



---

# Zeeguu Dashboard Architecture

---

**Version 4.0**

*Authors :*

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

*Clients :*

Mircea Lungu  
Carlos Paz Rodriguez

*Supervisor :*

Lars Holdijk

University of Groningen  
March 2018

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Zeeguu/Dashboard Architecture Overview</b>	<b>2</b>
<b>3</b>	<b>Zeeguu Dashboard Architecture Overview</b>	<b>4</b>
3.1	Overview of system . . . . .	4
3.2	Project structure . . . . .	5
<b>4</b>	<b>Functional Architecture</b>	<b>8</b>
4.1	The Concept of a Zeeguu Classroom . . . . .	8
4.2	Sending & Receiving Data To/From The Zeeguu API . . . . .	8
4.3	REST API . . . . .	8
4.4	Security . . . . .	9
4.4.1	User authentication . . . . .	9
4.4.2	Database attacks . . . . .	9
4.5	Updates to the Zeeguu Core & API . . . . .	10
4.5.1	Updates . . . . .	10
4.5.2	Conversion from old database to new database . . . . .	10
<b>5</b>	<b>GUI Architecture</b>	<b>10</b>
5.1	Base . . . . .	10
5.2	Homepage . . . . .	11
5.3	Create cohort . . . . .	12
5.4	Class Page . . . . .	13
5.5	Student page . . . . .	14
5.6	Error Pages . . . . .	14
<b>6</b>	<b>Implementation</b>	<b>16</b>
6.1	Front-end Implementation . . . . .	16
6.2	Back end Implementation . . . . .	17
6.2.1	API Dashboard . . . . .	17
6.2.2	Testing . . . . .	19
<b>7</b>	<b>Technology stack</b>	<b>19</b>
<b>8</b>	<b>Team Organisation</b>	<b>21</b>
8.1	The Team . . . . .	21
8.2	Version Control and Communication . . . . .	21
8.3	Scrum . . . . .	21
<b>9</b>	<b>Change Log</b>	<b>22</b>

## 1 Introduction

Zeeguu is an innovative new tool for learning languages, allowing its users to read articles in a foreign language while having real time translations at their finger tips. Zeeguu allows its users to perform interactive exercises with words for which they needed translating. Zeeguu offers the support of translating five different languages (English, Dutch, German, Spanish and French) to three base languages (English, Dutch and Chinese), allowing you to learn these languages in the most effective way.

The current Zeeguu system is not yet set up for teachers, individuals can sign up to the website and they can be added to classes manually by system admins. this means that 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 Zeeguu dashboard is to extend the Zeeguu website's functionality in order to help teachers use Zeeguu in their classrooms. This should include the creation, mangagement and analysis of classes and students.

## 2 Zeeguu/Dashboard Architecture Overview

Our work is an extension of functionality to Zeeguu. The source for Zeeguu is open and on Github. It spans three repositories called Zeeguu-Web, Zeeguu-Core annd the Zeeguu-API. Generally speaking, Zeeguu-Web is for running a the Zeeguu flask 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. Additionally, there is now our repository called 2018-ZeeguuDashboard. This is another Flask web framework which delivers all the dashboard related web pages to a teacher. Take a look at figure 5 to see the direction of communication between each of these systems. We see bidirectional arrows between the Zeeguu-API and the other systems since each system communicates with the API to access another system. This is because the Zeeguu-Web, dashboard and Zeeguu-Core shouldn't communicate between each other directly. One exception to this is that Zeeguu-Dashboard redirects you to a web page on the Zeeguu-Web when a teacher is requesting an article read by a student through the dashboard.

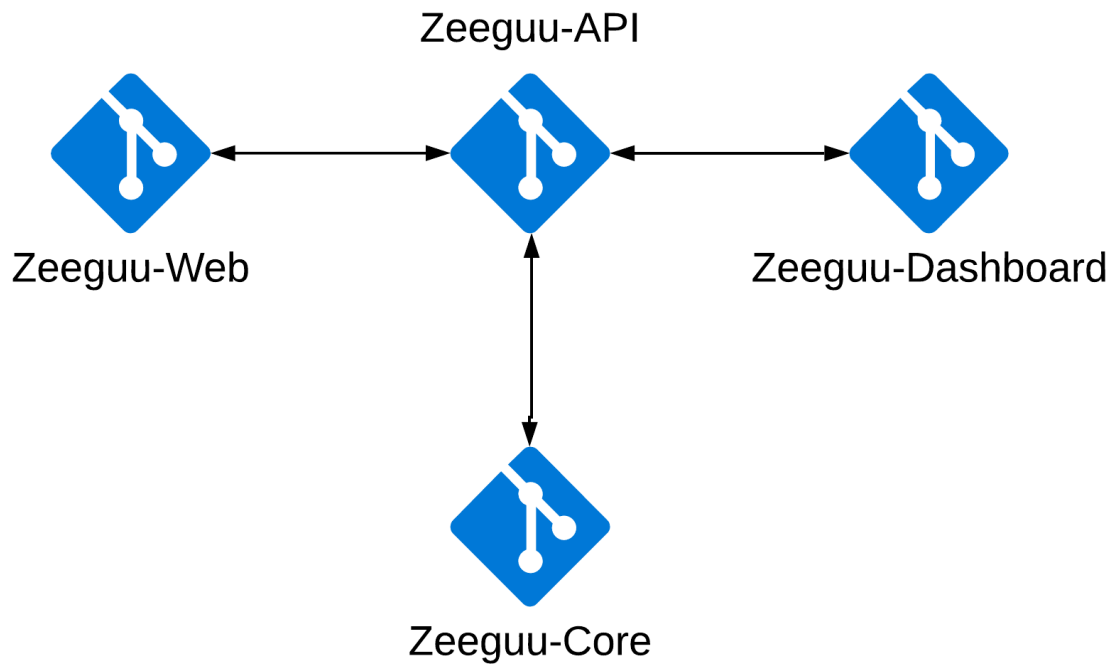


Figure 1: Communication between the Zeeguu repositories

## 3 Zeeguu Dashboard Architecture Overview

### 3.1 Overview of system

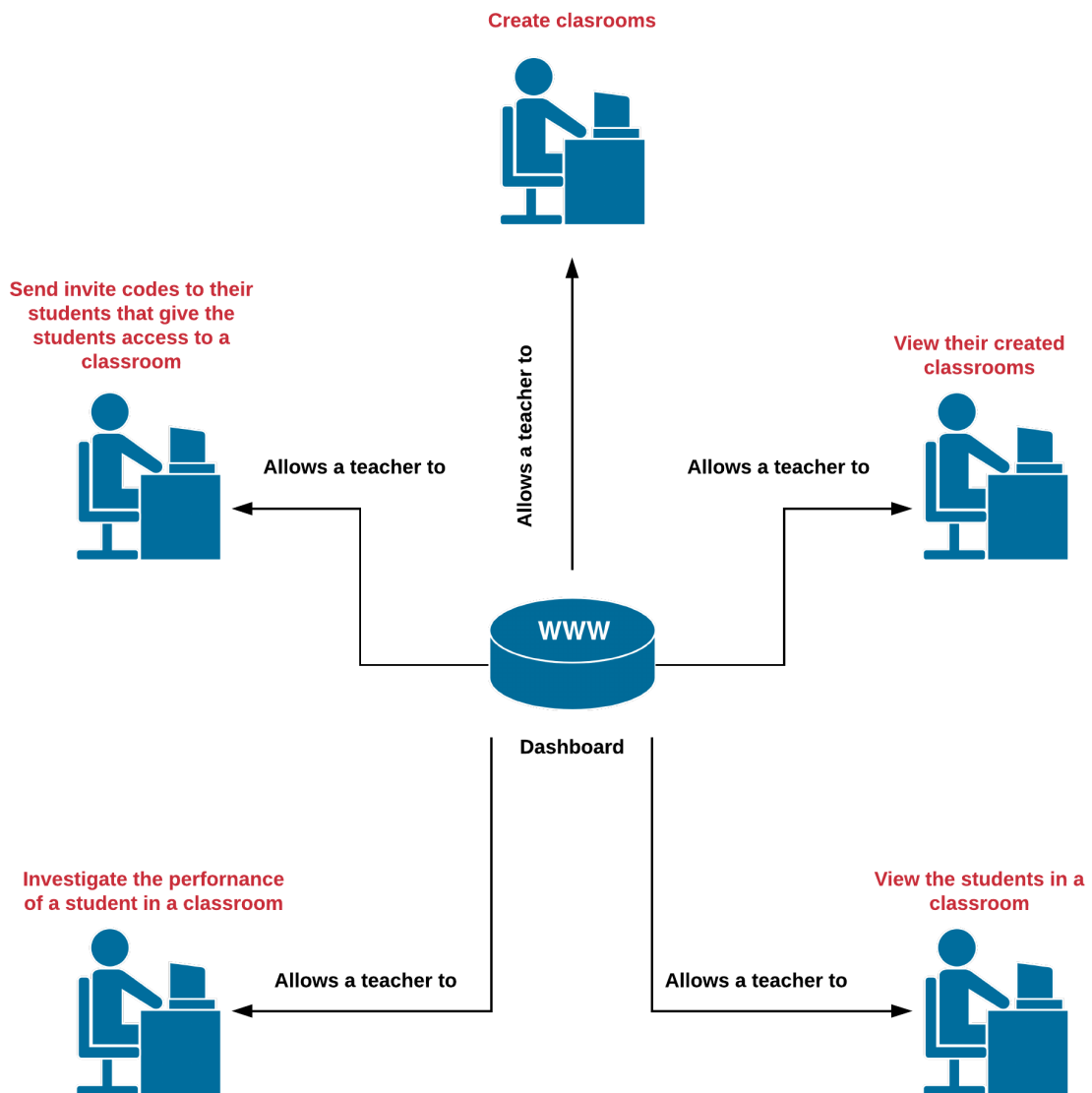


Figure 2: Abstract capabilities of the Zeeguu Dashboard

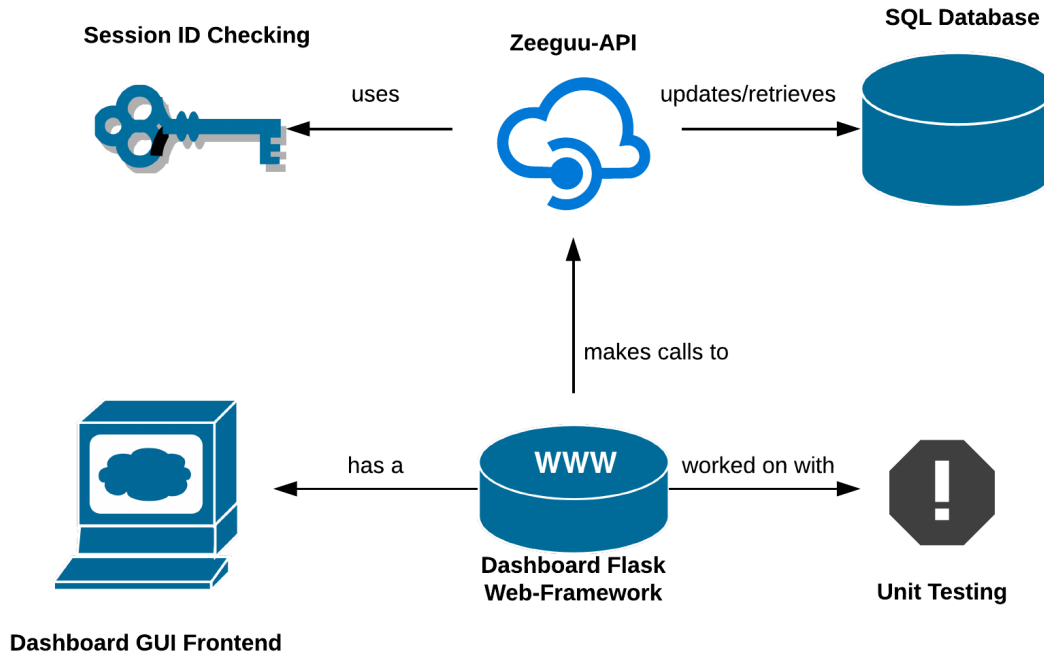


Figure 3: Conceptual elements of the Zeeguu Dashboard

## 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 module with the basic functions for making REST 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

The util folder contains all of the functions that change or format data before it can be displayed on the web page. They also contain flask code for WTForms as well as functions which check user permissions for visiting pages.

### 3.2 Project structure 3 ZEEGUU DASHBOARD ARCHITECTURE OVERVIEW

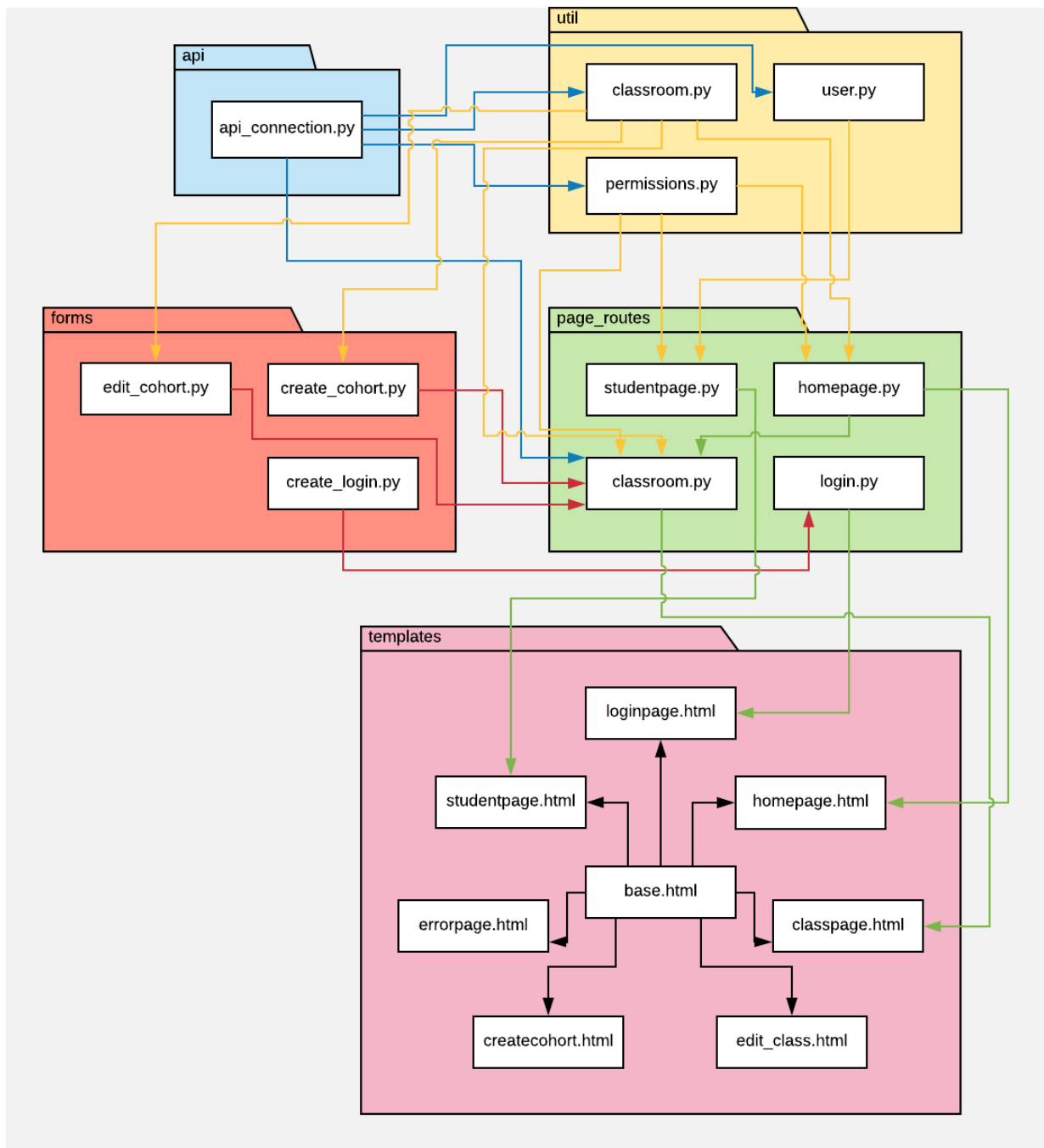


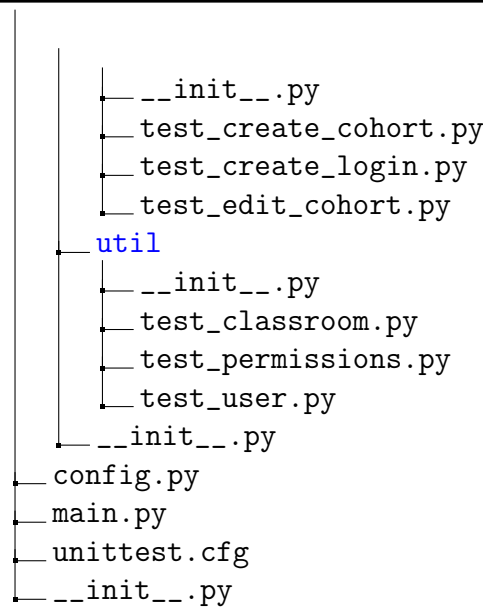
Figure 4: The project modules and their importing relationships

### 3.2 Project structure 3 ZEEGUU DASHBOARD ARCHITECTURE OVERVIEW

---

```
/
├── app
│   ├── api
│   │   ├── __init__.py
│   │   └── api_connection.py
│   ├── forms
│   │   ├── __init__.py
│   │   ├── create_cohort.py
│   │   ├── create_login.py
│   │   └── edit_cohort.py
│   ├── page_routes
│   │   ├── __init__.py
│   │   ├── classroom.py
│   │   ├── errorpages.py
│   │   ├── homepage.py
│   │   ├── login.py
│   │   └── studentpage.py
│   ├── static
│   ├── templates
│   │   ├── base
│   │   │   ├── base.html
│   │   │   ├── classpage_base.html
│   │   │   └── student_pag_base.html
│   │   ├── classpage.html
│   │   ├── createcohort.html
│   │   ├── edit_class.html
│   │   ├── empty_classpage.html
│   │   ├── empty_student_page.html
│   │   ├── errorpage.html
│   │   ├── homepage.html
│   │   ├── loginpage.html
│   │   └── studentpage.html
│   ├── util
│   │   ├── __init__.py
│   │   ├── classroom.py
│   │   ├── permissions.py
│   │   └── user.py
│   └── __init__.py
├── database
│   ├── reset_db.py
│   └── update_db.py
├── tests
│   └── forms
```





## 4 Functional Architecture

### 4.1 The Concept of a Zeeguu Classroom

Our main addition to the Zeeguu Ecosystem 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 be used so that the users get recommendations for the language they are being taught. The number of students makes sure that the classroom does not get clustered with fake accounts and the invitation code makes sure that the students and only the students can easily join their classroom, without having to search for it on Zeeguu.

### 4.2 Sending & Receiving Data To/From The Zeeguu API

We send and receive data using the requests package, more information here:

<http://docs.python-requests.org/en/master/>.

We use the JSON format for all of the data that is sent across.

All new API calls were added to the `teacher_dashboard.py` file. 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'

in the 'api\_connection.py' file. 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 is carried out using sessions. A session is an object stored in the database holding an ID and a User. This ID is therefore mapped to one and only one User. To start using the website (sending requests to the API) the user must first log in and receive a valid session ID (this session is created on the server when the log in credentials are correct). Once a user has a valid session ID it is sent along with each get/post request the website makes.

Any endpoints in the rest-api that return data require a valid session. Any 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. The dashboard 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.

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

With updating them database, the columns that needed to be added will be checking by information\_schema with mysql query. The latest database has `max_students` and `language_id` as the new columns and `invitation_code` changes to `inv_code`. Then adding an unique key to `inv_code` to check with unrepeatable code when creating or changing class invitation code into database.

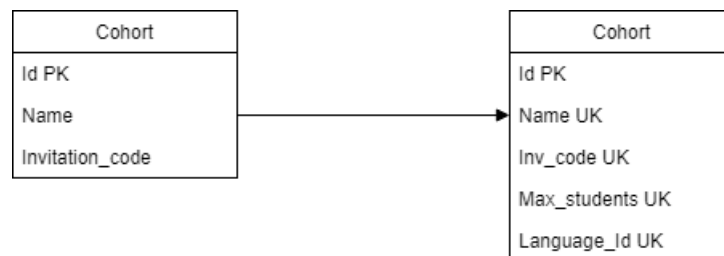


Figure 5: Cohort database conversion

## 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

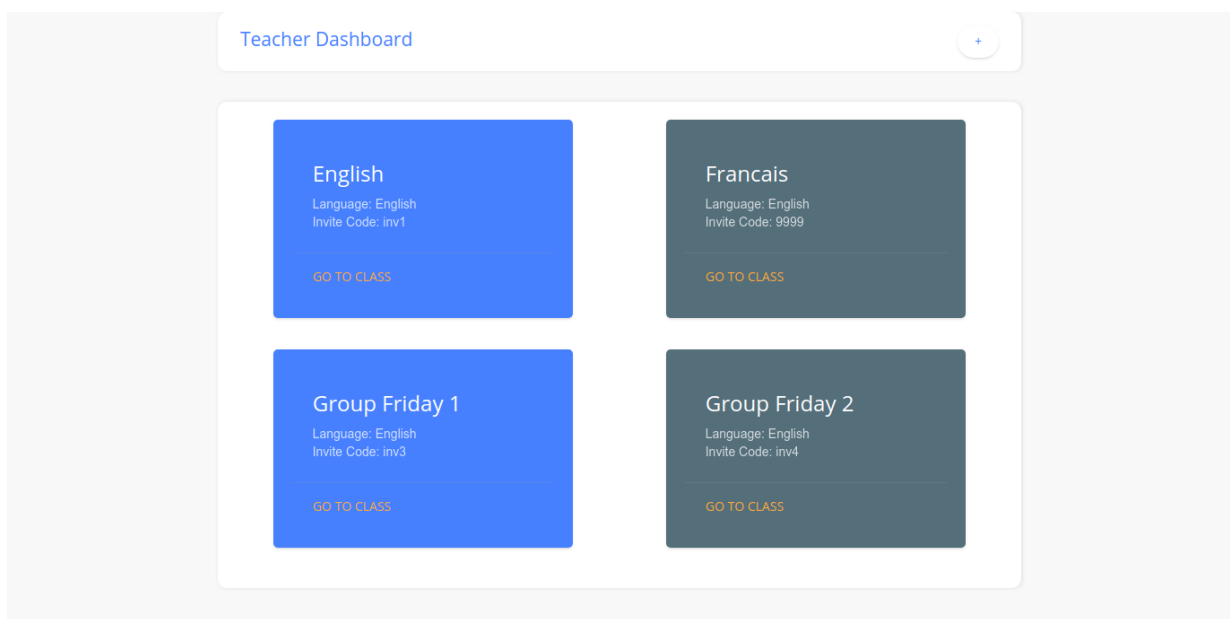


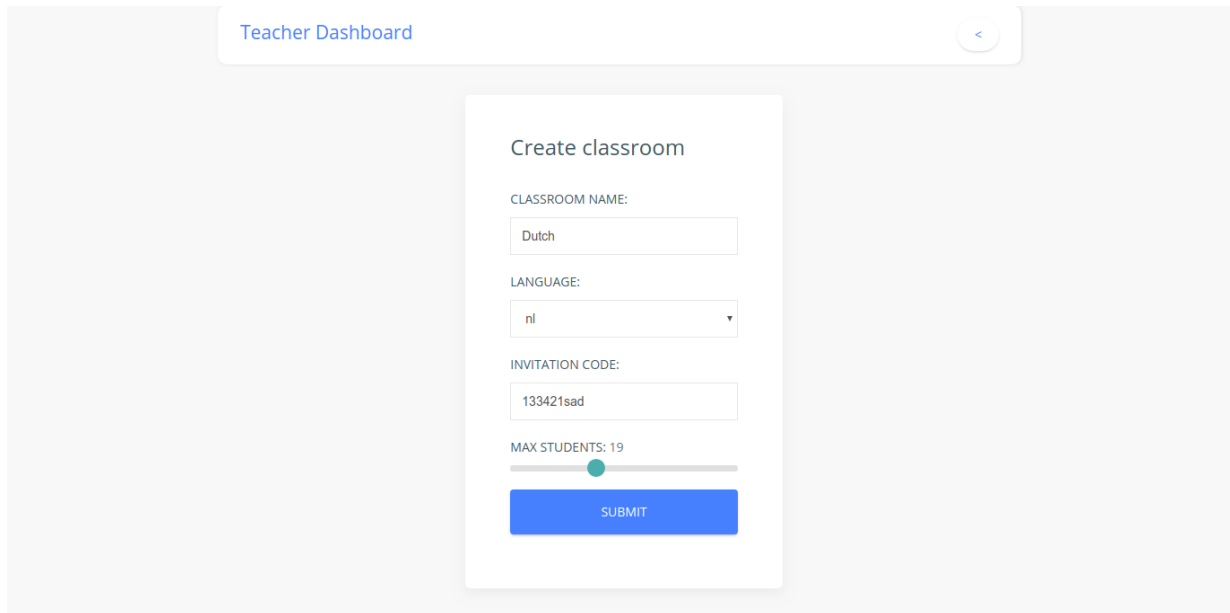
Figure 6: Homepage

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



The screenshot shows a 'Teacher Dashboard' with a 'Create classroom' form. The form includes the following fields and controls:

- CLASSROOM NAME:** A text input field containing the word 'Dutch'.
- LANGUAGE:** A dropdown menu with 'nl' selected.
- INVITATION CODE:** A text input field containing the code '133421sad'.
- MAX STUDENTS:** A slider control set to 19, with a green dot indicating the current value.
- SUBMIT:** A blue button labeled 'SUBMIT'.

Figure 7: 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

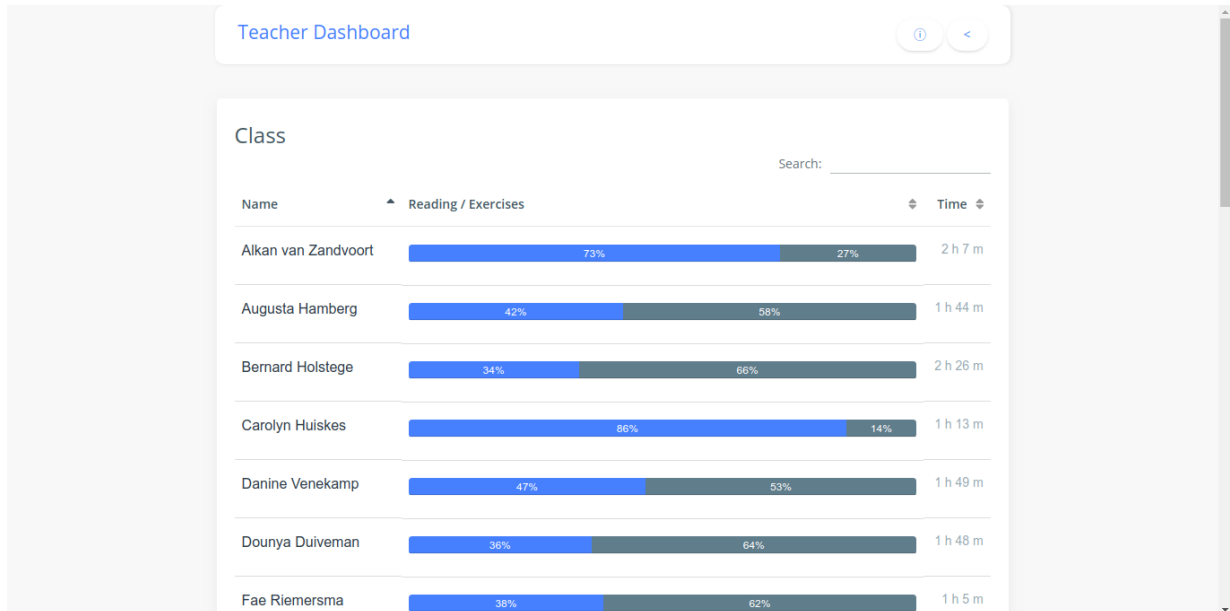


Figure 8: Homepage

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

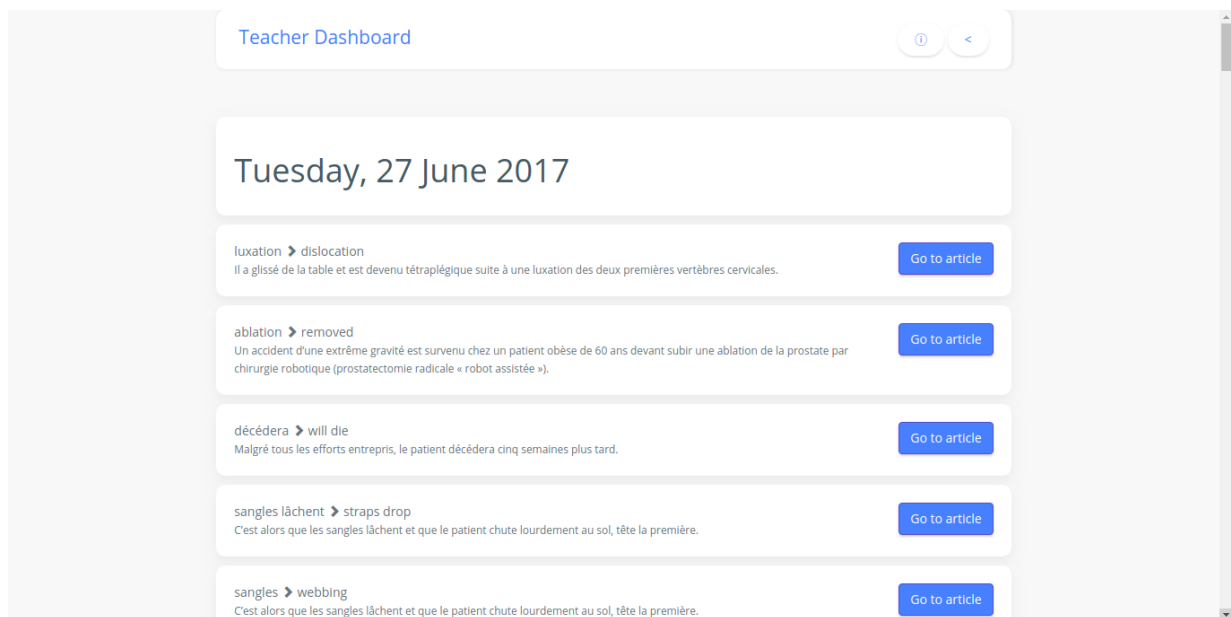


Figure 9: Homepage

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

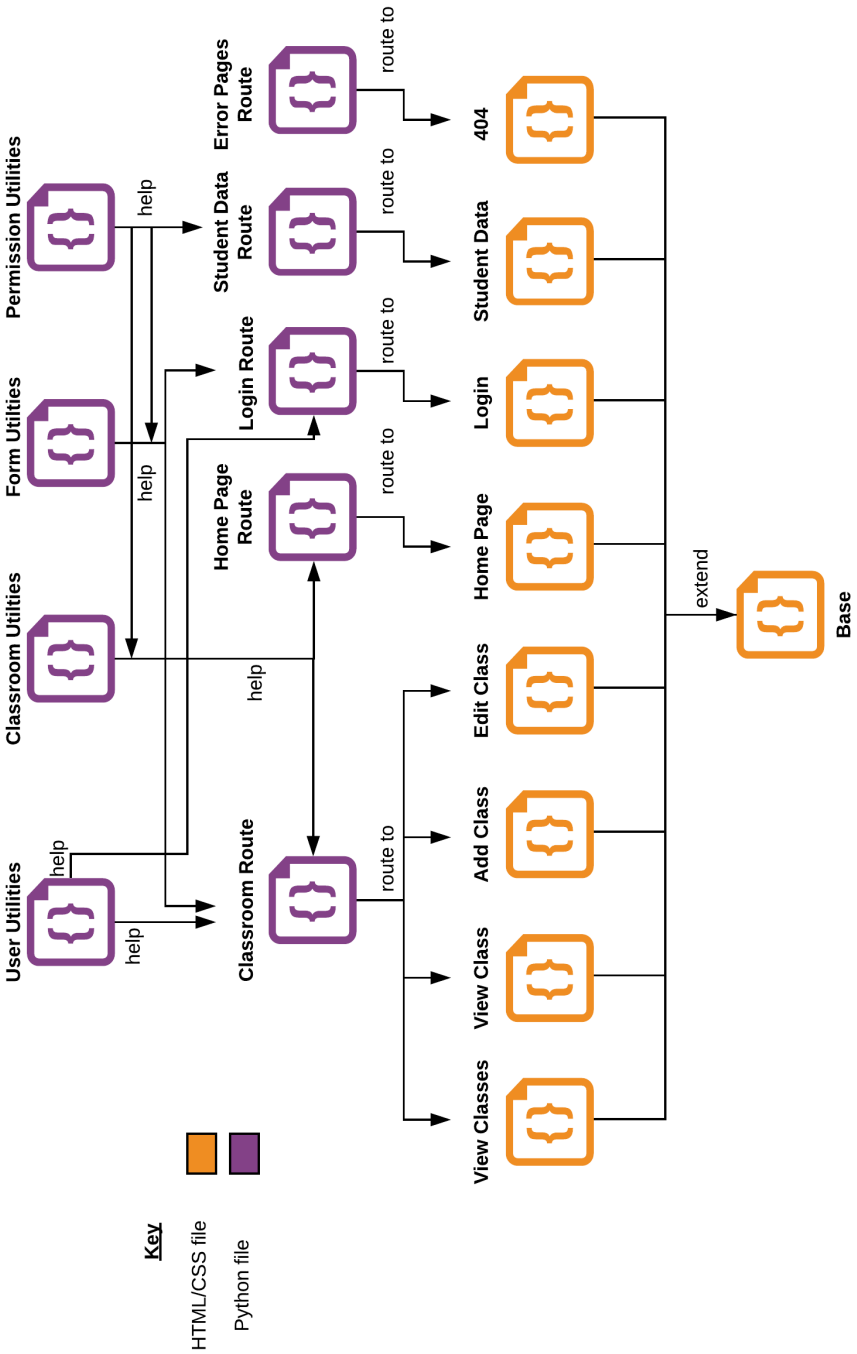


Figure 10: Front-end files and their intercommunication

## 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
  - Method:GET
  - Endpoint: `"/cohorts_info"`
- Function to return info on an individual class
  - Method:GET
  - Endpoint: `/cohort_info/<id>`
  - Variables: id: CohortId
- Function to return info on all students in a class
  - Method:GET
  - Endpoint: `"/users_from_cohort/<id>/<duration>"`
  - Variables:
    - \* id:CohortId
    - \* duration:the amount of days of which we want to see activity
- Function to return info on an individual student
  - Method:GET
  - Endpoint:`"/user_info/<id>/<duration>"`
  - Variables
    - \* id: UserId
    - \* duration: the amount of days of which we want to see activity
  - :
- Function to add a class
  - Method:POST
  - Endpoint: `"/create_own_cohort"`

- Function to add user and set their class (For Testing)
  - Method:POST
  - Endpoint: `"/add_user_with_cohort"`
- Function to link teacher to class (Private) (Called by class adding function)
- Permission checking functions (Private)
  - Method:GET
  - Endpoint: `"/has_permission_for_cohort/<id>"`
  - Variable: id: `userId`
- Permission querying functions for website
  - Method:GET
  - Endpoint:
    - \* `"Has_permission_for_cohort"`
    - \* `"Has_permission_for_user"`
    - \* `"with_session"`
- Remove class function
  - Method:POST
  - Endpoints: `"/remove_cohort/<cohort_id>"`
  - Variable: `cohort_id`: the `cohortId`
- Function to return user statistics
  - Method:GET
  - Endpoint: `"/cohort_member_bookmarks/<id>/<time_period>"`
  - Variables:
    - \* id: `userId`
    - \* time\_period: the amount of days of which we want to see activity
- Function that edits class
  - Method:POST
  - Endpoint: `"/update_cohort/<cohort_id>"`

### 6.2.2 Testing

We performed unit testing on the general functionality which Flask utilises. The unit testing is implemented in our python code via the unittest package and within that lies a very important module called Mock. You can find this test code separated for organisation within the tests sub directory. They are separated into directories that are labelled after a module sub directory of app. Therefore the tests for code in `app/util` lie in `tests/util`.

The tests are run via the pip package, nose2. This is in effect the same as calling unittest. This call is made from Travis CI. This is because we have a configuration file, `.travis.yml`, and Travis enabled on our github repo. Thus every time we push to our repo, Travis executes our tests.

Now let's consider how we really test code within a unit test. We use a combination of both white-box and black-box testing. With the Mock module, we can swap out a function or method call in some code file with a stub version. This for example allows us to stop all API calls from ever reaching the API server. We can additionally fake it's return.

We additionally performed unit testing on the dashboard components in the Zeeguu\_API repository. Specifically this is in `Zeeguu-API/tests_zeeguu_api/test_teacher_dashboard.py`. There we also used the unittest package but we didn't use the Mock module. We instead create a new temporary database for starting every test that is deleted after the test has completed. This is done via the `setUp` and `tearDown` methods that the unittest package offers. They are then implemented and defined in `Zeeguu-API/tests_zeeguu_api/api_test_mixin.py`.

By having unit tests, we can have a higher confidence in knowing that the functions we write do what they are expected to perform. If a function is accidentally edited then we have a higher chance of catching this. When adding new code, the unit tests provide a safety net for ensuring that we don't break code somewhere else.

## 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

### 8.1 The Team

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
- Chris Ausema

### 8.2 Version Control and Communication

For version control we use the site Github (<http://github.com>) this because it is the most used, popular and easiest way to use Git, it allows us to work in teams effectively and without much overlap and without the change of accidentally overwriting crucial code.

For our inter-team communication we use the tool Slack(<https://slack.com/>), this is a great tool because of the fact that you can make different text channels in it and then choose different team members that can see the channel. This reduces the amount of unneeded notifications.

### 8.3 Scrum

To optimize our workflow we use the project boards of Github as a Scrum board. At the start of every two week sprint we meet to pick tasks from our backlog based on importance and the amount of time we think it will take to finish it. from this we make a sprint planning. We then divide this work up under the different team members. We don't do the

daily scrums, this because it is not possible for the complete team to meet up everyday to talk and work on the project. Instead we inform each other of our work and troubles through Slack.

## 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