Zeeguu Dashboard Architecture

Version 1.0

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 for facilitating 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 The Zeeguu architecture

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 1 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.

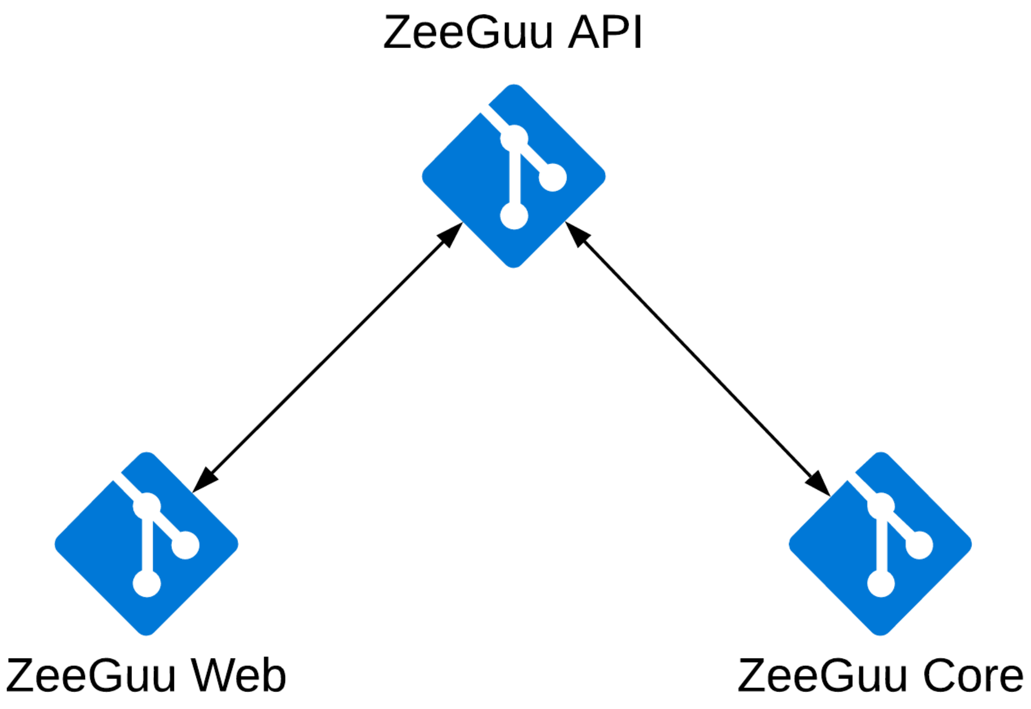


FIGURE 1 – Communication between the Zeeguu repositories

3 Implementation

3.1 Minimum viable product

First, the minimum viable product must be defined. This will be done by listing features the MVP should have.

— Teachers have the ability to create classes.
— Teachers have the ability to view created classes.
— Teachers have the ability to set an invite code to their classes.
— Teachers have the ability to view all the students in view class.
— Teachers have the ability to view each student in their class.

3.2 Front end Implementation

— Template to view classes

- Template to view a class
- Template to view student data
- Template to add class
- Utility function to load all classes belonging to teacher
- Utility function to load individual class info
- Utility function to load all students belonging to class
- Utility function to load individual student info
- Abstracted API GET request function
- Abstracted API POST request function
- Routing between templates with use of data loading functions to give templates data.
- Permission checking functions that wrap permission blocked routes.
- Template to edit class
- Utility function to edit class

3.3 Back end Implementation

3.3.1 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)

3.3.2 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)
  - Remoing 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)

3.4 Implementation choices

3.4.1 REST API efficiency

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'. These functions include the session with each request. This is done to reduce code duplication and unnecessary errors.

All data gathering functions are split into a utility package that handles the dejsonifying of objects and calling the API request functions. This is split from the routes.py to make each functions actions clear and to increase forward integration with the future API.

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.

3.4.2 Permissions/Sessions

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 Time line

1. API : Dashboard functionality implemented.
2. Add class form created on website.
3. Add class interacting with API, able to create cohort in the db now.
4. View classes added to website.
5. View Class added to website.
6. View classes and class linked to api and now functional.
7. API functions re-optimised for scale ability.
8. data-loading functions split and changed with regards to API changes.
9. GET and POST requests split into handler functions.
10. Log in implemented to website to activate sessions.
11. Sessions sent with GET and POST requests.
12. Session checking implemented on the API side.
13. Permission checking functions implemented on API.
14. Permission checking implemented on website using API functions.
15. View user data created on website and linked to API.
16. Removing of classes implemented on API.

17. Removing of classes implemented on Website.

18. Edit update class on both API and website.

5 Graphical User Interface

For the design aspect, our customer required a minimalist design that required as few clicks as possible to navigate. The design should be material-like. The homepage should simply list 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 :

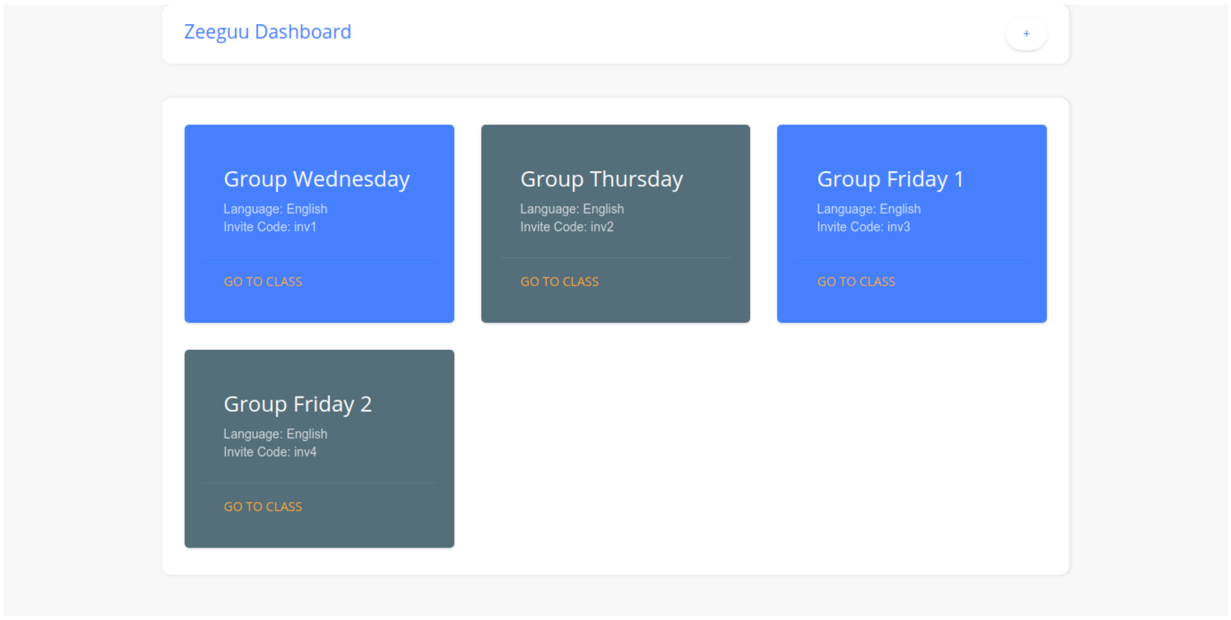
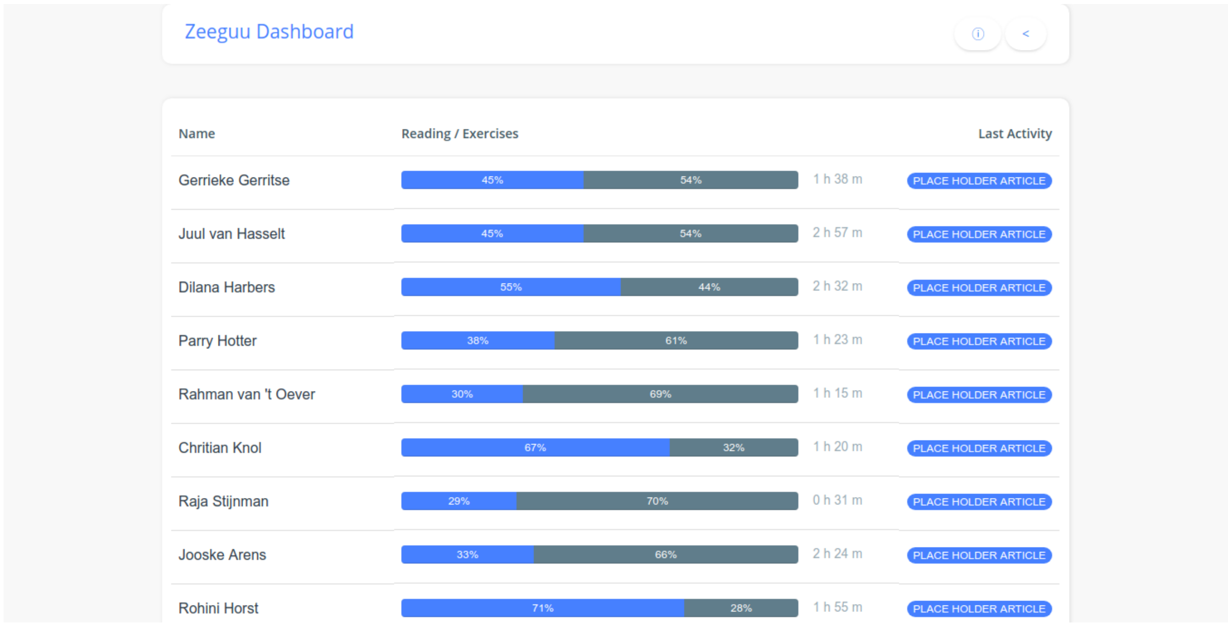


FIGURE 2 – Homepage

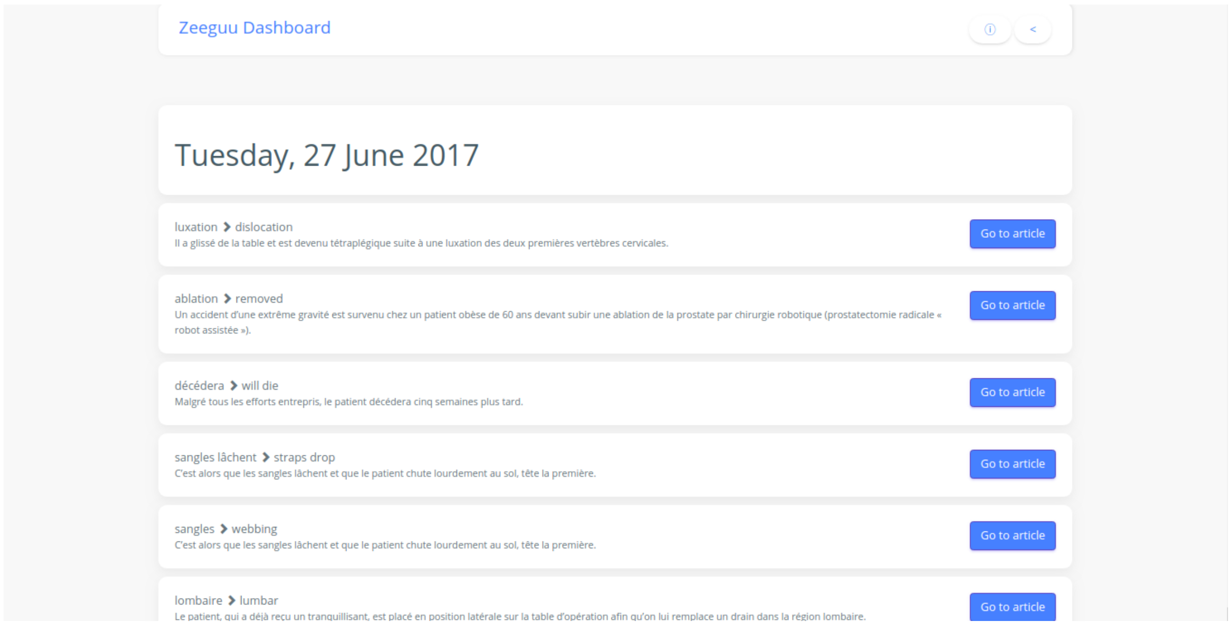
As for the class page, we simply return a list of all the students enrolled in the respective class. We display a progress bar that displays a ratio between the amount of time the student spent reading versus the amount of time spent in exercises. We also display the total time on the side. There is also a small alert of the last activity a student has done.


Zeeguu Dashboard

Name	Reading / Exercises			Last Activity
Gerrieke Gerritse	45%	54%	1 h 38 m	PLACE HOLDER ARTICLE
Juul van Hasselt	45%	54%	2 h 57 m	PLACE HOLDER ARTICLE
Dilana Harbers	55%	44%	2 h 32 m	PLACE HOLDER ARTICLE
Parry Hotter	38%	61%	1 h 23 m	PLACE HOLDER ARTICLE
Rahman van 't Oever	30%	69%	1 h 15 m	PLACE HOLDER ARTICLE
Chritian Knol	67%	32%	1 h 20 m	PLACE HOLDER ARTICLE
Raja Stijnman	29%	70%	0 h 31 m	PLACE HOLDER ARTICLE
Jooske Arens	33%	66%	2 h 24 m	PLACE HOLDER ARTICLE
Rohini Horst	71%	28%	1 h 55 m	PLACE HOLDER ARTICLE

Figure 3 – Homepage

Clicking the name of the student redirects the user to a page with further information :


Zeeguu Dashboard

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.
Go to article

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 »).
Go to article

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

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

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

lombaire > lumbar
Le patient, qui a déjà reçu un tranquillisant, est placé en position latérale sur la table d'opération afin qu'on lui remplace un drain dans la région lombaire.
Go to article

Figure 4 – Homepage

This page 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.

6 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 templates for better looking webpages.
     SQLalchemy - Integrates server queries into python.
3. MySQL - The framework on which the project server is built on.

7 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

Middle-back-end team :

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

Back-end team :

— Oliver Holder

8 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