



VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Minh Nguyen

A WEB APPLICATION FOR COMPENSATING COURSES WITH FREETIME ACTIVITIES

Technology and Communication
2022

ACKNOWLEDGEMENTS

I am thankful for my time at Vaasa University of Applied Sciences. I am sure that all the knowledge I have learnt will follow me forever.

I would like to thank Dr. Ghodrat Moghadampour for being my supervisor and helping me with my thesis.

Also, I want to show my appreciation to my CTO Simon Björklund for giving me this opportunity to apply what I have learnt and for helping and guiding me throughout this project. Thank you, my colleagues, especially Martin Fellman who helped me to solve problems and make this web application as perfect as possible.

I wish all the best to you all.

Helsinki, 13.02.2022

Nguyen Hieu Minh

ABSTRACT

Author	Minh Nguyen
Title	A Web Application for Compensating Courses with Freetime Activities
Year	2022
Language	English
Pages	78
Name of Supervisor	Ghodrat Moghadampour

The aim of this thesis was to create a web application with a platform for students to apply for compensation for courses which they have done outside of their schools. This application was developed for 10Monkeys company located in Helsinki as a part of Competence Disc.

The Osaamiskiekko – name of the web application in Finnish - displays how the content of validated training offered by NGOs and digital badges corresponds to studies leading to a qualification. The app showed the education institutes' recommendations for how competence gained through NGO activities could be identified and recognized in studies.

The project was successful in helping students find out what study course and credits they can compensate with the competence acquired from some organizations. As a result of the project, students can seek the most matching correlation between their degree and competence.

Keywords Web application, firebase, frontend, backend, nuxtjs

CONTENTS

ACKNOWLEDGEMENTS

ABSTRACT

1	INTRODUCTION	1
1.1	Motivation.....	1
1.2	Objectives.....	1
2	TECHNOLOGIES.....	2
2.1	HTML5	2
2.2	CSS3.....	2
2.3	VueJS	3
2.4	NuxtJS.....	4
2.5	Vue Bootstrap	5
2.6	Firebase.....	5
3	APPLICATION DESCRIPTION.....	8
3.1	Requirements.....	8
3.2	Requirements Analysis.....	9
3.2.1	Use Case Diagram	9
3.2.2	Search-For-The-Match Sequence Diagram	11
3.2.3	Contact Form Sequence Diagram	12
4	DATABASE AND GUI DESIGN	13
4.1	Database	13
4.2	GUI Design.....	15
4.2.1	Homepage	15
4.2.2	Students, Schools and Organizations Page	21
4.2.3	Sign-up Page.....	26
4.2.4	Competence and Degree Details Page.....	28
4.2.5	Navigation	30
4.2.6	Footer	32
5	IMPLEMENTATION.....	34
5.1	Environment Setup	34
5.1.1	Installing and Configuring VSCode	34

5.1.2	Installing NodeJS	34
5.1.3	Create Nuxt Application	35
5.1.4	Dependencies Installation.....	36
5.2	Architecture	37
5.3	Shared Components.....	39
5.3.1	Navigation	39
5.3.2	Footer	43
5.3.3	Default Layout.....	44
5.3.4	Content Fetching.....	46
5.3.5	Cookie Consent	49
5.4	Pages	51
5.4.1	Homepage	51
5.4.2	Students, Schools and Organizations Page	56
5.4.3	Sign-up Page.....	61
5.4.4	Competence and Degree Details Page.....	64
6	DEPLOYMENT AND TESTING	68
6.1	Deployment.....	68
6.2	Testing.....	70
6.2.1	Change the Language.....	70
6.2.2	View the Details of the Match	71
6.2.3	Send the User's Data Through Contact Form	72
7	CONCLUSIONS	75
7.1	Future Work	75
	REFERENCES	76

APPENDICES

LIST OF FIGURES

Figure 1.	Composition of a View in Nuxt [6]	4
Figure 2.	Comparison between traditional model and Firebase [9].....	6
Figure 3.	Firebase suite [9].....	6

Figure 4. User's functions	10
Figure 5. Admin's functions	10
Figure 6. Website owner's function	11
Figure 7. Sequence diagram for searching the match between competence and degree	12
Figure 8. Contact form sequence diagram	12
Figure 9. Database	13
Figure 10. A set of YAML files	13
Figure 11. Header design	15
Figure 12. Header design on a mobile	16
Figure 13. Choose a school or organization	16
Figure 14. Choose a school or organization on a mobile	17
Figure 15. Choose a degree or education	17
Figure 16. Choose a degree or education on a mobile	18
Figure 17. A list of options regarding user's choices	18
Figure 18. A list of options regarding user's choices on a mobile	19
Figure 19. One of the container designs	19
Figure 20. One of the container designs on a mobile	20
Figure 21. Category design	20
Figure 22. Category design on a mobile	21
Figure 23. Part 1 design	22
Figure 24. Part 1 mobile design	22
Figure 25. Part 2 and 3 design	23
Figure 26. Part 2 and 3 mobile designs	23
Figure 27. Part 4 design	24
Figure 28. Part 4 mobile design	24
Figure 29. Part 5 design	25
Figure 30. Part 5 mobile design	25
Figure 31. Sign up page design - part 1 on a mobile and a desktop	26
Figure 32. Part 2 design	27
Figure 33. Part 2 mobile design	27

Figure 34. Part 3 design	28
Figure 35. Part 3 mobile design	28
Figure 36. Competence and degree detail page design	29
Figure 37. Competence and degree detail page mobile design	29
Figure 38. Navigation links box	30
Figure 39. Navigation links box on a mobile	30
Figure 40. Navigation layout	31
Figure 41. Hover effect	31
Figure 42. Language dropdown selection	31
Figure 43. Mobile and tablet navigation	31
Figure 44. Mobile and tablet navigation when open	32
Figure 45. Footer design	32
Figure 46. Footer design on a mobile	33
Figure 47. Project structure	35
Figure 48. Pages folder	37
Figure 49. Content folder	46
Figure 50. Yml files	47
Figure 51. Cookie consent design	49
Figure 52. Inside data folder	52
Figure 53. Inside one of the folders	52
Figure 54. Import project	68
Figure 55. Edit build settings	68
Figure 56. Branches settings	69
Figure 57. Environment variable settings	69
Figure 58. Deployment	70
Figure 59. Before language change	71
Figure 60. After language change	71
Figure 61. Before choosing the match	71
Figure 62. After choosing one of the matches	72
Figure 63. The user fills in the information and submit	73
Figure 64. A thank you message	73

Figure 65. Website owner receives user's information.....	74
--	-----------

LIST OF CODE SNIPPETS

Code Snippet 1. Minimal example of Vue application [4].....	3
Code Snippet 2. Install dependencies [8]	5
Code Snippet 3. Adding bootstrap-vue/nuxt to modules section of nuxt.config.js file [8]	5
Code Snippet 4. Data structure in yaml file.....	14
Code Snippet 5. Convert database from yaml to JSON format.....	14
Code Snippet 6. An example of JSON file after conversion.....	14
Code Snippet 7. Creating a nuxt application	35
Code Snippet 8. Running the application in development environment.....	35
Code Snippet 9. Dependencies.....	36
Code Snippet 10. Install a dependency with npm	37
Code Snippet 11. Install a dependency for testing and local development.....	37
Code Snippet 12. Index.vue file	38
Code Snippet 13. Router config option in nuxt.config.json	39
Code Snippet 14. Example of one link item.....	39
Code Snippet 15. Language dropdown for a desktop	40
Code Snippet 16. Language list.....	40
Code Snippet 17. Page reloading function	41
Code Snippet 18. Mobile nav.....	41
Code Snippet 19. Closing language dropdown function	42
Code Snippet 20. Example of a link in footer	43
Code Snippet 21. Example of an external link.....	43
Code Snippet 22. Example of an image link	44
Code Snippet 23. Default template	44
Code Snippet 24. JS part.....	45
Code Snippet 25. Site list configuration	46
Code Snippet 26. Default SCSS	46
Code Snippet 27. Page data fetching.....	48

Code Snippet 28. Fetch content from content folder	48
Code Snippet 29. Retrieving data	48
Code Snippet 30. Example of home-en.yml file	48
Code Snippet 31. Component data fetching	49
Code Snippet 32. Customize the look and feel of cookie consent	50
Code Snippet 33. Language of the cookie consent	50
Code Snippet 34. Header implementation	51
Code Snippet 36. Importing functions of retrieving data.....	53
Code Snippet 37. Retrieve data	53
Code Snippet 38. Example of one container implementation	53
Code Snippet 39. Page section implementation	54
Code Snippet 40. Category section implementation	55
Code Snippet 41. Category item implementation	55
Code Snippet 42. Category data object	56
Code Snippet 43. Part 1 implementation	57
Code Snippet 44. Part 2 and 3 HTML implementation.....	57
Code Snippet 45. Part 4 HTML implementation.....	59
Code Snippet 46. Part 5 HTML implementation.....	59
Code Snippet 47. <navigation-links-box> HTML implementation.....	60
Code Snippet 48. <arrow-link> HTML implementation.....	61
Code Snippet 49. Part 1 HTML implementation.....	61
Code Snippet 50. One example of form HTML implementation.....	62
Code Snippet 51. Submit data	63
Code Snippet 52. Part 3 HTML implementation.....	63
Code Snippet 53. ContactInfo component HTML implementation.....	64
Code Snippet 54. Fetch data from backend API	65
Code Snippet 55. Find proper school, organization, and their level	65
Code Snippet 56. Data computed properties.....	66
Code Snippet 57. Initialize data with corresponding parts	67

LIST OF ABBREVIATIONS

YAML - YAML Ain't Markup Language

NGO - Non-Governmental Organization

HTML - Hypertext Markup Language

CSS - Cascading Style Sheets

SEO - Search Engine Optimization

SPA - Single Page Application

JS - JavaScript

YARN - Yet Another Resource Negotiator

NPX - Node Package Execute

NPM - Node Package Manager

SCSS - Sassy Cascading Style Sheets.

HTTP - HyperText Transfer Protocol

API - Application Program Interface

JSON - JavaScript Object Notation

UI - User Interface

UX - User Experience

MYSQL - My Structured Query Language

VSCODE- Visual Studio Code

1 INTRODUCTION

1.1 Motivation

The Youth Work Centre of Expertise (Kentauro) and Ministry of Education and Culture want to rebuild the whole Competence Disc (in Finnish Osaamiskiekkko) from the outdated version. Its purpose is to recognize the youth's skills from their free time as a part of their studies [1]. For example, when we join a course for our hobby or a volunteer activity, we will get some credits for that, and we can use the credits to compensate for some university's courses. Hence, we both reduce the study time and enjoy our free-time and hobby activity, which means everything counts.

1.2 Objectives

Students can find relevant information on the match between a university's degree and an organization's competence.

The web application must support responsiveness, accessibility, and be multilingual in English, Finnish, Swedish.

Users can contact the website owner to modify, update or add the latest information to the web application.

No corruption in data is allowed, which means when users select a school and a degree, they must have corresponding competences and organizations and vice versa. There is no room for faults here.

Students can search for every university and organization together with their degrees and competences in Finland.

2 TECHNOLOGIES

The project was made with VueJS, NuxtJS, HTML5, CSS3, JavaScript, Vue Bootstrap for frontend and Firebase, a set of YAML files for backend. The data is stored and handled with YAML file.

2.1 HTML5

HTML5, or Hypertext Markup Language, is the most recent version of the code that represents online pages. HTML, which gives structure, Cascading Style Sheets (CSS), which handles display, and JavaScript, which makes things happen, are the three types of code. [2]

HTML5 was created to perform practically everything we would like to do online without the use of third-party software such as browser plugins. It can accomplish everything from animation to applications, music to movies, and it can even be used to create extremely complex web applications. [2]

All the major browsers, including Internet Explorer, Edge, Firefox, Chrome, Safari, and Opera, as well as Mobile Safari and Android's browser, support HTML5, although not in the same way. [2]

2.2 CSS3

CSS3 (Cascading Style Sheets Level 3) is the third version of the CSS standard, which is used to style and format Web pages. CSS3 is based on the CSS2 standard, but with a few tweaks and enhancements. [3]

The splitting of the standard into different modules, which makes it easier to learn and understand, is a notable change. The World Wide Web Consortium (W3C) is continuing working on the standard as of February 2014, but several CSS3 properties have been implemented in recent versions of some Web browsers. [3]

CSS3 modifies the way a browser implements and renders various visual elements. Unlike CSS2, however, it is not a single cumbersome specification. To make development easier, CSS3 is divided into independent modules. This means that the specification will be released in pieces, with some modules being more stable than others. [3]

2.3 VueJS

According to its website, VueJS is one of the JavaScript frameworks best known for building user interfaces. It lays on top of the standard HTML, CSS, and JavaScript. It also supplies a declarative and component-based programming model that could make us efficient in developing user interfaces no matter how it is simple or complicated [4].

```
import { createApp } from 'vue'

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')

<div id="app">
  <button @click="count++">
    Count is: {{ count }}
  </button>
</div>
```

Code Snippet 1. Minimal example of Vue application [4]

The above code snippet shows us the two core features:

- **Reactivity:** JavaScript state changes are tracked automatically, and the DOM will be effectively updated by Vue when these changes happen [4].
- **Declarative Rendering:** Standard HTML will be extended by Vue with a template syntax which lets us declaratively describe HTML output depending on JavaScript state [4].

2.4 NuxtJS

NuxtJS is a free, open-source web application framework created from Webpack, Babel, Vue, and Node. Nuxt makes it possible to create JavaScript web views with the Vue single file component system and function not only as in-browser SPA views but also as server side-rendered web views which can together make a full SPA functionality [5].

Besides, it also allows users to see contents or parts of them pre-rendered and served in a static site [5]. This way makes sure to reduce response time and better SEO than traditional SPA since the server serves the whole content of the page before any client-side JS-action can be taken [5]. The most interesting and best part is that Nuxt simplifies the application setup and configuration. This truly enables coders to develop the UI faster and easier yet more effectively as Vue does [5].

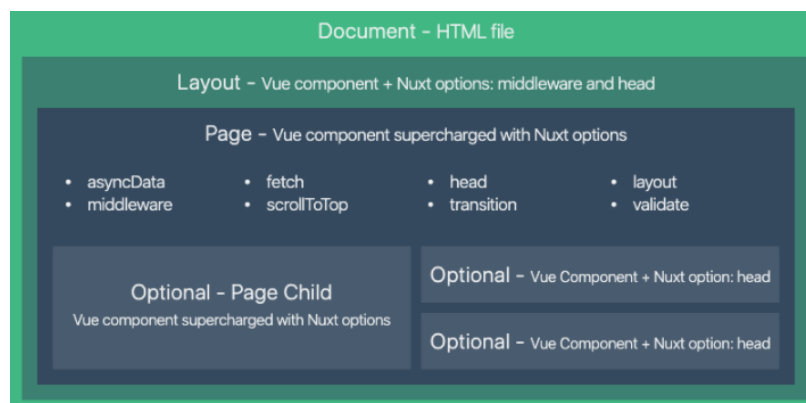


Figure 1. Composition of a View in Nuxt [6]

The above figure describes what data and views we need to configure for a specific route. It is called View which includes an application template, a layout, and the actual page [6].

2.5 Vue Bootstrap

Vue Bootstrap is a front-end CSS library like Bootstrap which is made for Vue.js. This tool helps to create mobile-first, responsive and accessible projects [7]. It embraces ready-made components such as Form, Card, Table, Modal and so on together with a full collection of icons that could really ease web making.

To get started with NuxtJS, dependencies must be installed, as shown in **Code Snippet 2** and **Code Snippet 3**.

```
# With npm
npm install bootstrap-vue
```

```
# With yarn
yarn add bootstrap-vue
```

Code Snippet 2. Install dependencies [8]

```
module.exports = {
  modules: ['bootstrap-vue/nuxt']
}
```

Code Snippet 3. Adding bootstrap-vue/nuxt to modules section of nuxt.config.js file [8]

2.6 Firebase

Firebase is a set of tools that helps developers to “build, improve and grow their app” by providing a lot of services which coders have to build themselves but do not really want to, since they prefer taking care of application experience. The services include analytics, databases, authentication, configuration, push messaging, file storage and more. These services are hosted in the cloud and will

be scaled easily according to the developer's need. As a result, the application development is more economical and requires less effort [9].

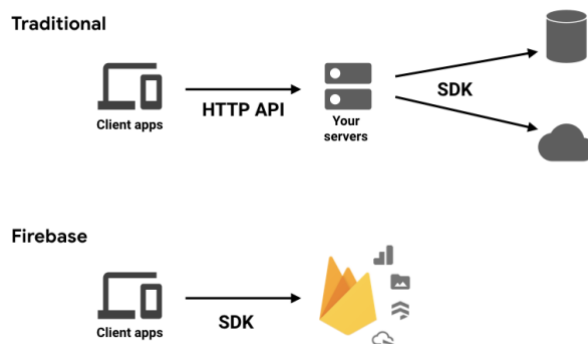


Figure 2. Comparison between traditional model and Firebase [9]

As we can see, all the backend components are fully operated and maintained by Google. These backend services will interact directly with client SDKs provided by Firebase without founding any middleware between the application and the service. This is not the case when it comes to traditional application development, where both frontend and backend must be written. The frontend will call the APIs exposed by the backend. Nonetheless, the traditional backend is avoided, and the work is pushed to client in Firebase products thanks to Firebase products [9]. **Figure 3** shows the services provided by Firebase.

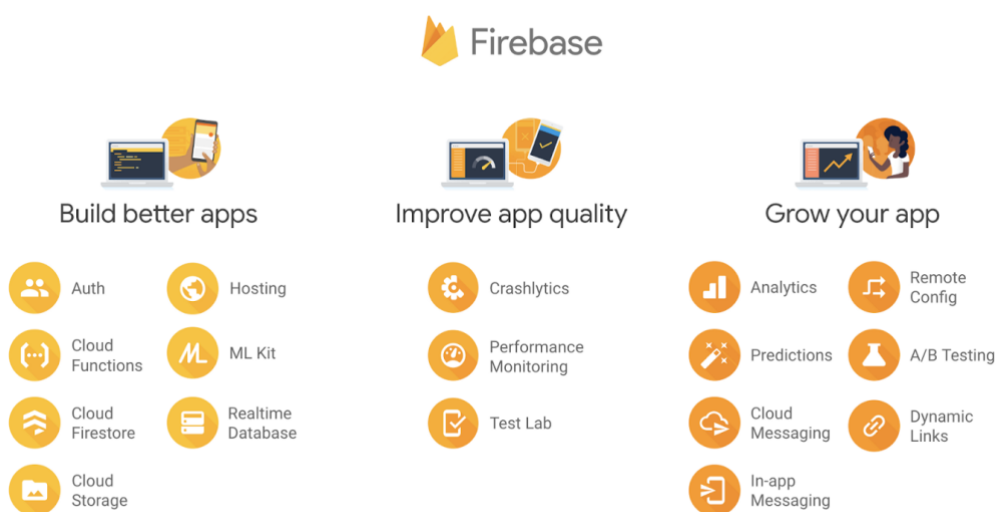


Figure 3. Firebase suite [9]

In this project, Firebase was used to create a backend for sign-up form data handling where the user would fill in the information. When it is sent, the data will be delivered to the owner of the project.

3 APPLICATION DESCRIPTION

The objective of the web application is mainly for students to look for what volunteer work or hobby activities can be exchanged for study credits in their desired school. On the other hand, they can find what study credit can be replaced by what activity.

Firstly, users start from the homepage, from which they can see some introductions on why they need to study the same thing twice, free credits and how to sign up. In addition, there is a search bar which helps them to find an organization or a school and the corresponding degree or competence. Afterwards, they can click at one of those links below to learn more details.

Secondly, users can choose Student page or School or Organization page depending on their role to see how the web application can benefit them. Next, users can go to Sign up site to contact and propose any school or competence they want to have in this web application. Furthermore, they can have a look at Privacy Policy and Accessibility Statement to learn more about the web application and the organization behind it.

3.1 Requirements

The requirements of the web application are present in this section.

The must-have requirements of the application are:

- Users can learn more about the match connecting the course from school and one from organization through a separate web page.
- The search function works well.
- No corruption in data.

- The web application supports three languages, Finnish, English and Swedish.

The should-have requirements of the application are:

- Users are able to view separate introduction pages depending on their role, for example, students or schools and organizations.
- Users can contact the website owner through a contact form when they want to add or modify a school, a degree, a competence, or an organization.
- Responsiveness and accessibility are supported.
- Have a privacy policy and accessibility statements pages

The nice-to-have requirements of the application are:

- One of the images in the background images collection is chosen randomly when the page is loaded.
- The color of the link changes when it is hovered.
- Navigation links box follows users' scroll on match detail page.

3.2 Requirements Analysis

In this part, we will analyze the use case diagram and sequence diagram to learn more about the web application making process.

3.2.1 Use Case Diagram

Figure 4 illustrates that the user can view the Student, School or Organization page, change the language of their preference, contact the website owner, and most importantly find and view the details of the match between the degree and competence.

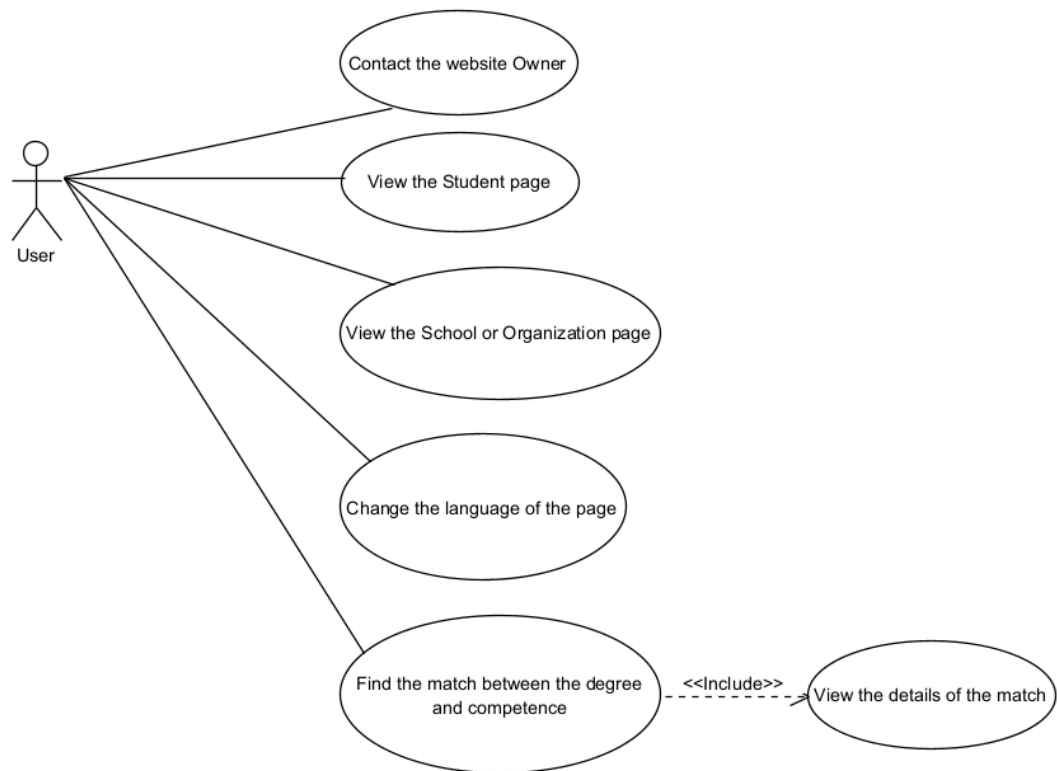


Figure 4. User's functions

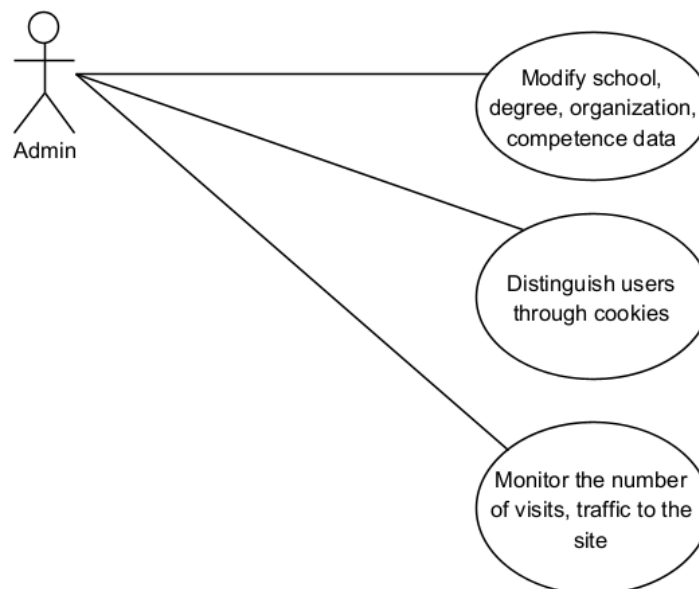


Figure 5. Admin's functions

The administrator can change, modify data related to schools, degrees, organizations, and competences. Also, one can monitor the visits, traffic, where the access is from through cookies accepted by users. Everything complies with the GDPR compliance.

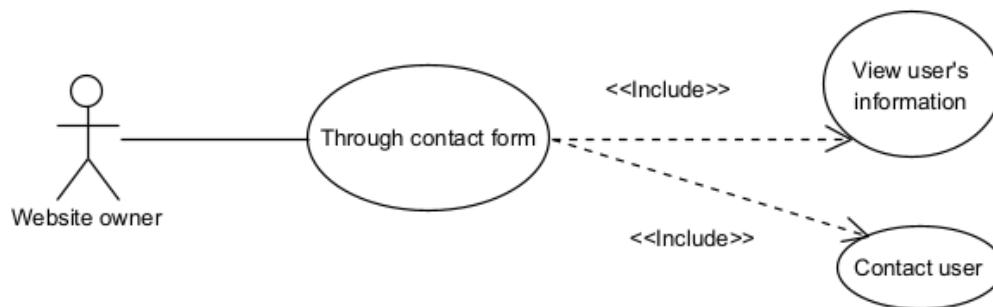


Figure 6. Website owner's function

The website owner can view the user's information including their role, name and message and is able to communicate with them through the contact form on the sign-up page.

3.2.2 Search-For-The-Match Sequence Diagram

The diagram below shows how users start searching. First, they select the school and degree. Secondly, the program will search in the database the corresponding organizations and their competences. It can start vice versa with organization and competence selection then it produces the schools and their degrees.

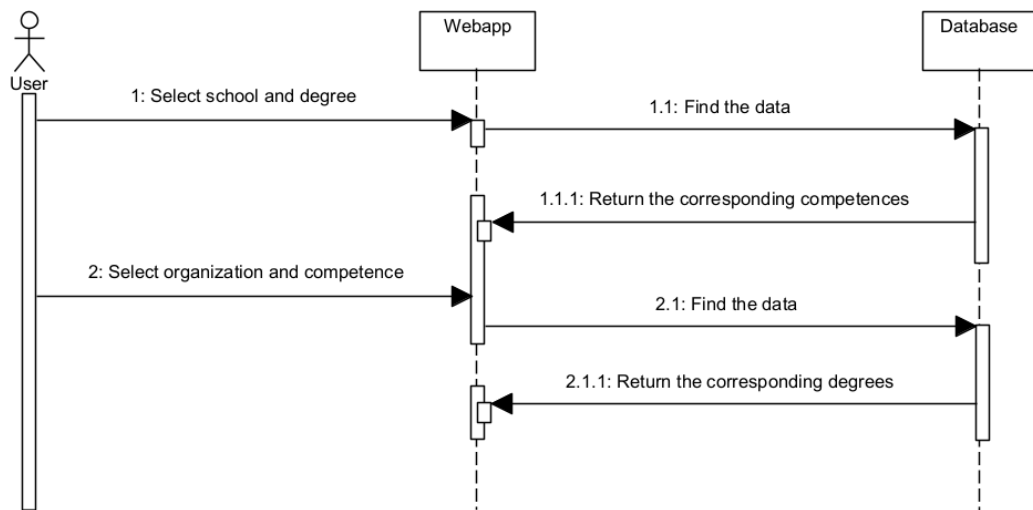


Figure 7. Sequence diagram for searching the match between competence and degree

3.2.3 Contact Form Sequence Diagram

Figure 8 shows the way data flows when users fill in and submit the contact form. When they submit it, the data goes to Firebase to validate and if everything is correct, the user information is sent to the project owner and the users will see the thank you message on the client side.

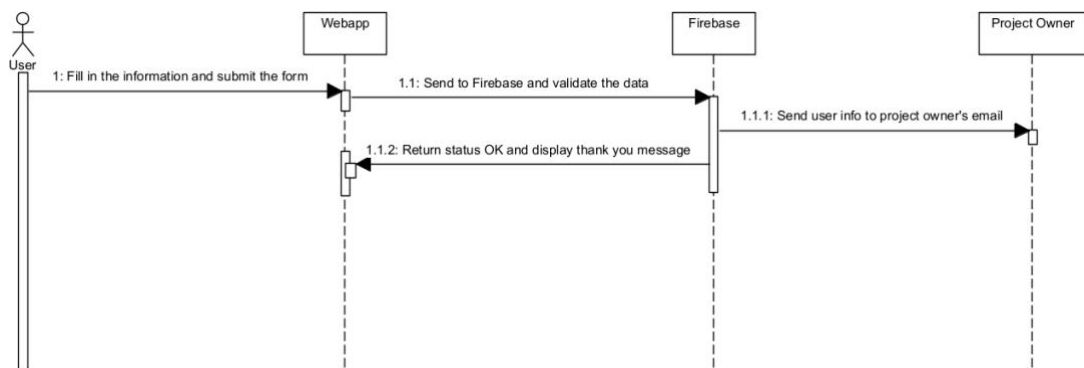


Figure 8. Contact form sequence diagram

4 DATABASE AND GUI DESIGN

4.1 Database

The database is separated into small folders (see **Figure 9**). Each holds the corresponding data according to its name. For example, “competences” has data of competences of organizations, “nqfs” has the data of the level of each degree.

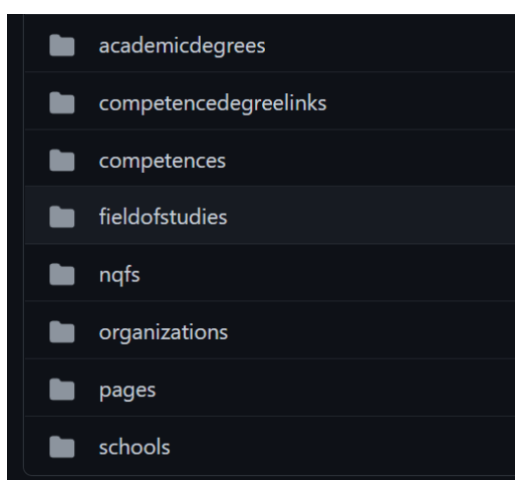


Figure 9. Database

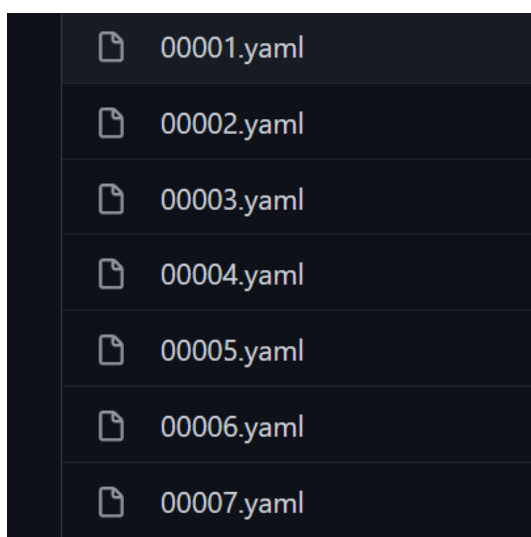


Figure 10. A set of YAML files

Inside these folders are the set of YAML files (**Error! Reference source not found.**) and each of which has a structure of a key-value pair (**Code Snippet 4**), for instance, "id: 1".

```
id: 1
identification: humak-yhteisopedagogi_amk
fieldofstudyId: 1
nqfId: 4
schoolId: 1
credits: 210
url:
"https://www.humak.fi/opiskelijaksi/yhteisopedagogi/"
name_fi: Yhteisöpedagogi (AMK)
name_en: "Community Educator, Adventure and Outdoor
Education (Bachelor of Humanities) "
name_sv: Samhällspedagog (YH)
```

Code Snippet 4. Data structure in yaml file

Later, all these YAML files are converted into JSON files in order to easily retrieve in the frontend by running the **Code Snippet 5**.

```
node makeDatabase.js
```

Code Snippet 5. Convert database from yaml to JSON format

```
{
  "id":10,
  "identification":"seurakuntaopisto-sote",
  "fieldofstudyId":5,
  "nqfId":2,
  "schoolId":2,
  "credits":180
}
```

Code Snippet 6. An example of JSON file after conversion

The JSON files (**Code Snippet 6**) then appear in “/frontend/static/data” folder. The makeDatabase.js is the file my CTO wrote to perform this conversion action.

4.2 GUI Design

This section is about the Graphic User Interface design for the application which was done by the company’s designer. The design implementation was done in this thesis.

4.2.1 Homepage

This section describes the implementation of the home page design. The homepage is divided into header and main parts.

In the header, the left side holds text and a search form while the right contains the background image. The `content` class keeps all the nested elements inside a safe zone so that regardless of the screen size, it will stay in the middle of the screen with certain width and height.

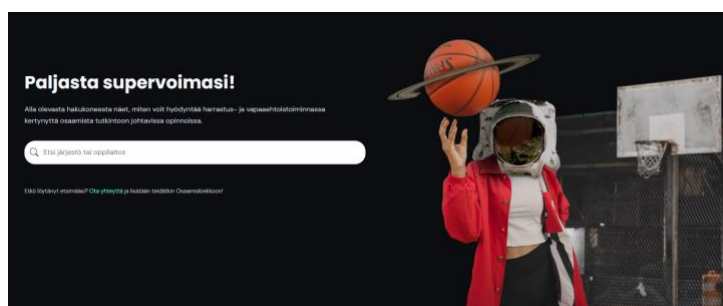


Figure 11. Header design

Figure 11 shows how the homepage looks on a desktop. The background image is randomly chosen in a collection of images when the user reloads or first enters the page.

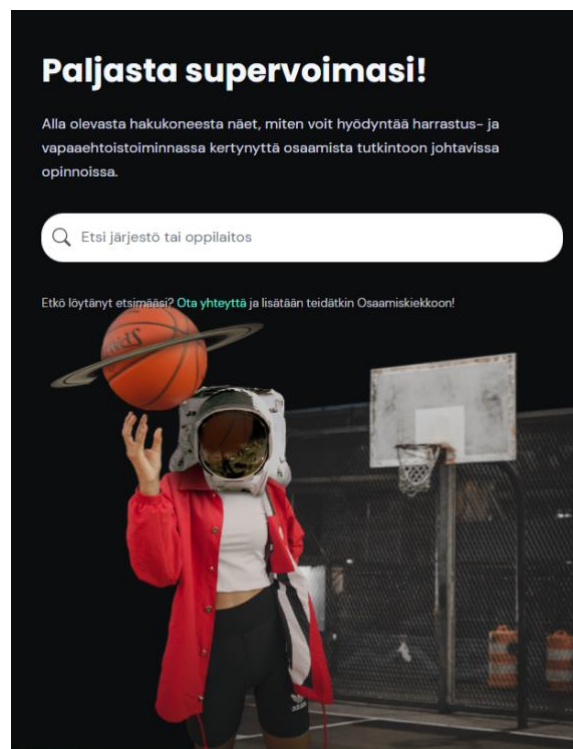


Figure 12. Header design on a mobile

Figure 12 shows what the header looks like on a mobile with a compact view.

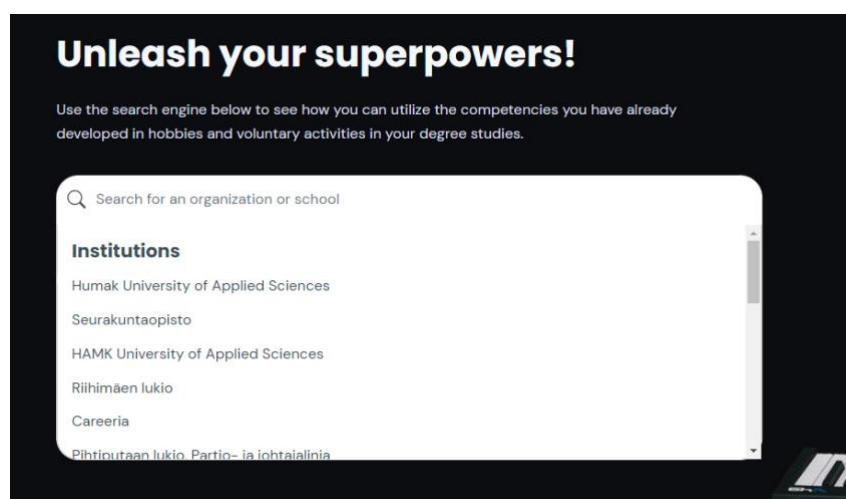


Figure 13. Choose a school or organization

Figure 13 and **Figure 14** show the views where users choose a school or organization from a dropdown menu.

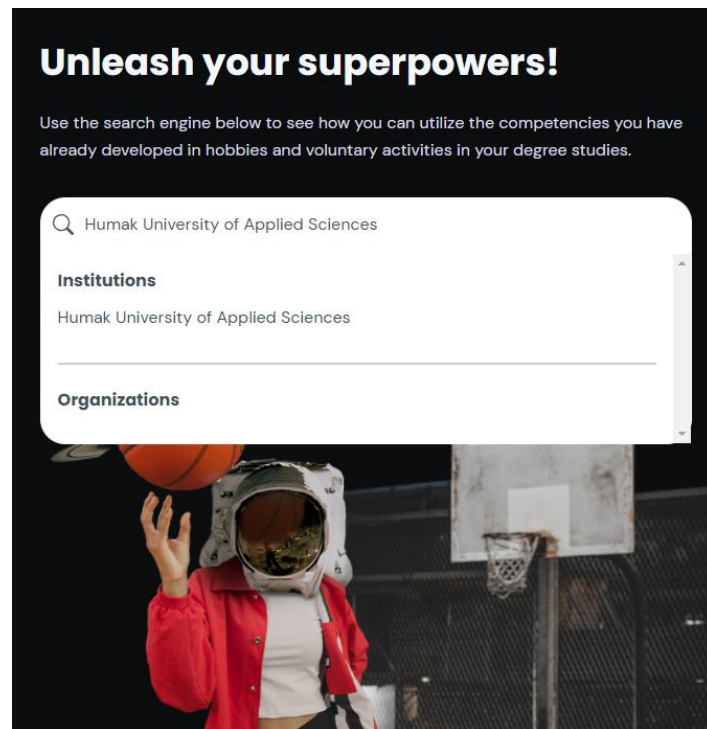


Figure 14. Choose a school or organization on a mobile

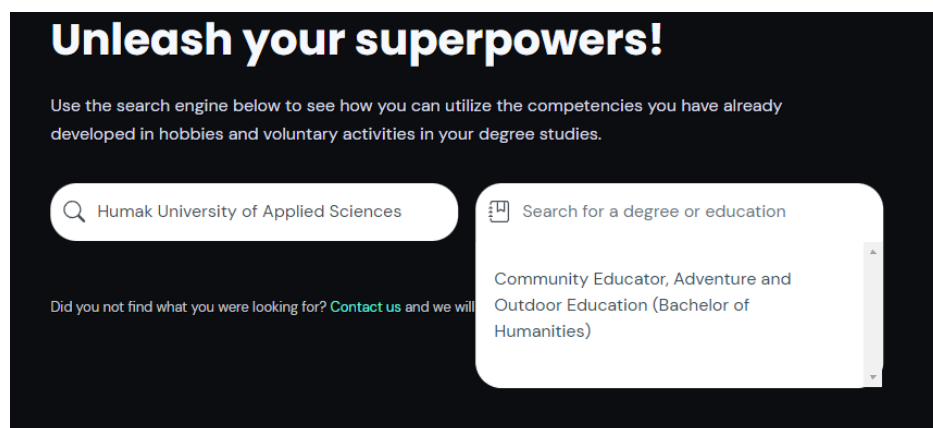


Figure 15. Choose a degree or education

The dropdown menu for a list of degrees or competences on a desktop is shown in **Figure 15** and **Figure 16** on a desktop and a mobile respectively.

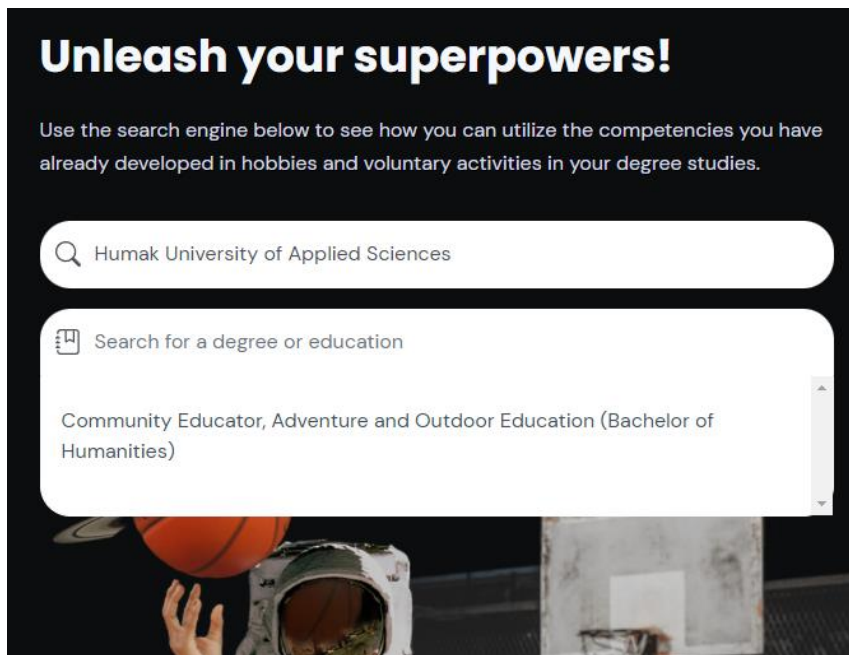


Figure 16. Choose a degree or education on a mobile

After selecting all options, the program then compares the selections and yields the proper result (**Figure 17**). Once the user clicks one of the results, it will take them to another page for more details.

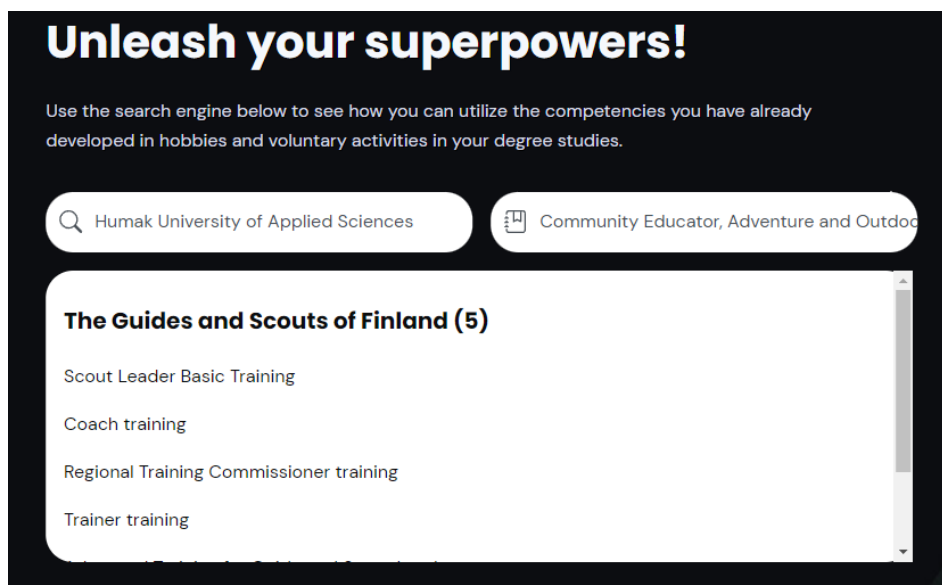


Figure 17. A list of options regarding user's choices

A list of degrees or competences pops up when all selections are satisfied on a desktop and on a mobile (**Figure 17** and **Figure 18**).

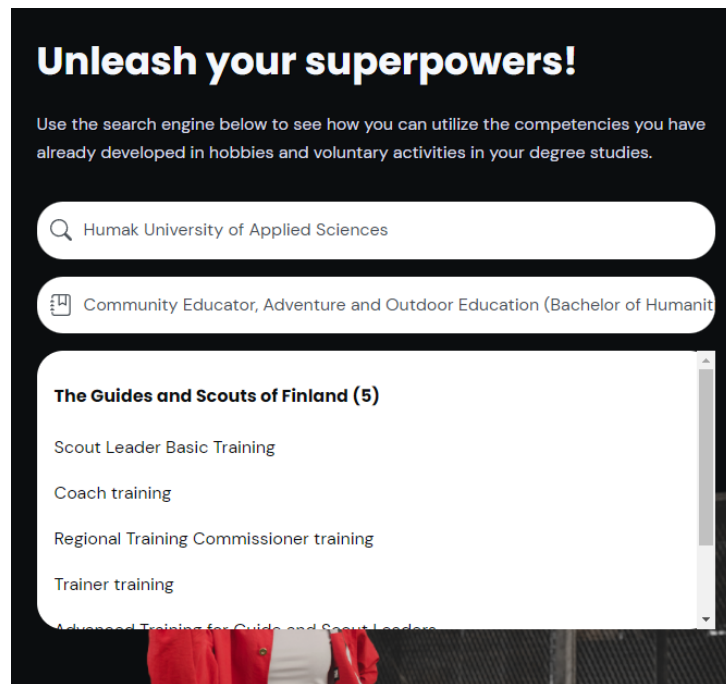


Figure 18. A list of options regarding user's choices on a mobile

In the main section, there are three blocks holding different texts and picture. **Figure 19** and **Figure 20** illustrate the design of one of the blocks on a desktop and a mobile.

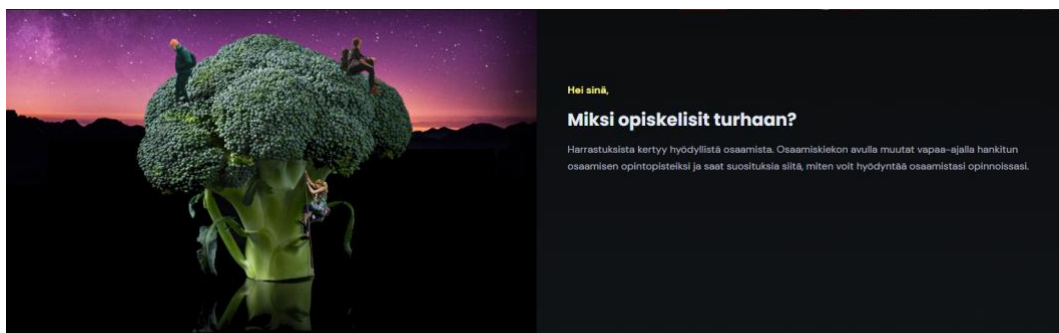


Figure 19. One of the container designs

The block has a picture on one side and the content on the other side on a desktop.

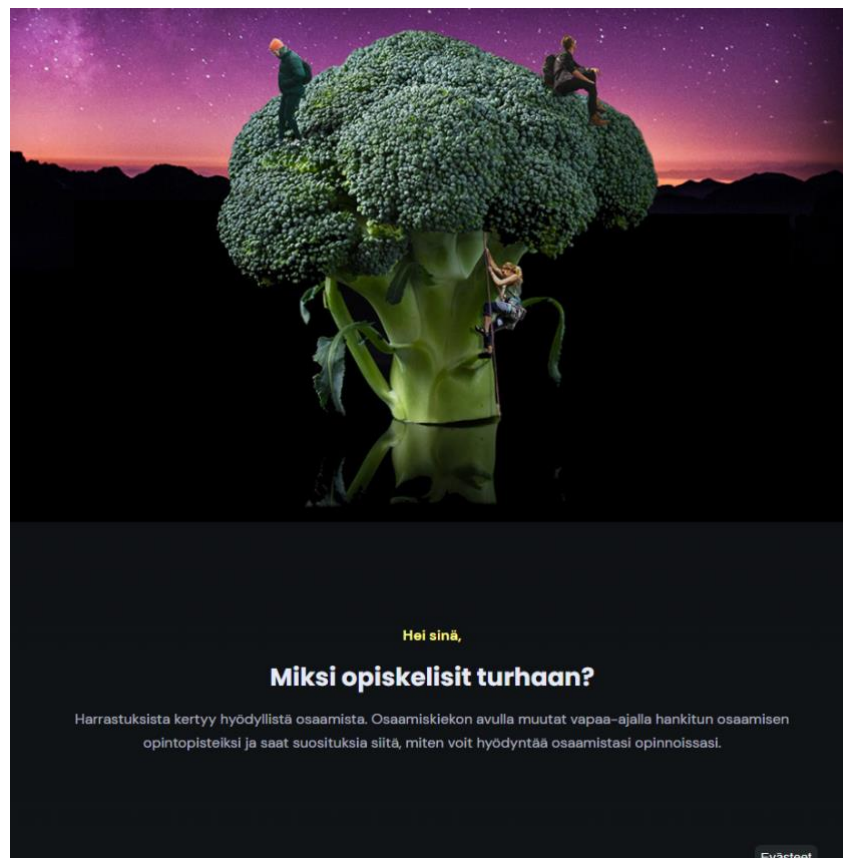


Figure 20. One of the container designs on a mobile

Besides, it also has one sub-section called Category where all four navigators to four main parts of the web application are shown (**Figure 21**).

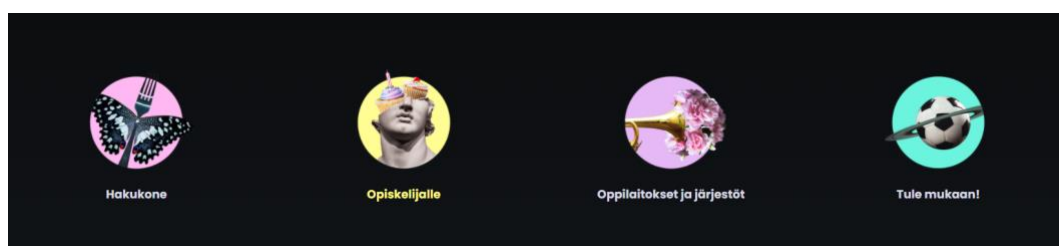


Figure 21. Category design

Through these, users can jump to the part they want faster and more conveniently without scrolling around. The hover color is the theme color of each part when users hover it. **Figure 22** shows the Category design on a mobile in a vertical list.

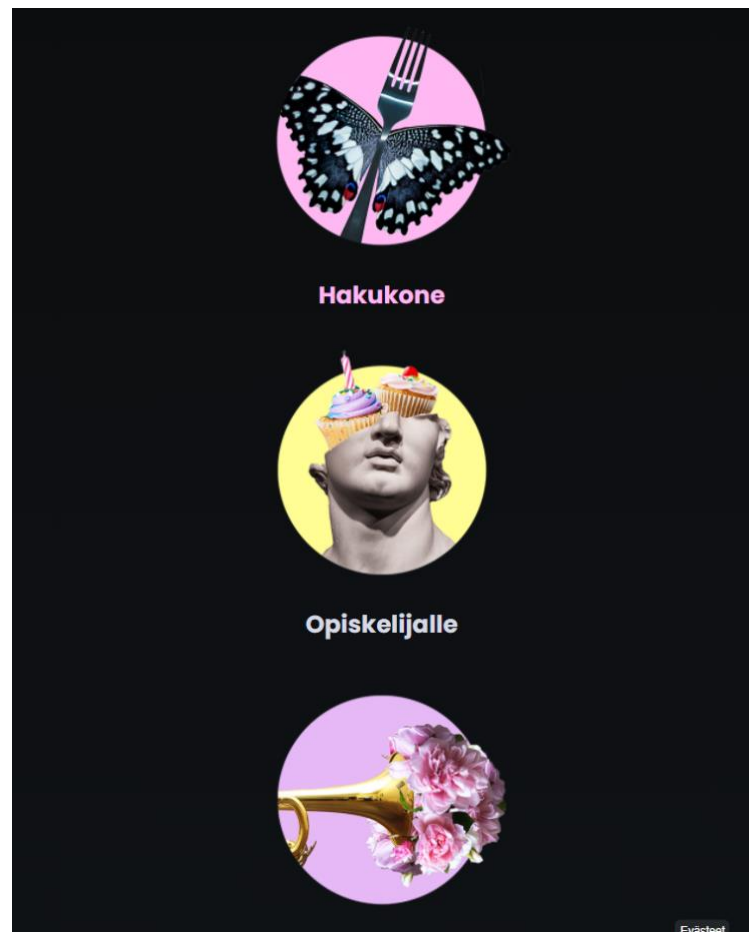


Figure 22. Category design on a mobile

4.2.2 Students, Schools and Organizations Page

Student, school and organization pages share the same layout and only content-wise different.

Part 1 contains text on one side and a picture on the other side. It also comes with a button to go back to the search engine. The design for a desktop has the content on the left and a picture on the right. On a mobile the alignment is vertical.

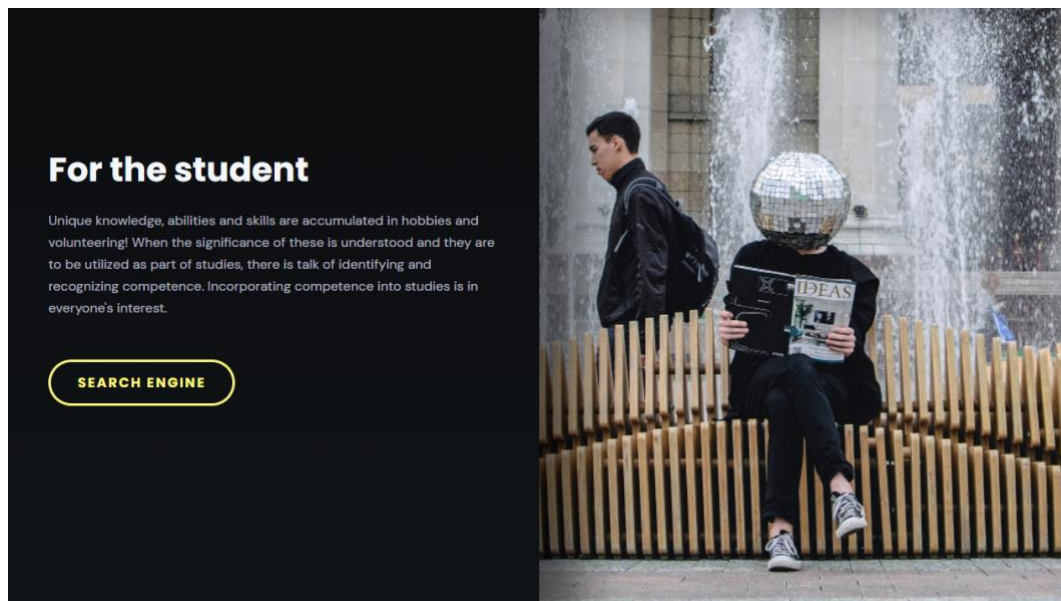


Figure 23. Part 1 design

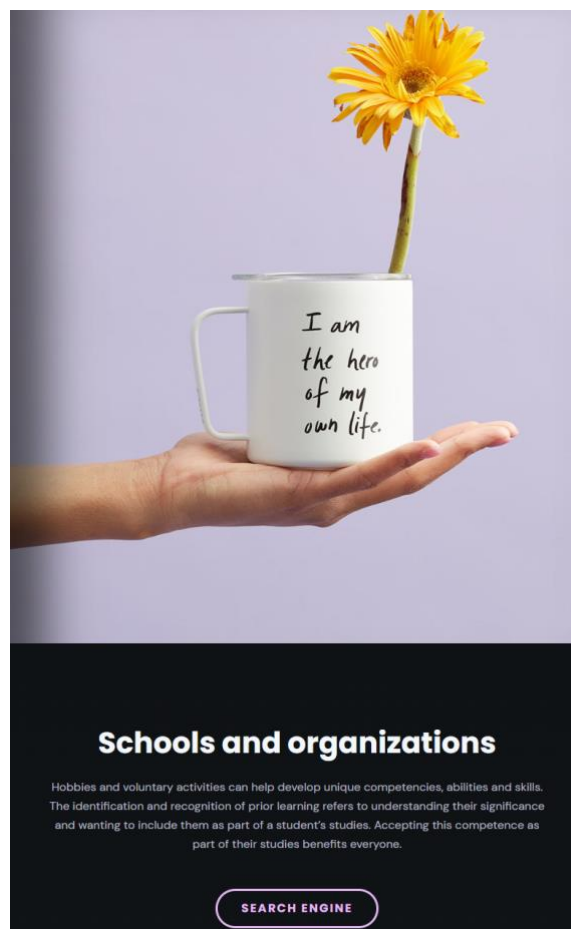


Figure 24. Part 1 mobile design

Parts 2 and 3 only interchange the position of text and picture and have different content regarding the topic of the page. See **Figure 25** for the design for a desktop and **Figure 26** for a mobile.

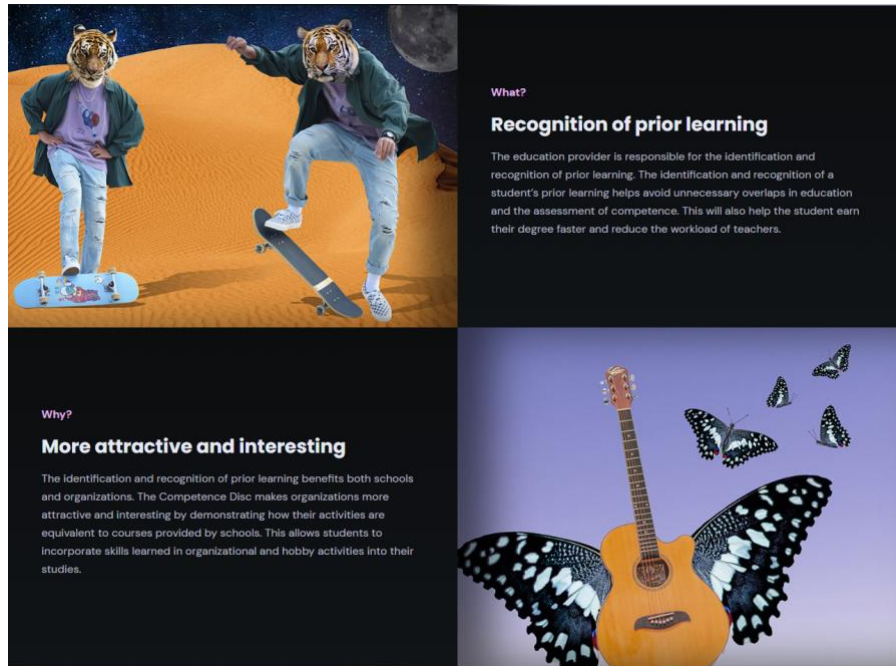


Figure 25. Part 2 and 3 design

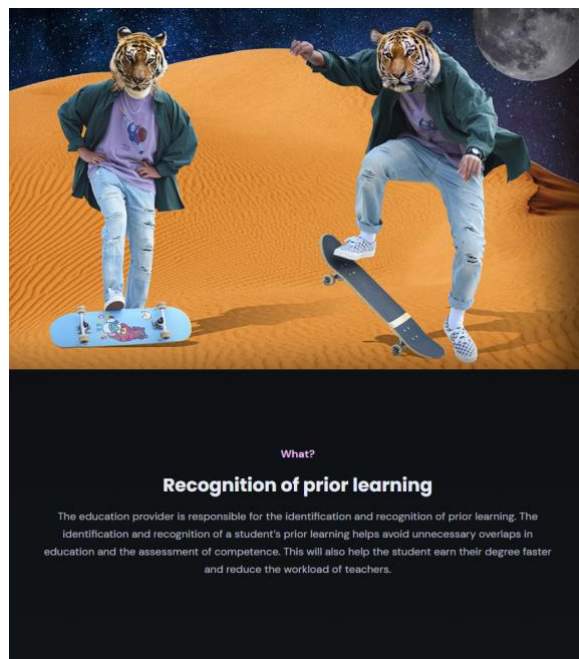


Figure 26. Part 2 and 3 mobile designs

Part 4 contains a picture and text regarding Duunikoutsu which is a mobile application helping young people to get a job. **Figure 27** shows the part 4 design on a desktop with a horizontal display and **Figure 28** shows the design for a mobile with a vertical display.

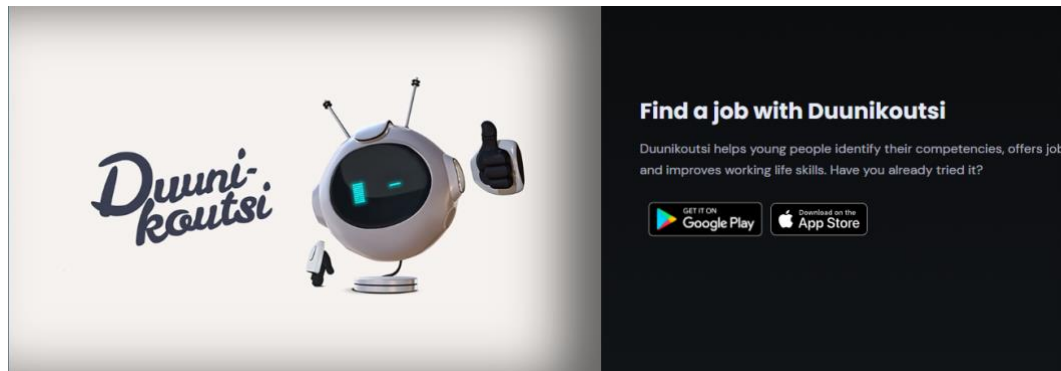


Figure 27. Part 4 design

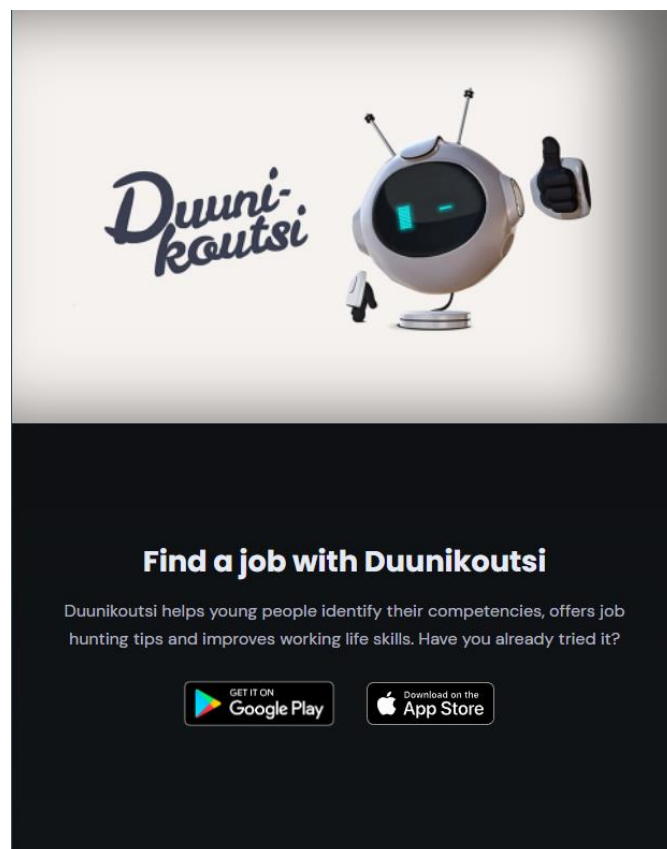


Figure 28. Part 4 mobile design

Part 5 also has one picture and text section which holds the navigators to each part of the web application. **Figure 29** and **Figure 30** show part 5 design for a desktop and a mobile.

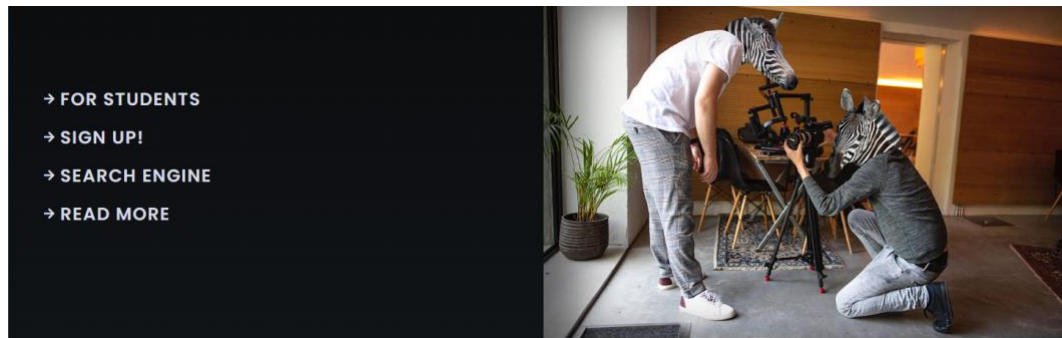


Figure 29. Part 5 design

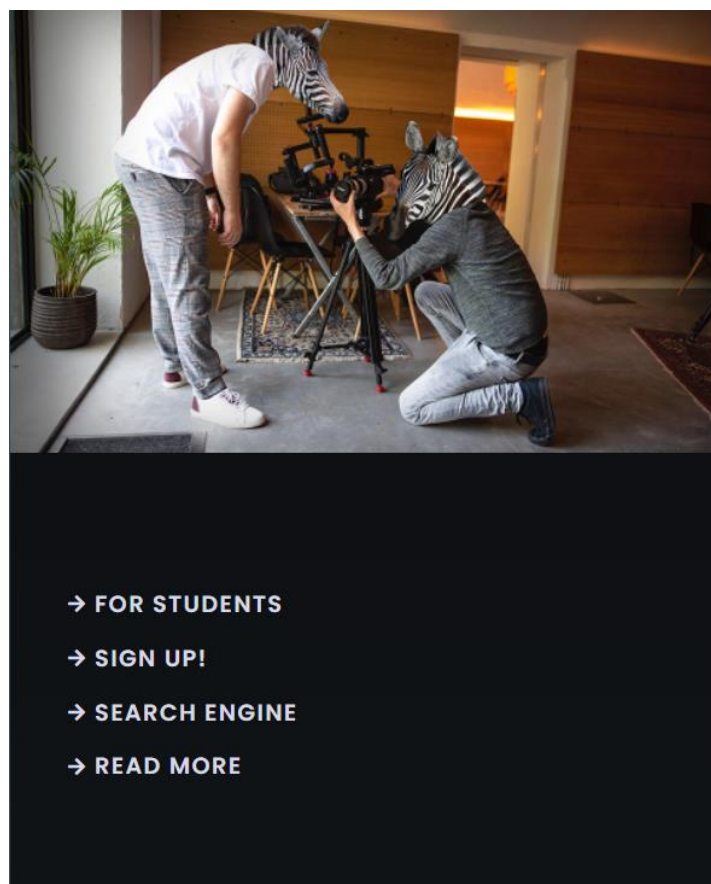


Figure 30. Part 5 mobile design

4.2.3 Sign-up Page

The sign-up page is where users come and leave their information for contacting and view the information of the website owner. **Figure 31** shows the design for part 5, the introduction page. **Figure 32** and **Figure 33** show the actual sign-up forms on a desktop and a mobile, respectively. Users will fill in the form to contact the website owner about what they want.

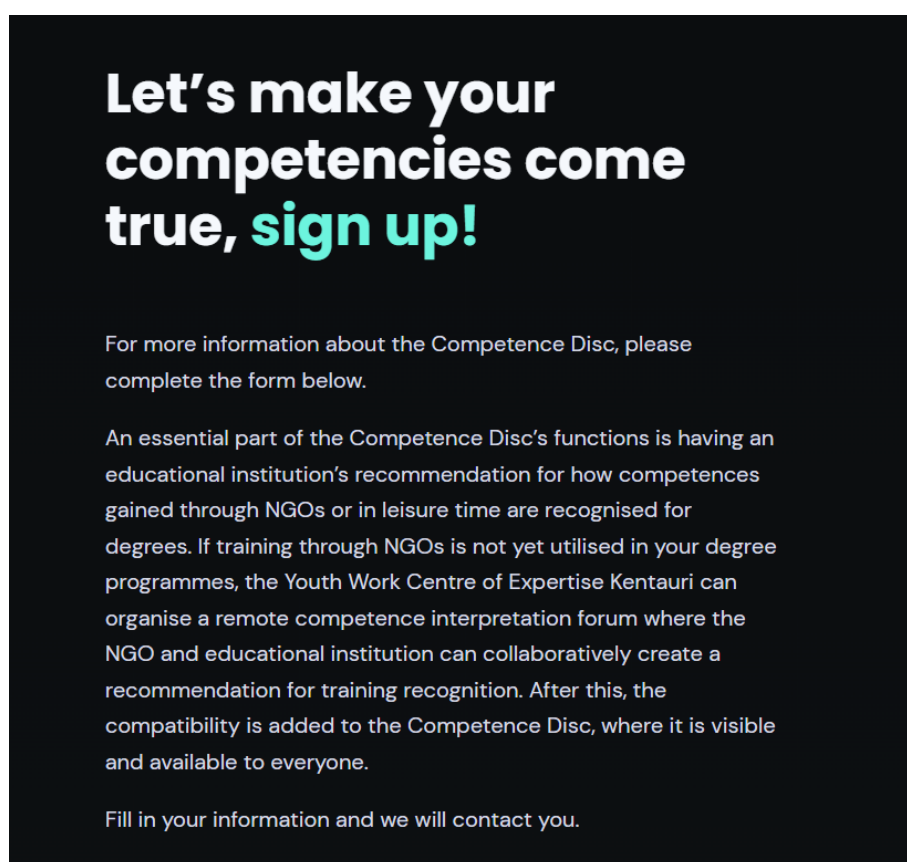


Figure 31. Sign up page design - part 1 on a mobile and a desktop

* All fields marked with an asterisk are required

* I am a
Select

* Reason for contacting us
Select

* Name
Name

* Email address
Email address

Phone

* Message

SUBMIT

Figure 32. Part 2 design

* All fields marked with an asterisk are required

* I am a
Select

* Reason for contacting us
Select

* Name
Name

* Email address
Email address

Phone

* Message

SUBMIT

Figure 33. Part 2 mobile design

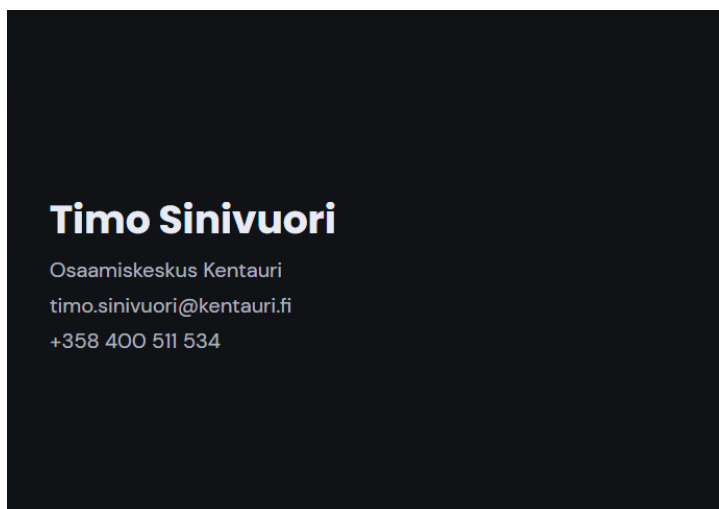


Figure 34. Part 3 design

Examples how the information of website owner is shown can be seen in **Figure 34** and **Figure 35**.

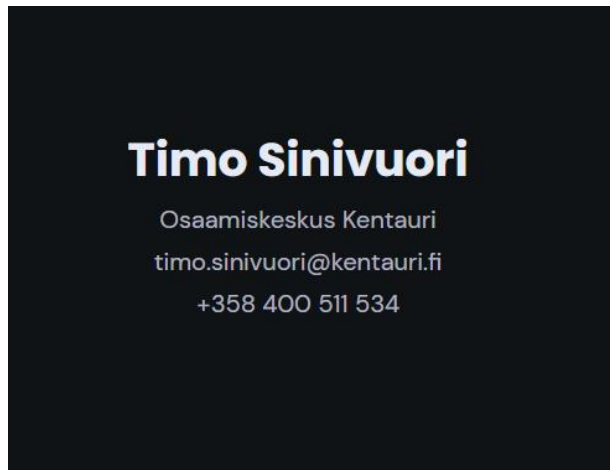


Figure 35. Part 3 mobile design

4.2.4 Competence and Degree Details Page

On competence and degree details page, users can view all vital information on the match between the degree and competence (see **Figure 36** and **Figure 37**). If they want to learn more, they can click the link “Read more on the school’s

website.” The image of the header will also be chosen randomly from a collection of images when the user reloads the page.

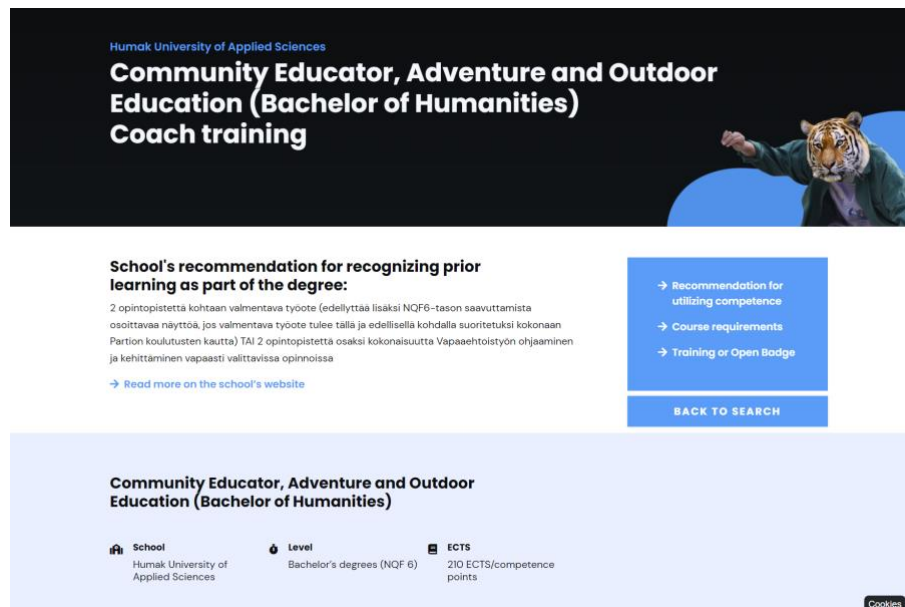


Figure 36. Competence and degree detail page design

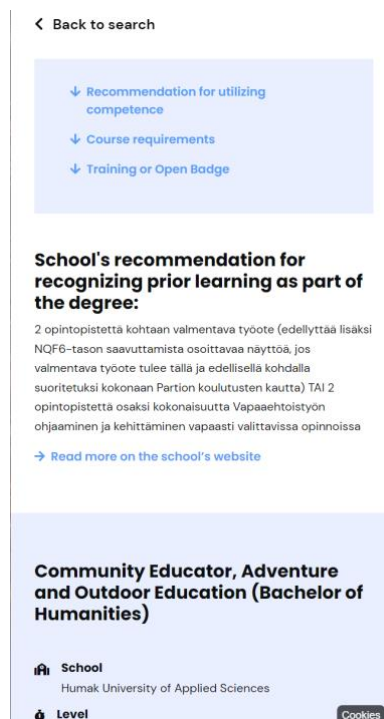


Figure 37. Competence and degree detail page mobile design

A box of navigation links (**Figure 38**) was also made for each part so that it can follow them while scrolling. No matter where the user is, they can click one of them and get right to the part they want instead of scrolling all around. On a mobile its position is fixed in one place due to limited space (**Figure 39**).

Navigation links box on a mobile

).

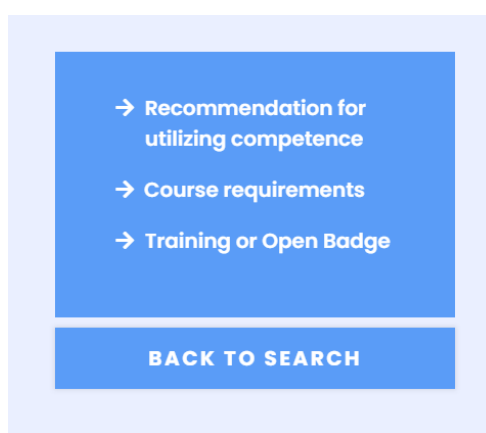


Figure 38. Navigation links box



Figure 39. Navigation links box on a mobile

4.2.5 Navigation

The navigation bar has 5 link items which are Home, Students, Schools, and Organizations, Sign up, logo and 1 language dropdown selection (**Figure 40**).

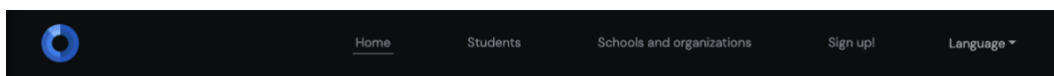


Figure 40. Navigation layout

Home and Logo direct to homepage, the rest direct to their own site accordingly. Whenever a site is visited, its name is highlighted with an underline.

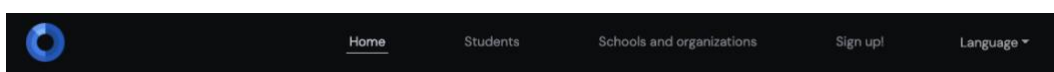


Figure 41. Hover effect

Also, when user hovers a link item, the color is a little brighter (**Figure 41**). Together it creates a nicer and more fluid UX.

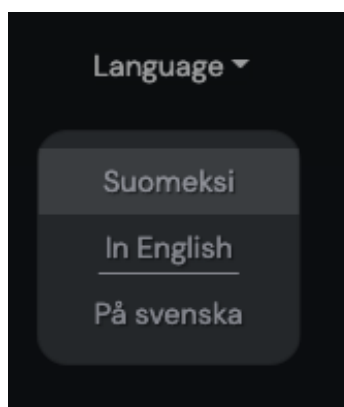


Figure 42. Language dropdown selection

For the language selection, it is presented as a dropdown (**Figure 42**). When being hovered, it has a slightly brighter background colour compared to others.



Figure 43. Mobile and tablet navigation

On a mobile and a tablet, it is made with a more compact design (**Figure 43**). All link items are packed into a hamburger menu.

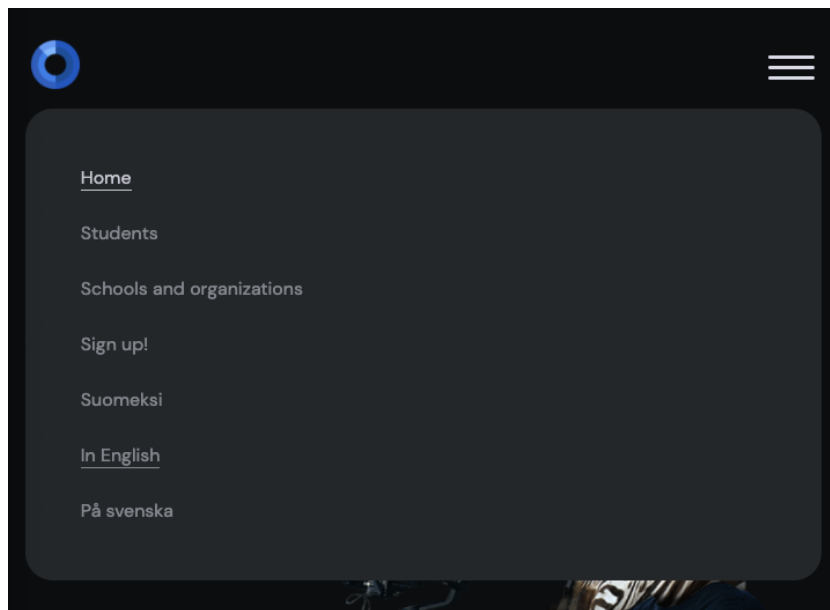


Figure 44. Mobile and tablet navigation when open

While in the open state, every link item will be organized into a vertical list with the same functions as in the desktop version (**Figure 44**).

4.2.6 Footer

The footer is divided into 3 sections: About, Navigation links and Logos of involved companies (**Figure 45**).

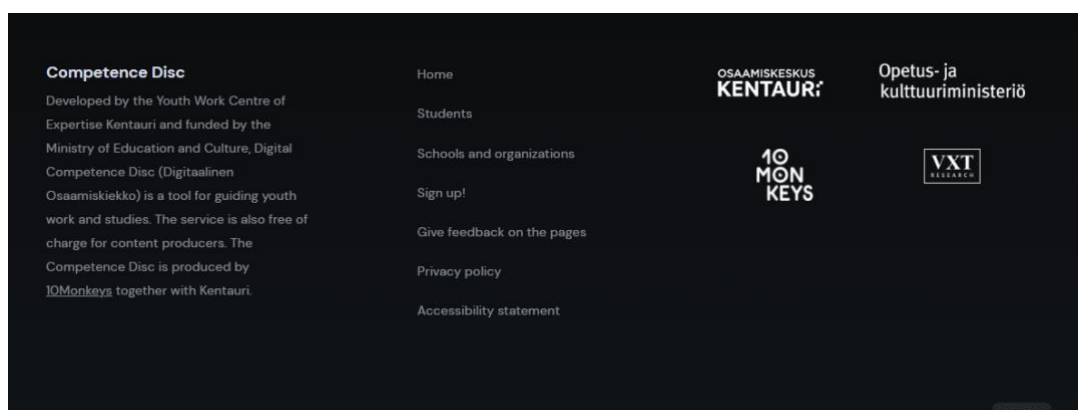


Figure 45. Footer design

The first column is about the information of the project. The second one is for navigators to each essential part. The last one is for all the parties involved in the development of the project.

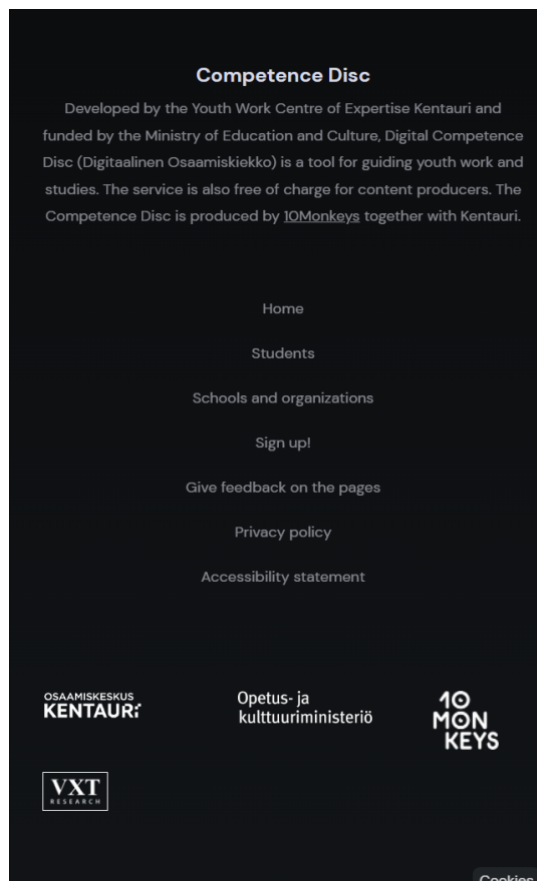


Figure 46. Footer design on a mobile

The flexbox is used for making the layout by separating the footer into left and right parts. The left block contains About, Navigation links section while the right one holds Logos. Inside the left block, flexbox is once again utilised to divide its container into 2 parts for About and Navigation links. By doing this, it will be easier to make the design responsive and maintainable.

5 IMPLEMENTATION

The application was created by shared components and pages. The code was written in a file with an extension of `.vue`. A Vue file consists of three parts: HTML, JavaScript and SCSS. Each part is structured in its own tag. For instance, HTML in a `<template>` tag, JavaScript in a `<script>` tag and SCSS in a `<style>` tag with an attribute of `lang` having the value of coder's choice, it is SCSS in this project.

5.1 Environment Setup

5.1.1 Installing and Configuring VSCode

VSCode is one of the most famous Integrated Development Environments (IDE) which gives a development environment with various tools, such as basic editing, IntelliSense which autocompletes what we are writing, code navigation, refactoring, debugging, and version control and is highly customizable [10].

The reason it was used in this project is because it has integrated a built-in terminal which can help run code right away without using any external terminal. Besides, it has the Git version management system which assists in adding, committing, and pushing files to the remote repo, switching branches.

Additionally, VSCode has such a big market of extensions where to find the suitable plugin. Some extensions used in this project are Vue VSCode Snippets, Vetur, ESLint, Better Comments, Bracket Pair Colorizer, Material Icon Theme.

5.1.2 Installing NodeJS

NodeJS is a JavaScript runtime based on Chrome's V8 JavaScript engine [11] and and a free, open-source server environment. It can run on famous platforms such as Windows, MacOS, Linux, Unix [12]. The reason it is used with Nuxt is that it

creates a web server for Nuxt components to run locally and get deployed to well-known hosting platforms, for instance Netlify.

5.1.3 Create Nuxt Application

To start a project template of Nuxt, we use the following command if yarn, npx or npm has been installed.

```
npx create-nuxt-app nuxt-app
```

Code Snippet 7. Creating a nuxt application

There will be no longer manual webpack, dependencies installation required. It does save a lot of time from setup to configuration.

Then we move to the project folder and run the project.

```
cd nuxt-app
```

```
npm run dev
```

Code Snippet 8. Running the application in development environment

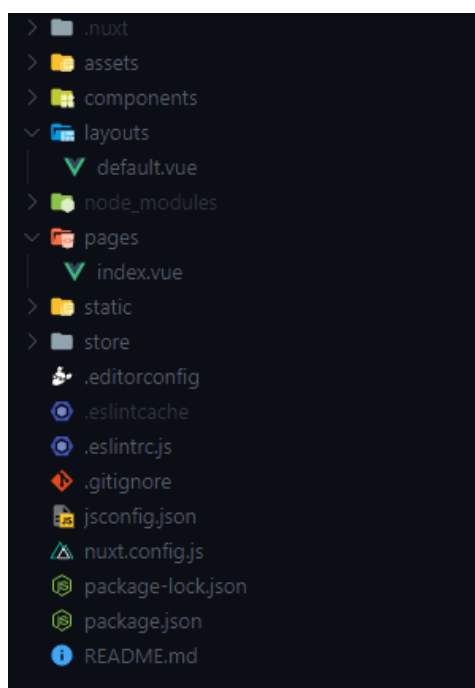


Figure 47. Project structure

After installation, we can see the project structure of a Nuxt application in **Figure 47**. The implementation can be started from `index.vue` file.

5.1.4 Dependencies Installation

A dependency is the code from third party which the application relies on [13]. From it, the app can have some external features without requiring the coder to rewrite everything from scratch. Below is a list of dependencies used in this project.

```
"dependencies": {
  "@nuxt/content": "^1.14.0",
  "@nuxtjs/axios": "^5.13.6",
  "bootstrap": "^4.6.0",
  "bootstrap-vue": "^2.21.2",
  "core-js": "^3.15.1",
  "moment": "^2.29.1",
  "nuxt": "^2.15.7"
},
"devDependencies": {
  "@babel/eslint-parser": "^7.14.7",
  "@nuxtjs/eslint-config": "^6.0.1",
  "@nuxtjs/eslint-module": "^3.0.2",
  "@nuxtjs/style-resources": "^1.2.1",
  "eslint": "^7.29.0",
  "eslint-plugin-nuxt": "^2.0.0",
  "eslint-plugin-vue": "^7.12.1",
  "fibers": "^5.0.0",
  "sass": "^1.32.12",
  "sass-loader": "^10.2.0"
}
```

Code Snippet 9. Dependencies

“devDependencies” comprise those necessary for local development and testing while “dependencies” are used in production [14].

These dependencies can be installed through the following command.

```
npm install "package-name"
```

Code Snippet 10. Install a dependency with npm

And for testing and local development only.

```
npm install --save-dev "package-name"
```

Code Snippet 11. Install a dependency for testing and local development

5.2 Architecture

The whole project is divided into frontend and backend parts. I focus mostly on the frontend while backend has been developed and maintained by my CTO and my colleague.

All pages in the frontend are served in pages directory. It comprises both Application Views and Routes. Nuxt does not only scan through all files ending with .vue but also generate the router configuration [15].



Figure 48. Pages folder

I use dynamic pages technique in this project because language preference will be fetched from a router parameter called id. That is the reason why `_siteId` folder is there to achieve this. Despite using dynamic pages, one index file has to be created in order to use `nuxt run generate` command to build the project

and use that for production stage. That must be done specifically only in this project as the project owner wants fi – Finnish language - will be the default site when user accesses the domain, which means it must be navigated automatically from “/” to “/fi/” by default. Without the index file (**Code Snippet 12**) and a router configuration (**Code Snippet 13**), Nuxt will not be able to fulfil this task.

```
<template>
  <p></p>
</template>
```

```
<script>
/**
 * "num run generate" won't work without this page.
 */
export default {
}
</script>
```

Code Snippet 12. Index.vue file

```
router: {
  // Extend the routes.
  // See: https://nuxtjs.org/docs/features/file-system-routing
  extendRoutes(routes, resolve) {
    const pagesToIndexList = ['index', 'schools-and-organizations', 'sign-up', 'students']

    for (const siteId in siteList) {
      for (const page of pagesToIndexList) {
        const name = siteId + '-' + page

        let path = '/' + siteId + '/' + page + '/'

        if (page === 'index') {
```



```

        path = '/' + siteId + '/'
    }

    routes.push({
      name,
      path,
      component: resolve(__dirname,
'pages/_siteId/' + page + '.vue')
    })
  }
}
}

```

Code Snippet 13. Router config option in nuxt.config.json

5.3 Shared Components

For every page, there is a navigation bar and a footer. Only the content will be rendered differently according to the topic of that page.

5.3.1 Navigation

The navigation is implemented in a file called Navigation.vue file. The navigation template is divided into two parts for responsiveness with respective styles: desktop and mobile. Each link item is put inside a `<nuxt-link>` wrapped in a `div` container so that it can be easily styled (**Code Snippet 14**). Each of them has a `to` attribute describing the route, for example: `"/students"`. The child of this tag is the content of the link.

```

<div>
  <nuxt-link :to="$urls.home">
    {{ content["navbar-home"] }}
  </nuxt-link>
</div>

```

Code Snippet 14. Example of one link item

The language dropdown (**Code Snippet 15**) is made with Vue Bootstrap component as `<b-dropdown>`.

```
<div class="language">
  <b-dropdown
    id="language-dropdown-btn"
    ref="language-dropdown-btn"
    :text="content.language"
    :class="languageCssClass"
  >
    <b-dropdown-item
      v-for="language in languagesList"
      :key="language.name"
      @click.native="reloadPage(language.path)"
    >
      <nuxt-link
        :to="getLanguageLink(language.path)" >
        {{ language.name }}
      </nuxt-link>
    </b-dropdown-item>
  </b-dropdown>
</div>
```

Code Snippet 15. Language dropdown for a desktop

Instead of adding option separately with a pure HTML, it can be done with an array (**Code Snippet 16**) using `v-for` (for loop) in `<b-dropdown-item>`.

```
languagesList: [
  { name: 'Suomeksi', path: '/fi' },
  { name: 'In English', path: '/en' },
  { name: 'På svenska', path: '/sv' }
]
```

Code Snippet 16. Language list

Every time user chooses a language, it will reload the page to apply the new language to the entire system (**Code Snippet 17**).

```
reloadPage (rootPath) {
    location.href = rootPath + this.urlPath
}
```

Code Snippet 17. Page reloading function

On a mobile and a tablet, all link items will be hidden and only be shown as modal if the user clicks the hamburger menu presented as three bars (**Code Snippet 18**).

```
<div
    class="mobile-button"

@click.prevent="toggleMobileMenuVisibilityClick()"
>
    <div class="bar" />
    <div class="bar" />
    <div class="bar" />
</div>

<div v-show="isMobileMenuVisible" class="mobile-
menu">
    <div>
        <nuxt-link
            class="blur-link"
            :to="$urls.home"
            @click.native="closeMenu()"
        >
            {{ content["navbar-home"] }}
        </nuxt-link>
    </div>
```

Code Snippet 18. Mobile nav

The modal is closed when user clicks the backdrop or item by the function inside the `mounted` function where all functions will be executed after component is mounted (**Code Snippet 19**).

```
mounted () {
  const path = this.$route.path
  this.chooseSiteName(path)
  if (this.$refs['language-dropdown-btn'] !==
undefined) {
    // Close mobile menu
    document.addEventListener('click', (ev) => {
      const collapse =
document.querySelector('.mobile-menu')
      const eventClass = ev.target.className
      // Check if the event target class name is not
'bar' either 'mobile-button'
      // then close the menu. Otherwise, every time
we click it will
      // open and close the menu immediately
      if (
        collapse &&
        !collapse.contains(ev.target) &&
        eventClass !== 'mobile-button' &&
        eventClass !== 'bar'
      ) {
        this.isMobileMenuVisible = false
      }
    })
  }
},
```

Code Snippet 19. Closing language dropdown function

5.3.2 Footer

Basically, it is implemented in the same way as in the navigation, in which a link embedded inside a `div` uses `<nuxt-link>` as a Nuxt component.

```
<div>
  <nuxt-link :to="$urls.home">
    {{ content["home"] }}
  </nuxt-link>
</div>
```

Code Snippet 20. Example of a link in footer

As for an external link, `<a>` tag is utilised instead of `<nuxt-link>` as `<nuxt-link>` is meant to be used for navigating between pages [16].

```
<div>
  <a
    target="_blank"
    href="https://link.webropol-surveys.com/Participation/Public/43bf4835-3f8d-4250-8bee-888d3719be70?displayId=Fin2412610"
  >
    {{ content["giveFeedback"] }}
  </a>
</div>
```

Code Snippet 21. Example of an external link

Besides, when embedding a link into an image, I simply just put an `` tag inside an `<a>` tag.

```
<div class="logo monkeys">
```

```

        <a target="_blank"
href="https://www.10monkeysdigital.com/">
        
        </a>
    </div>

```

Code Snippet 22. Example of an image link

5.3.3 Default Layout

In Nuxt, it has a component called `default.vue` in `layouts` folder where every page shares the same layout and data. With this, coders can create different layouts and decide what pages should have the same configurations and vice versa.

```

<template>
  <div id="layout">
    <div class="layout-top">
      <navigation />
      <Nuxt />
    </div>

    <div class="layout-bottom">
      <main-footer />
    </div>
  </div>
</template>

```

Code Snippet 23. Default template

As in **Code Snippet 23**, the layout is divided into two parts: top and bottom. The bottom contains only footer while the top holds the rest including navigation and

main content. `<Nuxt />` is the component which is rendered differently depending on what coders want.

```
<script>
export default {
  head () {
    return {
      htmlAttrs: {
        lang: this.$config.site &&
this.$config.site.localeId.substr(0, 2)
      }
    }
  }
}
</script>
```

Code Snippet 24. JS part

```
{
  "fi": {
    "id": "fi",
    "localeId": "fi_fi",
    "contentId": "fi"
  },
  "sv": {
    "id": "sv",
    "localeId": "sv_se",
    "contentId": "sv"
  },
  "en": {
    "id": "en",
    "localeId": "en_us",
    "contentId": "en"
  }
}
```

Code Snippet 25. Site list configuration

In **Code Snippet 24**, the page is set to its own language depending on what user selects. The algorithm is that it takes only the first two letters from the "localeId" of the chosen language (**Code Snippet 25**).

```
.page-container {  
  width: $content-width;  
  max-width: 90vw;  
  margin: 3rem auto 6rem auto;  
}
```

Code Snippet 26. Default SCSS

In every part of the page, this class of CSS (**Code Snippet 26**) is used to guarantee that in all devices with varied sizes, the content will always stay in the center of the screen without being stretched on an exceptionally large screen.

5.3.4 Content Fetching

With Nuxt Content, it is easy to fetch the content conditionally, for example, language. All the content is put inside the content folder.

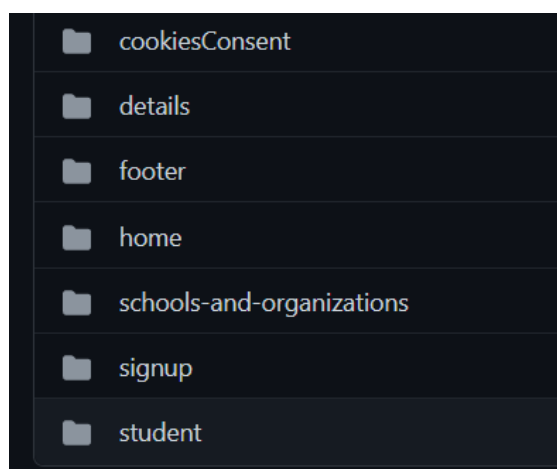


Figure 49. Content folder

Folders are divided according to sites and the `cookiesConsent` folder is meant for the cookies consent popup content (**Figure 49**). In each folder, files are named after the folder with an extension of `yml`, which stands for YAML file and postfix as chosen language (**Figure 50**). Thus, when a page is loaded, it will check for the site id which is `en`, either `fi` or `sv` and fetch the corresponding content.

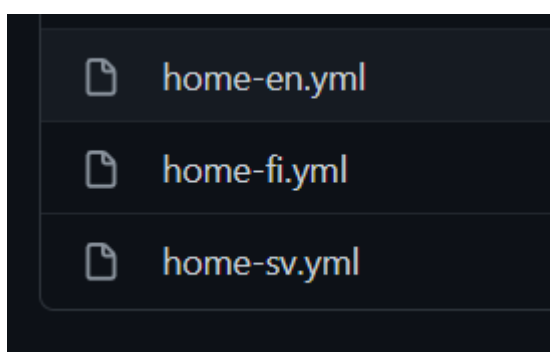


Figure 50. Yml files

There are two ways of fetching data asynchronously which are `asyncData` and `fetch hooks`.

`asyncData` is available only in the `page` component. It does not show a loading placeholder while rendered on the client-side but suspends route navigation until action is resolved, otherwise, shows an error page if it does not succeed [17]. Instead of `this`, only `context` parameter can be accessed in `asyncData` method. It can be torn down into smaller things by destructuring assignment of JavaScript such as `$content`, `$config`. The data returned in `asyncData` function will then be merged into `data()` method. Therefore, there is no need to initialize the variable beforehand.

```

async asyncData ({ $content, $config: { siteId } }) {
  const content = await $content('home/home-' +
siteId).fetch()
  return {
    content
  }
}

```

```
},
```

Code Snippet 27. Page data fetching

`$content` is the function to access content folder. The syntax is the following.

```
$content(file_path_inside_content_folder).fetch()
```

Code Snippet 28. Fetch content from content folder

The returned data is a list of objects including key and value. The data can be retrieved by the following.

```
<div>
  <nuxt-link :to="$urls.home">
    {{ content["navbar-home"] }}
  </nuxt-link>
</div>
```

Code Snippet 29. Retrieving data

```
navbar-home: Home
navbar-student: Students
navbar-school: Schools and organizations
navbar-signup: Sign up!
language: Language
```

Code Snippet 30. Example of home-en.yml file

`fetch` can be used in any component and supplies many ways to access rendering loading states (only in client-side) and errors if any [17].

```
async fetch () {
  const content = await this.$content(
    'home/home-' + this.$config.siteId
  ).fetch()
  this.content = content
},
fetchOnServer: true,
```

Code Snippet 31. Component data fetching

As it shows, the inner things are quite the same as in `asyncData`. Every `async` (asynchronous) action can be used in this function including HTTP request, API calls. With `fetch`, the coder can check the state of fetching whether it finishes or not, so that a loading activity indicator can be shown through `$fetchState.pending`. In case of errors, `$fetchState.error` is an effective way to start looking into and display the error message if needed [18].

`fetchOnServer` is one of the options that makes `fetch()` somehow like `asyncData()` because as its name states, it can be fetched on server-side instead of client-side [18].

To sum up, there are many options and features that `fetch()` and `asyncData()` expose may be checked on their websites.

5.3.5 Cookie Consent

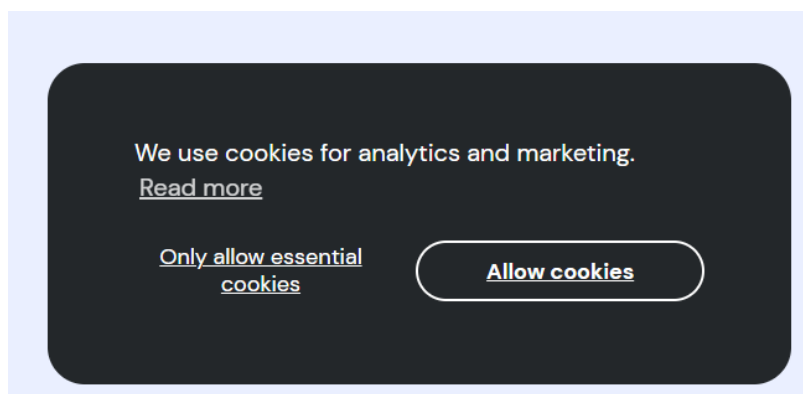


Figure 51. Cookie consent design

Cookie consent is made as a plugin to the Nuxt project. That is why it is in the plugin folder and imported in `nuxt.config.json` file. It is used for analyzing data.

```
window.cookieconsent.initialise({
  palette: {
    popup: {
```

```

        background: '#23272A',
        text: '#ffffff',
        outline: 'none'
    },
    button: {
        background: 'transparent',
        text: '#ffffff',
        border: '#ffffff'
    }
},
position: 'bottom-right',
type: 'opt-out',
content: {
    message: content.message,
    allow: content.allow,
    deny: content.deny,
    link: content.link,
    policy: content.policy,
    href: 'https://www.partio.fi/suomen-
partiolaiset/tietoa-meista/henkilotietojen-
kasittely/tietosuojaseloste/'
},
law: {
    countryCode: 'FI'
},
},

```

Code Snippet 32. Customize the look and feel of cookie consent

Here is where I customize the appearance of the cookie consent, from the look to the content inside.

```

const content = await app
    .$content('cookiesConsent/cookies-' +
app.$config.siteId)
    .fetch()

```

Code Snippet 33. Language of the cookie consent

The language and content of it is also fetched from the `siteId` as `en`, `fi` or `sv` (**Code Snippet 33**) as it is loaded.

5.4 Pages

This section is about pages implementation of the web application.

5.4.1 Homepage

This page is divided into header, main.

```
<header>
  <div class="content" :style="backgroundCSS">
    <div class="left">
      <h1>{{ content["header-heading"] }}</h1>
      <p>
        {{ content["header-content"] }}
      </p>

      <search-form />
    </div>
    <div class="right">
      
    </div>
  </div>
</header>
```

Code Snippet 34. Header implementation

In the header, the search form is implemented so that the user will enter the university name and bachelor's degree name they follow then the program will produce some courses of an organization that can be replaced with. The app also

works backwards when the user has credits from some courses and want to replace some degree in a university of his or her choice.

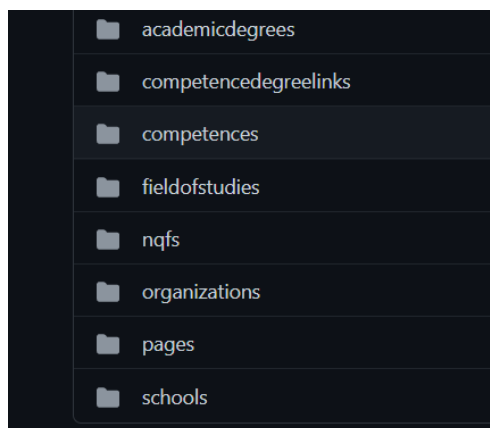


Figure 52. Inside data folder

It fetches data from a data folder (**Figure 52**) in static directory as JSON files (**Figure 53**).



Figure 53. Inside one of the folders

In order to retrieve the data, functions (**Code Snippet 35**, **Code Snippet 36**) are used. They can be put in `asyncData` or `fetch` function depending on where they are utilised.

```
import {  
  getCreditingInfosForCompetence,  
  getCreditingInfosForDegree,  
  getNqfs,
```

```

    getOrganizations,
    getSchools
  } from '../assets/js/old-api/Api'

```

Code Snippet 35. Importing functions of retrieving data

```

const schoolsList = await getSchools()
  const organizationsList = await getOrganizations()
  const nqfs = await getNqfs()

```

Code Snippet 36. Retrieve data

In the main section, there are three containers and each of them has an image on one side and text on the other side.

```

<page-section
  :is-header="false"
  class="part1"
  img-url="/images/osamiskiekko-broccoli.jpg"
  image-side="left"
  :image-alt="content['alt-osamiskiekko-
broccoli']"
  >
  <h2>
    <p class="sub-heading">
      {{ content["body-part1-h1"] }}
    </p>
    {{ content["body-part1-h2"] }}
  </h2>

  <p class="blur-text">
    {{ content["body-part1-p1"] }}
  </p>
</page-section>

```

Code Snippet 37. Example of one container implementation

```

<div
  class="page-section"
  :class="{
    'right-img-layout': isImgRight,
    'align-items-center': isHeader
  }"
>
  <div class="left">
    

    <div class="shadow-img" :class="{ 'shadow-img-
header': isHeader }" />
  </div>

  <div class="right">
    <div>
      <slot />
    </div>
  </div>
</div>

```

Code Snippet 38. Page section implementation

`<slot />` is where everything between `<page-section>` and `</page-section>` is rendered. Some containers may have different content so that is why I do not want to make it fixed here.

In the last part of the main section, there is one area called category where all logos of the page are shown. By means of which, user can navigate to their desired part right away with just a click.

```

<div class="category">
  <div>
    <category-item
      v-for="category in categoriesList"

```



```

        :key="category.title"
        :item-data="category"
    />
</div>
</div>

```

Code Snippet 39. Category section implementation

```

<div :style="cssStyle" class="category-item">
  <nuxt-link :to="itemData.link">
    <div class="avatar">
      
    </div>

    <h3>
      {{ itemData.title }}
    </h3>
  </nuxt-link>
</div>

```

Code Snippet 40. Category item implementation

In `categoriesList` there is an array of objects containing some values needed to fetch the category-item component (**Code Snippet 41**).

```

{
  imgUrl: '/images/icon_hakukone.png',
  title: this.content['category-1'],
  link: this.$urls.searchForm,
  hoverColor:
    process.client &&
    // Retrieve CSS variables
    window
    .getComputedStyle(document.documentElement)
      .getPropertyValue('--blusher-pink-
color'),

```

```

        content:
          'Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor.'
      },

```

Code Snippet 41. Category data object

5.4.2 Students, Schools and Organizations Page

These two pages share the same layout but only different regarding the content, so I make a component that they can share with.

This component has five parts, and they are named from one to five. Each part is put inside its own file, for example, Part1.vue.

In part 1, it separates text and picture parts into two sides with the help of flexbox. In the bottom, user can go back to homepage and use the searching tool by clicking the button.

```

<template>
  <page-section
    :image-alt="imageAlt"
    image-side="right"
    :img-url="imgUrl"
    :is-header="true"
  >
    <h1 class="mb-4">
      {{ heading }}
    </h1>
    <p class="blur-text">
      {{ text }}
    </p>
    <div class="button">
      <button class="btn" :class="btnClass"
        @click="$goto('searchForm')">
        {{ buttonName }}
      </button>
    </div>
  </page-section>
</template>

```

```

        </button>
    </div>
</page-section>
</template>

```

Code Snippet 42. Part 1 implementation

In this component, we check if the picture is situated on the right or left through `image-side` prop, if this section is part 1 through `is-header` prop so that the text on left will be aligned vertically center, also get the image path from `img-url` prop and alt text (in case the image is not shown, that text will be shown instead) from `image-alt`.

```

<template>
  <!-- This is for part 2 and part 3 -->
  <page-section :image-alt="imageAlt" :image-
side="imageSide" :img-url="imgUrl">
    <h2>
      <p class="sub-heading" :style="subHeadingStyle">
        {{ subHeading }}
      </p>
      {{ heading }}
    </h2>
    <p class="blur-text">
      {{ text }}
    </p>
  </page-section>
</template>

```

Code Snippet 43. Part 2 and 3 HTML implementation

Parts 2 and 3 reuse the `<page-section>` component in order not to repeat anything which is easier to maintain and develop further because everything has been implemented in the `<page-section>` component.

```

<template>
  <page-section
    image-alt="Duunikoutsi"
    class="part4"
    img-url="/images/duunikoutsi.jpeg"
  >
  <h2>{{ heading }}</h2>

  <p class="blur-text">
    {{ text }}
  </p>

  <div class="download-imgs">
    <div>
      <a
        aria-label="xxxMartinxxx - Go to Google Play
store"
        target="_blank"

href="https://play.google.com/store/apps/details?id=fi.
tat.duunikoutsi&hl=fi"
      >
        
      </a>
    </div>

    <div>
      <a
        aria-label="xxxMartinxxx - Go to Apple app
store"
        target="_blank"

href="https://apps.apple.com/fi/app/duunikoutsi/id14629
03853"
      >

```

```

        
    </a>
</div>
</div>
</page-section>
</template>

```

Code Snippet 44. Part 4 HTML implementation

In part 4, both pages have the same image and text talking about the Duunikouts app which can help users get a job and how to download it.

```

<template>
  <page-section :image-alt="imageAlt" :img-url="imgUrl"
image-side="right">
    <navigation-links-box :is-student-page="isStudent"
/>
  </page-section>
</template>

```

Code Snippet 45. Part 5 HTML implementation

Part 5 includes a picture and a group of navigation links to other pages. These links are implemented inside a component called `<navigation-links-box>`. In which, we will have four links. The first one will be rendered as “For students” if user is in Schools-and-Organization page and vice versa (**Code Snippet 46**).

```

<template>
  <ul class="navigation-links-box">
    <li v-for="itemData in data" :key="itemData.text">
      <arrow-link
        :class="{

```

```

        'student-page': isStudentPage,
        'organization-page': !isStudentPage
    }"
    :to="itemData.link"
    :is-external-link="itemData.isExternalLink"
  >
    {{ itemData.text }}
  </arrow-link>
</li>
</ul>
</template>

```

Code Snippet 46. <navigation-links-box> HTML implementation

Each of these links is taken care by ArrowLink component which will attach an arrow in the beginning (**Code Snippet 47**).

```

<template>
  <div class="arrow-link d-flex">
    <div class="left">
      <!-- is-right-arrow: arrow pointing right -->
      <div v-show="!isDegreePage" :class="{ 'not-degree-page': !isDegreePage }">
        <i class="fas fa-arrow-right" />
      </div>

      <!-- is-down-arrow: arrow pointing down -->
      <div v-show="isDegreePage" :class="{ 'degree-page': isDegreePage }">
        <i class="fas fa-arrow-right" />
        <i class="fas fa-arrow-down" />
      </div>
    </div>

    <div class="right">
      <nuxt-link v-if="!isExternalLink" :to="to">

```

```

        <slot />
    </nuxt-link>

    <a v-else target="_blank" :href="to"><slot /></a>
</div>
</div>
</template>

```

Code Snippet 47. <arrow-link> HTML implementation

5.4.3 Sign-up Page

This page includes a bunch of texts and a form for the user to stay connected with the website owner. The last part of the page is where the contact information of the website owner lies.

```

<div class="sign-up">
  <div class="content">
    <h1 v-html="headingHtml" />

    <template v-if="submitted">
      <p>
        {{ content["thank-you"] }}
      </p>
    </template>

    <template v-else>
      <div class="mt-5" v-html="content['normal-
text']" />

      <p>* {{ content["special-text"] }}</p>
    </template>
  </div>
</div>

```

Code Snippet 48. Part 1 HTML implementation

Firstly, there are texts about the main content of the site. When user finishes the form, they will be replaced with a thank-you text in one's own language.

```

<b-form-group :label="content['phone-label']" label-
for="phone">
    <b-form-input id="phone" v-
model="form.phone" type="text" />
</b-form-group>

```

Code Snippet 49. One example of form HTML implementation

Every form item is done by using `<b-form-group>` and `<b-form-input>` element of Vue Bootstrap, which is more convenient and requires less effort to make the form more structured. Its goal is to associate form controls with a legend or label as well as supply help text, invalid/ valid feedback text also visual contextual state feedback [19].

```

// Set post data.
    // Note that if devMode is true, the sender
    (this.form.email) will receive the email.
    const postData = new URLSearchParams({
        role: this.form.role,
        subject: this.form.subject,
        name: this.form.name,
        email: this.form.email,
        phone: this.form.phone,
        message: this.form.message,
        devMode: this.$config.contact.devMode ? '1' :
'0'
    })
    await fetch(this.$config.contact.url, {
        method: 'post',
        body: postData,
        headers: {
            'Content-Type': 'application/x-www-form-
urlencoded'
        }
    })
    .then((response) => {

```



```

    if (response.ok === false) {
        throw new Error(response.statusText)
    }
    if (process.client) {
        window.scrollTo(0, 0)
    }
    this.submitting = false
    this.submitted = true
  })
  .catch(() => {
    this.submitting = false
    alert('Unknown Error')
  })

```

Code Snippet 50. Submit data

When the user submits the form, the data will be collected and turned into URLSearchParams object so as to be put in the body of the request (**Code Snippet 50**). When the request is successful, the page will scroll itself to the top so that user can see the thank you message.

```

<footer>
  <div>
    <contact-info
      v-for="contact in contactInfoList"
      :key="contact.name"
      :item-data="contact"
    />
  </div>
</footer>

```

Code Snippet 51. Part 3 HTML implementation

In **Code Snippet 51**, I use an array of contact information objects in which it contains name, company, email, and phone. Thus, it will be rendered dynamically and whenever I want to remove or add another person, I just need to go to that

array and modify it. In **Code Snippet 52**, I receive contact data as props from sign up page, style them and display them properly.

```
<div class="contact-info">
  <h2>{{ itemData.name }}</h2>

  <p class="blur-text">
    {{ itemData.company }}
  </p>

  <a :href="mailtoLink" class="blur-text">
    {{ itemData.email }}
  </a>

  <br>

  <a :href="phoneToLink" class="blur-text">
    {{ itemData.phone }}
  </a>
</div>
```

Code Snippet 52. ContactInfo component HTML implementation

5.4.4 Competence and Degree Details Page

This page is about the information of the match between the school and organization, for example, the description of courses. The data are fetched from API endpoints of the backend (**Code Snippet 53**). From which, we look for school, organization, and their data to show (**Code Snippet 54**).

```
const { linkId } = params
  const degreesList = await
getCompetenceDegreeLink(linkId)
  const degree = degreesList[0]
  const schoolsList = await getSchools()
```

```

const nqfsList = await getNqfs()
const organizationsList = await getOrganizations()

```

Code Snippet 53. Fetch data from backend API

```

const school = schoolsList.find(
  school => school.id ===
degree.academicdegree.school
)
const organization = organizationsList.find(
  organization => organization.id ===
degree.competence.organization
)
const academicLevel = nqfsList.find(
  item => item.id === degree.academicdegree.nqf
)
const competenceLevel = nqfsList.find(
  item => item.id === degree.competence.nqf
)

```

Code Snippet 54. Find proper school, organization, and their level

```

detailsDescription () {
  return this.competenceDegree &&
this.competenceDegree[this.description]
},
academicDegreeName () {
  return (
    this.competenceDegree &&
this.competenceDegree.academicdegree[this.name]
  )
},
competenceName () {
  return (
    this.competenceDegree &&
this.competenceDegree.competence[this.name]
  )
},

```

```

    schoolName () {
        return this.school && this.school[this.name]
    },
    organizationName () {
        return this.organization &&
this.organization[this.name]
    },
    academicDegreeDescription () {
        return (
            this.competenceDegree &&
this.competenceDegree.academicdegree[this.description]
        )
    },
    competenceDegreeDescription () {
        return (
            this.competenceDegree &&
this.competenceDegree.competence[this.description]
        )
    },

```

Code Snippet 55. Data computed properties

I make everything as computed properties so they can be easily retrieved in many places. Otherwise, I have to write such a long code and perhaps repeat myself, which is not a clean code.

```

mounted () {
    this.part3List[0].heading = this.content.institute
    this.part3List[1].heading = this.content.level
    this.part3List[2].heading = this.content.credits
    this.part3List[0].text = this.schoolName
    this.part3List[1].text =
this.levelInformation(this.academicLevel)
    this.part3List[2].text = this.creditInformation(

```

```
        this.competenceDegree.academicdegree
    )
    this.part4List[0].heading = this.content['training-
open-badge']
    this.part4List[1].heading = this.content.level
    this.part4List[2].heading = this.content.credits
    this.part4List[0].text = this.organizationName
    this.part4List[1].text =
this.levelInformation(this.competenceLevel)
    this.part4List[2].text = this.creditInformation(
        this.competenceDegree.competence
    )
},
```

Code Snippet 56. Initialize data with corresponding parts

When the page finishes loading, all data will be given to corresponding parts and rendered to the user.

6 DEPLOYMENT AND TESTING

When finished, the project was deployed to Netlify. The deployment is explained in this chapter.

6.1 Deployment

Our team choose Netlify to deploy the web application because it is easy to use and powerful for SPAs.

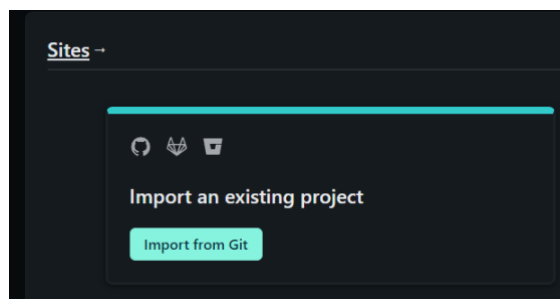


Figure 54. Import project

Firstly, the project was imported through GitHub.

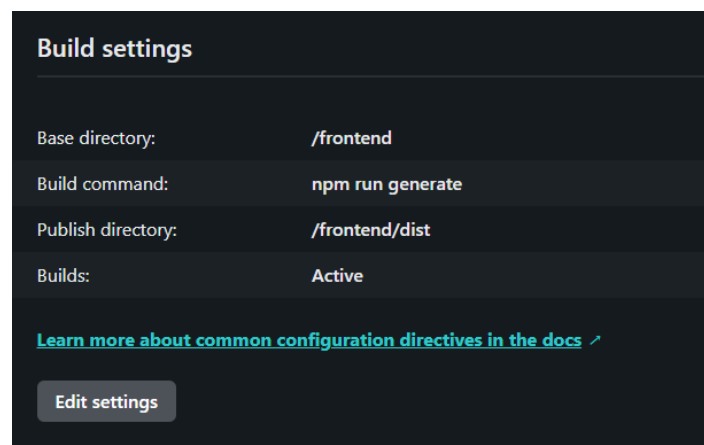


Figure 55. Edit build settings

After I logged in to Netlify, the project was imported from GitHub. Firstly, the build settings would be configured. Because both frontend and backend were deployed, a base directory and a public directory had to be specified so that Netlify could

detect where the build directory was. The build command was where I put the command to build the project.

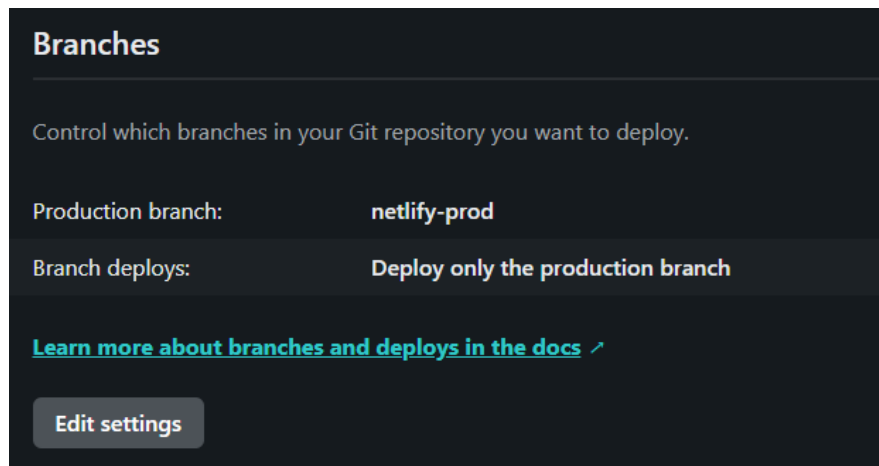


Figure 56. Branches settings

Figure 56 was where the production branch was set so whenever this branch was updated, Netlify would automatically run the build command and deploy the recent version.

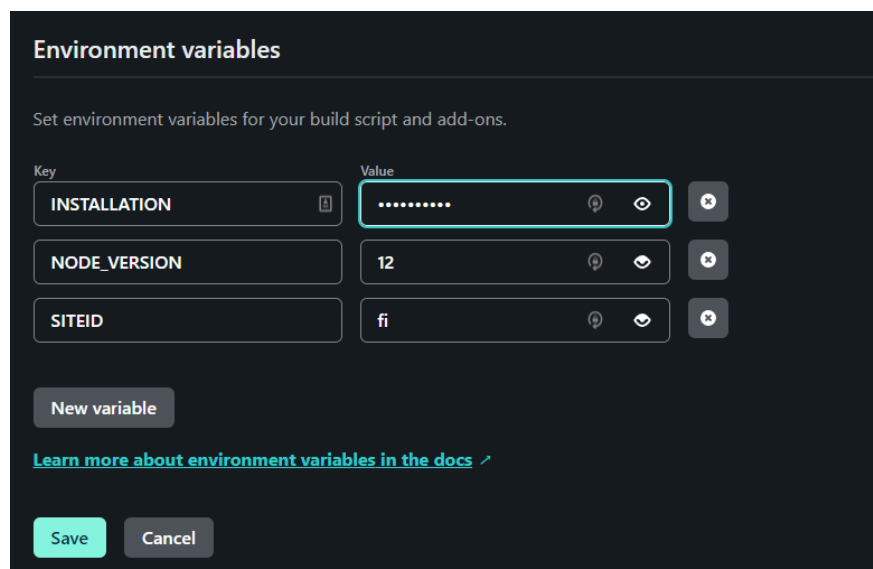


Figure 57. Environment variable settings

In order for the app to work well with Netlify, the node version had to be set to 12. By default, the language was Finnish thus the SITEID was set to fi.

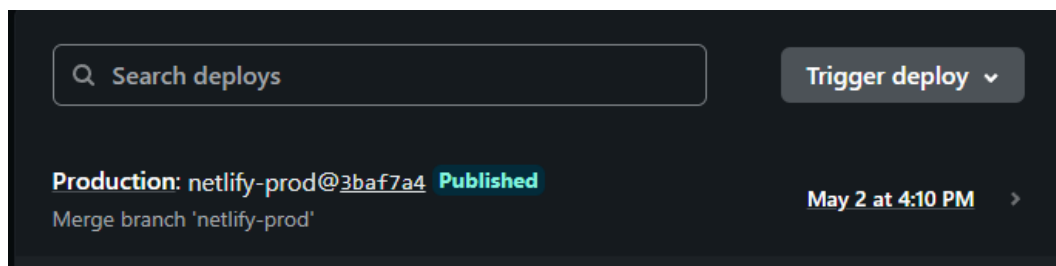


Figure 58. Deployment

When the latest change was pushed to the production branch on GitHub, Netlify would be notified, build, and deploy the latest version.

6.2 Testing

This part focuses on test cases and test result analysis. It was tested with all browsers such as Edge, Chrome, Firefox, Safari to make sure everything was working synchronously throughout all platforms and with screen reader to ensure accessibility. It was also SEO friendly because all content of the project was loaded in the server (SSR – server-side rendering) and sent to the client browser, which made Google algorithm happy and put the page on top of searching list.

6.2.1 Change the Language

- **Testing step**

Click the language dropdown and select the desired language

- **Result**

All the content of the page is shown in that language.

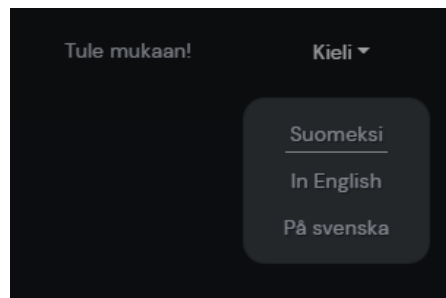


Figure 59. Before language change

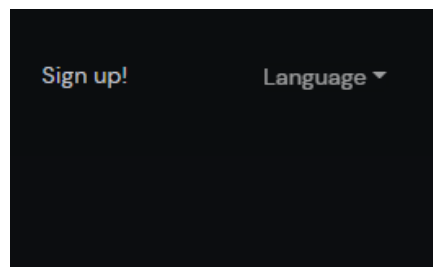


Figure 60. After language change

6.2.2 View the Details of the Match

- **Testing step**

After choosing a school and degree or organization and competence, the list of matches will be shown. Click one of those to get us to the detail page.

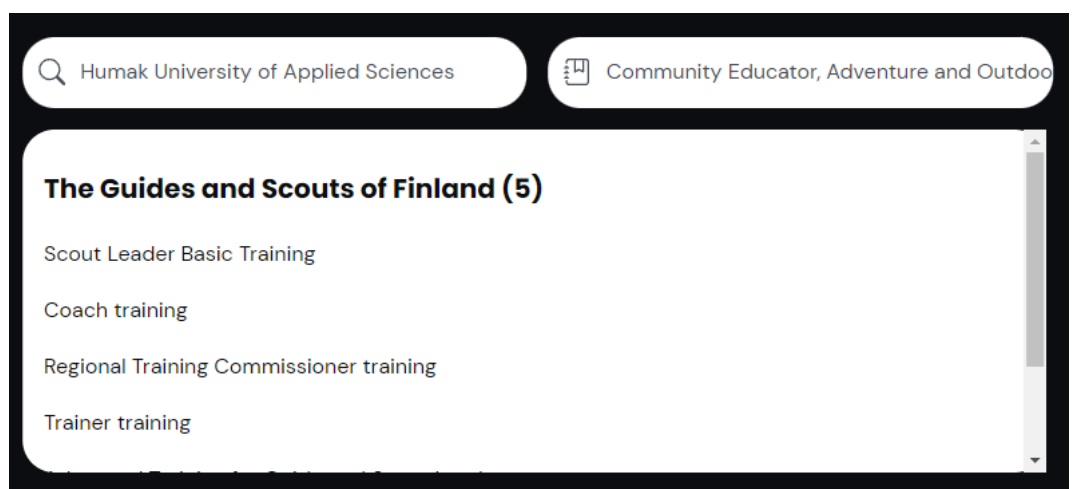


Figure 61. Before choosing the match

- Result

The detail page is shown after the click.

[← Back to search](#)

[↓ Recommendation for utilizing competence](#)

[↓ Course requirements](#)

[↓ Training or Open Badge](#)

School's recommendation for recognizing prior learning as part of the degree:

5 ECTS credits distributed in the manner determined annually in the study modules "Promotion of agency" and "Coaching-based approach" (usually substituting the 5 ECTS credits of practical training included in these modules) OR 5 ECTS credits as part of the module "Guidance and development in voluntary work" in the optional studies.

[→ Read more on the school's website](#)

Figure 62. After choosing one of the matches

6.2.3 Send the User's Data Through Contact Form

- **Testing step**

First, the user fills in all necessary information and submits it.

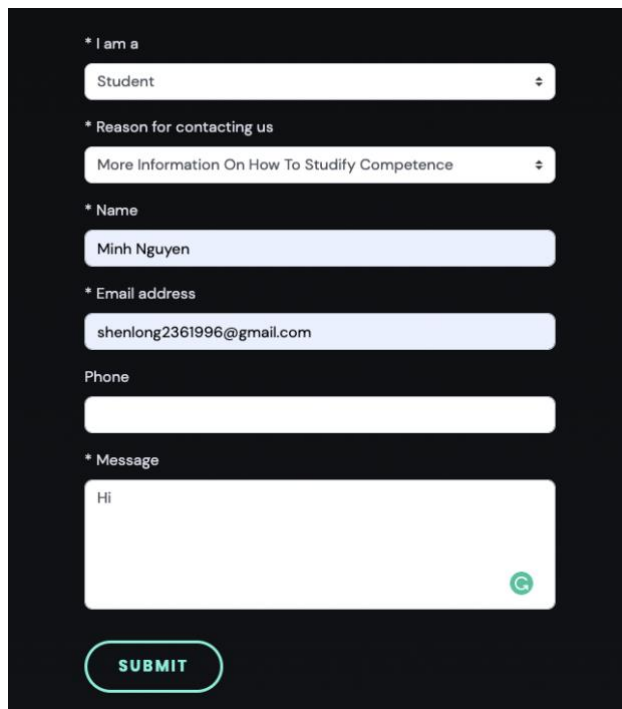


Figure 63. The user fills in the information and submit

- **Result**

After submission, the user will see a thank you message.

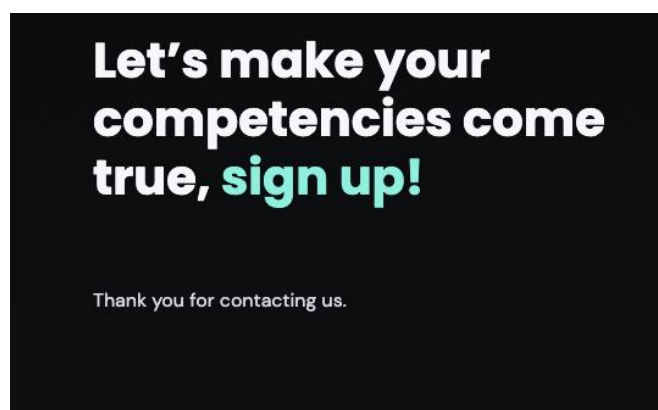


Figure 64. A thank you message

And the website owner will receive user's information and may contact them. In this case, in the development stage, the sender is the receiver but in the production stage, the receiver is the website owner.

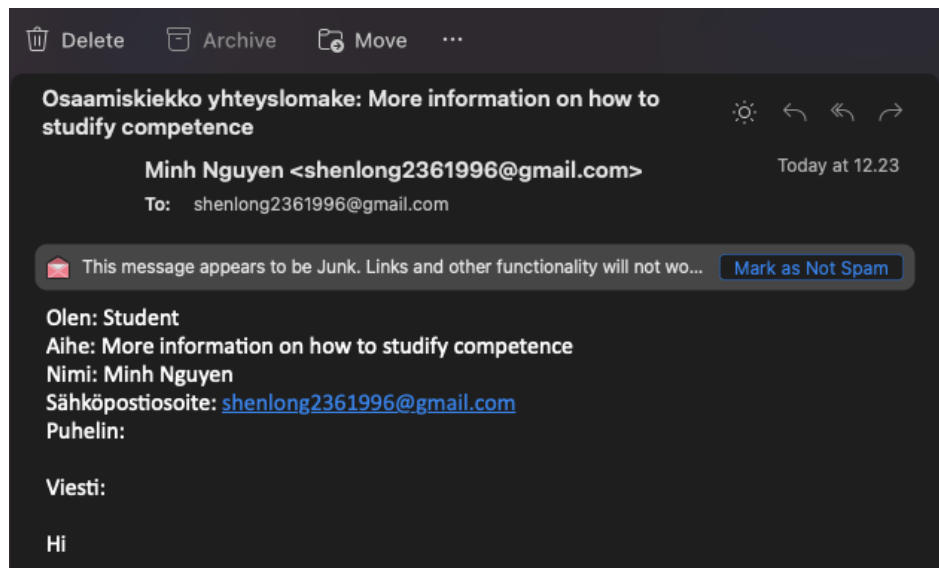


Figure 65. Website owner receives user's information

7 CONCLUSIONS

The main purpose of this thesis was to make a web application where students can visit and look for how their volunteer activities or competences can benefit them in exchanging those for study credits and vice versa.

The web application is for students with objectives of making sure that they can find the match between the school and organization they desire. Other features of the application are responsiveness, accessibility and multilanguage support, contacting the website owner, no data corruption, and the ability to search for as many schools and organizations as possible.

All the targets were achieved except the last one which is in progress because it is hard to have all data immediately. Our client's specialized department has to evaluate and connect schools and organizations, which is a long procedure.

7.1 Future Work

The project will be developed further with version 3.0. It focuses on data suggested by AI and the migration from an old database to a relational database such as MySQL.

REFERENCES

- [1] Competence Disc - 10Monkeys Digital. Accessed 9 April 2022.
<https://www.10monkeysdigital.com/en/work/osaamiskiekkoo>
- [2] TechRadar. HTML5: what is it? Accessed 17 April 2022.
<https://www.techradar.com/news/internet/web/html5-what-is-it-1047393>
- [3] Techopedia. What is Cascading Style Sheets Level 3 (CSS3)? Accessed 17 April 2022. [https://www.techopedia.com/definition/28243/cascading-style-sheets-level-3-css3#:~:text=Cascading%20Style%20Sheets%20Level%203%20\(CSS3\)%20is%20the%20iteration%20of,easier%20to%20learn%20and%20understand](https://www.techopedia.com/definition/28243/cascading-style-sheets-level-3-css3#:~:text=Cascading%20Style%20Sheets%20Level%203%20(CSS3)%20is%20the%20iteration%20of,easier%20to%20learn%20and%20understand)
- [4] Vue.js. Introduction. Accessed 23 February 2022.
<https://vuejs.org/guide/introduction.html>
- [5] What is Nuxt.js? Accessed 25 February 2022.
<https://www.popwebdesign.net/what-is-nuxt.js.html>
- [6] NuxtJS. Views. Accessed 25 February 2022.
<https://nuxtjs.org/docs/concepts/views>
- [7] BootstrapVue. Accessed 28 February 2022. <https://bootstrap-vue.org/>
- [8] BootstrapVue. Getting Started. Accessed 28 February 2022.
<https://bootstrap-vue.org/docs>

- [9] Stevenson, Doug. 2018. What is Firebase? The complete story, abridged. Accessed 1 March 2022. <https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>
- [10] Documentation for Visual Studio Code. Accessed 2 March 2022. <https://code.visualstudio.com/docs/>
- [11] Node.js. Accessed 4 March 2022. <https://nodejs.org/en/>
- [12] Node.js Introduction. Accessed 4 March 2022. https://www.w3schools.com/nodejs/nodejs_intro.asp
- [13] Glover, Ryan. n.d. What are JavaScript packages and dependencies?. Accessed 7 March 2022. <https://cleverbeagle.com/blog/articles/what-are-javascript-packages-and-dependencies>
- [14] Npm Docs. Specifying dependencies and devDependencies in a package.json file. Accessed 7 March 2022. <https://docs.npmjs.com/specifying-dependencies-and-devdependencies-in-a-package-json-file>
- [15] NuxtJS. Pages directory. Accessed 26 March 2022. <https://nuxtjs.org/docs/directory-structure/pages/>
- [16] NuxtJS. Built-in Components. Accessed 19 March 2022. <https://nuxtjs.org/docs/features/nuxt-components/>
- [17] NuxtJS. Data Fetching. Accessed 19 March 2022. <https://nuxtjs.org/docs/features/data-fetching/>
- [18] NuxtJS. The Fetch Hook. Accessed 26 March 2022. <https://nuxtjs.org/docs/components-glossary/fetch/>

- [19] BootstrapVue. Form Group. Accessed 3 April 2022. <https://bootstrap-vue.org/docs/components/form-group>

