "examples": [
109                   0.8
110                 ]
111               }
112             }
113           }
114         ]
115       }
116     },
117     "students": {
118       "type": "array",
119       "items": {
120         "type": "object",
121         "properties": {
122           "id": {
123           "type": "string",
124           "description": "Unique string identifying a student. Preferably
    anonymized in the form of a hash or similar"
125         },
126           "performance": {
127             "type": "number",
128             "description": "Overall student performance. Values between 0
    and 1"
129           },
130           "units": {
131             "type": "array",
132             "items": {
133               "type": "object",
134               "allOf": [
135                 {"$ref": "#/definitions/unit"},
136                 {
137                   "properties": {
138                     "opened": {
139                       "type": "boolean",
140                       "description": "True if the student has opened the
    unit."
141                     },
142                     "completed": {
143                       "type": "boolean",
144                       "description": "True if the student has completed
    the unit."
145                     },
146                     "recommended": {
147                       "type": "boolean",
148                       "description": "True if the unit is recommended to
    the student.",
149                       "default": false
150                     },
151                     "timeSpent": {
152                       "type": "integer",
153                       "description": "How many seconds the user has spent
    on the unit.",
154                       "minimum": 0
155                     }
```

```
                    }
                }
            ]
        }
    }
  }
}
```

Listing 1: The VSON schema definition

D The Hoov application

The Hoov application (Hoov is a play on the fact that it is supposed to hoover up information) was written upon request from Hallvard Trætteberg so as to function as an API endpoint where he could configure the JExercise plugin to post its gathered data. It was written to specifications supplied by Trætteberg and was completed and deployed on January 5th 2018, but the functionality needed in JExercise was never implemented. Thus Hoov has stood unused since its deployment. It has been added to the appendix as it is mentioned earlier in the thesis, and to illustrate work that has been done, although without any tangible results.

```
var express = require("express");
var app = express();
var port = 3000;
var bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

var mongoose = require("mongoose");
mongoose.Promise = global.Promise;
var mongo_adr = process.env.MONGO_ADR;
mongoose.connect("mongodb://" + mongo_adr + ":27017/hoov1");

var logSchema = new mongoose.Schema({
    key: Object,
    payload: Object
});

var LogEntry = mongoose.model("LogEntry", logSchema);

app.get("/entries", (req, res) => {
    LogEntry.find({}, function(err, doc){
        res.json(doc)
    });
});

app.post("/entries/add", (req, res) => {
    // Sort request body to assure sorted data in key
    var sortedBody = sortKeys(trimObj(req.body));
    var query = {'key': sortedBody.key};
    LogEntry.findOneAndUpdate(query, sortedBody, {upsert:true}, function(
    err, doc){
```

```
        if (err) return res.send(500, { error: err });
        return res.send(sortedBody);
    });
});

sortKeys = function(x) {
    if(typeof x !== 'object')
        return x;
    if(Array.isArray(x))
        return x.map(sortKeys);
    var res = {};
    Object.keys(x).sort().forEach(k => res[k] = sortKeys(x[k]));
    return res;
}

trimObj = function(obj) {
  if (!Array.isArray(obj) && typeof obj != 'object') return obj;
  return Object.keys(obj).reduce(function(acc, key) {
    acc[key.trim()] = typeof obj[key] == 'string'? obj[key].trim() :
    trimObj(obj[key]);
    return acc;
  }, Array.isArray(obj)? []:{});
}


app.listen(port, () => {
    console.log("Server listening on port " + port);
});
```

Listing 2: The Hoov application code

```
{
  "name": "hoov",
  "version": "1.0.0",
  "description": "Application for recieving (hoovering up) data from the
    JExercise plugin",
  "main": "app.js",
  "scripts": {
    "dev": "node app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "node",
    "mongodb",
    "mongoose",
  ],
  "author": "Nils Herde",
  "license": "MIT",
  "dependencies": {},
  "devDependencies": {
    "body-parser": "^1.18.2",
    "express": "^4.16.2",
    "mongoose": "^5.0.0-rc0"
  }
}
```

Listing 3: The Hoov application package.json definition