Zeeguu Dashboard Architecture

Version 4.0

Authors :
Oliver Holder
Christian Grier Mulvenna
Henry Salas
Tai-Ting Chen
Evi Xhelo
Ai Deng
Jakob Vokac

Clients :
Mircea Lungu
Carlos Paz Rodriguez

Supervisor :
Lars Holdijk

University of Groningen
March 2018

1 Introduction

Zeeguu is an innovative new tool for learning languages. Zeeguu allows its users to read articles in a foreign language while having real time translation at their finger tips. Zeeguu allows its users to undertake interactive exercises with words they needed translating. Zeeguu supports learning five different languages (English, Dutch, German, Spanish and French) from three base languages (English, Dutch and Chinese).

The current Zeeguu system is not set up for teachers, individuals can sign up to the website and they can be added to classes manually by system admins. But an automated and graphically supported system is needed to facilitate teachers in running a class with Zeeguu. This brings us to the project at hand. The aim of the Zeeguu dashboard project is to extend the Zeeguu website's functionality in order to help teachers use Zeeguu in their classrooms. This functionally should include the creation, mangagement and analysis of classes and students.

2 Zeeguu Architecture Overview

The source for Zeeguu is open and on Github. The source however spans three repositories where each repository contains code for an individual component of Zeeguu. These repositories are the Zeeguu-API, Zeeguu-Core and Zeeguu-Web. Each component is a system that communicates with the other components. When each of these separate systems are successfully running, you have a copy of the Zeeguu website going. Generally speaking, Zeeguu-Web is for running a web framework, Zeeguu-Core is for making operations on a database and Zeeguu-API provides to both other systems procedures like crawling web articles and assessing language difficulty. According to the Zeeguu-API readme, the API is a thin layer on top of the core. Take a look at figure 4 to see the direction of communication between each system. We see bidirectional arrows between the Zeeguu-API and the other two systems since the Zeeguu-Web and Zeeguu-Core want a response from the Zeeguu-API sometimes. The Zeeguu-Web and Zeeguu-Core shouldn't communicate between each other directly, although they currently do in practice. Our implementation does not use this direct communication, but is rather completely reliant on the transitive communication path. That path therefore allows for the Zeeguu-Web to request the Zeeguu-Core to create a new user account in the database for example.

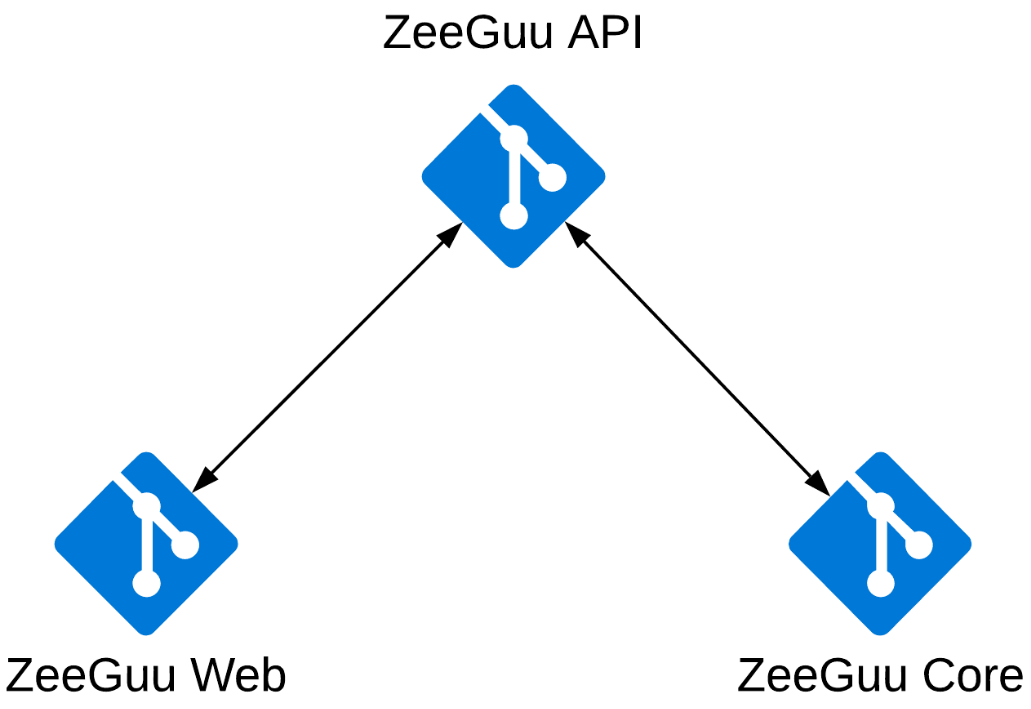


Figure 1 – Communication between the Zeeguu repositories

3 Zeeguu Dashboard Architecture Overview

3.1 Overview of system

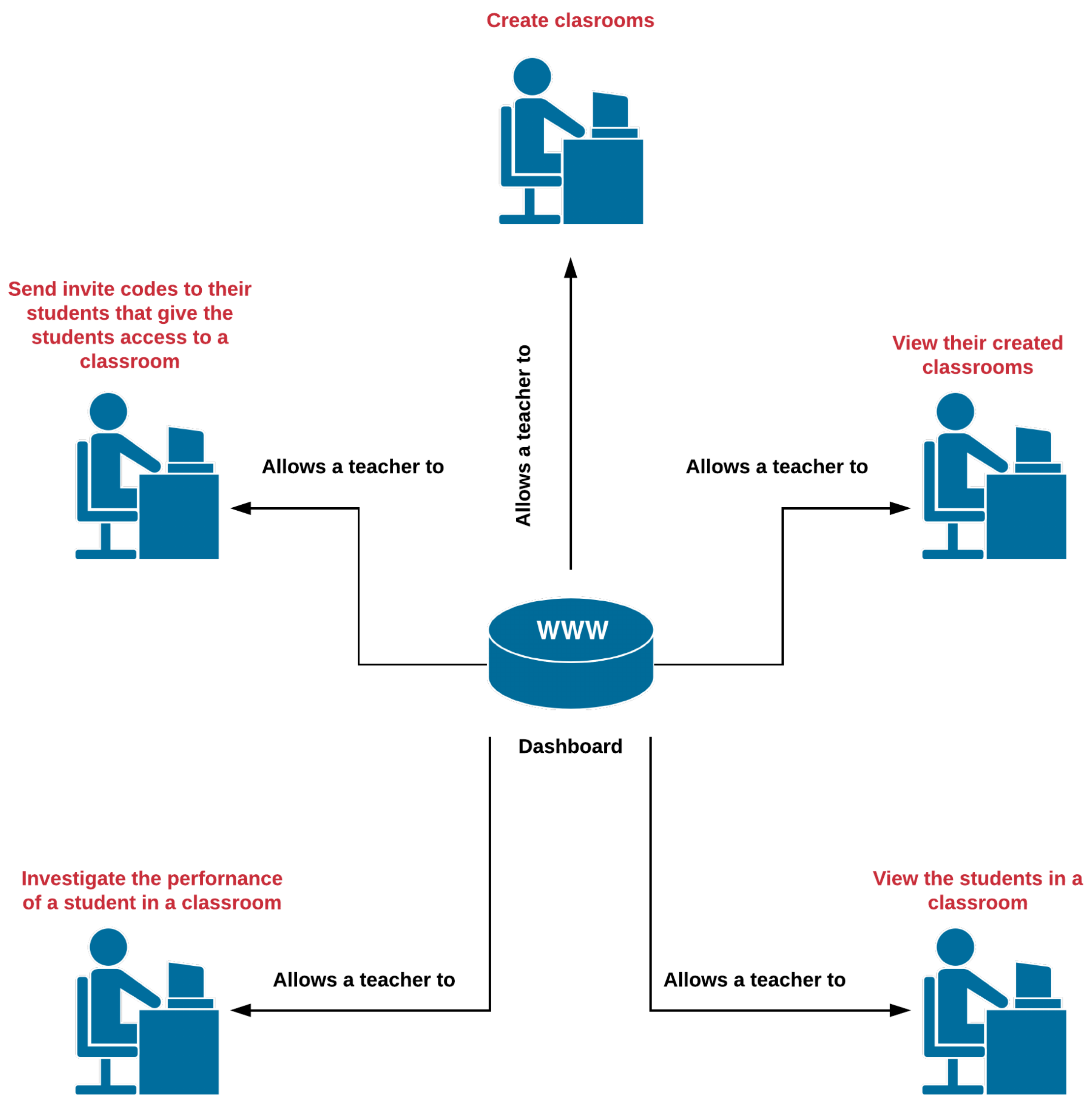

Create clasrooms
Send invite codes to their students that give the students access to a classroom
View their created classrooms
Allows a teacher to
Allows a teacher to
Allows a teacher to
WWW
Dashboard
Investigate the perfornance of a student in a classroom
View the students in a classroom
Allows a teacher to
Allows a teacher to

Figure 2 – Abstract capabilities of the Zeeguu Dashboard

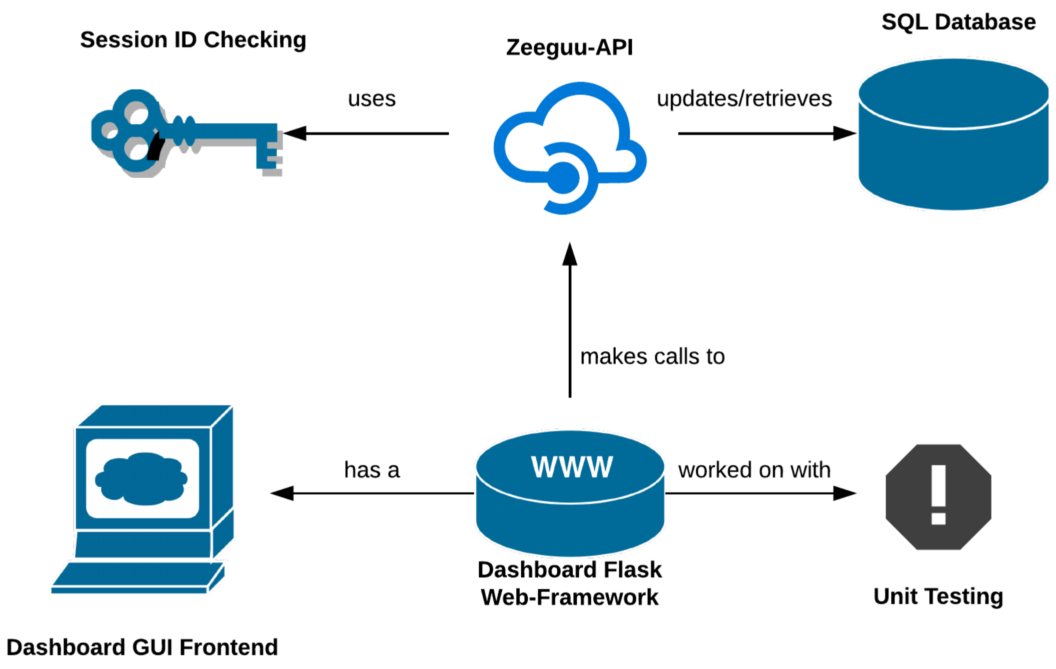


FIGURE 3 – Conceptual elements of the Zeeguu Dashboard

3.2 Project structure

The project is split up in four main parts (plus the static folder, which contains the dependencies like css and javascript).
The api folder contains parts regarding connections to the Zeeguu-API. For now this is just a single modules with the basic functions for making calls.
The page routes folder contains all of the modules with functions that are called based on what endpoint is called. These are divided based on which web page they represent (the classroom page, the error pages, the login page, the homepage and the student info page).
The templates folder contains html templates for all of the web pages. It contains a base page, which is used in all of the other pages (it, for instance, contains the navigation bar).
The util folder contains all of the functions that change data or format it before it can be displayed on the web page. They also contain flask code for WTForms as well as functions which check user permission for visiting pages.

```
/
├── app
│   ├── api
│   │   └── api_connection.py
│   ├── page_routes
│   │   ├── classroom.py
│   │   ├── errorpages.py
│   │   ├── homepage.py
│   │   ├── login.py
│   │   └── studentpage.py
│   ├── static
│   ├── templates
│   │   ├── 404.html
│   │   ├── base.html
│   │   ├── classpage.html
│   │   ├── createcohort.html
│   │   ├── edit_class.html
│   │   ├── homepage.html
│   │   ├── loginpage.html
│   │   └── studentpage.html
│   ├── util
│   │   ├── classroom.py
│   │   ├── forms.py
│   │   ├── permissions.py
│   │   └── user.py
│   └── __init__.py
├── database
│   ├── reset_db.py
│   └── update_db.py
├── config.py
└── main.py
```

4 Functional Architecture

4.1 The Concept of a Zeeguu Class

Our main addition to the Zeeguu Ecosystem with this update is the idea of a classroom, with which the teacher interacts and manages students.

The classroom has : a name, a specified language, a specified number of students and an invite code.

The specified language will make sure that the users get recommendations for the language they are being taught, without having to specify it. The number makes sure that the class does not get cluttered with fake accounts and the invitation code makes sure that the students and only the students can easily join their class, without having to search for it on Zeeguu.

4.2 Sending & Receiving Data To/From The Zeeguu API

We send and receive data using requests, more information here :
http ://docs.python-requests.org/en/master/.
We use the JSON format for all data that is sent across.
All new API calls were added to one file, teacher_dashboard.py. The functions that were added are listed below under Back end Implementation.

4.3 REST API

All information sent between the website and API is in Java-Script object notation.

All 'GET' and 'POST' requests are called through the functions 'api_get' and 'api_post'. This is done to reduce code duplication and unnecessary errors. These functions connect to a private '_api_call' function, which handles the session parameter, that authenticates the call, as well as handles exceptions.

All data gathering formatting and filtering functions are split into corresponding modules in the utility package. These operations include de-jsonifying of objects and filtering out entries in dictionaries. This is split from the routes.py to make each functions actions clear and to increase forward integration with the future features.

All data interactions have been optimised to reduce 'GET' requests to the API. Initially the website would request the API one time for every class and then one time for every student in a class. The request functions were modified to return all class or student info at once. This will reduce the strain on the API server as the project scales.

4.4 Security

4.4.1 User authentication

User authentication
A user must first log in to receive a valid session. Once a user has a valid session it is sent along with each get/post request. Any Endpoints in the rest-api returning data require a valid session Some Endpoints in the rest-api that return protected data require a session that identifies the user as an individual/teacher that has access to that information.

Functions that return data should be protected from users without the correct permissions to view the data. This requires checking the permissions of the individual requesting data on each request. The individual identifies themselves with a Session.

The website will use wrapper functions to call the permission checking functions in the API. It will then either allow the request to be called or redirect to a permissions required page. The use of these wrappers makes it easy to implement permission checking on future pages/functions.

The API is to have both permission checking functions for the website, and use the permission checking functions when called to return data.

Login page was added to dashboard for the attaining and using of sessionID's to test the dashboard.

4.4.2 Database attacks

SQL-alchemy handles SQL injections internally, so we did not have to implement any extra security for this issue. As for user authentication, we will rely on Zeeguu's already implemented interface for this.

4.5 Updates to the Zeeguu Core & API

4.5.1 Updates

The model for cohort was changed. The variable invitation_code was changed to inv_code. The variables max_students, language_id and language. 'language' is a foreign key linked to 'language_id'

The methods 'cohort_still_has_capacity' and 'get_current_student_count' were added to cohort.

get_current_student_count returns the current number of students in the database that are connected to the cohort.

cohort_still_has_capacity returns true is get_current_student_count is lower than max_students. And false otherwise. This function is used when assigning users to a cohort.

4.5.2 Conversion from old database to new database

The script of changing old database to the new one is using the connector in Python for communicating with MYSQL server. We then have connection to the database with the host and its root, password and the database which need to be used. Then, with updating the database, we check if the column exist in the table. The method of checking the column is using information_schema from mysql query to see whether we are able to get any columns with the same column name. Then information_schema is providing the access to database metadata, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. In short, it provides you the specific table or column name in the database. If there is no result for the searching, adding columns into the table, this are the case for column name of max_students and language_id. For the invitation code, we change it into inv_code as the column name and add it an unique key to make sure that invitation code is not repeatable. After those steps, we close the connection to the database.

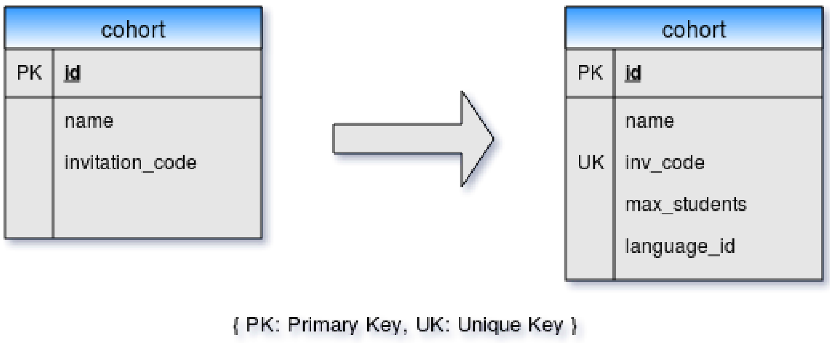


FIGURE 4 – Cohort database conversion

5 GUI Architecture

5.1 Base

All pages should inherit from the `base.html`. This file gives the basic outline for how a page should look and also declares all the dependencies the project has such as bootstrap and jQuery. The components included in this file are the :

1. Navigation bar : The navigation includes the title of the website along with an optional jinja2 block to add buttons. These buttons are added when an html file extends from base.
2. Content block : The jinja2 block to extend to add content to the website.

For instance, one of the simplest pages are the homepage, which looks as follows :

5.2 Homepage

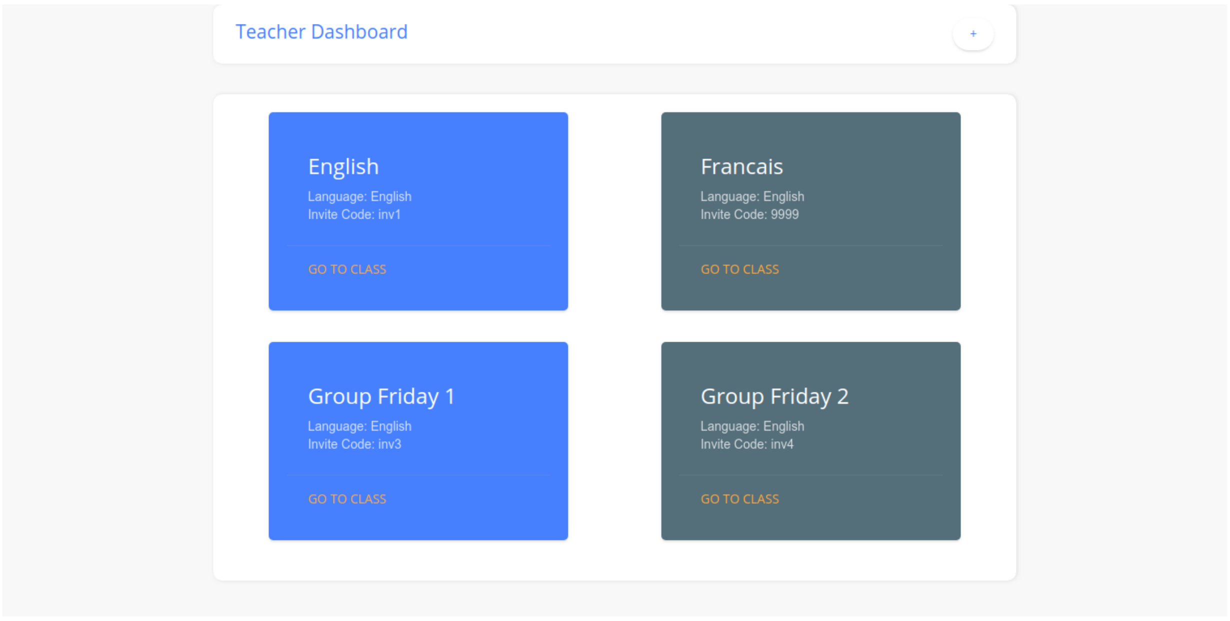


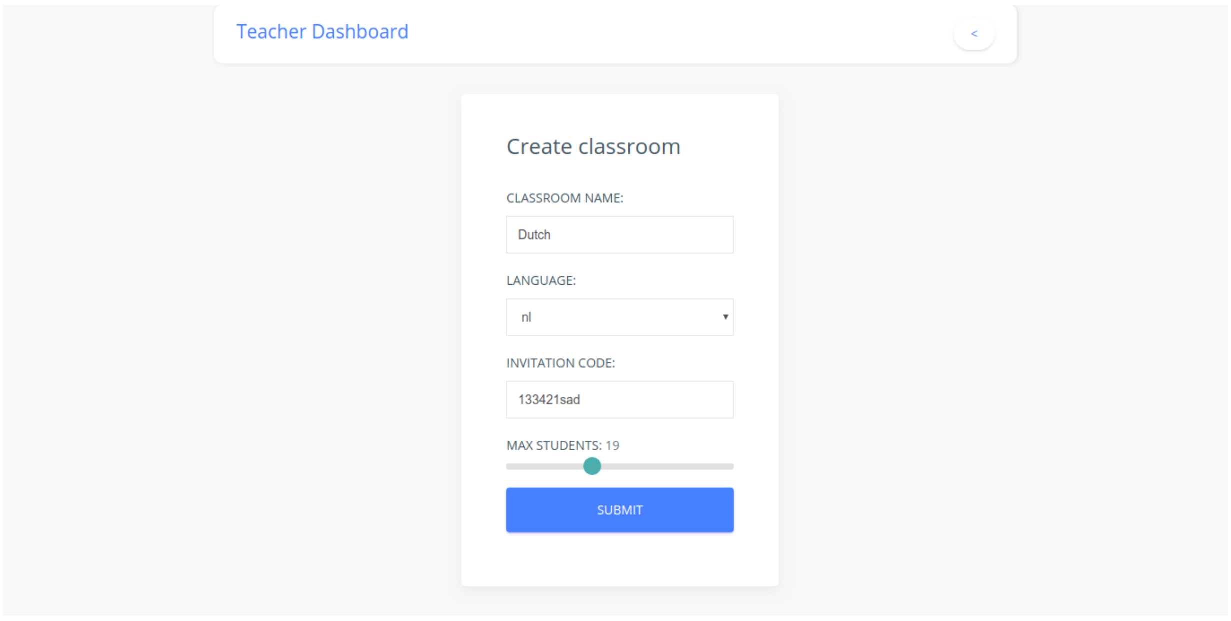
FIGURE 5 – Homepage

As expected, the homepage extends from base. The design is material-like, and should obey general material theme principles all throughout the website.

For the content block, the homepage simply lists all the classes that a teacher is linked to. We chose to display these as cards along with some quick information displayed. For now, we have two main colors : blue and grey, but more may be added.

For the buttons block, we added a button in order to add a new class. The file is `createcohort.html` which looks as :

5.3 Create cohort



Create classroom

CLASSROOM NAME: Dutch

LANGUAGE: nl

INVITATION CODE: 133421sad

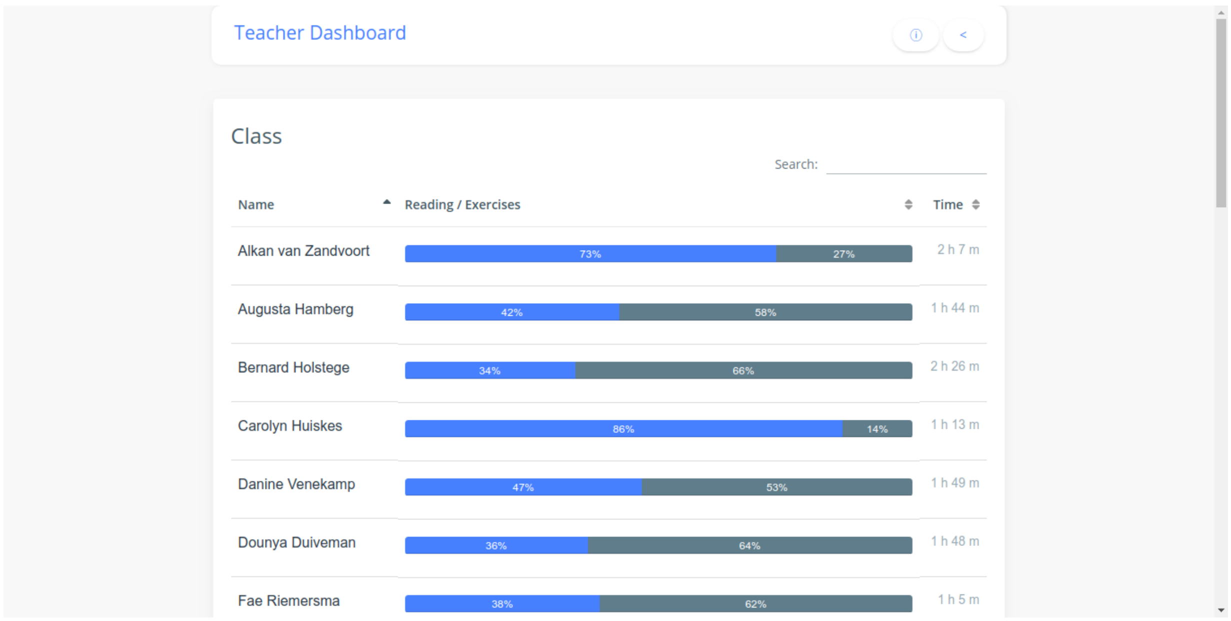
MAX STUDENTS: 19

FIGURE 6 – Homepage

This page extends from base and is composed of a form. The functionality of this page is self explanatory and all design choices here should be intuitive to understand.

If a user goes to a class, they will see the classpage :

5.4 Class Page



Name	Reading / Exercises		Time
Alkan van Zandvoort	73%	27%	2 h 7 m
Augusta Hamberg	42%	58%	1 h 44 m
Bernard Holstege	34%	66%	2 h 26 m
Carolyn Huiskes	86%	14%	1 h 13 m
Danine Venekamp	47%	53%	1 h 49 m
Dounya Duiveman	36%	64%	1 h 48 m
Fae Riemersma	38%	62%	1 h 5 m

FIGURE 7 – Homepage

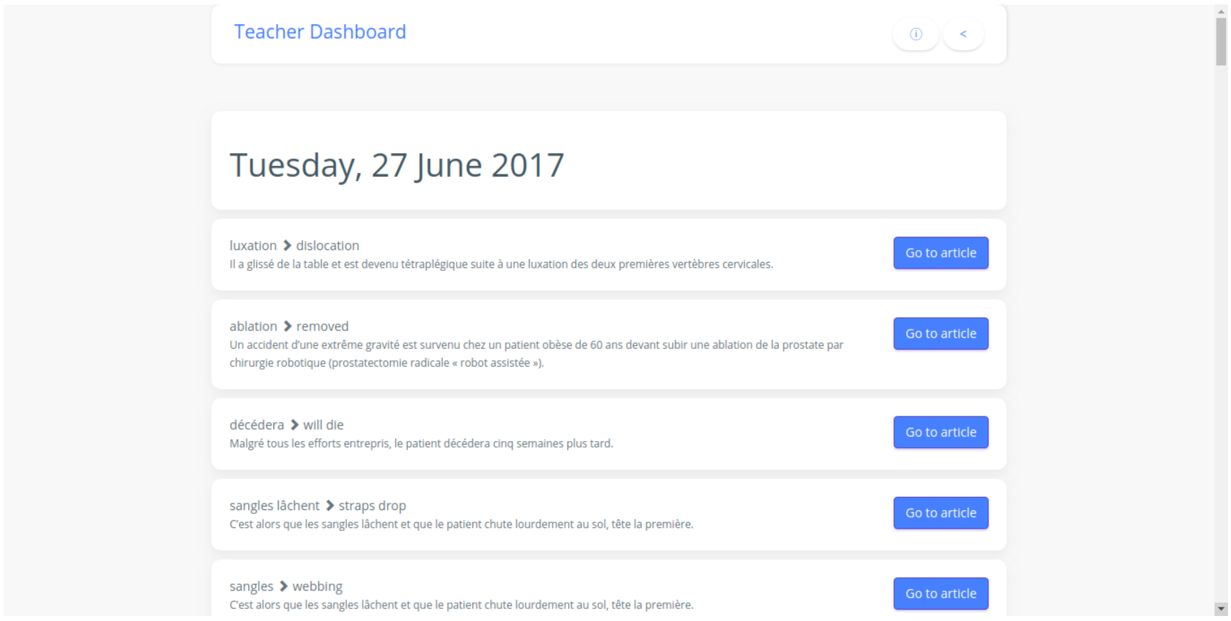
Again, the classpage extends from base. The page's content block will return a list of all the students enrolled in the respective class. We display a progress bar that displays a ratio between the amount of time the student spent reading versus the amount of time spent in exercises. We also display the total time on the side.

With the help of jQuery, we were able to add sorting options for each column along with a search menu.

For the buttons, we have an information button which opens up a modal with information about that class. The back button is also added here (to go back to the previous page).

Clicking the name of the student redirects the user to the student page.

5.5 Student page



Tuesday, 27 June 2017

luxation > dislocation
Il a glissé de la table et est devenu tétraplégique suite à une luxation des deux premières vertèbres cervicales.

ablation > removed
Un accident d'une extrême gravité est survenu chez un patient obèse de 60 ans devant subir une ablation de la prostate par chirurgie robotique (prostatectomie radicale « robot assistée »).

décédera > will die
Malgré tous les efforts entrepris, le patient décédera cinq semaines plus tard.

sangles lâchent > straps drop
C'est alors que les sangles lâchent et que le patient chute lourdement au sol, tête la première.

sangles > webbing
C'est alors que les sangles lâchent et que le patient chute lourdement au sol, tête la première.

FIGURE 8 – Homepage

This page's content block shows the translation of each word they struggled with along with the context of which the word was chosen from. It is very similar to Zeeguu's own statistics page for each student.

The buttons are the same as the previous page but of course the information button shows different information.

5.6 Error Pages

Error pages were added that extend from base that simply print out the error at hand.

6 Implementation

6.1 Front-end Implementation

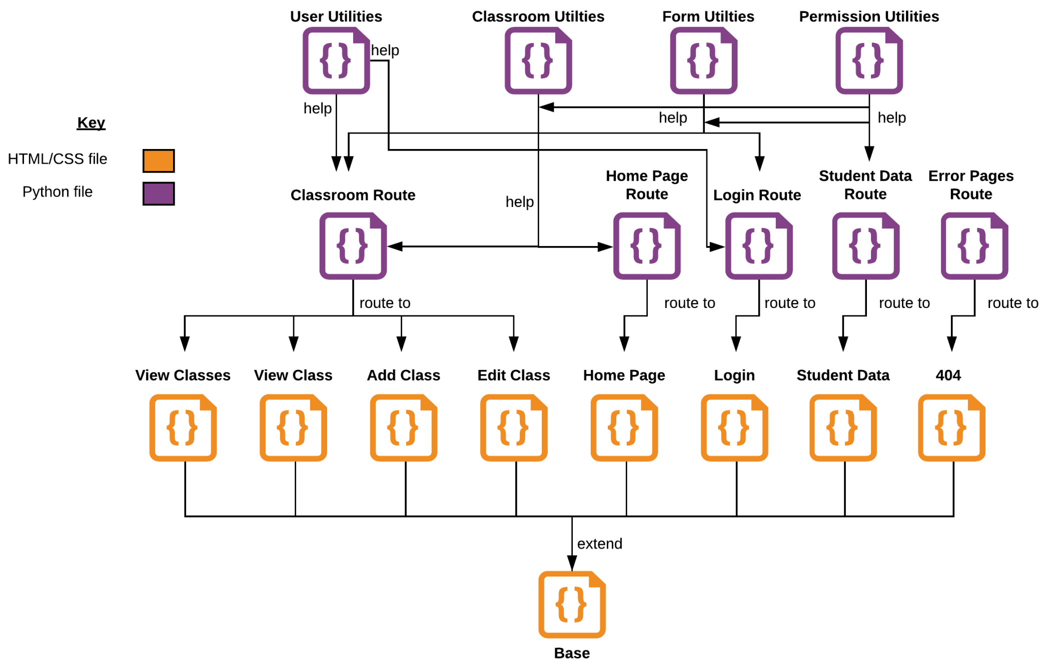


FIGURE 9 – Front-end files and their intercommunication

6.2 Back end Implementation

6.2.1 API Dashboard

Functionality that will be added to the new API file dashboard.py.

— Function to return info on all classes belonging to a teacher (GET)
— Function to return info on an individual class (GET)
— Function to return info on all students in a class (GET)
— Function to return info on an individual student (GET)
— Function to add a class (POST)
— Function to add user and set their class (POST) (For Testing)
— Function to link teacher to class (Private) (Called by class adding function)
— Permission checking functions (Private)
— Permission querying functions for website (GET)
— Remove class function (POST)

- Function to return user statistics (GET)
- Function that edits class (POST)

6.2.2 Testing

This is the testing that is included into test_dashboard.py. Each test is to be followed by its expected result (Whether it should return something and the status_code)

- **Test adding of classes**
  - Acceptable class (PASS 200)
  - Negative max_students (FAIL 400)
  - Taken inv_code (FAIL 400)
  - Invalid language_id (FAIL 400)
- **Test getting info from classes**
  - Class belonging to another teacher (PASS 200)
  - Class belonging to current teacher (FAIL 401)
  - Class that doesn't exist (FAIL 401)
- **Test session checker function (Permissions)**
  - Correct session (PASS 200)
  - Incorrect session (FAIL 401)
- **Test removing classes**
  - Removing class belonging to teacher (PASS 200)
  - Removing class not belonging to teacher (FAIL 401)
  - Removing class that does not exist (FAIL 401)
- **Test getting of student info**
  - Student belonging to class that belongs to current teacher (PASS 200)
  - Student belonging to class that belongs to another teacher (FAIL 401)
  - Student that doesn't exist (FAIL 401)
  - Student that belongs to no class (FAIL 401)
- **Test updating class**
  - Class that belongs to teacher, with valid input (PASS 200)
  - Class that does not belong to teacher, with valid input (FAIL 401)
  - Update max_students to be negative or 0 (FAIL 400)
  - Update that tries to update inv_code to a taken code (FAIL 400)

7 Technology stack

1. HTML - Used for designing the page

   CSS - Used for styling the web-page.

   JQuery - Used for easier web-page design.

2. Python - Used for the back-end of the project.

    Flask - Used in Python for developing websites.

    Jinja2 - Used for dynamically constructing web-pages in HTML.

    WTForms - Used for easy form creation and usage in HTML.

    Bootstrap - Provides a framework for CSS and JS webpage design.

    SQLalchemy - Integrates server queries into python.

3. MySQL - The framework on which the project server is built on.

8 Team organisation

The group project team is made up of seven people. We have put ourselves into three teams the front-end team, the middle-end team and the back-end (API) team. Each team is listed below, including each team member.

Front-end team :

— Henry Salas
— Evi Xhelo
— Ai Deng

Back-end team :

— Jakob Vokac
— Christian Grier Mulvenna
— Tai-Ting Chen
— Oliver Holder

9 Change Log

Created requirements document on the 27th at 5pm.

When	**Where**	**What**
Mar 17th, 20 :00	The document	Draft made for the structure ; Intro written
Mar 22nd, 11 :00	The Zeeguu architecture	Images included ; Most of the text written
Mar 23rd, 11 :00	The Zeeguu architecture	Picture fixing
Mar 24th, 15 :30	Functionality we want ; Implementation	Some text and images made
Mar 24th, 15 :30	Technology stack	Text made
Mar 25th, 15 :00	Everywhere	Last touches before the end of the second sprint
Mar 31st, 13 :00	Development	Updated development details and added time scale
Mar 31st, 15 :00	Implementation	Completely changes implementation section
Mar 31st, 15 :00	Introduction	Changed introduction a bit
Mar 31st, 16 :00	Design	Added in design section
April 30st, 16 :00	Everywhere	Overhaul