UNIVERSITY OF GRONINGEN

Extensible Exercises Platform

Martin Avagyan (S2628074)
Vlad Turbureau (S2755858)

February 13, 2017

Contents

1 Abstract

The following document describes the architecture, design decisions and the structure of the exercises platform designed for Zeeguu API [1].

In the section *Architecture* we will describe all the design patterns and construction principles used in the project. Later we will expand using UML schema's to presents in detail the internal working of the source code.

In the section *Design goals*, we will discuss the design goals we had in mind at the beginning of the project and reflect the final results with it.

The section *Logic View* will describe the structure of the code in terms of logical modules and interactions between them.

In the section *Usage* we will talk about how to use the code: how to install all the dependencies and how to deploy it. After which we will expand on how to extend the platform.

In the *Appendix* you can find the initial planning and the vision of the project.

2 Architecture/Design

2.1 Design Goals

While developing the platform we had the following design values in mind:

1. The design should minimize complexity.
2. The design should be intuitive, and result in easily extensible code.
3. The design should prioritize efficiency.

In order to meet all those requirements we have decided to develop the project in one-pager application manner. It allows us flexibility in controlling the user interface and allows inheritance done within JavaScript We will elaborate more about this in the next sections.

2.2 Structure

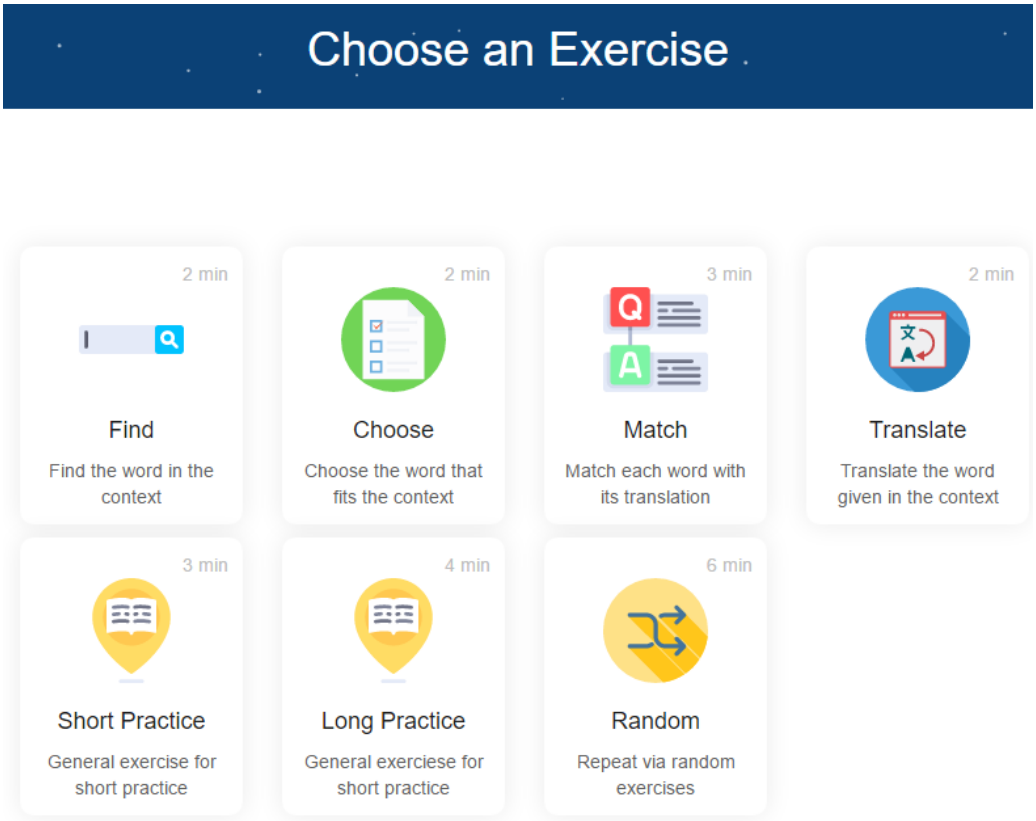
The application starts from the server loading the $index.html$ file in the browser (loading all relevant JavaScript and styles files). From this point on, all the following $html$ files are loaded in this $index.html$ by using Ajax requests. This principle followed by our application is also known as a one-page application. As mentioned before it allows flexibility in controlling the UI and exploiting the functionality of JavaScript. This also helps to balance the load to the server since JS runs on the client side in the browser. This approach comes with a set of its own challenges, namely memory leaks. In order to avoid memory leaks we have taken special care of modularizing the application, such that each reference to an object that no longer exists is removed therefore allowing the garbage collector to delete the objects.

2.2.1 Home Screen

The first page loaded into the $index$ is the home page we created ($home.html$). It allows the user to choose the a specific type of exercise (or a mixed, general exercise) to practice. The image below is a screen-shot of

[1] https://zeeguu.unibe.ch/

the home screen populated by exercise cards.



The home page is created dynamically. We use *Mustache.js* to populate the exercise cards. The framework allows flexibility in templating. Consider an example of exercise card below.

Here we can observe that the card template consists of four parts. The header, containing the time, the content having the icon, and the footer containing the name and short description about the card. Each of those components are dynamically created for each exercise through Mustache.js.

2.2.2 EXERCISES

When an exercise type is selected, we load *exercise.html*, which represents the template of an exercise. It is designed as a wrapper , containing all the common elements (in this case progress bar, situated in the header). The content of an exercise is later loaded in the *div* named "*custom* − *content*" inside the *exercise.html*. The custom content is represented by: a custom header (with a description) which is added below the main header, the content (context) and the footer. As a demonstration consider the mock-up below. It represents the distribution of the elements of one page:

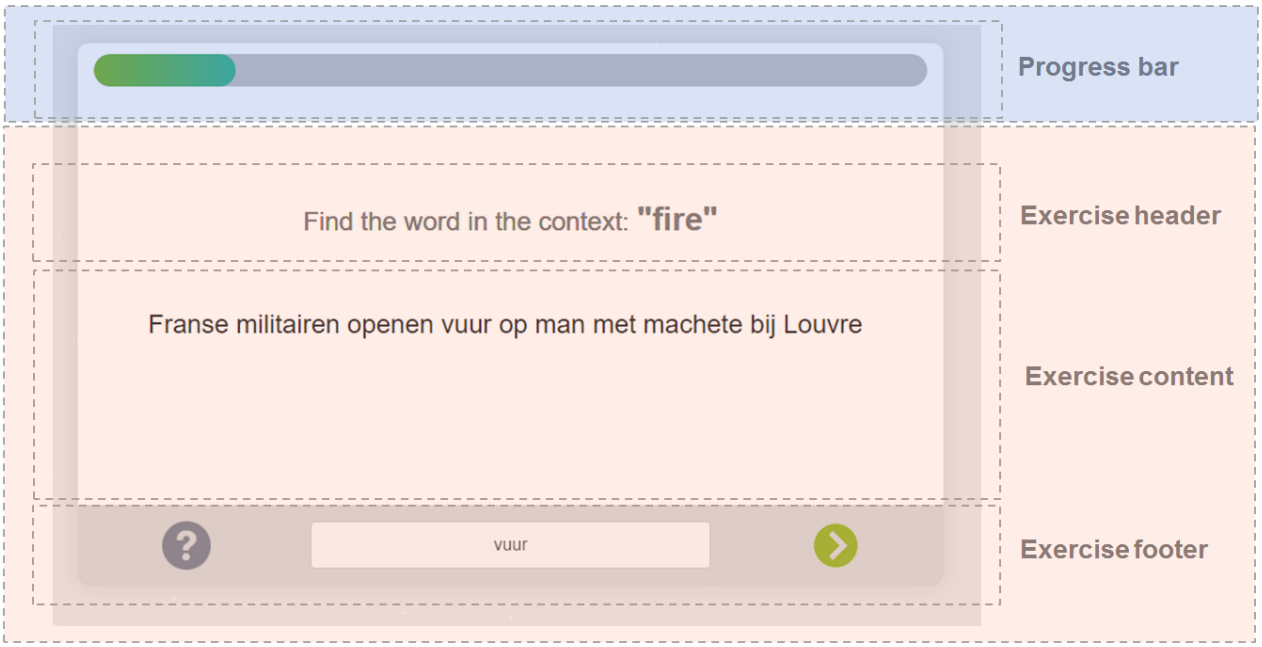
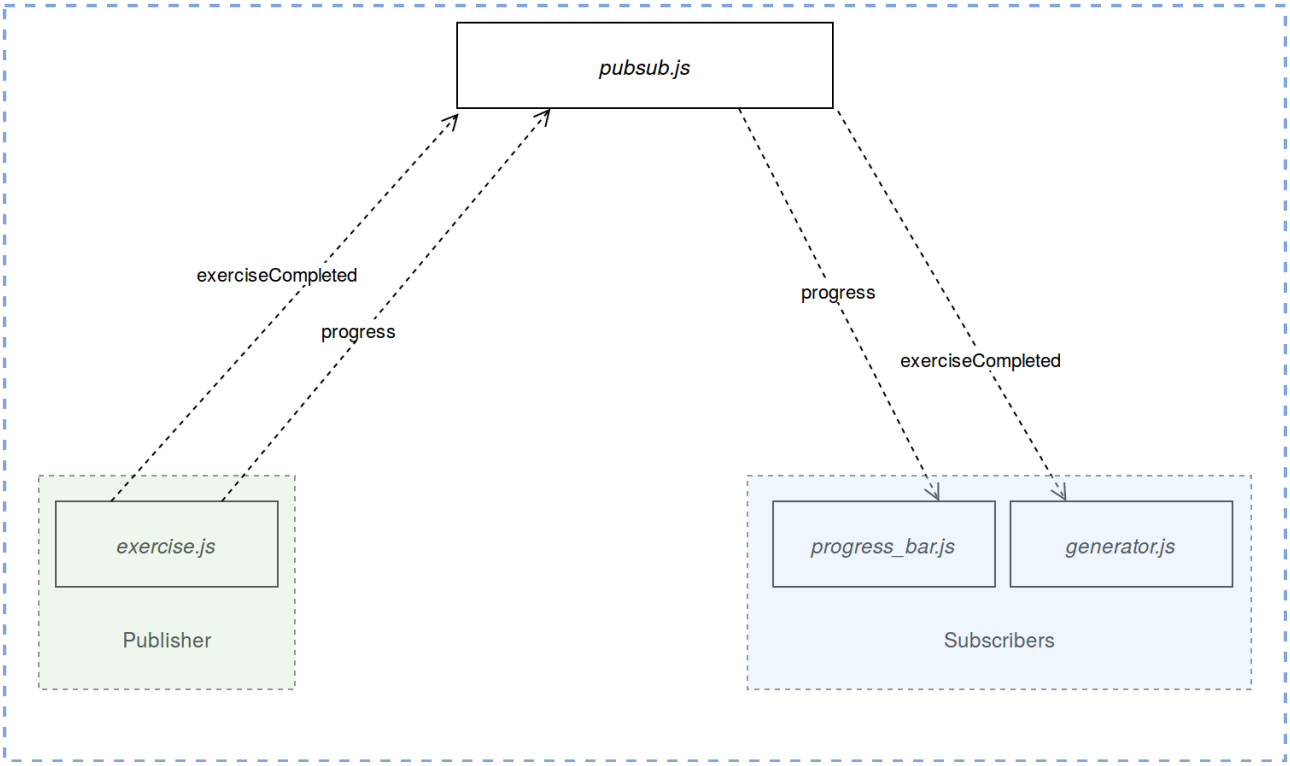


Figure 2.1: Common content is depicted in blue
Custom content (loaded from *exX.html*) is depicted in red

Therefore, for example, if the user chooses the first exercise, then the "main-content" div of *index.html* (that now contains the home page) is reloaded with the content of *exercise.html*. Given the modular structure of the application the garbage collector then destroys all the objects related to previous module, since all the binding elements are no longer referenced. The inheritance of the exercises allows less code repetition. This way first we load the most common DOM elements, then for each particular exercise we load the relevant content. This is loaded in the "custom-content" div of *exercise.html*, in this case *ex*1.*html* is added (since we chose the first exercise). This cumulates into the final page, as seen by the user (presented above).

2.3 Design Patterns

2.4 Publish Subscribe Pattern



The publish-subscribe pattern is an important part of the application. It gives the flexibility in controlling JS modules without coupling them. In order to explain the pattern consider the image above. In our project progress bar is an object literal, that is independent of any other module. Therefore to keep its independence any exercise should only emit changes to it, while progress bar does not need to know about the existence of the exercise. This way each running exercise sends a 'progress' signal whenever it was successful. The progress bar listens to the signal and updates when it receives.
Secondly, it allows full control of the generator of the exercises without heavily coupling them. When a set of the some type of exercises are completed, an 'exerciseCompleted' signal is send (from $exercise.js$). The generator receives the signal and creates the next set.
In addition, at the end of a set of exercises, the user is asked if it wants to redo; if he accepts, then the entire set is restarted. On the other hand, if he presses the cancel button, then a signal is send and he is returned to the home page. This allows us to remove any binding reference to generator and delete the object from the memory.
In addition the pattern is very similar to observer observable patter in other languages. The exercise has no knowledge of the existence of any subscribers and the subscribers do not know the origin of the signal they receive. The fact that the subscribers and the publisher do not need to know about each other maintains the modularity of the software.

2.5 Other tools

2.5.1 Hotjar

We have integrated Hotjar[2] script in order to get insight into the users activities and help us in making decisions while designing the user interface.

2.6 Bootstrap

To support different screen sizes (e.g. mobile phone support) we have used the bootstrap framework. The application thus supports any platform.

[2] https://www.hotjar.com/

2.7 Logic View

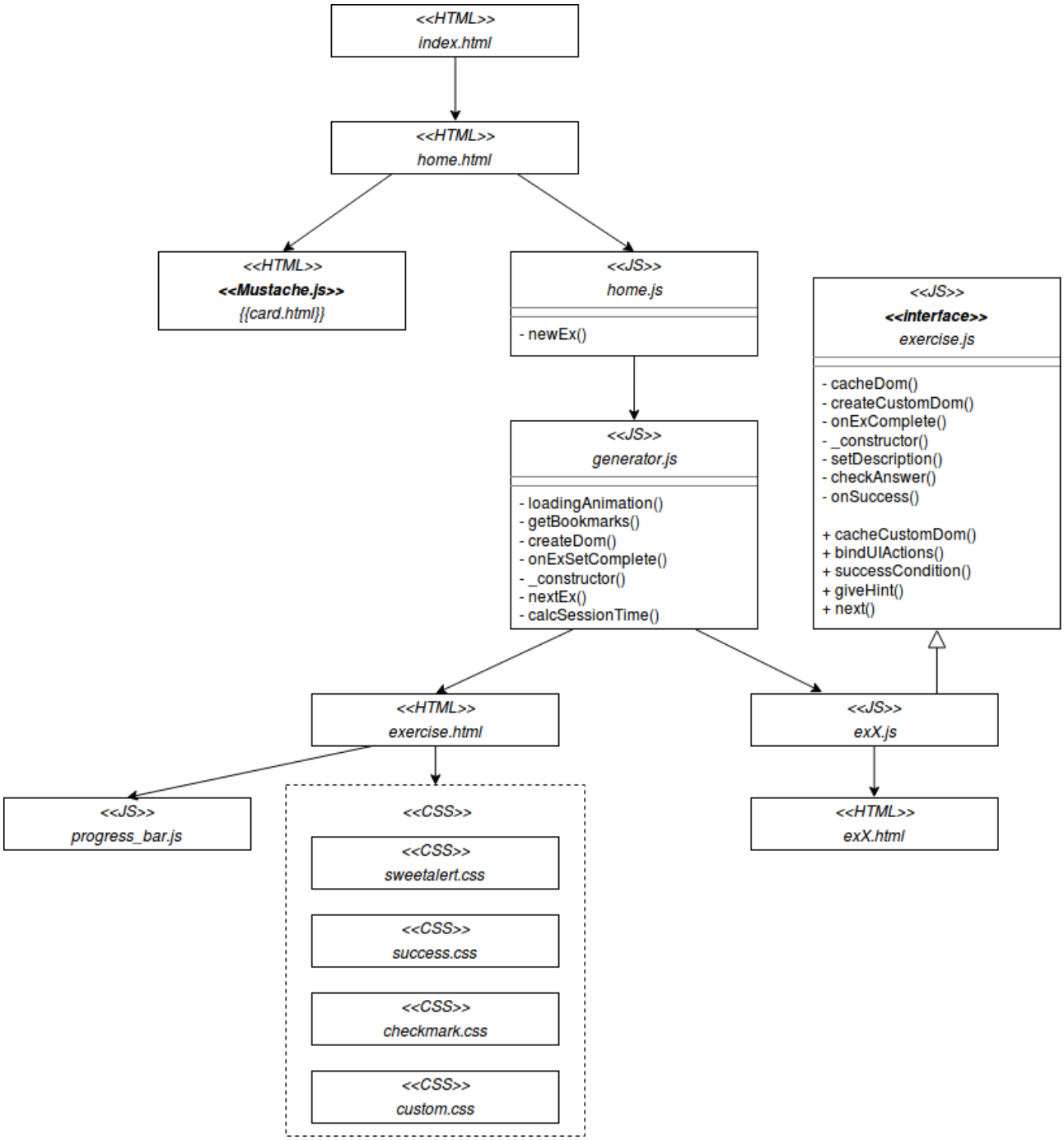


Figure 2.2: UML Diagram

3 Usage

3.1 How to run the code on local machine

At first clone the repository from https://github.com/martinavagyan/zeeguu-exercises.git
In order to run the code on the local machine, we need to install the following dependencies: Python Virtual environment, *Flask*, *Gunicorn*, *Requests*.

The following tutorial shows how to install *virtualenv* and *Flask*:
http://flask.pocoo.org/docs/0.12/installation/

After installing virtual environment in the project folder, activate it using:

```
venv/Scripts/activate
```

Make sure the environment is activated before proceeding with the instructions

To install *gunicorn* use the following command:

```
pip install gunicorn
```

More details here:
http://docs.gunicorn.org/en/stable/install.html

To install the *requests* package simply do:

```
pip install requests
```

Afterwards, you need to (re)generate the requirements file:

```
pip freeze > requirements.txt
```

And then save it:

```
pip install -r requirements.txt
```

A batch file (*batchfile.bat*) is provided to easily activate the virtual environment and start the local host, given that all the previous installations are successfully completed.

3.2 Heroku Deployment

The code is already deployed on the *Heroku* server[3]. The deployment is done through Github.
The link to our app:
zeeguu.herokuapp.com
The repository connected to heroku is:
https://github.com/martinavagyan/practice-as-a-service
In order to contribute to the project you can fork the repository, connect it to heroku to have your own running version.
How to deploy a Phyton app on Heroku:
https://devcenter.heroku.com/articles/getting-started-with-python#introduction

[3] https://www.heroku.com/

4 Appendix

4.1 Plan

4.1.1 Overview

The purpose of the project is to develop a dashboard and add learning tools for the Zeeguu platform. The Zeeguu is designed to speed up vocabulary learning in a new language and make the overall activity more interesting. The platform is evolving and aims to improve the learning experience both by traditional and innovative methods, for example by free reading with vocabulary building and language proficiency assessment.
The platform consists of the following pieces of software, most of which are currently in development:

1. an iOS app
2. an Android app
3. a web-based dashboard

4.1.2 Goals and Scope

This project will focus on the the dashboard, more precisely, we will extend its functionality, by adding multiple new learning tools in the form of exercises. They will include interactions with texts, images and sounds. The mentioned exercises will be of four types:

1. written text-based learning (e.g. choose the right meaning of a word in a given context, select the right synonym of a word)
2. learning using images (e.g. write the word that best fits the object in the image or select from multiple choices)
3. audio learning (e.g. write the word that is dictated to you)
4. reorganizing sentences. (e.g. learning a Zeeguu platform is)

4.1.3 Project constrains and scope

We envision the tools to have their application user interface (API) and provide flexibility in reusing them in different parts of the Zeeguu platform.
The formal constraints:

1. reusable (the code must be easy to understand, use and extend)
2. responsive (the dashboard must be working properly on different platforms)

4.1.4 Users

The primary users of the final product are all individuals that want to learn a new language. However, our initial user group will be composed of students from a secondary school. Another user, later, customer group, can be companies who provide language learning. Since the project will provide API for learning tools, other parties that provide language learning can use these tools.

4.1.5 Risks

The main risk of the project is the competition in the market. There are multiple other platforms that provide tools and API's for language learning. Mainly language learning through exercises in multiple platforms.