

Enrichment of the user and developer experience through modern web technologies.

Jonathan Sparvath Nielsen
Mads Kristian Brodt Nielsen

josn@itu.dk
mkrn@itu.dk

Bachelor's thesis
May 2019
Supervised by Dr. M. Lungu
Class code: 2723287-Spring 2019

Abstract

We present a system designed to aid teachers of foreign languages in their classes by implementing a teacher dashboard using the JavaScript framework React. We present an exploration of modern web technologies, to decide if using a framework can enhance the user experience. We report on the user feedback gathered by deploying the new dashboard with select teachers, as well as the overall code quality. The results turn out to be very positive. Finally, we present some ideas for future work and improvements to the system.

Contents

1	Introduction	6
1.1	Structure	7
2	Related work	8
2.1	The Zeeguu Ecosystem	8
2.1.1	Zeeguu Unified Multilanguage Reader (UMR)	9
2.1.2	Teacher Dashboard	9
2.2	Duolingo classroom	10
3	Problem description	11
3.1	Improved teacher dashboard	11
3.2	Requirements of the system	11
3.2.1	Non-functional requirements	12
3.2.2	Functional requirements	12
4	Exploration of modern web technologies	13
4.1	Evolution of JavaScript	13
4.2	Usage and developer satisfaction	14
4.3	Single Page Application contra Multi Page Application	15
5	Realization	17
5.1	Introduction	17
5.1.1	Reverse engineering and extending the existing dashboard . . .	17
5.1.2	Re-implementing a new dashboard	18
5.2	Design	19
5.3	Home page	20
5.4	Classroom page	21
5.4.1	Adding texts	21
5.4.2	Managing and deleting texts	23
5.5	Student page	23
5.6	User Experience	23

5.6.1	Loading Spinners	24
5.6.2	SPA	25
5.6.3	Asynchronous Data Fetching	25
6	Architecture	26
6.1	Introduction	26
6.2	React	26
6.2.1	Components	26
6.2.2	JSX	27
6.2.3	Hooks	27
6.3	Details of the system	27
6.4	Modules	28
6.5	Communicating with the Zeeguu API	28
6.5.1	Example API communication	28
6.6	React component hierarchy	29
6.7	Routing	30
6.8	Adding new texts	31
6.9	ArticleList component	32
6.9.1	Generalizing the ArticleList component	32
6.10	Expansion of other Zeeguu projects	34
6.10.1	API & Core	34
6.10.2	Unified Multilanguage Reader (UMR)	35
6.10.3	Conventions and naming	35
7	Results and Evaluation	36
7.1	User Feedback	36
7.2	Code quality	38
7.2.1	Lighthouse	38
7.2.2	Sonarcloud	39
7.3	Further evaluation	41
8	Conclusion	42
9	Future work	43
9.1	Uploading different files	43
9.2	Integrating exercises with the teacher dashboard	43
9.3	Student interaction with a text	43
9.4	Lazy loading reading sessions	44
9.5	Real-time updating reading activity	44
	Bibliography	45
A	Appendix	47
A.1	Workflow	47

A.2	Figures	49
A.3	Feedback email sent to teachers	50

Introduction

Technology is evolving rapidly. The methods we use to build websites and applications today are quite different from how we did it 20, 10 or even 5 years ago. As new technologies emerge, so does new opportunities and challenges. User expectations also change - in 2006, users expected a page to load in 4 seconds or less. In 2009, only three years later, the expected load time was half of that - 2 seconds or less [1]. Nowadays, 53% of mobile users will leave a website if it takes more than 3 seconds to load [2].

It is the job of the developer to familiarize themselves with new development tools and practices in order to create the best possible user experience. **This report will feature an exploration of modern web technologies with a focus on the front end.**

The evolution of technology has made travelling the world and communicating with other countries easier than ever - increasing the importance of learning new languages [3]. The evolution of technology has also had an important impact on the way we learn and teach languages [4]. Moving learning material to a web platform can offer many benefits for teachers - for example, easy distribution of learning material and monitoring student progress.

In this project, we will be working with the Zeeguu ecosystem¹. The Zeeguu ecosystem is a collection of projects with the purpose of accelerating acquisition of a foreign language. A main part of its functionality is the ability to parse an arbitrary text and provide on-demand translation of each word to the reader. This opens up the possibility for language teachers to use it in their lessons. Our project will extend the Zeeguu ecosystem by implementing a teacher dashboard, allowing teachers to monitor their students' reading activity and provide them with custom texts.

The main focus of the report will be how to implement the teacher dashboard while leveraging modern web technologies to enrich the user experience for the end users, the teachers. Additionally, we demonstrate how to implement such a system with a focus on code quality and maintainability.

¹<https://github.com/zeeguu-ecosystem>. More details in section 2.1

1.1 Structure

This report features the following chapters:

2. **Related work:** Presents work related to the problem at hand and establishes the surrounding context of the project.
3. **Problem description:** Presents the overarching theme of the project and requirements for the new system.
4. **Exploration of modern web technologies:** Features a data-driven and objective analysis of the current front-end web landscape.
5. **Realization:** Demonstrates the implementation of the system with a focus on the *why*.
6. **Architecture:** Covers the architecture of the implementation with a focus on the *how*.
7. **Results and evaluation:** Presents an evaluation of the new system in terms of the research questions and system requirements.
8. **Conclusion:** Draws conclusions on the results of the project.
9. **Future work:** Discusses future changes and possible improvements to the system.

Appendix: Provides more detail on the workflow and development of the project.

Related work

2.1 The Zeeguu Ecosystem

This project is created within the context of the Zeeguu ecosystem as mentioned in chapter 1. Zeeguu is an open source project, allowing anybody to contribute and use the software.¹ The Zeeguu ecosystem consists of several projects designed to aid personalized language acquisition. These projects are listed below:

- Zeeguu Core
- Zeeguu API
- Zeeguu Unified-Multilanguage-Reader (UMR)
- Zeeguu Teacher Dashboard

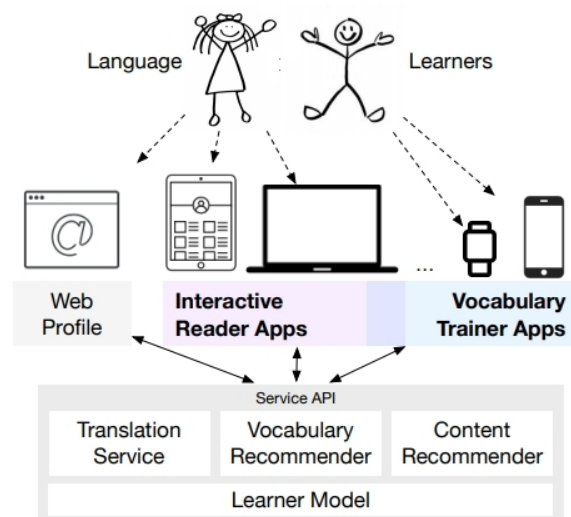


Figure 2.1: A high level overview of the Zeeguu Ecosystem. Image credit goes to [5]

How the systems work together is best described in figure 2.1. Zeeguu Core contains the data models used within the Zeeguu ecosystem as well as the Learner

¹ Available at <https://github.com/zeeguu-ecosystem/>

Model which is comprised of the Translation Service, Vocabulary Recommender and Content Recommender. The Zeeguu API exposes endpoints for communicating with Zeeguu Core. Finally, the Zeeguu UMR and Teacher Dashboard are applications that allow users to interact with Zeeguu through the API.

More details about Zeeguu can be found in [5], including the results of deploying Zeeguu with multiple high school classes for one month.

2.1.1 Zeeguu Unified Multilanguage Reader (UMR)

The UMR is a web application where language learners can select their topics of interest as well as a language they wish to learn² [6, 7]. The system will then present articles in a distraction free environment based on the users interest. Each word can be clicked to obtain a translation, and the system will attempt to translate the words in context. The UMR is content agnostic, meaning it can take any plain text and allow the user to translate words from that text.³ [6]



Figure 2.2: Zeeguu UMR. Left image is a list of articles for the user to choose from. Right image is the distraction free environment to read articles in.

2.1.2 Teacher Dashboard

The teacher dashboard is a part of the Zeeguu website where teachers can create classes and manage their students.⁴ The dashboard allows teachers to view their students activity, including how much time a student has spent reading and doing exercises, and which texts they have read. The dashboard also displays a detailed overview of which words a student has translated in a certain text, including the surrounding context. A more thorough analysis of the teacher dashboard follows in chapter 5.

²The project can be found at <https://github.com/zeeguu-ecosystem/Zeeguu-Reader>

³This will prove useful later, when teachers are allowed to upload their own texts.

⁴The project can be found at <https://github.com/zeeguu-ecosystem/Zeeguu-Teacher-Dashboard>

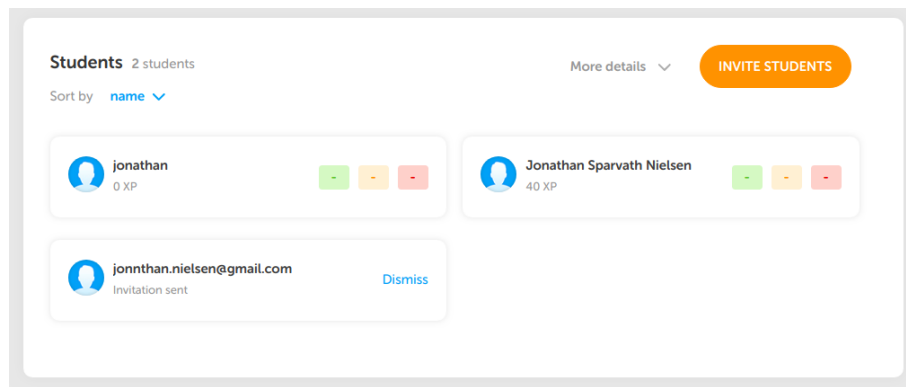


Figure 2.3: List of students from Duolingo’s teacher dashboard

2.2 Duolingo classroom

Duolingo is another application for learning languages⁵. Duolingo has a new feature called Duolingo Classroom⁶. Duolingo Classroom offers the possibility of creating classes and inviting Duolingo users as students. Some of the key features offered by Duolingo Classroom include:

- See all your students in a list with info about the amount of assignments completed, and experience points earned (XP). See figure 2.3.
- Give assignments to your students.
- In-class activities such as practicing words together.

⁵<https://www.duolingo.com/>

⁶<https://schools.duolingo.com/>

Problem description

3.1 Improved teacher dashboard

As discussed in the previous section, the Zeeguu ecosystem already has a teacher dashboard. However, several teachers requested additional features to help them teach their students. One of these features was the ability for teachers to upload their own texts. This will make it easier for teachers to adjust the difficulty level of texts based on the class they are teaching.

Another feature explicitly requested by teachers was the ability to view all of their students in a single table. This should provide an overview of each students activity, allowing teachers to quickly view any particular students interactions with the texts.

With this in mind, we present the two main research questions we aim to answer throughout this project:

- **RQ1. How can we leverage modern technologies to enhance the user experience of the teacher dashboard while implementing the new, user requested features?**
- **RQ2. How can we ensure good code quality and maintainability of the system to aid developers in future support and extension of the project?**

The system should be easy to use and support the users in their jobs. The code base should be maintainable and scalable to allow implementation of future user feedback. This includes providing a nice developer experience that speeds up development time and minimizes error, which will ultimately result in a better user experience.

The research questions present a choice: reverse engineer and expand upon the current teacher dashboard implementation, or create an entirely new teacher dashboard using modern technologies. Both options will be explored in chapter 5.

3.2 Requirements of the system

We can narrow down some non-functional and functional requirements to better illustrate how the teacher dashboard should work.

3.2.1 Non-functional requirements

1. The system should be intuitive to use. This includes creating a simple and seamless user experience that empowers the users to complete their tasks.
2. The system should be maintainable to easily adapt to changing requirements and future user feedback. To achieve this, the system should consist of high quality code with focus on modularity and extensibility.
3. The system should be accessible¹. This includes being usable by people with various disabilities, such as visibility or mobility impairments.

3.2.2 Functional requirements

In addition to these non-functional, high-level requirements, we define the following functional requirements:

1. The user should be able to create, edit and delete a class and see an overview of all their classes.
2. The user should be able to see how much time their students have spent on reading and doing exercises.
3. The user should be able to select a specific time period in which to view student activity²
4. The user should be able to view a student's interactions with a text. This includes how long time the student has spent on the text and which words they translated.
5. The user should be able to see an overview of all their students.
6. The user should be able to add their own texts to a class. It should also be possible to view a list of added texts, as well as deleting a text if it is no longer relevant to the class.

These features will be using the functionality of the Zeeguu API and Core projects. When necessary, the API and Core will be extended to support the new use cases of adding, viewing and removing texts. In addition, the Zeeguu UMR will be expanded to include a way for students to read teacher uploaded texts.

¹https://developer.mozilla.org/en-US/docs/Learn/Accessibility/What_is_accessibility

²For instance, how much time a student has spent reading and doing exercises in the last 2 weeks.

Exploration of modern web technologies

4.1 Evolution of JavaScript

Front-end development has been moving fast the past few years. JavaScript has now adopted many of the features that once made jQuery¹ so popular; so much so that Github.com has removed jQuery completely from their front end in favor of *vanilla JavaScript*² [8]. Even though JavaScript as a language has evolved³, new JavaScript frameworks have gained a lot of traction because they alleviate many of the pain points that still exist with vanilla JavaScript. Some of the benefits of using a front-end framework include:

- **Application structure.** Frameworks often have an opinionated way of organizing the code with focus on making the codebase more scalable.
- **On-boarding new developers.** Since frameworks often have an opinionated way of doing things, new developers who are already familiar with the framework can quickly contribute to projects using the framework. On the other hand, developers who are not familiar with the framework can consult the documentation and community.
- **Development speed.** Frameworks often include a solution to common problems such as state management and routing, and their community often develop open source reusable components that can easily be included in the code.

However, there are also downsides to using a front end framework:

- **Framework lifespan.** New frameworks appear and disappear all the time.

¹<https://jquery.com/>

²"Vanilla JavaScript" simply refers to writing JavaScript without the use of major frameworks or libraries

³Most notably with the release of ECMAScript 2015 (ES6) in June of 2015, <https://en.wikipedia.org/wiki/ECMAScript>

“Frameworks and tools don’t last in JavaScript. The average framework has a phase of peak popularity of 3–5 years,...“ [9]

- **Bundle size.** Client side frameworks naturally come with some overhead in the form of larger file sizes.
- **Complexity.** For small and trivial applications, introducing a framework can be unnecessary and introduce confusion.

4.2 Usage and developer satisfaction

An important metric to take into consideration when choosing a framework is its current usage. This is both increasing the likelihood that new developers already possess the skill to jump into the project, and that a healthy amount of components exist in the community that can be utilized. Additionally, it is important to look at developer satisfaction - the percentage of developers who have used a framework before, and would use it again. This is a good indicator of the developer experience when using the framework.

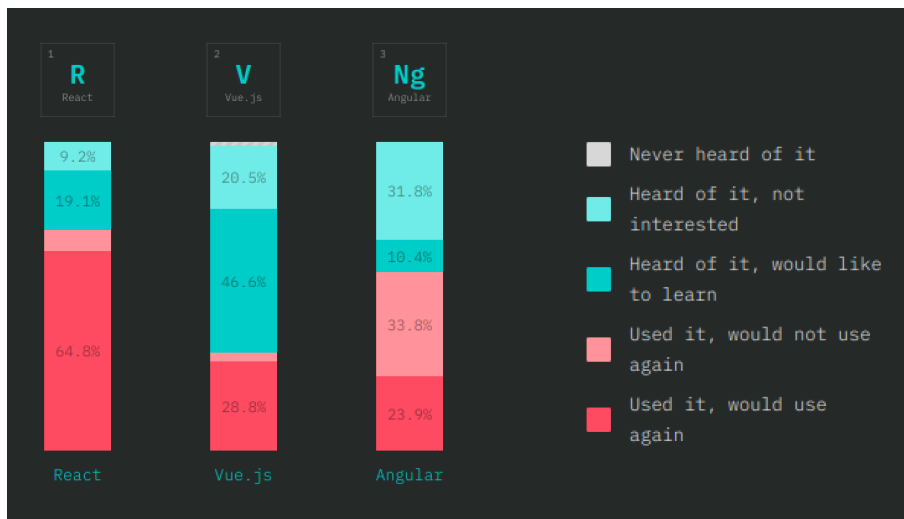


Figure 4.1: Overall popularity of three of the major front end frameworks: React (React is usually considered a library, but for simplicity’s sake we will regard React as a framework.), Vue and Angular. (This image is modified from the source, leaving out some of the lesser used frameworks. To see the original please visit the source)

Source: <https://2018.stateofjs.com/front-end-frameworks/overview/>

As seen in figure 4.1, 71.5% of the respondents said that they have used React before; 91% of those, would use it again. 31.6% have used VueJS; of those, 91% would use it again. 57.7% have used Angular, but of those only 41% said they would use it again.

These usage statistics are also reflected in downloads from NPM⁴. As seen in Appendix A.2, on the 5th of February 2018 the React package had 5,205,381 down-

⁴Node Package Manager (<https://www.npmjs.com/>)

loads, Vue had 754,802 downloads and @angular/core and AngularJS⁵ together had 1,504,262 downloads.

In conclusion, React is the most used framework with a high percentage of developer satisfaction. React also supports the largest user base of the major frameworks, increasing the likelihood of other developers already knowing React.

4.3 Single Page Application contra Multi Page Application

Another important thing to consider when implementing the front end of a web application is the page architecture, specifically: Should the application be implemented as a Single Page Application (SPA) [10] or as a Multi Page Application (MPA)⁶ [11]? Here is a short description of the two:

- SPA. The content of the website is rendered in the browser by JavaScript.
- MPA. Upon receiving a request, the corresponding HTML is rendered on the server and sent back to the requester.

The advantages of disadvantages of the two approaches can be seen as an inverse relationship. That is, the advantages of the SPA architecture listed below is a correlated disadvantage of an MPA and vice versa.

Advantages of a SPA:

- More control - the SPA model change the route without refreshing the webpage, thus, allowing seamless and fast page transitions.
- Performance - Many interactions can be performed without contacting the server, lowering wait times for the user and avoiding potential latency / network issues.
- Decoupled back end and front end - Since a SPA is just one initial HTML document and subsequent data requests, the server is not concerned with *how* the data is displayed in the browser.

Disadvantages of a SPA:

- Search Engine Optimization (SEO) - Since routing is simulated on the client side, extra attention to SEO is needed for applications where SEO is important.
- More edge cases regarding route configuration. If an application is powered by a server routing framework like Flask for Python, and one of those routes

⁵@angular/core is the NPM package for Angular 2+, which is a complete rewrite of the now deprecated AngularJS. see <https://blog.angular.io/stable-angularjs-and-long-term-support-7e077635ee9c>

⁶Also known as a Traditional Web Application.

points to the SPA,⁷ then you need to configure the server routing framework to reroute all subroutes to that root, because the route does not really exist, but is only emulated by the SPA.

⁷In our case, the teacher dashboard lives at zeeguu.org/teacher-react. So if a user hits this URL, an initial `index.html` file is returned that takes care of routing. However, if a user requests a resource for example at zeeguu.org/teacher-react/classroom, this file does not exist on the server.

Realization

5.1 Introduction

For realizing our vision of the improved teacher dashboard, we had to make a choice: Reverse engineer and extend the existing teacher dashboard, or create a new dashboard from scratch.

5.1.1 Reverse engineering and extending the existing dashboard

Extending the existing dashboard would mean that some functionality was already present. Features like adding, updating and deleting a class would not need to be re-implemented. However, the existing dashboard has several limitations:

1. The dashboard uses a mix of different programming languages and approaches on the front end. For example, some template files contain inline CSS, JavaScript and jQuery mixed with Jinja2 templating logic with data coming from Python.¹ This hurts the maintainability of the project by making it harder to modify and extend in the future.
2. Every HTML page is server rendered. This introduces a usability problem when a page takes a while to process - for instance, a students entire reading history in the last year. This forces the user to wait potentially multiple seconds while the page is generated, without the user knowing what is going on or if the application has crashed.
3. Parts of the design is inaccessible. Some text does not adhere to the WCAG contrast ratio requirements of text having at least 4.5:1 contrast ratio in order to be deemed accessible [12]. Furthermore, some buttons (like the "Create new class" button) are visually hard to notice because the color of the button matches the background, hurting the user experience. See figure 5.1 for examples.

¹Example at https://github.com/zeeguu-ecosystem/Zeeguu-Teacher-Dashboard/blob/master/src/zeeguu_teacher_dashboard/templates/classpage.html



Figure 5.1: The text on the left is deemed inaccessible with a contrast ratio of 2.38 as determined by the Chrome Developer Tools. The grey borders have been added for visual distinction here, but are not present in the design.

4. The teacher has no control over the content of the texts their students can read.
5. It is not possible for a teacher to view all of their students at the same time. The teacher has to enter a class, and can then only view the students of that class and their activity.

5.1.2 Re-implementing a new dashboard

Creating a new dashboard from scratch would allow the use of a modern JavaScript based framework. It would also mean a clean slate to implement a new design with better accessibility, an enhanced user experience and more maintainable code. Based on the analysis in chapter 4, combined with the limitations of the existing dashboard, **we decided to use React to implement the new dashboard from scratch as an SPA.** This provided several key benefits:

1. **Development speed.** React is easy to set up and get started with, and since the project had a well-defined deadline, being able to rapidly prototype was very important.
2. **Reduced complexity.** By only using React for the front end, we avoid the complexities and maintainability issues that can arise when mixing languages.
3. **Active framework development.** React is backed by Facebook and is under active development with near daily commits² and major new features being added regularly³. Active development is a good indicator of longevity of the framework.
4. **Improved UX.** Utilizing the SPA architecture enhances the user experience by allowing seamless page switching without waiting for the server.⁴

²<https://github.com/facebook/react/commits/master>

³Like Hooks, added in version 16.8 in February 2019. <https://reactjs.org/blog/2019/02/06/react-v16.8.0.html>

⁴More on this in section 5.6

With the decision to re-implement the teacher dashboard in React, several new questions arose:

- How should the design look?
- How can we make it easy for the user to accomplish their tasks?
- What should the new features look like and how should they work?

To help anchor the project, we decided to focus on one key principle: **The user should have the best possible experience.** This means creating a simple and intuitive user interface that emphasizes the user's "Jobs to Be Done" [13] and empowers the user to complete their tasks. The design should also be accessible, in correspondence with the non-functional requirements.⁵

5.2 Design

We decided to use Material Design developed by Google⁶ as our styling framework. Material Design is known for its intuitive design, well-crafted hierarchy and attention to detail. The Zeeguu UMR⁷ also uses an implementation of Material Design, so our choice helps to increase cohesiveness between the different Zeeguu projects. We decided to use the React library Material UI⁸ to implement the Material Design components.

Material UI presents a list of components that have been developed and battle-tested over long periods of time, such as tabs, input fields, buttons, accordions, dialogs and dropdowns. These components allow us to quickly get a nice design up and running that is accessible by default⁹. They also implement animations and micro interactions¹⁰ that enrich the user experience. However, some elements of the new dashboard required custom styling to fully support our vision of the system. For this, we used the CSS preprocessor Sass¹¹ to give us additional functionality on top of regular CSS, and to implement a styling system that is easier to maintain and scale. This is accomplished by dividing the Sass code into separate files mirroring the JavaScript files containing the React components. Additionally, all common styling variables are kept in its own file to make updating things like colors and font sizes easier.

⁵See section 3.2.1

⁶<https://material.io/design/>

⁷See section 2.1.1

⁸<https://material-ui.com/>

⁹For example, the Material UI components are all fully tab navigable, making the application much easier to use for keyboard-only users

¹⁰<https://www.nngroup.com/articles/microinteractions/>

¹¹<https://sass-lang.com/guide>

5.3 Home page

The home page of the teacher dashboard features an overview of all the teacher's classes. Each class is represented as a card component, to easily distinguish classes from each other while making each class more visually appealing than a traditional list. Each class card contains the relevant information to the class - its name, language, invite code, number of students and a way to access the "Classroom page."¹² Additionally, the home page features a way for the user to create a new class. Unlike the button from figure 5.1, our button is very prominent. It easily captures the focus of the user and lets them know that an action is possible.

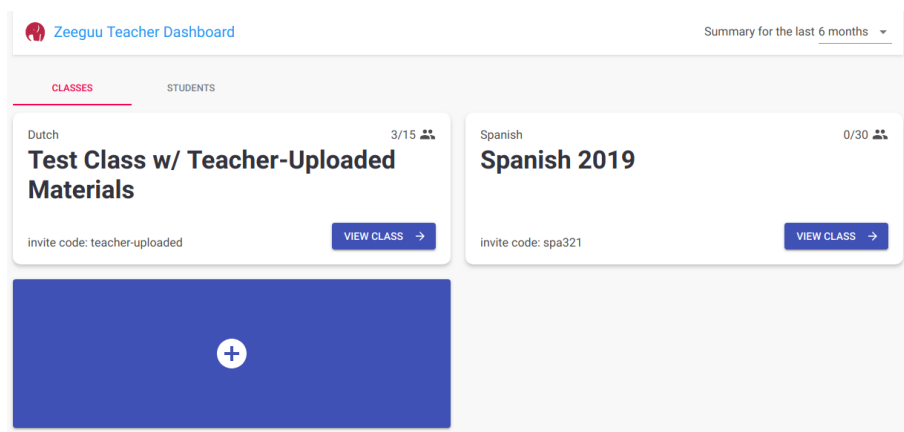


Figure 5.2: The class overview on the home page of the teacher dashboard.

One of the key new features requested by users was the ability to see a list of all their students.¹³ It makes sense to keep this on the home page to make it easy for the user to find. However, having it directly next to or below the class overview seemed confusing - and ultimately not every user will be interested in this list of all their students. Hence we decided to use the Material UI Tab component¹⁴ to allow two different views to exist on the same page.

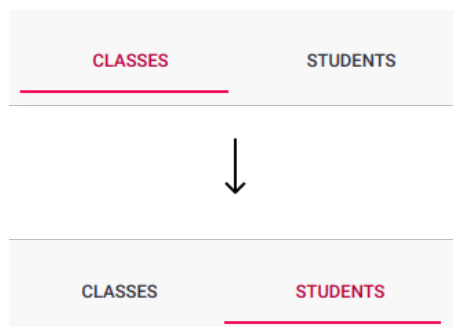


Figure 5.3: The tab component on the home page

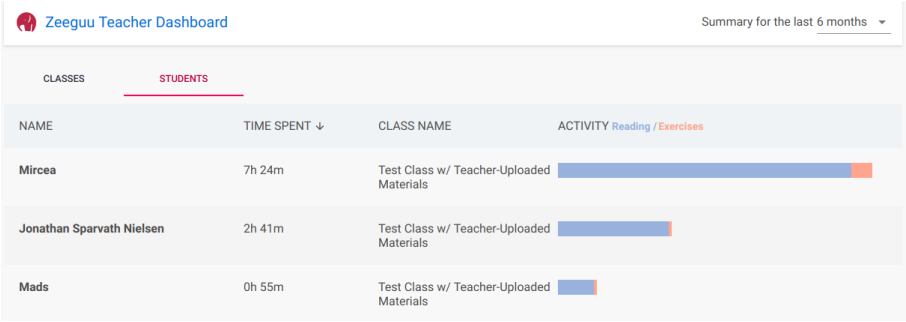
The student overview tab displays a table of all the user's students. The table contains data of how much time a student has spent reading and doing exercises on

¹²More on this in section 5.4.

¹³See section 3.2.2

¹⁴<https://material-ui.com/demos/tabs/>

Zeeguu in a given time period.¹⁵ This information can be useful for the teacher to structure their lessons based on how their students are spending time. Each row in the table can be clicked to visit that students page¹⁶. Focus was on creating a simple UI to present data in a useful format to the users. The student table is an example of a custom made component. In the Material UI table component it is not possible to have an entire row be a link. We also wanted a more visually interesting look, so ultimately it made sense to create our own component in this case.



Zeeguu Teacher Dashboard			
Summary for the last 6 months			
CLASSES		STUDENTS	
NAME	TIME SPENT ↓	CLASS NAME	ACTIVITY Reading / Exercises
Mircea	7h 24m	Test Class w/ Teacher-Uploaded Materials	<div><div></div></div>
Jonathan Sparvath Nielsen	2h 41m	Test Class w/ Teacher-Uploaded Materials	<div><div></div></div>
Mads	0h 55m	Test Class w/ Teacher-Uploaded Materials	<div><div></div></div>

Figure 5.4: List of all the user's students.

5.4 Classroom page

The classroom page can be entered by clicking the "View Class" button in the class overview, and has three main features:

1. An overview of students.
2. The ability to upload, write and delete texts.¹⁷
3. A way to modify information regarding the class¹⁸, or delete the class.

The student table is identical to the table of the previous section, except now it only contains students that are part of this particular class. The most interesting feature of this page is the new ability for the user to add and delete texts that will be presented to students of the class in the Zeeguu UMR.

5.4.1 Adding texts

The user can upload any plaintext (.txt) files through the UI. This can be done by clicking on the "Manage Articles" button on the classroom page. This will open a dialog where the user can upload, write and delete texts.

In the dialog, a file can be uploaded by dragging and dropping the file directly onto the grey-bordered box. It is also possible to click on this box, which will open a file browser and allow the user to select the texts they wish to upload. Once one or

¹⁵Chosen by the dropdown in the top right corner of figure 5.4.

¹⁶More on this in section 5.5

¹⁷See section 3.2.2

¹⁸Like the name or invite code

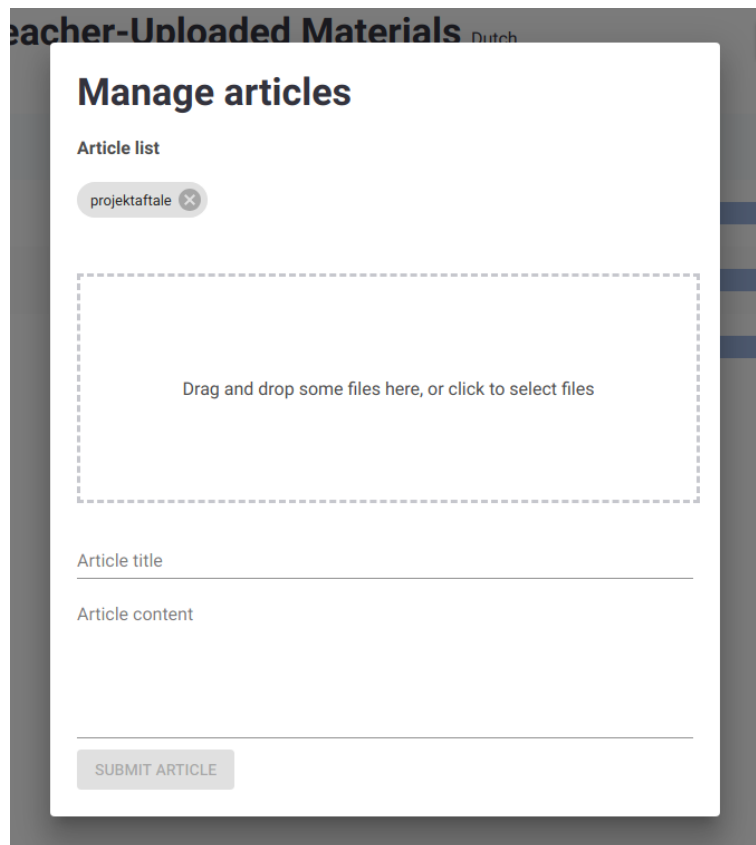


Figure 5.5: The UI for creating and deleting texts

more texts has been added, the UI will change to reflect which files the user is about to upload. A new button will appear, which will handle the actual uploading to the server.



Figure 5.6: The UI for uploading texts once they have been selected for upload

Currently, our file upload solution only supports .txt files. This decision was made as .txt files are easy to parse and allowed us to quickly get the feature working. It also helps avoid confusion, since a user might expect images to also be part of the article if they upload a different kind of file.¹⁹ However, there are alternatives to only supporting .txt files. More on this in chapter 9.

¹⁹Which is not the case, as the database stores article content as plaintext and disregards any images.

It is also possible for the users to write their own texts by providing a title and some body content. This feature could prove useful for teachers to craft texts that are perfectly in line with the level of the students. It would also enable the teacher to decide exactly which topics and words the students should be focusing on.

5.4.2 Managing and deleting texts

With the new possibility of adding texts, it stands to reason that the user should be able to view a list of the texts associated with a class. It should also be possible to delete texts if they are no longer relevant to the students. This feature exists within the "Manage Articles" dialog as well, to keep all article logic contained in one place. Our solution presents the texts with the Material UI Chip component,²⁰ see figure 5.7. This component elegantly solves the problem of displaying the texts while also providing an obvious way to delete a text. Implementation details can be found in section 6.9.

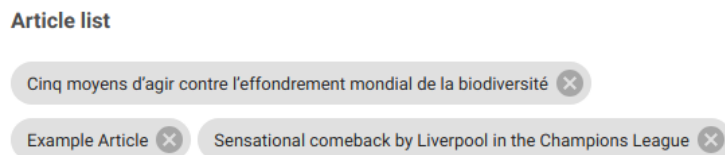


Figure 5.7: The list of articles in a class.

5.5 Student page

Every student on Zeeguu has their own personal student page. The student page features an overview of the student's reading sessions²¹ and how much time they have spent on each reading session. Each reading session is a Material UI Expansion Panel²² that can be expanded to show which words the student has translated, including the translations. The expanded panel also shows the sentence the word was translated from, and if several words were translated in a row. This data can be useful for teachers to learn which words their students are struggling with. It is also helpful to know how long a student has spent on the article and how many words they translated, as this can give an idea of the difficulty level of the text compared to the students current level. An example of a student page can be seen in figure 5.8.

5.6 User Experience

As mentioned, a core part of the new teacher dashboard was to provide a great user experience. This is partly achieved by having an intuitive UI with features that are beneficial to the users - but there are other methods we can employ to further enhance

²⁰<https://material-ui.com/demos/chips/>

²¹A reading session is recorded any time a student opens an article in the UMR.

²²<https://material-ui.com/demos/expansion-panels/>

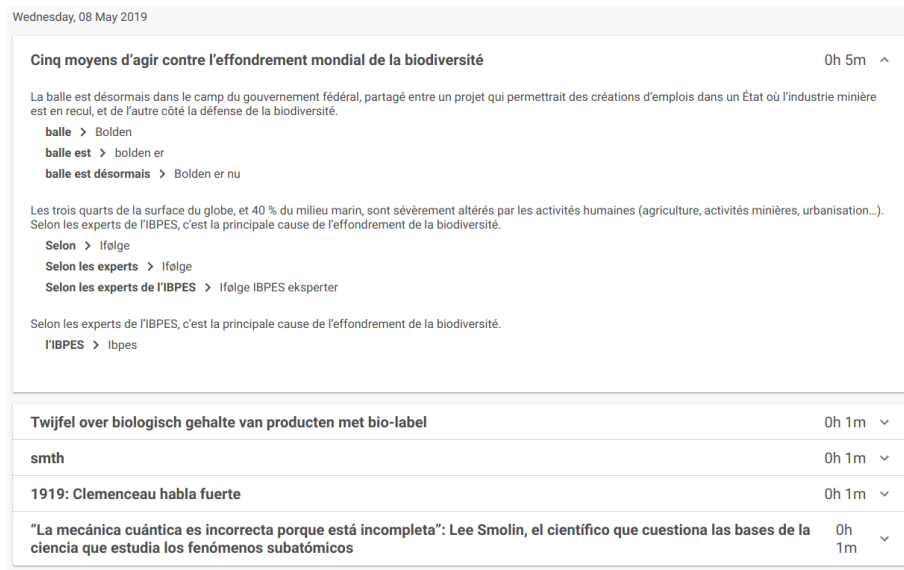


Figure 5.8: A student page showing all articles the student has interacted with.

the user's experience with the system. However, to properly test the user experience, we need to put the system in the hands of the users. More on this in chapter 7.

5.6.1 Loading Spinners

A method for enhancing UX is using loading spinners. This is useful when we need to wait longer periods of time for the server to return some data, like the entire reading history of a student over the last year. Loading spinners let the user know that something is happening. They help to ensure the user that they did not do anything wrong, and that the application has not crashed - it is simply performing some work in the background.

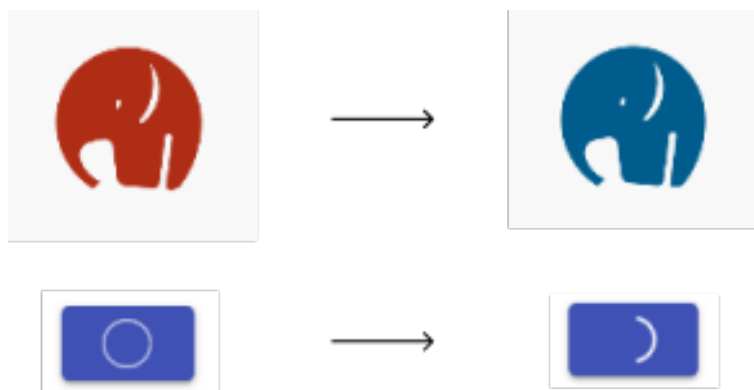


Figure 5.9: Loading spinner examples

We use an elephant spinner animation when retrieving data on a page level, like fetching all a users classes or a students reading sessions. We use a simple loading spinner within buttons when the user is performing a form action, like adding or deleting a class.

5.6.2 SPA

Part of the improved user experience stems from the fact that our application is a SPA²³ and thus does not require any page reloads. For example, when changing a page route, the page changes immediately and while the users are waiting for the data to load, they can be entertained with a loading spinner like the bouncing color-changing elephant that we use in our implementation.

5.6.3 Asynchronous Data Fetching

By fetching data asynchronously in the browser without refreshing the page, the user experience is improved by saving time, since the server doesn't need to generate a new HTML document and the browser doesn't need to render a new HTML document from scratch. Some examples where asynchronous data fetching²⁴ is improving the UX in our project:

- When a user is creating a new class, it is almost instantly available on the page.
- When a user is updating class information, the changes are reflected almost instantly.
- When a user uploads a file, we can keep the "Manage Articles" dialog open. This allows the user to see their newly uploaded article added directly to the list of available articles.²⁵

²³See section 4.3.

²⁴AJAX, <https://developer.mozilla.org/en-US/docs/Glossary/AJAX>

²⁵An example of this is explained in detail in section 6.8.

Architecture

6.1 Introduction

This chapter will provide a detailed description of the architecture of the teacher dashboard. As our focus of the project has been the front end of the dashboard, which is implemented in React, we will begin by describing some key React concepts¹. This is important to understand how our system leverages React to create a better user experience. Afterward, we will dive into the concrete implementation of some key features of the new dashboard. All of the front-end code is available on GitHub.²

Note: We've previously referred to user uploaded material as "texts" and a group of students managed by a teacher as a "class". In the code, these are called "articles" and "cohorts" respectively. This is because Zeeguu previously only supported actual articles sourced from news media, and *class* is a reserved word in Python. With these conventions being enforced on the back end, it made sense to use the same wording on the front end for consistency.

6.2 React

6.2.1 Components

React is "A JavaScript library for building user interfaces".³ The most important concept to understand in React is the idea of "components". A component is a way to encapsulate some behavior which can be reused throughout an application. A component can be thought of as a Lego block. By itself, it has limited use - but if you compose it together with other blocks, it allows you to create complex UIs and rich user experiences. By splitting the UI into components, it becomes easier to think of and test each component in isolation. In practice, a component is just a JavaScript function that returns some JSX.

¹For a more thorough explanation of React, we recommend reading through the React docs at <https://reactjs.org/docs/getting-started.html>

²<https://github.com/zeeguu-ecosystem/Zeeguu-Teacher-Dashboard--React>

³<https://reactjs.org/>

6.2.2 JSX

JSX is React's way of handling templating, and is a syntax extension to JavaScript⁴. In a traditional web application, we would require some templating engine to merge logic together with HTML.⁵ With JSX, all component logic is written in JavaScript. This gives the developer access to a full-fledged programming language and allows complex logic to determine what HTML ultimately gets rendered to the screen. This example from the React documentation illustrates JSX in practice:

```
const name = "John Doe"
const element = <h1>Hello, {name}</h1>
```

Figure 6.1: An example of JSX

Any valid JavaScript expression can be put inside the curly braces, and will be evaluated by React before the JSX is turned into JavaScript for the browser to execute.

6.2.3 Hooks

Hooks are a new feature of React introduced in version 16.8.⁶ Hooks allow function components to implement "state" and other React features. State is a way for a component to store internal data consistently across multiple renders. If a component needs a specific piece of data from its parent state, this is passed down as a "prop"⁷. Some examples of hooks are the *useState* and *useEffect* hooks. The *useEffect* hook presents an API for interacting with the lifecycle of a React component. The lifecycle of a component is a way to model the process of when the component is initially rendered to the screen, when some change makes it necessary for the component to re-render, and when the component is being removed from the screen. Hooks allow code execution at different points in the lifecycle - for example, to fetch some data from an API when a component is first being rendered. *useEffect* takes in a function and a dependency array. When any of the dependencies change, the function is executed.⁸

6.3 Details of the system

The dashboard has been bootstrapped using the program "Create React App" developed by Facebook.⁹ Create React App allows us to quickly get a web application up and running, with several important tools pre-configured. In particular, Create React App initializes a project that allow us to write JavaScript using ES6 standard syntax, which

⁴<https://reactjs.org/docs/introducing-jsx.html>

⁵This is how the previous teacher dashboard works, using Jinja2 with Python.
<https://github.com/zeeguu-ecosystem/Zeeguu-Teacher-Dashboard/>

⁶<https://reactjs.org/docs/hooks-intro.html>

⁷Which is just an argument passed to the child function

⁸See figure 6.5 for an example

⁹<https://github.com/facebook/create-react-app>

is then transpiled by Babel¹⁰ to work correctly in browsers that do not support ES6. Additionally, all our JavaScript files are bundled together using webpack.¹¹ Create React App also configures webpack to include hot module reloading, which allow code changes to be reflected in the browser without needing to reload the page in development. This instant feedback helps create a smooth developer experience.

6.4 Modules

The teacher dashboard does *not* use classes like traditional OOP systems. Instead, we use functions as the primary level of abstraction. This goes for any React component, hook, or helper function that is used throughout the application. In order to split logic into different files and separate concerns, we use ES6 modules. ES6 modules allow us to export functions, components and objects in their own files, and then import them whenever we need. An example can be seen in the `apiCohort.js` file¹², which exports several functions that all communicate with the Zeeguu API. These functions can then be imported and invoked from anywhere in the application.

6.5 Communicating with the Zeeguu API

The teacher dashboard is built around the Zeeguu API. The API exposes several endpoints that we can use, like creating classes and getting student data, and communicate through standard GET and POST requests. Any front-end action that require persistent storage has to go through the API¹³, which then connects and stores the data in the database. For communicating with the API, we create a JavaScript function for each endpoint we want to hit. Each of these functions then call the more general `apiGet` or `apiPost` functions¹⁴, which take in an endpoint and, if required, some data. These functions will then retrieve the `sessionID` from cookies and pass it along with the request, allowing the server to verify if the user has permission for the requested action.

6.5.1 Example API communication

For actually sending the requests, we use a library called `axios`.¹⁵ `Axios` is based on ES6 Promises¹⁶, which is a different approach to traditional callback functions. The point is that we can not guarantee when the API will respond. We also do not know if it will respond successfully with the requested data or throw an error. By returning a

¹⁰<https://babeljs.io/>

¹¹<https://webpack.js.org/>

¹²<https://github.com/zeeguu-ecosystem/Zeeguu-Teacher-Dashboard--React/blob/master/zeeguu-teacher-dashboard/src/api/apiCohort.js>

¹³Except for a users `sessionID` and preferred time period to display data for, which are stored as cookies in the browser

¹⁴Which can be seen in the `apiEndpoints` file, <https://github.com/zeeguu-ecosystem/Zeeguu-Teacher-Dashboard--React/blob/master/zeeguu-teacher-dashboard/src/api/apiEndpoints.js>

¹⁵<https://github.com/axios/axios>

¹⁶https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Promise when hitting an API endpoint, we allow the component submitting the request to decide what should happen when the Promise *resolves* with the data or *rejects* with an error. A practical example of this is the `onUpdateCohort` function invoked when a user is updating class information through a form, see figure 6.2.

```
38  const onUpdateCohort = form => {
39    setFormStateIsError(false)
40    updateCohort(form, cohortId)
41      .then(result => {
42        setFormIsOpen(false)
43        getGeneralCohortInfo(cohortId).then(({ data }) => {
44          setCohortInfo(data)
45        })
46      })
47    .catch(err => setFormStateIsError(true))
48  }
```

Figure 6.2: `onUpdateCohort` example of using Promises for API communication and error handling

This function first removes any errors that might be present from earlier form submissions. Then it calls the `updateCohort` endpoint, passing in the current form data and its own `cohortId`. Since `updateCohort` returns a Promise, there are two scenarios to consider. The `.then` block is executed if and when the Promise resolves successfully, passing in the returned data¹⁷. The component then closes the open form dialog, before hitting the `getGeneralCohortInfo` endpoint to retrieve the updated cohort information from the server. This returns a new Promise. If this Promise resolves, the data is used to update the component state, triggering a re-render. If any of the promises fail, all errors are caught in the `.catch` block, and the `formState` is updated to show an error in the UI.

6.6 React component hierarchy

React creates a component hierarchy by binding its root level component to a specified DOM node. From here, each component will recursively render all of its child components until a tree has been fully constructed, see figure 6.3.

The root component in our case is called `App` by convention. The `App` component is responsible for handling routing within the application, and to verify that a user is logged in with the proper permissions to access the teacher dashboard. As mentioned earlier, each component can have state. In order to keep each component as reusable and encapsulated as possible, `App` contains state that is relevant to most of the component tree - like the currently authenticated user. This is then made available to the tree through `React Context`,¹⁸ allowing any component that requires user details to pull those out of the `App` state rather than having to maintain that piece of state themselves.

¹⁷here named "result"

¹⁸<https://reactjs.org/docs/context.html>

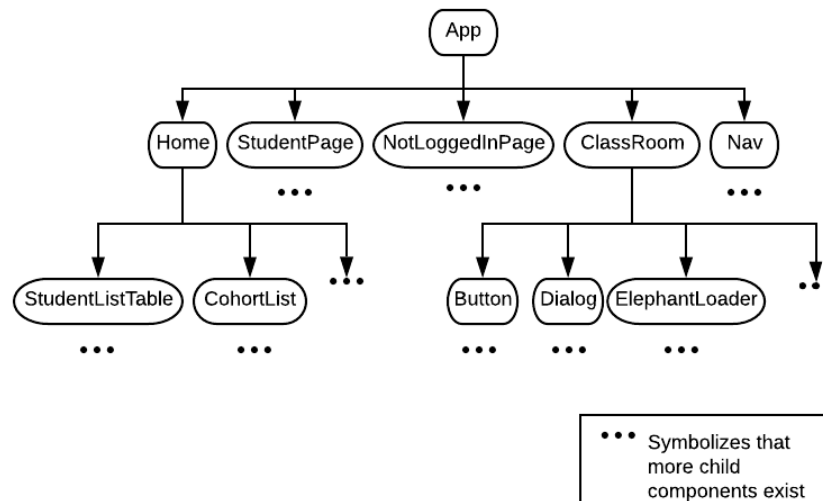


Figure 6.3: Representation of a React compnenet tree

6.7 Routing

Inside the App component, we leverage a Router component exported from an external NPM package.¹⁹ The router is what powers our SPA while making the application *feel* like a "traditional" website with different pages. This works by parsing the URL of the browser, and looking at the predefined paths of the Routers child components. If the URL matches any of the child components path, that component is rendered. This concept, while simple, provide us a powerful API to write declarative routing as seen in figure 6.4.

```
<div>
  <Nav />
  <Router>
    <Home path={` /` } />
    <Classroom path={` /classroom/:cohortId` } />
    <StudentPage path={` /student/:studentId` } />
  </Router>
</div>
```

Figure 6.4: Declarative navigation with Reach Router

In this example, the entire page content is rendered within a `<div>`. We want the `<Nav>` component to be present on all "pages"²⁰ of the application, which is why it is declared outside of the Router component. Each child of the Router is a component with a specified path. This is the path that gets matched to the browser URL. The colons in front of `cohortId` and `studentId` indicate that the ids are variable and will

¹⁹<https://www.npmjs.com/package/@reach/router>

²⁰Which are really just DOM elements being added and removed from the same page, but feel like different pages

be parsed from the URL. This allows us to, for instance, navigate to `/classroom/1`, where the `1` will then be passed to the Classroom component as a prop. The Classroom component then uses this id to hit the API and fetch the appropriate cohort data.

Because all routing is handled on the client side, the browser only ever requests one HTML page from the server: `index.html`, when the user navigates to the root URL of the teacher dashboard.

6.8 Adding new texts

As described in section 3.2.2, one of the desired features of the system is for the user to be able to add their own texts to a class. This is implemented by allowing the user to upload `.txt` files, or write a text directly in the UI. When a text is uploaded, we parse the content directly on the client using the `FileReader` browser API. This gives us a string that we can use to generate a summary of the text using the first 30 words. The name of the uploaded file becomes the text title, and the teacher becomes the author. Using this information, we construct an article object with the following properties, matching the properties of an article in the database:

- Title
- Content
- Author
- Summary
- Language Code, which is retrieved from the current class

Since we allow the user to upload several files at once, and we have no control over the size of these files, we wait until all files have been parsed and turned into JavaScript objects. When this is done, we send an array of objects to the server, which stores them in the database. If the texts are stored successfully, the server will return a 200 OK response. When this happens, we hit the API again to retrieve the list of all texts associated with the class. This is then used to re-populate the UI with the updated list, creating a nice user experience with nearly instant feedback. Of course this can be done with vanilla JavaScript, but using a framework like React greatly reduces the complexity of the implementation. This effect can be elegantly written in 5 lines of code using a hook,²¹ see figure 6.5. In this instance, whenever the `refetchArticles` variable changes, the hook is run and hits the `getArticles` API endpoint with the current cohort id. When the API returns, we call `setArticles` with the retrieved list of articles to update the components local *articles* state, which triggers a re-render of the UI.

The existing Zeeguu API and Core projects required some modification to support the addition of custom texts. More on this in section 6.10.

²¹See section 6.2.3

```

58   useEffect(() => {
59     |   getArticles(cohortData.id).then(result => {
60     |     |   setArticles(result.data)
61     |   })
62   }, [refetchArticles])

```

Figure 6.5: An example of the useEffect hook to fetch articles from the server

6.9 ArticleList component

To further illustrate the benefits of using React for a project like this, we take a look at the ArticleList component. This component is responsible for rendering the list of articles discussed in section 5.4.2. The code can be seen in figure 6.6.

```

147  const ArticleList = ({ articles, deleteArticle }) => {
148    |   return (
149    |     <div className="article-list-container">
150    |       <h4>Article list</h4>
151    |       <ul className="article-list">
152    |         {articles.map(article => {
153    |           |   return (
154    |           |     <li key={article.id}>
155    |           |       <Chip
156    |           |         |   label={article.title}
157    |           |         |   onDelete={() => deleteArticle(article)}
158    |           |         |   className="article"
159    |           |       </li>
160    |         </li>
161    |       )
162    |     })}
163    |   </ul>
164    | </div>
165    | )
166  }

```

Figure 6.6: ArticleList component

This component takes two arguments: an array of articles to display, and a deleteArticle function. Both of these arguments are defined elsewhere - the component is simply responsible for rendering the list. It does this by iterating through the array using the JavaScript *map* function, which returns a list item with a chip component inside for each item in the array. Each chip is bound to call the deleteArticle function when the user clicks the cross in the chip, passing the associated article.

6.9.1 Generalizing the ArticleList component

What we notice is that the ArticleList component does not have any logic related to articles. In fact, almost all the article specific details of this component are just names, except for the id and title properties. We could pass it an array of any objects,

and it would render just fine assuming the objects have an id and a title property. By changing the naming, we can create a more generic component, see figure 6.7.

```
147 | const ItemList = ({ items, deleteItem }) => {  
148 |   return (  
149 |     <div className="item-list-container">  
150 |       <h4>Item list</h4>  
151 |       <ul className="item-list">  
152 |         {items.map(item => {  
153 |           return (  
154 |             <li key={item.id}>  
155 |               <Chip  
156 |                 label={item.title}  
157 |                 onDelete={() => deleteItem(item)}  
158 |                 className="item"  
159 |               />  
160 |             </li>  
161 |           )  
162 |         })}  
163 |       </ul>  
164 |     </div>  
165 |   )  
166 | }
```

Figure 6.7: A more generic implementation that allow for better reusability

Now, this component can be used anywhere we want to display a list using the chip component. For example, the UMR has a list of topics that the user is not interested in, see figure 6.8. If the UMR was also implemented in React, it would require next to no coding to implement this list using our now generic ItemList component.

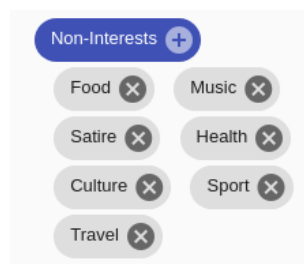


Figure 6.8: The UMR list of non-interesting topics

This is one of the biggest benefits of using a framework like React that support the component model. We can create components that any React project can implement with minimal work. This speeds up development time and minimizes the amount of custom code. Custom code requires separate tests to verify behavior. If we use a general component, we can test it once and be confident that the behavior is identical across all instances of the component.

6.10 Expansion of other Zeeguu projects

Most of the back end functionality of the new teacher dashboard was already present in the Zeeguu API and Core projects. However, the new features required some work on the server as well. This section will detail the necessary modifications on the Python server in order to support the new features. Furthermore, the UMR needed some adjustments to display the new teacher submitted texts to the students.

6.10.1 API & Core

The API is a Flask app running on the server. The API required three new endpoints to support uploading and deleting articles, as well as getting all articles associated with a cohort. These actions all require that the user is a teacher of the cohort. The signatures of the new endpoints can be seen in figure 6.9.

```
@api.route("/upload_articles/<cohort_id>", methods=["POST"])
@with_session

@api.route("/remove_article_from_cohort/<cohort_id>/<article_id>", methods=["POST"])
@with_session

@api.route("/cohort_files/<cohort_id>", methods=["GET"])
@with_session
```

Figure 6.9: Signatures of the new API endpoints

The *upload_articles* endpoint accepts a POST request containing a list of JSON objects. Each object contains data like the title, content, summary etc. of an article.²² The API will then iterate over the list, and use the Article model defined in Core to construct an article object for each item in the list. Before this is saved in the database, the function will construct a CohortArticleMap object, linking the new article to the provided cohort.²³ If all goes well, the API returns a 200 OK HTTP response.

The *remove_article_from_cohort* endpoint takes in two variables as query parameters: The cohort from which to delete an article, and the id of the article to delete. The function will then look up the article in the CohortArticleMap and remove it from this table. It is important not to fully delete an article from the database. This would break the displaying of reading sessions in the student page²⁴ because reading sessions rely on data from the article. By simply deleting the link between a cohort and an article, we make sure the article will not show up in the UMR or in the list of articles associated with a cohort in the teacher dashboard.

The *cohort_files* endpoint simply takes a cohort_id and retrieves all the articles associated with this cohort. This is used to display a list of the articles in the "Manage File" dialog from section 5.4.1.

²²The same data as described in section 6.8

²³Obtained from the cohort_id submitted as a query parameter of the endpoint

²⁴see section 5.5

6.10.2 Unified Multilanguage Reader (UMR)

Naturally, the UMR needed to be expanded to show teacher uploaded articles. The UMR is written in JavaScript, and previously consisted of two tabs in the UI: one "main" tab presenting all the articles available for reading, and another tab presenting all articles the user had "liked" or marked for later reading.

From a design perspective, we came up with two valid ways to show teacher uploaded / class articles in the UMR:

- Have the main tab display class articles on top, above all regular news articles.
- Create a new tab in the UMR interface, next to the previous two tabs. This new tab would then contain a list of all the articles uploaded to the students class.

The problem with the first approach is that an article disappears from the list when a student has given feedback on it, which is not immediately clear. Thus, a class article might disappear from the students reading list when it really should persist until the teacher decides to remove it. Additionally, the class articles are more likely to blend in with the news articles, making it less obvious which articles the teacher has added.

The second approach provide a much better solution. The class articles are separate from news articles, making it clear which articles are teacher uploaded. The articles also persist despite the user giving feedback. This solution proved easier to implement as well. As the functionality of this tab is basically identical to the other tabs, except for hitting a different endpoint, we can reuse much of the code. This works by requesting all the class articles of the current logged in user when the page loads, then using this data to construct and render an HTML list of articles.

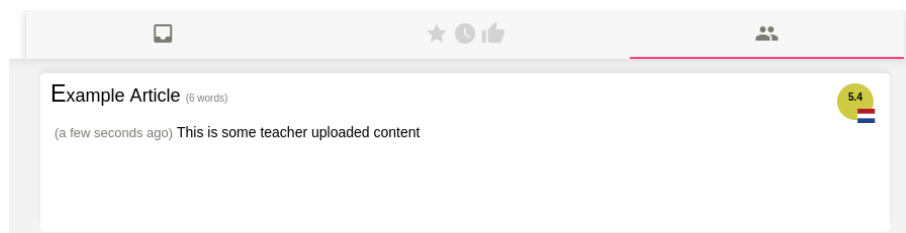


Figure 6.10: The updated UMR UI, featuring a new tab for class articles

6.10.3 Conventions and naming

When implementing functionality on the server, focus was on following the established conventions in terms of naming and documentation. Each endpoint is documented with a comment briefly explaining the purpose of the endpoint. If the code fails, we return the appropriate 400 or 401 status codes to notify the caller. Descriptive names are preferred for every declared function and variable. We also avoid hard-coded constants, as the meaning of those are typically only clear to the author of the code. Furthermore, the code follows the Pep8 Style Guide to improve readability and promote cohesiveness in the codebase.

Results and Evaluation

This chapter will seek to evaluate our solution based on the research questions presented in chapter 3. The research questions focus on different areas, namely implementing the new features and providing a nice user experience while also ensuring good overall code quality. The evaluation will be split to evaluate the questions individually.

The system supports all of the functional requirements. This is easily verified by using the system. The non-functional requirements are more open-ended and harder to measure. In order to test whether the system is intuitive to use, we contacted several teachers and asked for their feedback. The results are presented in section 7.1. To evaluate if the new system is maintainable and accessible, we use two different analysis tools. The tools and results of this evaluation are presented in section 7.2.

7.1 User Feedback

As mentioned in chapter 1, the primary focus has been on creating a system that aid teachers in their job of teaching new languages. Thus, having actual teachers test and evaluate the new dashboard was of paramount importance. For this purpose, we decided to go with a qualitative analysis with selected teachers. We deployed the new dashboard live on the Zeeguu.org website and contacted 18 teachers that all had previous experience with Zeeguu. Of those 18, 6 responded saying they would like to try out the new dashboard. We then asked those 6 to test out the new dashboard and presented them some questions¹:

- Do you find the new dashboard easy / intuitive to use?
- Did you run into any problems?
- Does it suit your needs?
- Do you have any ideas for improvements?

¹The full email we sent to the teachers can be seen in A.3 in the Appendix

We also asked these teachers to accomplish some basic tasks using the new dashboard, like uploading a file, writing a new article and exploring the student page. We encouraged the teachers to provide us *any* feedback they had on the new dashboard, either via email or, if they preferred, a UX session on Skype or similar.

Of the 6 teachers that expressed interest in the new dashboard, 2 responded and provided thorough and valuable feedback. Some of the positive feedback can be seen in figure 7.1.

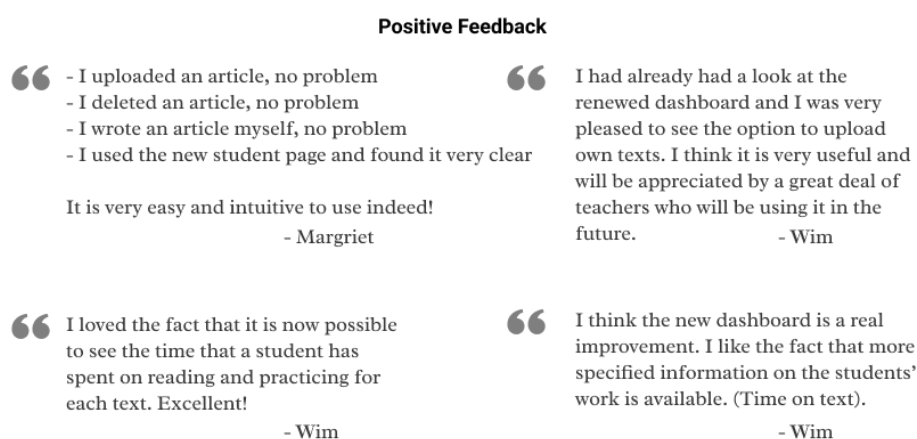


Figure 7.1: An extract of the positive feedback received from teachers

The teachers also had some suggestions for improvements, see figure 7.2.

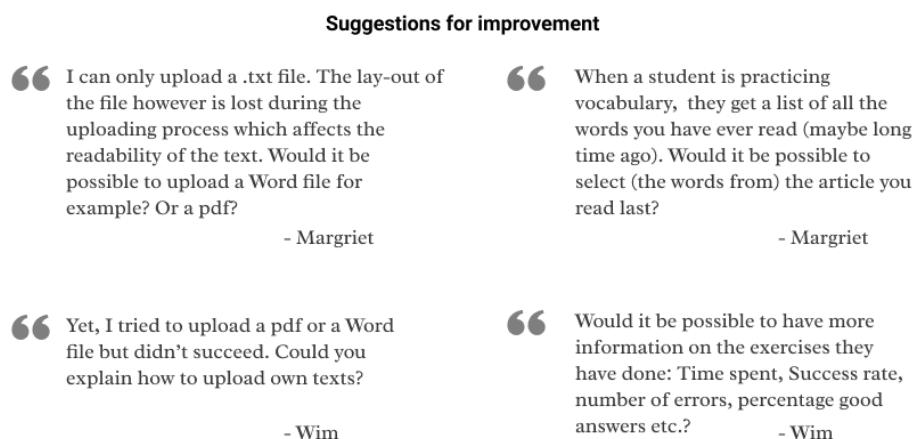


Figure 7.2: Some of the suggestions and improvements received from teachers

Overall, a thorough evaluation would require more teacher feedback. Ideally the teachers should have the opportunity to use the new dashboard with their classes for a few months, to assess if the new dashboard provide value and assist the teachers in their jobs. However, the feedback gathered so far is very positive. Both participants feel that the new dashboard is easy to use and that the ability to upload their own texts will be valuable in the future. One participant also mentioned that the new dashboard is a real improvement².

²see figure 7.1

Worth noting is that both of them expressed interest in being able to upload other types of files. They also mention better support for exercises within the dashboard. This is inline with our own analysis in chapter 9, and would be nice additions in the future.

7.2 Code quality

Another important goal of the new dashboard was to have a codebase that is extendable and easy to maintain. This can be hard to measure quantitatively, as code maintainability is vaguely defined and depends on the individual developer. However, some tools exist that can statically evaluate a codebase based on certain parameters. We evaluate the new teacher dashboard, written entirely in React and Sass with Material UI, against the old teacher dashboard written in Python with Jinja2, jQuery/JavaScript and CSS.

7.2.1 Lighthouse

One tool to analyze webpages is called Lighthouse³. Lighthouse is developed by Google, and is available as a browser extension or directly in the Chrome Developer Tools⁴. Lighthouse can evaluate on up to 5 parameters:

1. Performance
2. Accessibility
3. Best Practices
4. Search Engine Optimization (SEO)
5. Progressive Web App (PWA)

The evaluation is performed based on several checks. For instance, Performance is measured on "Time to Interactive" and "First Meaningful Paint" among others. Some of the Accessibility checks are that all images and buttons have "alt" attributes to improve the web experience for users using screen readers. It also tests if text has an appropriate contrast ratio in accordance with WCAG. Best Practices tests things like if the site is using HTTPS and that no browser errors are logged to the console. Lastly, SEO checks for semantic HTML, that font sizes are appropriately large etc. For this evaluation, we leave out PWA as neither the old or the new teacher dashboard are implemented as a PWA.

The results of running Lighthouse against the old and new teacher dashboards can be seen in figure 7.3.

³<https://developers.google.com/web/tools/lighthouse/>

⁴Under the "Audits" tab.

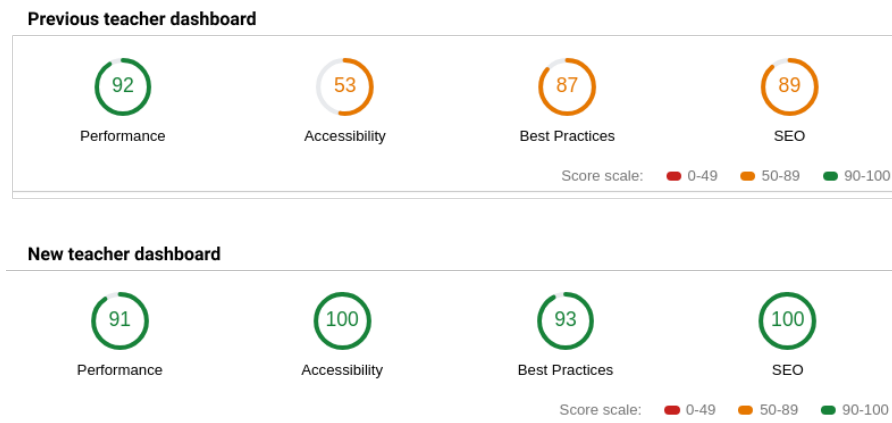


Figure 7.3: Audits performed on the previous and new teacher dashboard using Lighthouse from Chrome Developer Tools.

We notice that Performance is practically identical, while the new dashboard features a much better Accessibility score, with a slightly better SEO and Best Practices score. Accessibility is important for the user experience, allowing people with disabilities to use the website on equal footing with people who do not suffer from a disability. The European Accessibility Act⁵ is a proposal that would require businesses to be accessible by law - meaning the codebase would need to be accessible at one point or another.

7.2.2 Sonarcloud

Sonarcloud⁶ is a tool used to statically analyze a codebase. Sonarcloud attempts to find bugs, code smells and security violations in the code, while also highlighting the amount of duplication and test coverage. The results of Sonarcloud run against both dashboards can be seen in figure 7.4.

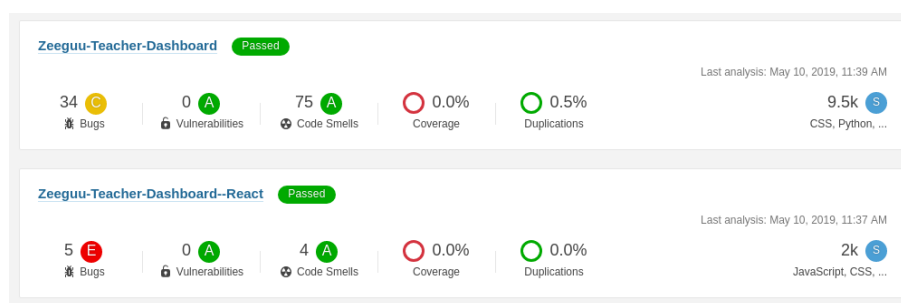


Figure 7.4: Top is the analysis of the **old** dashboard. Bottom is the analysis of the **new** dashboard with Sonarcloud. The results can be seen at <https://sonarcloud.io/organizations/zeeguu-ecosystem/projects>

As shown, the new dashboard has fewer bugs and code smells than the old one. Sonarcloud determines something as a bug if it breaks a certain rule. However, several

⁵<https://ec.europa.eu/social/main.jsp?catId=1202>

⁶<https://sonarcloud.io/>

of the "bugs" found in the new dashboard are not actually bugs - just Sonarcloud not understanding a specific syntax.⁷ The code smell metric is more interesting to look at, as it points out inconsistencies in the code and failure to follow best practices. An example includes exporting a component with a name that does not match the name of its file, or declaring unused variables. The new dashboard heavily outclasses the old one in this category.

Another interesting metric is Lines of Code (LoC). The old dashboard uses 9.5k LoC while the new one only uses 2k. It is worth noting that almost 7k lines in the old dashboard are CSS, most likely imported from a framework. A detailed look of the language breakdown can be seen in figure 7.5.

Old dashboard



New dashboard



Figure 7.5: A breakdown of languages used in the different dashboards

The old dashboard uses a mix of different languages and approaches that make it harder to maintain, while the new dashboard is almost entirely JavaScript. It is also interesting that, ignoring styling, the old dashboard is almost 2600 lines of Python, HTML and JavaScript while the new dashboard is only 1600 lines of JavaScript despite supporting more features.

⁷In our case, a .scss file exposes some variables using :export. This is deemed a bug by Sonarcloud, but works in practice.

7.3 Further evaluation

Besides the maintainability issues that can arise when mixing languages, the old dashboard has some other problems. The dashboard makes heavy use of inline styles, which is bad for several reasons:

- **Maintainability.** As styling is declared individually on elements, updating styles require changes in multiple places, making it hard to enforce consistency in the application.
- **Readability.** The HTML is harder to read at a glance, as it is cluttered with styles. It is also harder to visually imagine what element on the page you are looking at.⁸
- **Caching.** The browser will cache CSS files when possible, but has no way to cache styling from inline styles.

The old dashboard also makes heavy use of inline JavaScript, which is hard to maintain for the same reasons. Comparatively, the new dashboard take advantage of best practices regarding separation of concerns, like keeping related JavaScript in its own files, and breaking Sass styles into manageable chunks for better modularity. The component model used in the new dashboard enforces all logic and behavior to remain within a component itself, making the overall system easier to maintain.

It is difficult to objectively verify that the new dashboard is more maintainable and has a higher code quality than the old one. The tools employed in this section clearly favor the new dashboard, but it is unclear exactly how the code is evaluated with Sonarcloud. For this reason the results should be taken with a grain of salt.⁹ Qualitative analysis of developers modifying the code over time would provide more useful data to evaluate the maintainability and quality of the code.

⁸Compared to using a class like "card", where it is typically obvious which element in the browser is the card.

⁹Especially since Sonarcloud also reported a bug in the new dashboard that is not actually a bug, as mentioned earlier

Conclusion

In this project, we explored the front-end web development landscape. Through data-driven analysis, we decided to leverage the modern JavaScript framework React to implement a new teacher dashboard to support language teachers in their jobs. We chose React due to its developer satisfaction, large community and its component model. React, combined with Material Design, allowed us to rapidly build out a new teacher dashboard that supports all the functional and non-functional requirements defined in chapter 3. A large focus has been on creating a system that provides a simple and intuitive user experience. To decide whether or not this has been accomplished, we deployed and tested the system with select teachers. Another important element has been the code quality of the new system. This is compared against the old teacher dashboard, using multiple code analysis tools.

The user feedback, although limited, clearly indicates the excitement surrounding the new features. The users find the system easy and intuitive to use and they like the re-design. The ability to upload files is especially valuable to the users, as they have already expressed their interest in using the feature as it is, but also how it can be improved in the future. Furthermore, the code quality and maintainability appears to have improved with the new teacher dashboard React implementation. This is hard to verify, but the code analysis tools employed in chapter 7 clearly favor the new dashboard.

To more thoroughly evaluate the user experience of the new dashboard, more testing is necessary. Ideally, several teachers would use the dashboard in their classes over extended periods of time. This would give a more indicative picture of the dashboard's usefulness in practice. Code maintainability is also better measured over time, when other developers have had a chance to modify the system. However, the initial results of the project are very promising and constitute a solid foundation for future work and improvements.

Future work

As with any software product, there is always room for improvement. Our system has been designed with maintainability and extensibility in mind, allowing improvements to be implemented with relative ease in the future. This section will present some possible improvements to the teacher dashboard. Some of these features are based on user feedback gathered in the evaluation phase. The rest are new features we would consider useful or smaller improvements to the current implementation.

9.1 Uploading different files

In the current implementation, it is only possible to upload .txt files as articles. In the future, it could be nice to allow .docx or .pdf files to be uploaded as well. This would require more code to parse the file, but would benefit the user since they could write the content in an editor like Microsoft Word and upload it directly. Pdf is also a widely used format for online documents, increasing the value of being able to upload pdf files. Some design decisions would have to be made, including how to handle images and layout as part of an uploaded file.

9.2 Integrating exercises with the teacher dashboard

The new teacher dashboard does not focus much on the part of Zeeguu where students can perform exercises. Feedback gathered in chapter 7 indicate that it would be nice to have more data on how students are using the exercises - for instance how often they get an exercise right and their number of errors.¹.

9.3 Student interaction with a text

An interesting idea would be to create a view for each text associated with a class. This view could contain data on how the students have interacted with the particular text, for example:

¹See figure 7.2

- How long did each student spend on this text, and what is the average?
- Which words were translated, and with what frequency? This could be presented visually using graphs, to give the teacher an overview of which words or sentences the students are mostly translating.
- How many people finished the article, and what feedback did they give?

9.4 Lazy loading reading sessions

When visiting a student page, all their reading information is retrieved for the selected time period. This could be hundreds of articles with thousands of translations. Currently, the system will show a loading spinner until all the data has been received. This could be improved to lazy load the reading sessions based on time, eg. begin by loading the last 10 reading sessions and show those to the user right away. If the user scrolls down to see more, then load the next 10 sessions and so on. This would improve the user experience significantly, by not forcing them to wait for content they might not interact with.

9.5 Real-time updating reading activity

When a teacher is browsing a list of students, the reading time will not be updated in real-time. If a teacher does not refresh the page for a while, the information might be outdated if a student has been reading in this period. Although minor, the system could be modified to update the UI in real-time when a student's reading data changes.

Bibliography

- [1] <https://www.akamai.com/de/de/about/news/press/2009-press/akamai-reveals-2-seconds-as-the-new-threshold-of-acceptability-for-ecommerce-web-page-response-times.jsp>. Akamai reveals 2 seconds as the new threshold of acceptability for ecommerce web page response times. 2009.
- [2] <https://www.marketingdive.com/news/google-53-of-mobile-users-abandon-sites-that-take-over-3-seconds-to-load/426070/>. Google: 53% of mobile users abandon sites that take over 3 seconds to load. 2016.
- [3] Montclair State University. The importance of learning a modern language in a globalized world. <https://www.studyinternational.com/news/importance-learning-modern-language-globalized-world/>, April, 2017.
- [4] Mohammad Reza Ahmadi¹. The use of technology in english language learning: A literature review. 2018.
- [5] Mircea F. Lungu, Luc van der Brand, Dan Chirtoaca, and Martin Avagyan. As we may study: Towards the web as a personalized language textbook. <https://github.com/zeeguu-ecosystem/CHI18-Paper/blob/master/!AsWeMayStudy--Preprint.pdf>, April, 2018.
- [6] Dan Chirtoaca. Simplicity and intuitiveness in a personalized multilingual reading tool. <http://fse.studenttheses.ub.rug.nl/15434/1/ChirtoacaDan.pdf>, June, 2017.
- [7] Luc van der Brand. Efficiency and usability in a personalized multilingual feed manager. http://fse.studenttheses.ub.rug.nl/15435/1/Prometheus_-_Luc_van_den_Brand.pdf, June, 2017.
- [8] <https://github.blog/2018-09-06-removing-jquery-from-github-frontend/>. Removing jquery from github.com frontend. 2018.
- [9] <https://blog.npmjs.org/post/180868064080/this-year-in-javascript-2018-in-review-and-npms>. This year in javascript: 2018 in review and npm's predictions for 2019. 2018.

- [10] <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>. Single-page applications: Build modern, responsive web apps with asp.net. November, 2013.
- [11] <https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/choose-between-traditional-web-and-single-page-apps>. Choose between traditional web apps and single page apps (spas). 2019.
- [12] <https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-contrast.html>. Contrast (minimum): Understanding sc 1.4.3. 2016.
- [13] Clayton M. Christensen, Taddy Hall, Karen Dillon, and David S. Duncan. Know your customers' 'jobs to be done'. <https://blogs.baruch.cuny.edu/ibmwatson2019/files/2019/02/Know-your-customer-Jobs-to-be-done-Chrstensen.pdf>, September, 2016.

Appendix

A.1 Workflow

Throughout the project, we used git for version control and stored all code freely available on GitHub. We created a new repository for the new teacher dashboard, containing a *master* and *development* branch. Whenever we worked on a new feature, a new branch would be created with the *feature/example* syntax. When a feature was done, we merged it into development to verify that it worked with the rest of the system. When we felt like the system was ready for a release, we would open a pull request to a repository in the Zeeguu Ecosystem containing the new teacher dashboard code. The pull request would generally contain some feedback, some of which would require code changes. Once everyone was happy with the changes, the pull request would get merged and deployed live to the Zeeguu website.

We also used GitHub projects to manage smaller tasks and assignments. GitHub projects works like a Kanban board, having different columns that each task can be put in. Figure A.1 shows a snapshot of the board at a random point in the project.

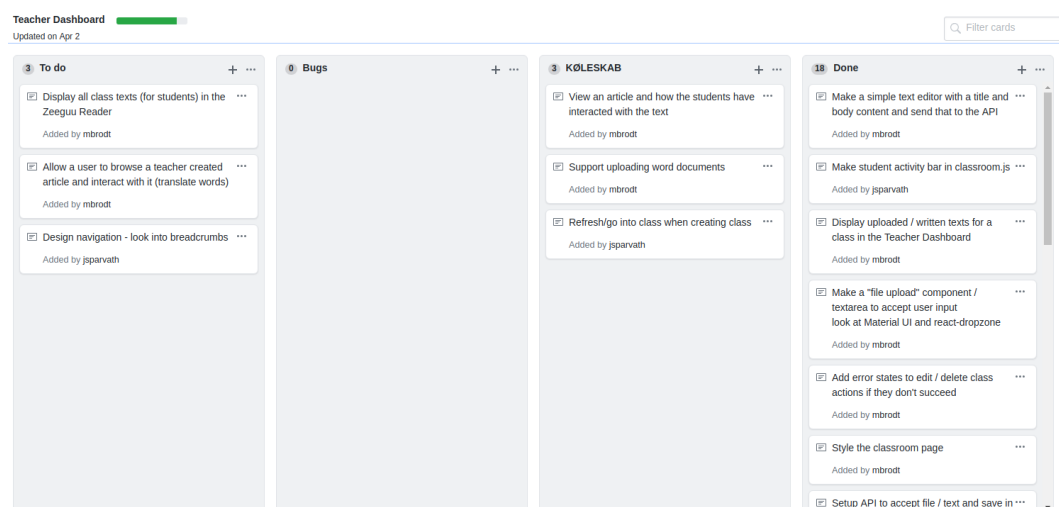


Figure A.1: A snapshot of our GitHub projects board at some point in the project

Using a board like this really improved our workflow. It allowed us to break

features into smaller and more manageable chunks, and focus on one of them at a time. It also allowed us to store features that were more "nice to have", but which we did not have time to prioritize.

A.2 Figures

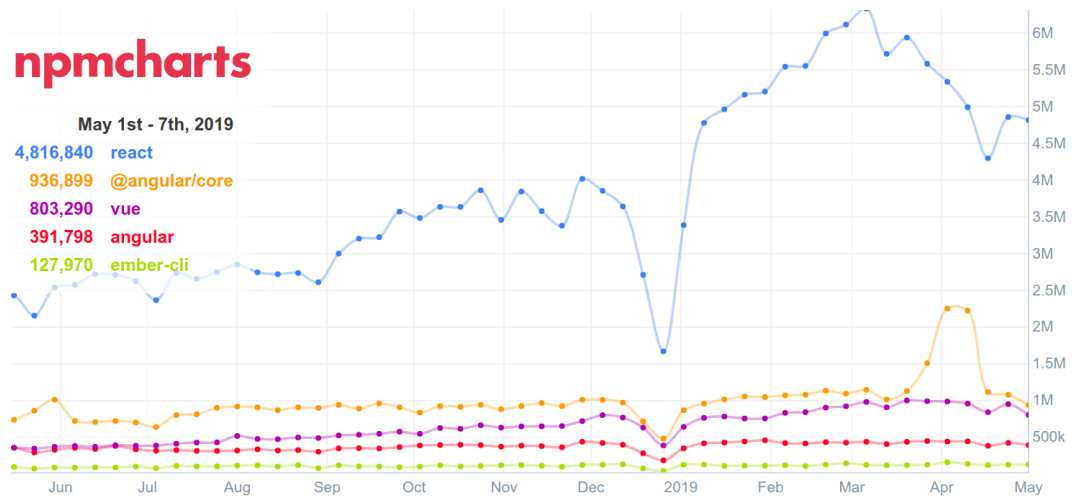


Figure A.2

Source: <https://npmcharts.com/compare/react,angular,@angular/core,ember-cli,vue?minimal=trueinterval=7>

A.3 Feedback email sent to teachers



Figure A.3: The email sent to teachers who expressed interest in providing feedback on the new teacher dashboard