

|  |
| --- |
| **VILNIAUS KOLEGIJA**  **ELECTRONICS AND INFORMATICS FACULTY**  **SOFTWARE DEVELOPMENT DEPARTMENT** |
| **FINAL PRACTICE REPORT**  6531BX028 PI18E |
| |  |  | | --- | --- | | PRACTICE PLACE: | UAB Visma | |
| |  |  |  | | --- | --- | --- | | COMPANY ADVISOR | 2021 - 11 – 04 | AURIMAS CHOLOMSKIS | |
| |  |  |  | | --- | --- | --- | | STUDENT | 2021 - 11 – 04 | DŽIUGAS PEČIULEVIČIUS | | FACULTY ADVISOR | 2021 - 11 - 04 | DR. ROMANAS TUMASONIS | |
| 2021, VILNIUS |

**TABLE OF CONTENTS**

[1. PRACTICE ASSIGNMENT FORMULATION 7](#_Toc86776434)

[1.1. Deep problem description 7](#_Toc86776435)

[1.2 Functional requirements 7](#_Toc86776436)

[1.3. Non-functional requirements 8](#_Toc86776437)

[2. PRACTICE ASSIGNMENT ANALYSIS 9](#_Toc86776438)

[2.1. Analysis on functional requirements. Use case diagrams, scenarios. 9](#_Toc86776439)

[2.2. Analysis on non-functional requirements. 16](#_Toc86776440)

[3. SOFTWARE 19](#_Toc86776441)

[3.1 Explanation of development tools selection. 19](#_Toc86776442)

[3.1.1 Jira 19](#_Toc86776443)

[3.1.2 Git and GitHub 20](#_Toc86776444)

[3.1.3 Angular and TypeScript 22](#_Toc86776445)

[3.1.4 Unit and End-to-end testing 24](#_Toc86776446)

[3.1.5 Accessibility tools 26](#_Toc86776447)

[3.2 Description of third-party libraries, modules. 30](#_Toc86776448)

[3.2.1 Nordic Cool 4 (NC4) 30](#_Toc86776449)

[3.2.2 LaunchDarkly 31](#_Toc86776450)

[3.2.3 Moment.js and Moment timezone 32](#_Toc86776451)

[3.2.4 RxJS 33](#_Toc86776452)

[3.2.5 Axe core and Axe HTML reporter 34](#_Toc86776453)

[3.2.6 Cypress and Cypress axe 34](#_Toc86776454)

[4. USER GUIDE 36](#_Toc86776455)

[4.1 Software local setup, installation, and uninstallation guide. 36](#_Toc86776456)

[4.2 Description and identification of steps necessary to take in order to fulfil all functional requirements. 36](#_Toc86776457)

[4.2.1 Login with ID-porten 36](#_Toc86776458)

[4.2.2 Logout after clicking logout button 38](#_Toc86776459)

[4.2.3 Should see all its children listed in a list and should be able to select one child at a time 38](#_Toc86776460)

[4.2.4 Should be able to fill out and send forms 39](#_Toc86776461)

[4.2.5 Should be able to add more children on leave application form to send for more than one child 42](#_Toc86776462)

[4.2.6 Should be able to see sent applications 44](#_Toc86776463)

[4.2.7 Should be able to sort sent applications from newest to oldest and vice versa 45](#_Toc86776464)

[4.2.8 Should show error messages if mandatory fields are left empty 46](#_Toc86776465)

[4.2.9 Should display warning if user tries to leave application process 47](#_Toc86776466)

[4.2.10 Should display success page if form was sent successfully 48](#_Toc86776467)

[4.2.11 Should be able to navigate and see contact information for schools and municipality 49](#_Toc86776468)

[CONCLUSIONS AND FUTURE STEPS 52](#_Toc86776469)

[LITERATURE 53](#_Toc86776470)

**TABLE OF FIGURES**

Figure 1 - Use case diagram for functional requirements 9

Figure 2 - My Kanban board 19

Figure 3 - Checkout into a new branch 20

Figure 4 - Check what changes were made 20

Figure 5 - Commit changes 21

Figure 6 - Git log 21

Figure 7 - GitHub PR 21

Figure 8 – Folder structure for CRUD application 22

Figure 9 - App component for application 23

Figure 10 – Application routing 24

Figure 11 - Service method to get types 25

Figure 12 - Service unit test to test if it returns types 25

Figure 13 - Cypress test example 26

Figure 14 - Wave - accessibility testing tool 27

Figure 15 - Accessibility issue example 27

Figure 16 - Working accessibility feature example 28

Figure 17 - axe DevTools - another accessibility testing tool 28

Figure 18 - axe DevTools - issues found 29

Figure 19 - Google Lighthouse accessibility tool 29

Figure 20 - Google lighthouse accessibility score 30

Figure 21 - Nordic Cool 4 - web library 30

Figure 22- Nordic Cool 4 - web library 31

Figure 23 - Nordic Cool 4 icons 31

Figure 24 – LaunchDarkly – feature toggling service 32

Figure 25 - moment.js 32

Figure 26 - Observable example - typescript file 33

Figure 27- Observable example - HTML file 33

Figure 28 - Cypress test to visit a page 34

Figure 29 - Cypress test to click a button 35

Figure 30 - Application landing page 36

Figure 31 - Application login step 37

Figure 32 - Choosing how to login 37

Figure 33 - Home page after login 38

Figure 34 - Logout button in user dropdown menu 38

Figure 35 - Children list 39

Figure 36 - First step of application form 40

Figure 37 - Second step of application form 40

Figure 38 - Third step of application form 41

Figure 39 - Final step of application form - Form overview 41

Figure 40 - Application sent success page 42

Figure 41 - Add additional children button 43

Figure 42 - Adding children popup modal 43

Figure 43 - List with added children 44

Figure 44 - Application sent success page 44

Figure 45 - Application history button 45

Figure 46 - Application history page 45

Figure 47 - Application history sort button – display newest 46

Figure 48 - Application history sort button – display oldest 46

Figure 49 - Error messages - missing fields 47

Figure 50 - Warning when trying to leave the form 47

Figure 51 - Application form overview step 48

Figure 52 - Application sent success page 49

Figure 53 - User dropdown menu 50

Figure 54 - Contact information page 51

**INTRODUCTION**

Professional practice was performed at UAB Visma that took a total of 8 weeks from September 1st of 2021 to November 5th of 2021 as a front-end developer role. The company is one of the Europe ‘s leading software companies and offers software/services that simplify core business processes in the private and public sector. Visma has diversity of projects – from small applications to large J2EE (Java 2 Enterprise Edition) systems.

After the two-month internship at the company we were allocated to a team. I was allocated to a team where they recently began working on remaking an old school application system and most of its functionality. In a team I had just joined, there were other two front-end developers working together with me: one of them was a mid and the other was a senior developer. All the tasks that had to be done, were logged, and tracked in Jira. After completing a task, we open a Pull Request (PR), most of the time senior is the one who looks over the code, leaves feedback if he has anything to add and either approves or asks to make changes to the current branch.

Currently our goal is to write most necessary functionality first so users could start using most necessary functionality on the new environment instead of using the old one. We also have a deadline by when we must finish all the initial functionality and release into production. Basically, our goal is to add initial functionality and release the application for one municipality in Norway which is called Tønsberg by November 12th. The initial functionality includes parents to be able to select their children and send forms such as school transport or leave applications.

# 1. PRACTICE ASSIGNMENT FORMULATION

## 1.1. Deep problem description

So, every front-end developer is responsible for implementing visual and interactive elements that user engages with through their web browser. Developers write code and create components from very simple UI elements like buttons to complex user forms. Front-end developers are also responsible for connecting front-end code to back-end by handling HTTP requests to the server and displaying responses in the user interface of the application.

During this two-month professional practice, I worked on various tasks, from simple bug fixes and quite simple UI functionality to writing complex user forms and handling the data that has been entered by the user.

The problem with the old system that we’re currently rewriting, was that the system is quite old, slow and uses outdated company design library - Nordic Cool 3 (NC3) and needs renewal. So, at this time as I’m writing this report, our designer is making designs for the renewed system that would use the latest Nordic Cool version (NC4) and we’re (us, developers) adding the functionality. The project itself is quite young so we’re finishing up main functionality that is needed and is most important so we can release the first version of the application. And we’re also working on accessibility as well as bug fixing and preparing the application for the release. The backend is practically going to stay the same as it was for the old system, but with minor improvements to work with our new application. As for testing our codebase, we focus on testing what is most important for us. We test all the data service methods that fetch data from the backend as well as we also write tests for components that include more complex logic such as a nested if statement or calculations. Basically, we try and focus on testing the business logic first and what would cause most issues for the user if it happened to break.

# 1.2 Functional requirements

Since the application is quite new and with main functionality for the first release, there won’t be that many functional requirements:

1. Users should be able to login using ID-porten (public login solution to online services. ID porten basically provides access to many services from central and local government).
2. The user should be able logout when they click the log out button.
3. The user should be logged out automatically after 15 minutes of inactivity.
4. The user should be able to see all children listed that belong to a parent when logged into the system.
5. The user should be able to select one of their children at a time.
6. The user should be able to fill out and send the forms for selected child.
7. The system should allow to add more children on some forms to send for more than one child at a time.
8. The system should allow users to check application history and view all the applications sent by the user.
9. The system should allow user to sort application history from newest first to oldest first and vice versa.
10. The system should show error messages if some of the mandatory form fields are missing.
11. The system should show warnings if user tries to leave the application progress without it being filled and sent.
12. The system should show success page if the form was sent successfully.
13. They system should let user navigate and see municipality and school contact information.

## 1.3. Non-functional requirements

Non-functional requirements:

1. The application should be built with JavaScript framework – Angular.
2. The application should be built with strongly typed language - TypeScript.
3. All data should be fetched from the existing API.
4. Application should have accessibility support and be readable for smaller screens.
5. Application should work with screen readers.
6. Application should be responsive and should show screen size accordingly.
7. It should display user a spinner (loading indicator) if any data is being fetched or page is being refreshed now.
8. Application should have a success page to redirect user once the process is complete.

# 2. PRACTICE ASSIGNMENT ANALYSIS

## 2.1. Analysis on functional requirements. Use case diagrams, scenarios.

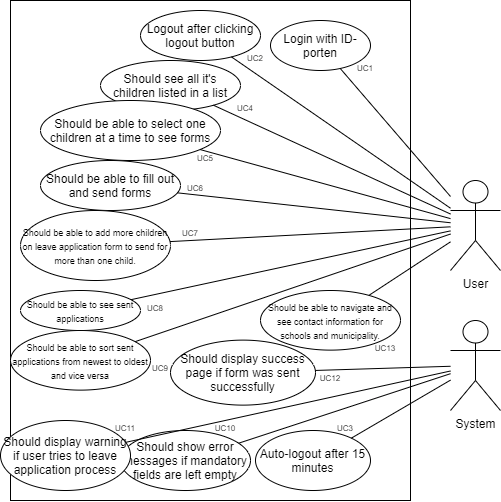


Figure 1 - Use case diagram for functional requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **FUNCTIONAL REQUIREMENT** | **USER REQUIREMENT** | **USE-CASE** | **PRIORITY** |
| FR1 | Login with ID-porten | User | UC1 | 1 |
| FR2 | Logout after clicking logout button | User | UC2 | 2 |
| FR3 | Auto-logout after 15 minutes | System | UC3 | 3 |
| FR4 | Should see all its children listed in a list | User | UC4 | 4 |
| FR5 | Should be able to select one child at a time to see forms. | User | UC5 | 5 |
| FR6 | Should be able to fill out and send forms | User | UC6 | 7 |
| FR7 | Should be able to add more children for leave application form to send for more than one child | User | UC7 | 11 |
| FR8 | Should be able to see sent applications | User | UC8 | 8 |
| FR9 | Should be able to sort application history from newest to latest and other way round. | User | UC9 | 9 |
| FR10 | Should show error messages if any mandatory fields are left empty | System | UC10 | 10 |
| FR11 | Should display warning if user tries to leave application process. | System | UC11 | 13 |
| FR12 | Should display success page if form was sent successfully. | System | UC12 | 12 |
| FR13 | Should be able to navigate and see contact information for schools and municipality. | User | UC13 | 6 |

Describing the use cases:

|  |  |
| --- | --- |
| **Function** | Login with ID-porten |
| **ID** | UC1 |
| **Description** | User can login into the system using login system. |
| **User role** | User |
| **Pre-conditions** | There is an internet connection  There are valid login details |
| **Inputs** | User ID, password and 5-digit code |
| **Outputs** | Displays the system interface |
| **Action** | 1. The use enters the website 2. User click to login with ID-porten 3. User enters details 4. Confirms and clicks ‘continue’ |
| **Post-conditions** | After entering login details, user should be redirected back to the system web address and should see the system interface. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Logout after clicking logout button |
| **ID** | UC2 |
| **Description** | User can logout out of the system |
| **User role** | User |
| **Pre-conditions** | There is an internet connection |
| **Inputs** | Go to user dropdown menu and click logout. |
| **Outputs** | User should be logged out. |
| **Action** | 1. The use enters the website. 2. Clicks user dropdown menu. 3. Clicks on the ‘logout’ button. |
| **Post-conditions** | After clicking the logout button, user should be logged out and redirected to the login screen. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Auto-logout after 15 minutes |
| **ID** | UC3 |
| **Description** | Should automatically logout user if the user hasn’t been active for 15 minutes. |
| **User role** | System |
| **Pre-conditions** | There is an internet connection  User being inactive for 15 minutes. |
| **Inputs** | - |
| **Outputs** | Logs the user out of the system. |
| **Action** | 1. No inputs required from the user. 2. After 11 minutes of inactivity user should get a popup with a countdown that a user has 4 minutes left till auto-logout. 3. Logs the user out. |
| **Post-conditions** | If a user hasn’t made any actions the system will automatically log the user out and bring the user back to the login screen. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | User should see all belonging children listed in a list |
| **ID** | UC4 |
| **Description** | User can see children. |
| **User role** | User |
| **Pre-conditions** | There is an internet connection  Should be logged in. |
| **Inputs** | Enters the website and logs in. |
| **Outputs** | Displays all the children belonging for the user. |
| **Action** | 1. User enters the website 2. Logs in if needed. |
| **Post-conditions** | User should be able to see a list of children that belong to the parent. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Should be able to select one child at a time to see forms. |
| **ID** | UC5 |
| **Description** | User can select a child and see a list of applications that can be done for specific child. |
| **User role** | User |
| **Pre-conditions** | There is an internet connection  Should be logged in |
| **Inputs** | User should be logged in and then is able to select one of the children at the top. |
| **Outputs** | Displays applications that are available to be filled for specific child. |
| **Action** | 1. User enters the website 2. Logs in if needed. 3. Selects a child to see applications. |
| **Post-conditions** | After selecting one of the children, user can now select one of the applications and fill it out for previously selected child. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Should be able to fill out and send forms |
| **ID** | UC6 |
| **Description** | User should be able to select child, then select a form that needs to be filled for a child and then send it. |
| **User role** | User |
| **Pre-conditions** | There is an internet connection.  Should be logged in |
| **Inputs** | Once logged in, selects a child, and can now select forms that need to be filled for a child. |
| **Outputs** | Displays applications that can be filled for selected child. |
| **Action** | 1. The use enters the website. 2. Logs in if needed. 3. Selects a child. |
| **Post-conditions** | After selecting a child, user can complete child related actions. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Should be able to add more children for leave application form to send for more than one child. |
| **ID** | UC7 |
| **Description** | User can add more children that could be sent for a few children at once. |
| **User role** | User |
| **Pre-conditions** | There is an internet connection.  Should be logged in. |
| **Inputs** | Go to the website, and once user clicks to create ‘leave application’ for a child, it would already show that this form is being filled for specific child. User can add more children in the form itself to send the application for few children at once. |
| **Outputs** | Sends the form for few children at the same time. |
| **Action** | 1. The use enters the website. 2. Logs in if needed. 3. Selects ‘Leave application’ 4. Aggress to rules. 5. Clicks ‘Add children’ 6. Selects children that need to be added for the form. 7. Accepts changes. 8. Selected children are now added to the form. |
| **Post-conditions** | After adding the children to the form, user should be able to send the form for few children at once. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Should be able to see sent applications |
| **ID** | UC8 |
| **Description** | Once a user sends one of the applications, user should be able to see sent applications in application history page. |
| **User role** | User |
| **Pre-conditions** | There is an internet connection.  Should be logged in.  Should have at least one application sent. |
| **Inputs** | After sending the application, user can click ‘Application history’ at the main page of the system, and user will be redirected to the page where user can view sent applications. |
| **Outputs** | Displays all sent user applications. |
| **Action** | 1. The use enters the website. 2. Logs in if needed. 3. Sends an application if there are none sent. 4. In the main applications page clicks ‘Application history’. |
| **Post-conditions** | After sending the application, user should be able to go to ‘Application history’ page and see sent applications for chosen child. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Should be able to sort application history from newest to latest and other way round |
| **ID** | UC9 |
| **Description** | In the application history page, above applications user is able to click a button which sorts the applications to show either newest or oldest first. |
| **User role** | User |
| **Pre-conditions** | There is an internet connection.  Should be logged in.  Should have at least one application sent. |
| **Inputs** | This needs at least one application sent. Then go to application history page and click sort icon at the top of the page above all sent applications. |
| **Outputs** | Sorts applications in the application history page. |
| **Action** | 1. The use enters the website. 2. Logs in if needed. 3. Sends an application if there are none sent. 4. In the main applications page clicks ‘Application history’. 5. Clicks sort icon to sort applications. |
| **Post-conditions** | After clicking the sort applications icon, they system shows either newest or oldest applications. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Should show error messages if any mandatory fields are left empty |
| **ID** | UC10 |
| **Description** | The system displays errors below unfilled fields when user tries to go to the next step. |
| **User role** | System |
| **Pre-conditions** | There is an internet connection.  User should be logged in. |
| **Inputs** | User tries to proceed in the application without filling the necessary data. |
| **Outputs** | Displays error messages below unfilled fields. |
| **Action** | 1. The use enters the website. 2. Logs in if needed. 3. Tries to fill the application but leaves empty fields. 4. Error occurs. |
| **Post-conditions** | After user tries to continue into the next page, system prevents user on continuing and lets the user know that some of the fields are left unfiled. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Should display warning if user tries to leave application process |
| **ID** | UC11 |
| **Description** | The system displays a warning that all data will be lost if the user has begun filling up the information required for the application to be sent. |
| **User role** | System |
| **Pre-conditions** | There is an internet connection.  User should be logged in. |
| **Inputs** | User has begun filling up the form and tries to leave in the middle of filling up the data for the application. |
| **Outputs** | Displays a warning that data will be lost if user decides to leave the process. |
| **Action** | 1. The use enters the website. 2. Logs in if needed. 3. Selects child that the application needs to be filled. 4. Starts filling up the application. 5. Tries to leave without sending the application. |
| **Post-conditions** | After trying to leave application process, system will display a warning. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Should display success page if form was sent successfully |
| **ID** | UC12 |
| **Description** | System should display a success page if user has filled out all the information and the application has been successfully to the backend. |
| **User role** | System |
| **Pre-conditions** | There is an internet connection.  User should be logged in.  User should have filled and sent the application. |
| **Inputs** | Fill out the application and click to send it. If it has been sent, the system will redirect the user to the success page. |
| **Outputs** | Displays success page. |
| **Action** | 1. The use enters the website. 2. Logs in if needed. 3. Selects child and form that need to be filled. 4. After filling up all the necessary data, and confirming the application details, system redirects user to success page. |
| **Post-conditions** | After sending the application successfully, the system displays a success page. |
| **Side effects** | None |

|  |  |
| --- | --- |
| **Function** | Should be able to navigate and see contact information for schools and municipality. |
| **ID** | UC13 |
| **Description** | User can navigate to contacts page and find all the necessary information. User can see municipality contact information at the top, and then school contact information. If there are more children that go to different schools, all of that schools and their data will be displayed there. |
| **User role** | User |
| **Pre-conditions** | There is an internet connection.  Should be logged in. |
| **Inputs** | Go to contact information page. |
| **Outputs** | Displays contact information. |
| **Action** | 1. The use enters the website. 2. Logs in if needed. 3. Clicks on the dropdown user menu at the top right of the screen. 4. Clicks on contact information. |
| **Post-conditions** | After clicking ‘contact information’, user gets redirected to contact information page. |
| **Side effects** | None |

# 3. SOFTWARE

At work, we use all kinds of tooling. From project management tools like Jira, software development tools like Angular and even accessibility testing tools like built-in google lighthouse.

Here’s all the tools we work with and I’m going to explain how we use them within our team later in the report.

For project management we mainly use Jira. For developing software, we use all kinds of tools. First, probably the most important tool that we use, I’d say it’s Git and GitHub, which helps us to track code, changes and store our code in a single repository. The application is written using JavaScript framework Angular. The code itself is written using TypeScript. We not only write unit tests for our application, but we also use end-to-end (e2e) testing. For unit testing we use jest. For writing and testing our application with e2e tests, we use cypress. We also spend a lot of our time testing and writing code, so our applications are accessible for users that have disabilities such as bad eyesight. To test the accessibility of our application, we use tools such as Wave (web accessibility evaluation tool), axe DevTools and Google Lighthouse which I’ll talk about all of them later.

## Explanation of development tools selection.

### 3.1.1 Jira

To track our progress, we use Jira. Jira allows teams to manage projects, view roadmaps and track bugs, which helps us to keep up to date with everything and helps plan our time on when something needs to be done. This is how my Kanban board currently looks like:

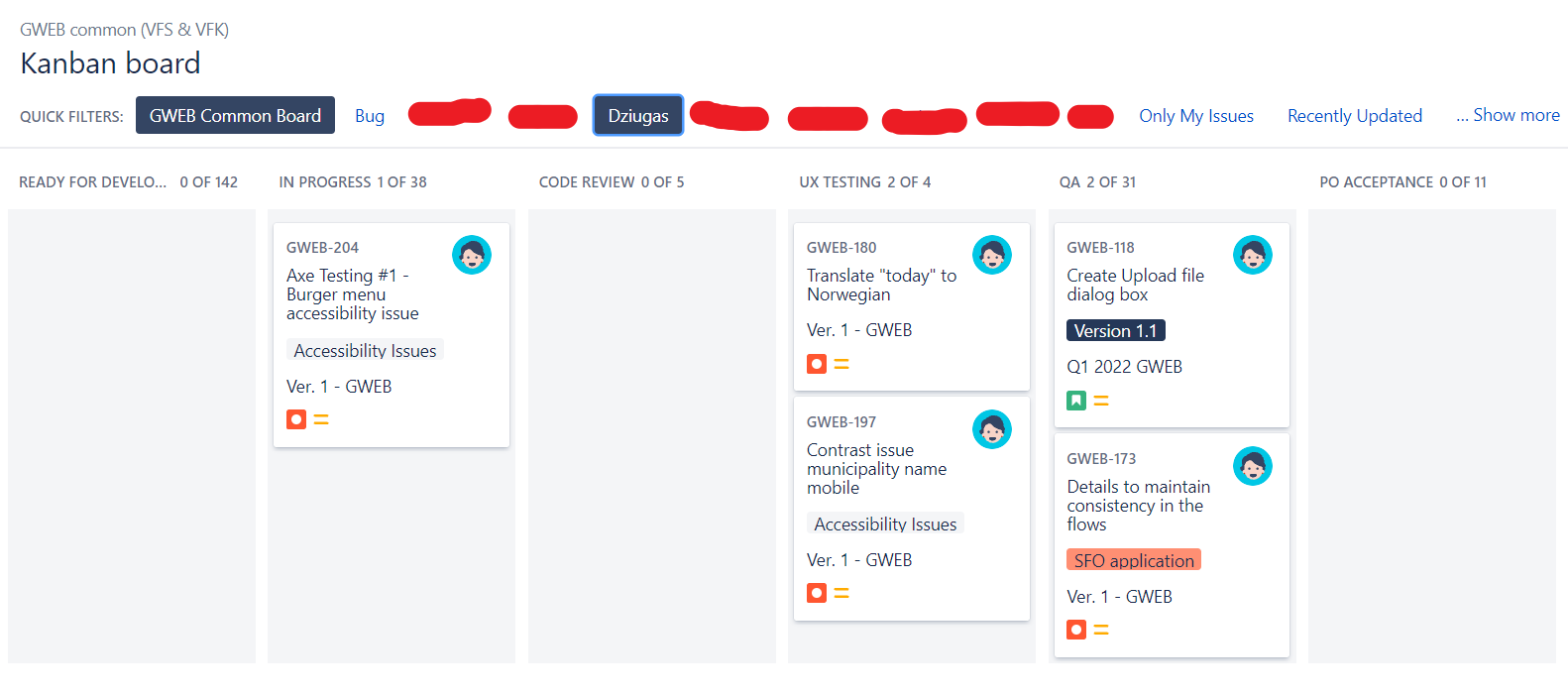


Figure 2 - My Kanban board

At this moment I’m writing this report, we have less than two weeks till our application release for one of the municipalities in Norway, and since we have finished all the main functionality the first version is going to have, we’re fixing all the accessibility issues we’re having. We’re aiming for the application to work seamlessly with screen readers and keyboard navigation. I will talk more about it when we reach accessibility tools section.

### 3.1.2 Git and GitHub

Just like most of the other development teams we use git as our versioning system and GitHub as our code storage. Basically, how we work with Git is once when we start working on a bug, adding, or fixing some functionality, or adding a completely new feature to our project, we create a new branch for it and name it accordingly to the ticket in our bord in Jira. Let’s say, a task in Jira is called “NAME-323 – fix a bug that occurs when user does X or Y”. In this case, we would take the task, and name our new branch as “NAME-323” Once we finish working on the task, we would open a Pull Request (PR) and wait for somebody to review it. Once it’s reviewed, we either make changes if needed or merge our code into the main branch of the application. It would look something like this:

First, we create and switch to our new branch that we’re going to be working on.

Text

Description automatically generated

Figure 3 - Checkout into a new branch

After making changes to files, we can see that we created or changed files in our project.

Text

Description automatically generated

Figure 4 - Check what changes were made

If we’re okay with all the changes we made, we can add all files to stage and then commit them. As a commit message, we want to write our ticket ID first and then a commit message describing what we have changes within that commit.

Text

Description automatically generated

Figure 5 - Commit changes

After we have made a commit, we can check our log using git log.

Text

Description automatically generated

Figure 6 - Git log

If we’re happy with what we did, we can push our code to GitHub and open it up for review. Once somebody reviews our project, we can merge our changes we made into the main branch.

Graphical user interface, text

Description automatically generated

Figure 7 - GitHub PR

That’s our workflow with Git and GitHub in our team.

### 3.1.3 Angular and TypeScript

Angular applications are unique not just because it uses TypeScript, has better tooling but it has a higher scalability. The way angular applications are structured, it makes it easy to get going and simple to maintain large applications. Unfortunately, I’m not allowed to show any of the code in our project we’re using, so I will try writing similar code examples and explain what we’re trying to achieve.

This is something how our Angular application folder structure looks like while not showing too much of it:

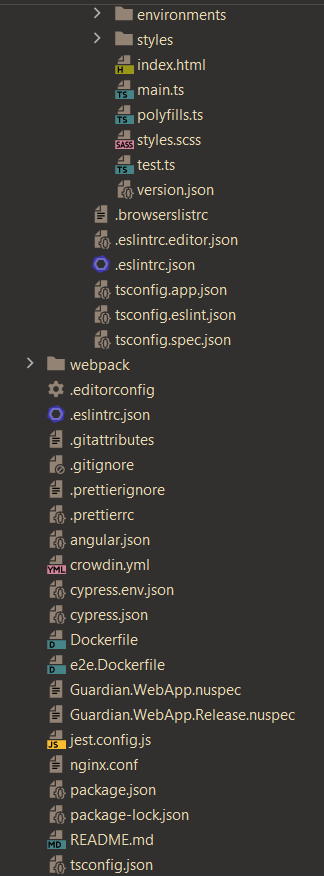
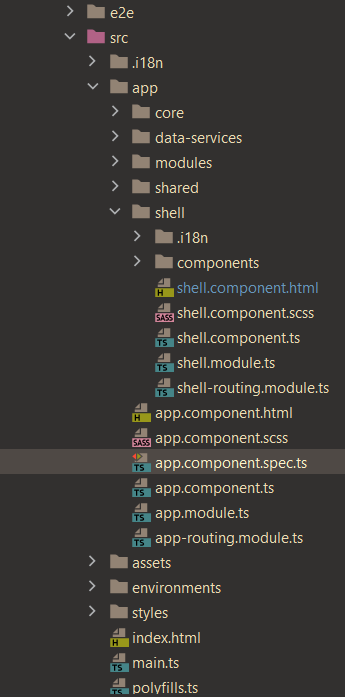


Figure 8 – Folder structure for CRUD application

At the top we can see e2e folder, we write and keep our end-to-end tests and configuration there. Then in the source folder, we keel all the files that are needed for our application to run.

We have .i18n which stands for internalization. We have translations for other languages, so a user can switch between English and Norwegian. In the application core folder, we keep all files like constants, models, utilities, or services. Then we have data-services folder which has services that we have in the backend itself.

Our modules folder is like a pages folder, we keep our components that are related to a page. Like for example, we have contact page, so we have a contact information component as one of the folders in that folder. Shared folder has all our global components that we use around the project. Then we have our shell folder, which where we keep all the files like navbars, child list, basically the main components of the project. In the app component we have our main logic how the application behaves on the initialization of the project. And finally, we have our setup files where we setup tools like linter, prettier and application configuration.

So, once we start our application, the application starts at this point, it loads whatever we have passed into the navbar, loads main content into the router-outlet where we can switch between pages and look at different content and the footer of the application.



Figure 9 - App component for application

Since the app component is ran first and it is the root component of the application, we want to define components that are going to be replaced across the application like navbar and footer and everything that is going to change, we want to put them into the router outlet component.

What is the router outlet? It is basically a component where we can write our routes that is going to visit, and once the route is visited, the page renders the component accordinly, it none of the routes are found, then the application renders a “not found” component. This how “app-routing.module.ts” file looks like:



Figure 10 – Application routing

At this point in this app routing component, if the user is authenticated, we load shell module which has routing just as this component but with all the components that can be switched between. If the user isn’t logged in, we load login page. Basically, that’s core angular functionality and how angular application works.

when we reach accessibility tools section.

### 3.1.4 Unit and End-to-end testing

Just like most of the other development projects, application testing is one of the most crucial parts of the development. In our project, we test all the data services and components that have some more complex logic and might be more likely to break. So, in this case when we create a service, we would cover all the service methods with tests. If we have created a component, we would only test parts that contain more complex methods or some conditional functionality like if else statements.

This is an example of how we write our services and how we test them. This is a very simple service method that gets types from the backend, we pass a required id, and it returns to us with the types:



Figure 11 - Service method to get types

And now that we have this very simple method to fetch the types, we can write some unit tests:



Figure 12 - Service unit test to test if it returns types

In this example we can see two tests, one test is there to check if the service method gets initialized, and then starting line 5, we can see our service method that we’re testing. What we’re seeing here, is we’re calling one of the methods and mocking the return value of two types on line 10. Then we write and call our created component from the service and write the assertion, so it expects two types to be called which we mocked previously.

So that’s unit testing, but that is not all. We also write end-to-end tests, and they automatically run every time we merge our features into the main branch through PR. So, if anything has unexpectedly changed, some of e2e tests would’ve failed and they would let us know.

For end-to-end testing, we use Cypress. Cypress is a JavaScript end to end testing framework. Basically, when you run your cypress run command, it will run your application, and it will perform step by step actions what a user would do. It would automatically try and login, it would click through the application and check if the application behaves on how the tests are written. So, for example, if we have a test run to test if it visits a website and loads it, we would write a test to something like this. Of course, we would want to write our hosted domain here.



Figure 13 - Cypress test example

But of course, we can have more complex tests, and have it enter data into the fields and navigate deep into the page, but I will not go deep into this.

### 3.1.5 Accessibility tools

For testing the accessibility of our application, we use tools like Wave, axe DevTools and Google Lighthouse. These tools display different results and problems that the applications have. Currently we are trying to minimize the number of the results (problems) returned using these tools.

First of them is Wave tool, I would say this is my least favorite tool to work with since it’s quite cluttered up most of the time.

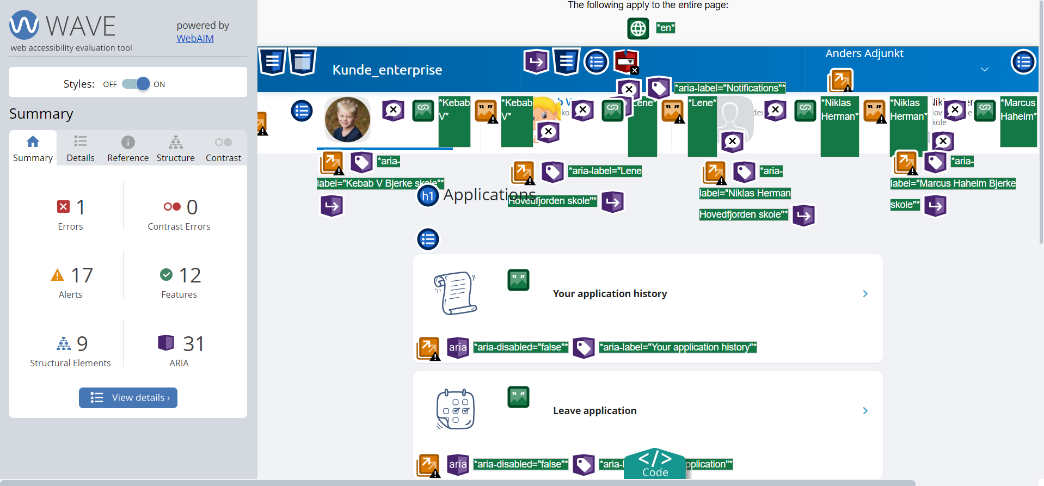


Figure 14 - Wave - accessibility testing tool

We can see all the elements that belong to the page.

We have been having some problems with screen readers on our navigation bar. Basically, the library we’re using has these accessibility issues, so we’re currently working on fixing these.

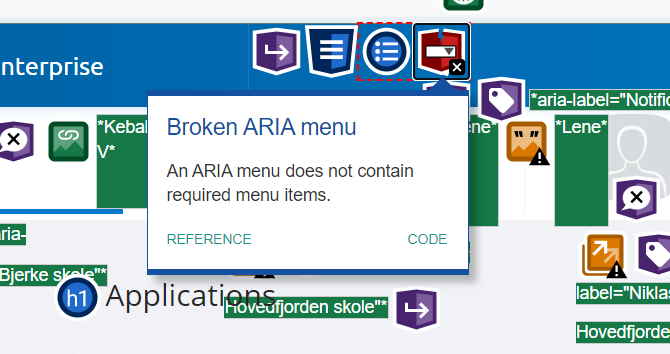


Figure 15 - Accessibility issue example

In the list, each of the item has so called aria labels. These aria labels are what allows screen readers to read out and let the user know where in the page user currently is. So, when user navigates when the keyboard, it would call out “Your application history”. If we had aria label disabled, screen readers would read the item as disabled, and a user would know that he cannot click on the item.

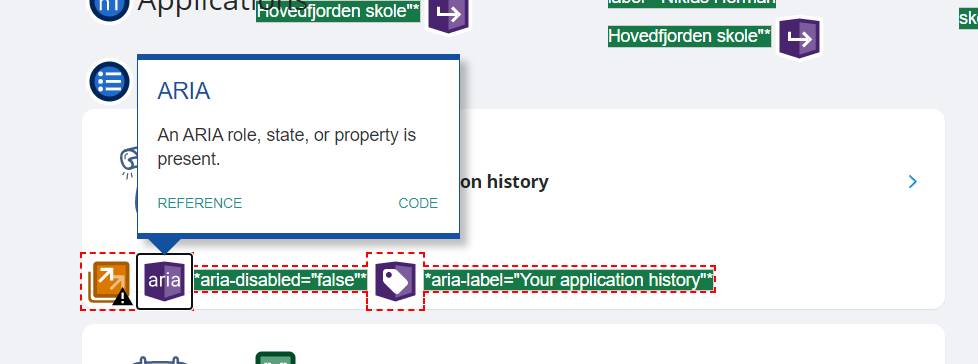


Figure 16 - Working accessibility feature example

The next accessibility tool I’m going to talk about is axe DevTools. Axe dev tools allows us to scan our page and then it lists all the problems that are wrong with the page.

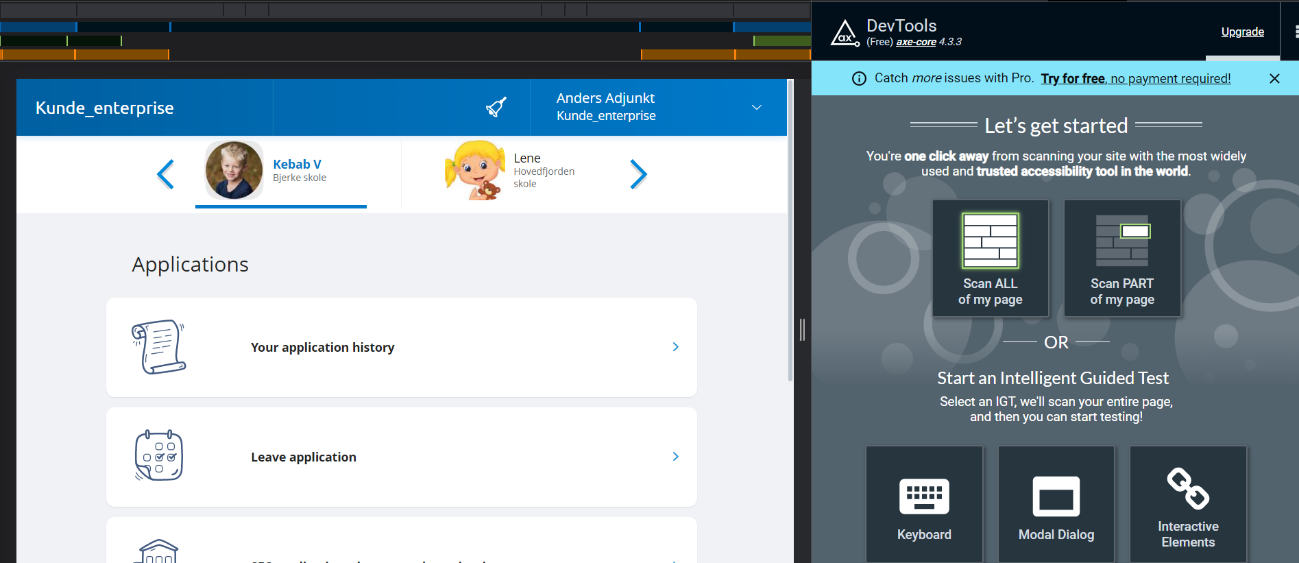


Figure 17 - axe DevTools - another accessibility testing tool

We can see, that currently we have 9 issues that need to be fixed. What I also like about this tool, that it allows us to highlight where the issue is, it also provides us with some resources and redirects us to websites which might give us a hint what exactly is going on with the accessibilty issues.

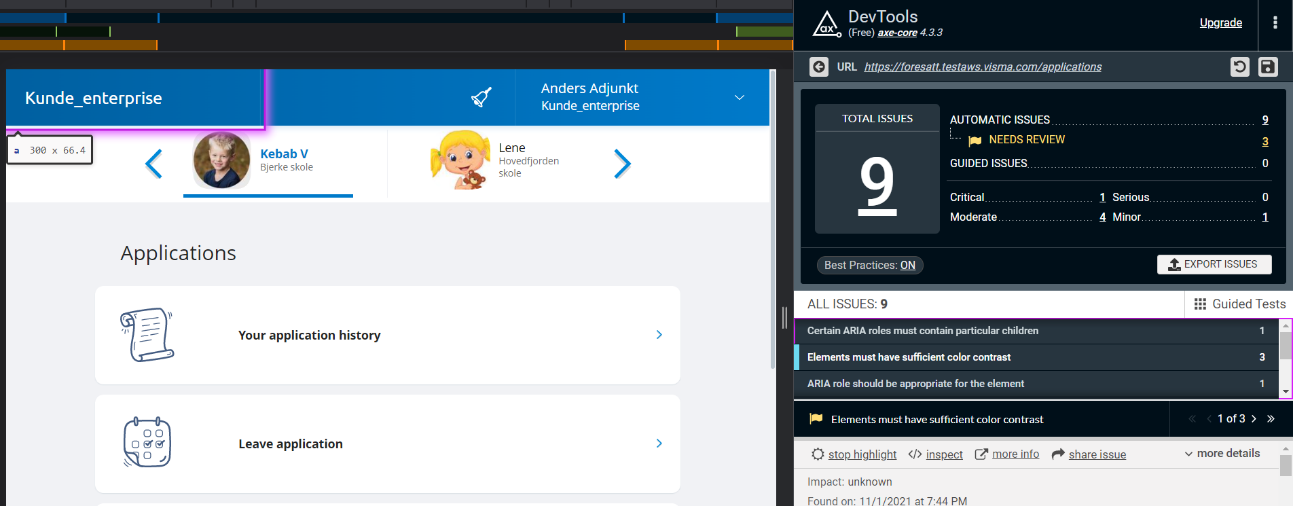


Figure 18 - axe DevTools - issues found

And the last accessibility tool we use, and I like it to use to get the picture of how our progress on the accessibility is going is called Google lighthouse. Google allows us to generate reports on many things like performance, SEO and so on, but currently we’re focusing on accessibility. We can choose a screen size if we want to generate a report for a phone or a desktop version and once we click the button to generate a report, it will give us some details on our website.



Figure 19 - Google Lighthouse accessibility tool

So, as we can see, currently we have pretty good accessibility which according to Google Lighthouse is 92%. It also provides us with the issues it found in our website, which in this case is also our navigation bar again.

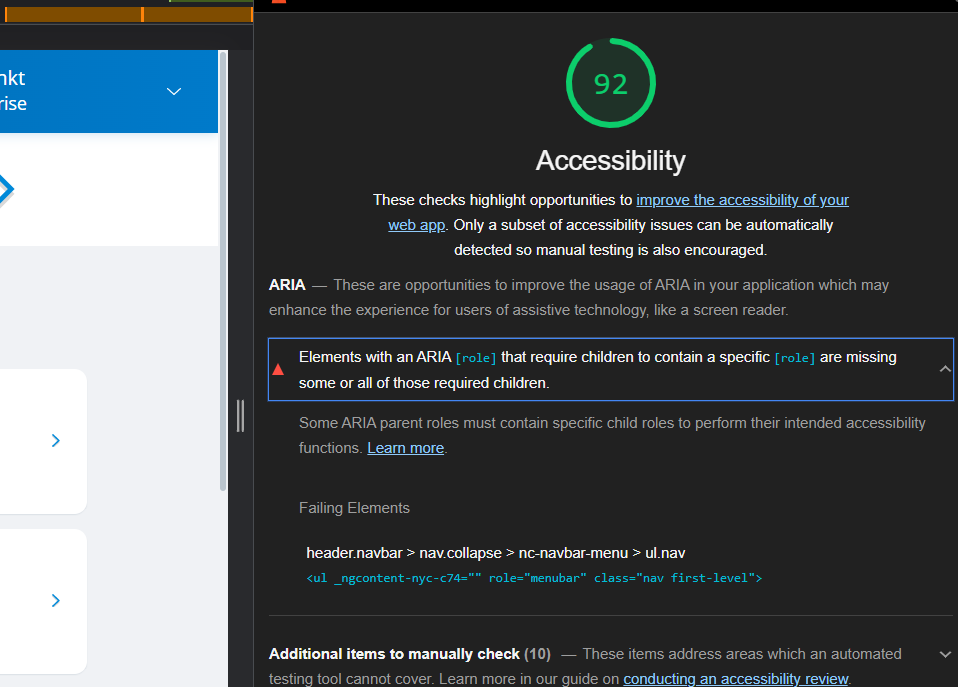


Figure 20 - Google lighthouse accessibility score

## Description of third-party libraries, modules.

### Nordic Cool 4 (NC4)

Nordic Cool 4 – is a component library we use in our project. It provides us with built-in components, styling, and code examples on how to use components such as drawers, cards, or navigation bars which we’re currently having accessibility issues with. Here’s a few examples what we’re working with.

This is the page with examples of what we can use to show alerts for our users:

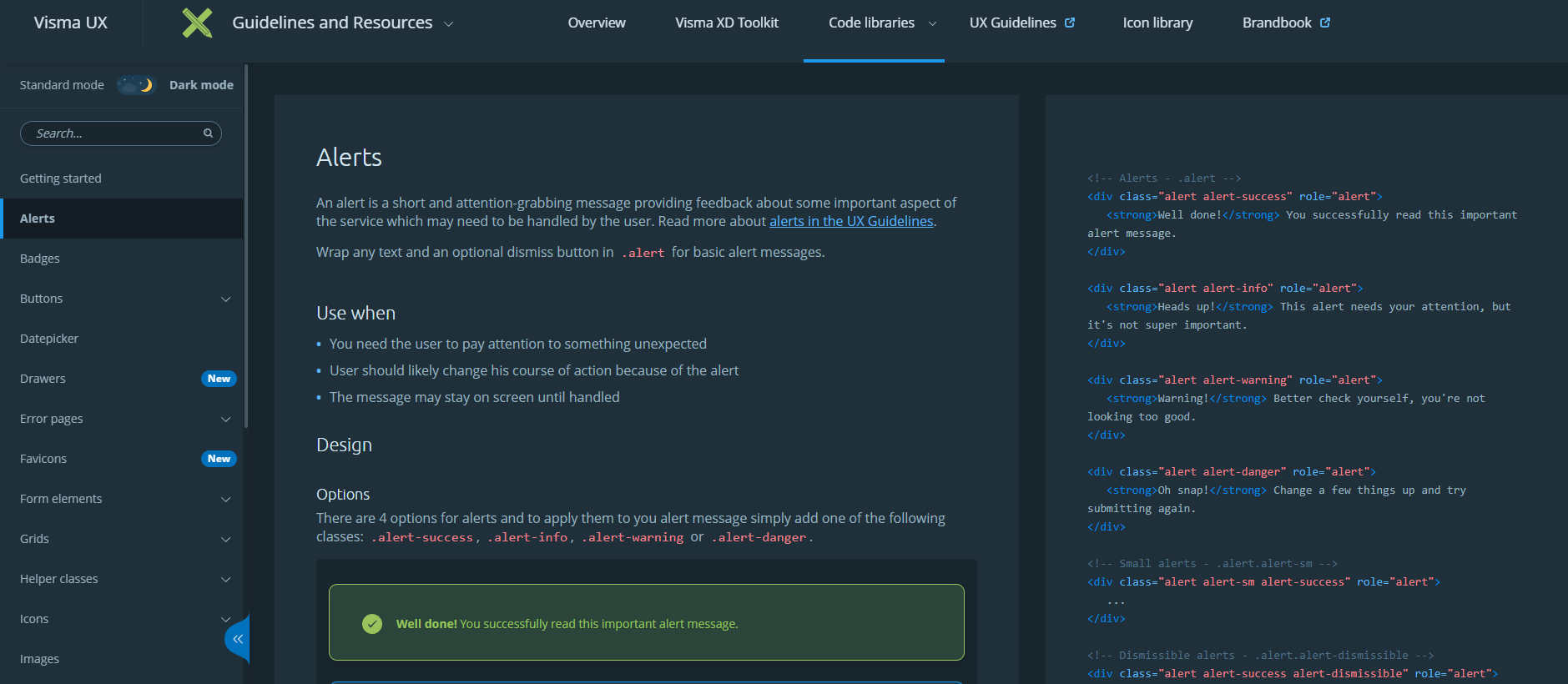


Figure 21 - Nordic Cool 4 - web library

It provides us with bunch of components to use like these buttons:

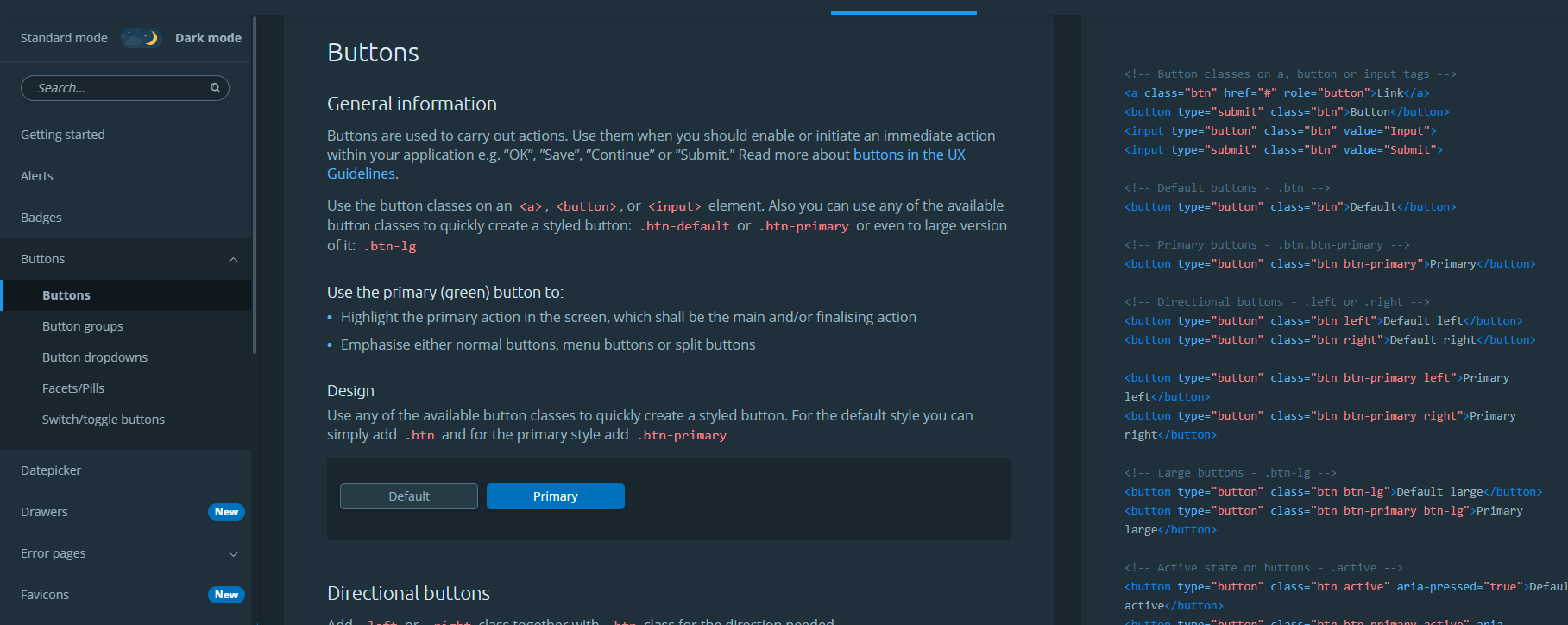


Figure 22- Nordic Cool 4 - web library

NC4 also provides us with icons to use:



Figure 23 - Nordic Cool 4 icons

### LaunchDarkly

Launch darkly allows us to manage feature flags. What are feature flags? Feature flags allow us to turn on and off selected functionality during runtime without deploying new code.

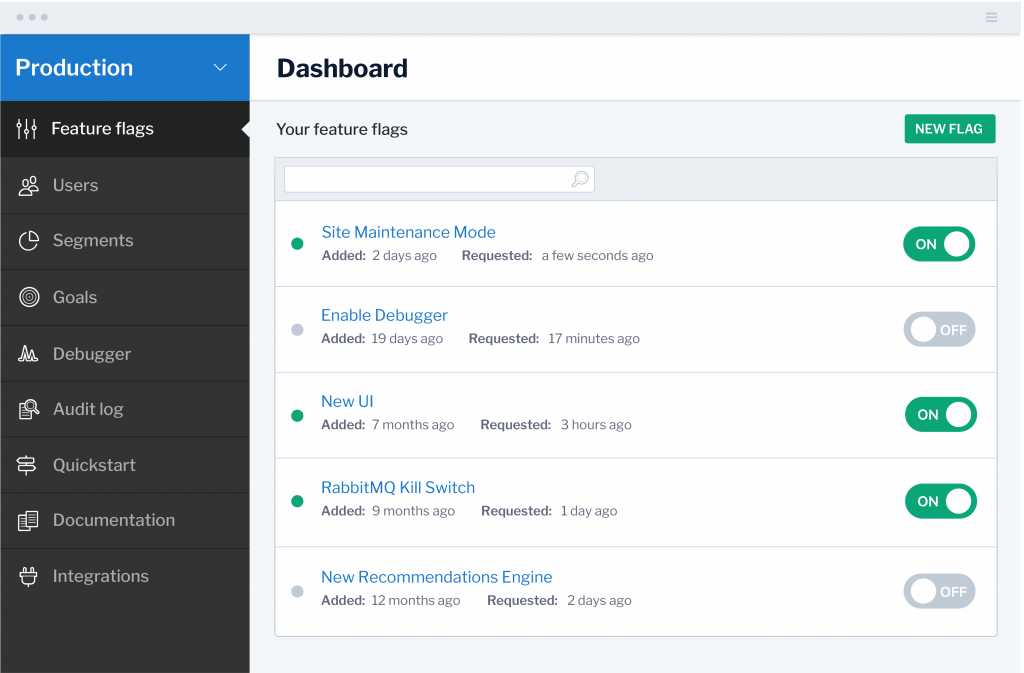


Figure 24 – LaunchDarkly – feature toggling service

### Moment.js and Moment timezone

Since we use dates in our forms, we needed a way to handle dates and display them in specific formats. In this case, Moment.js came in handy. It allows developers to easily parse, validate, manipulate, and display dates and times in JavaScript code. And moment timezone allows us to display dates and times according to the location user currently is located.

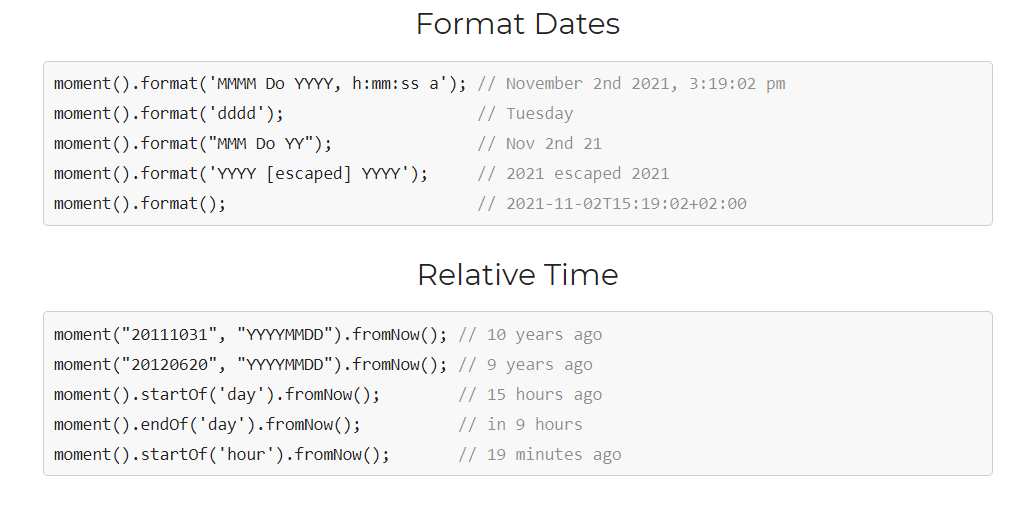


Figure 25 - moment.js

### RxJS

RxJS is a library which allows us to use Observables in our project. It makes it easier for us to handle asynchronous or callback-based code. Basically, RxJS is defined as a library for composing asynchronous and event-based programs by using observable sequences. It provides one core type, the Observable, and operators like (map, filter, reduce, every, etc.) to allow handling asynchronous events as collections.[2].

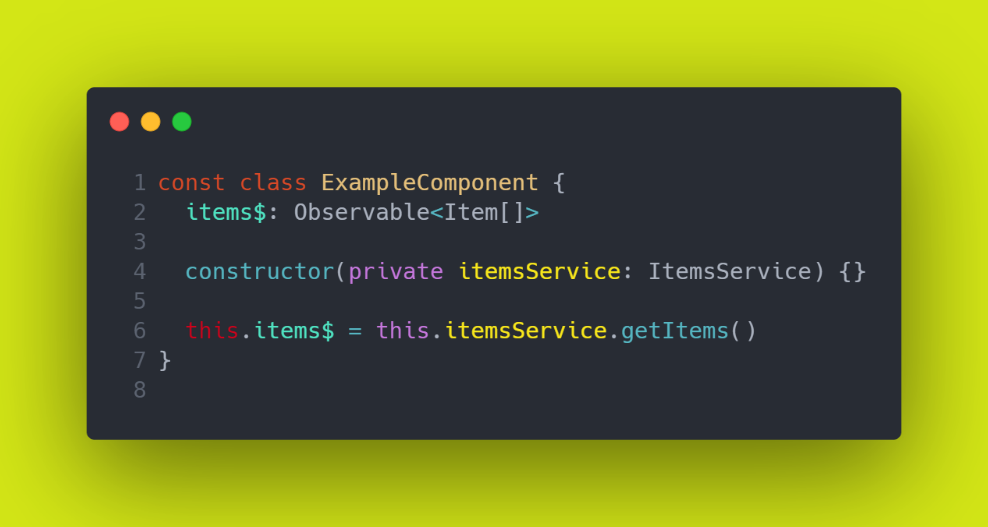


Figure 26 - Observable example - typescript file

Observables are quite easy to grasp. We can define an array of items as an observable. Then when we want to call a method from a service, we can assign it to the observable.

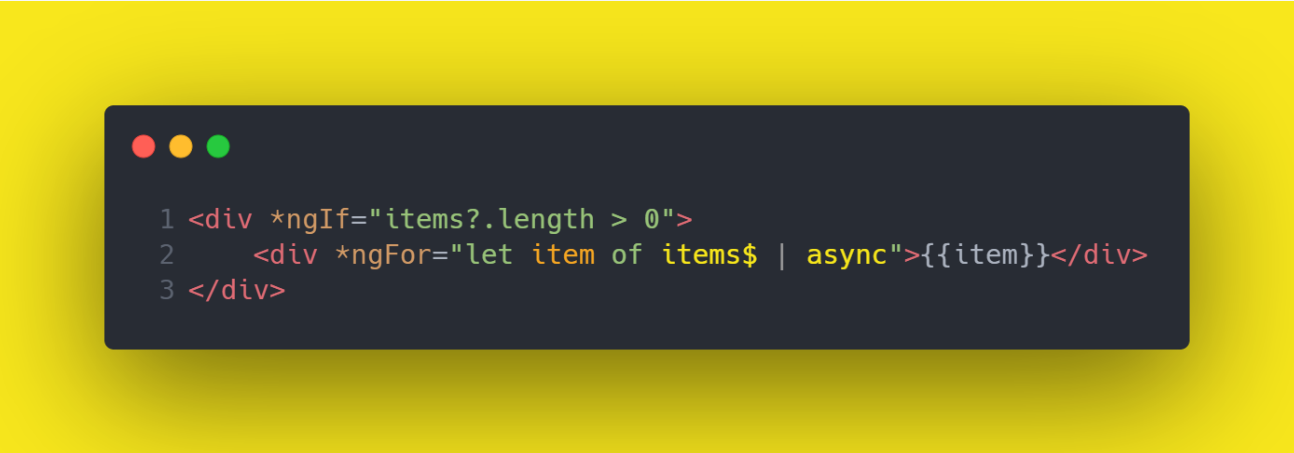


Figure 27- Observable example - HTML file

And then we can get all our items just by writing that observable in our html file. First, we check if items observable is more than 0, if not, the div will not render at all, otherwise, we loop through the items observable and render each item into a separate div.

### Axe core and Axe HTML reporter

Axe core is an accessibility testing tool for HTML-based websites. Axe core has a number of best practices that helps us identify common accessibility practices like ensuring every page has an h1 heading and helps us avoid any mistakes. Axe HTML reporter on the other hand, is the tool that creates an HTML report from axe-core library listing violations, passes, incomplete and incompatible results.

### Cypress and Cypress axe

We use Cypress in our project to write end-to-end testing.



Figure 28 - Cypress test to visit a page

This cypress test will run and open a localhost page. After this test is ran, it will run the next test which opens localhost and clicks a button that contains ‘type’ as text.



Figure 29 - Cypress test to click a button

# USER GUIDE

## 4.1 Software local setup, installation, and uninstallation guide.

Since we’re creating a web application, there’s no need to do any setup configuration to start using the application. All users got to do is just visit the website address and they can start using our application. However, if you would want to use the application locally (only people within our team can do that), you would need to fetch the application from GitHub, install needed dependencies and run the application locally using the given commands in the README.md file in the repository.

## 4.2 Description and identification of steps necessary to take in order to fulfil all functional requirements.

### 4.2.1 Login with ID-porten

Once a user visits the website, user will be greeted with this login page, for now we only have a login option with ID-porten only.

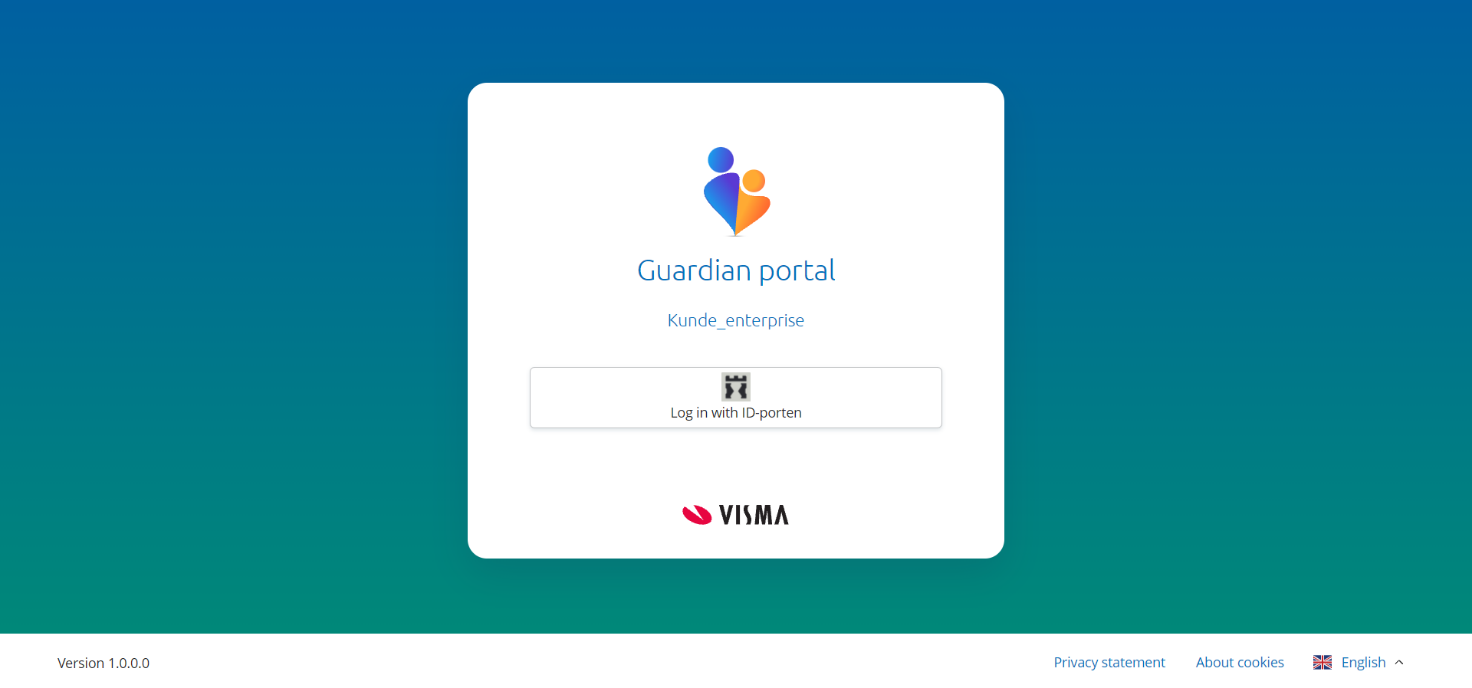


Figure 30 - Application landing page

This page is only shown for us, since this is a test application, so we click sign in with ID-porten again.

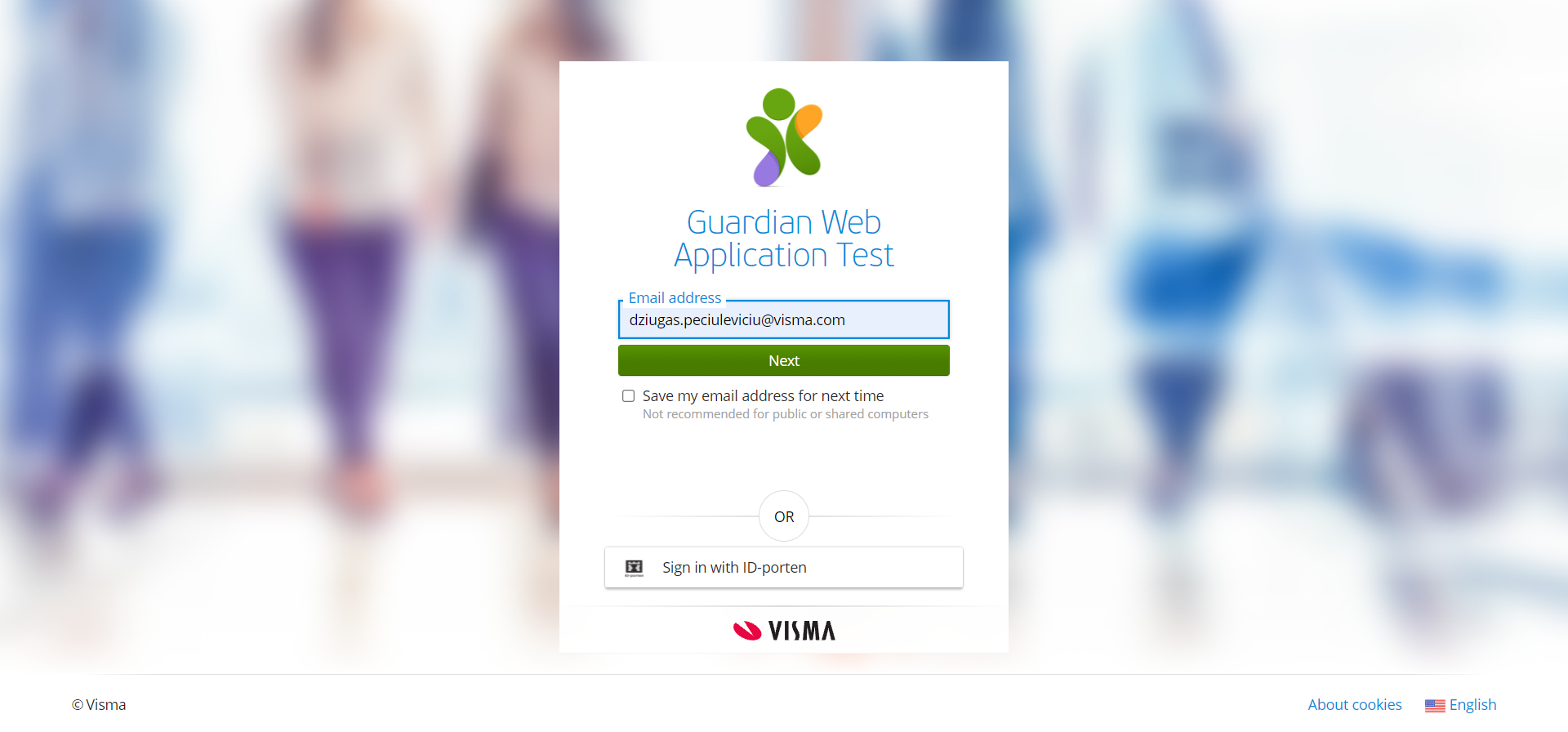


Figure 31 - Application login step

Once a user chose to login with ID-porten, a user will get to chose whatever service they feel comfortable logging in with into the system.

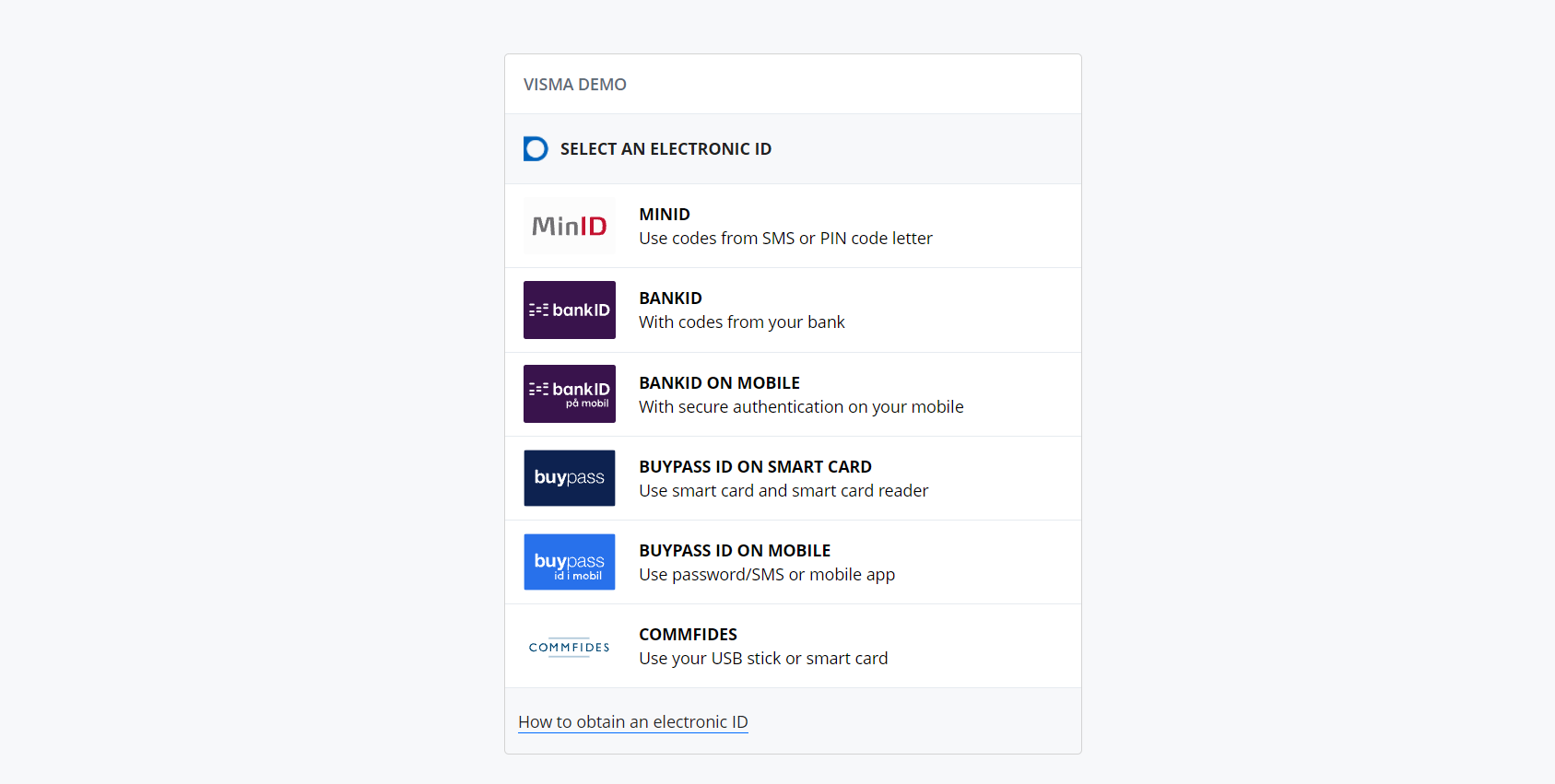


Figure 32 - Choosing how to login

Once a user has entered the correct login details, user will be greeted with the homepage of the application.

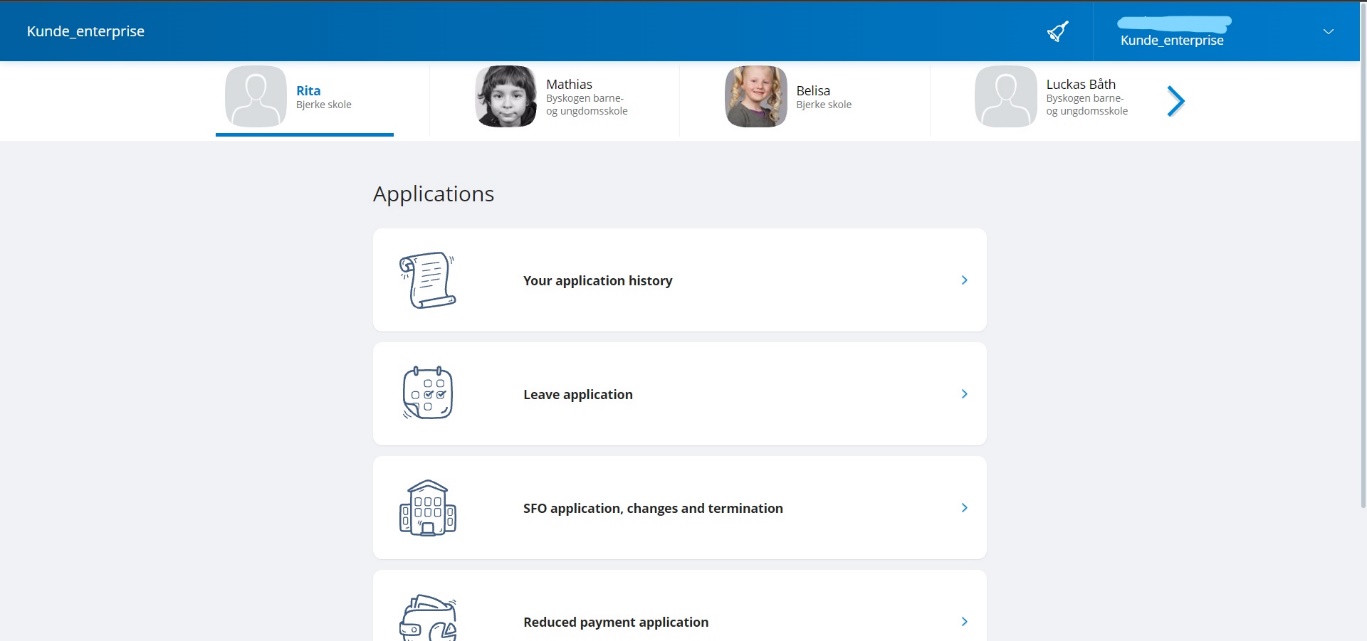


Figure 33 - Home page after login

### 4.2.2 Logout after clicking logout button

If user decides to logout, user can click onto the dropdown menu which user can find at the top right corner and click log out in the dropdown which will log the user out and bring them back to the login page.

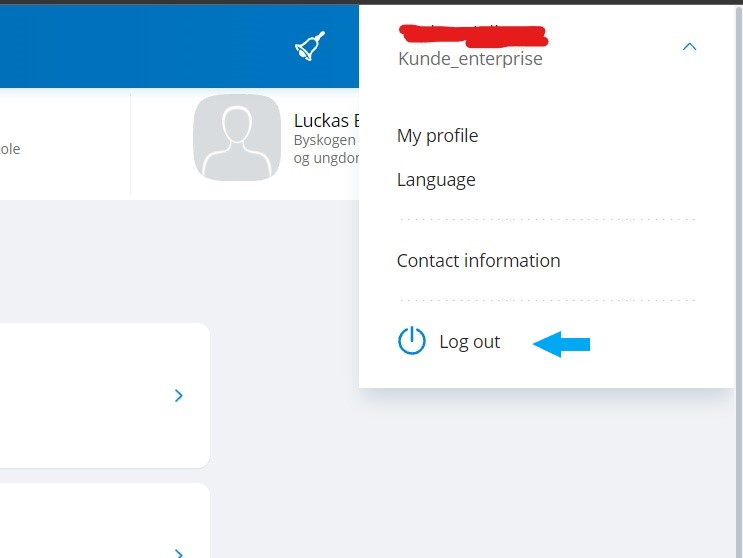


Figure 34 - Logout button in user dropdown menu

### 4.2.3 Should see all its children listed in a list and should be able to select one child at a time

When a user is logged in, user should be able to see children list just below the header.

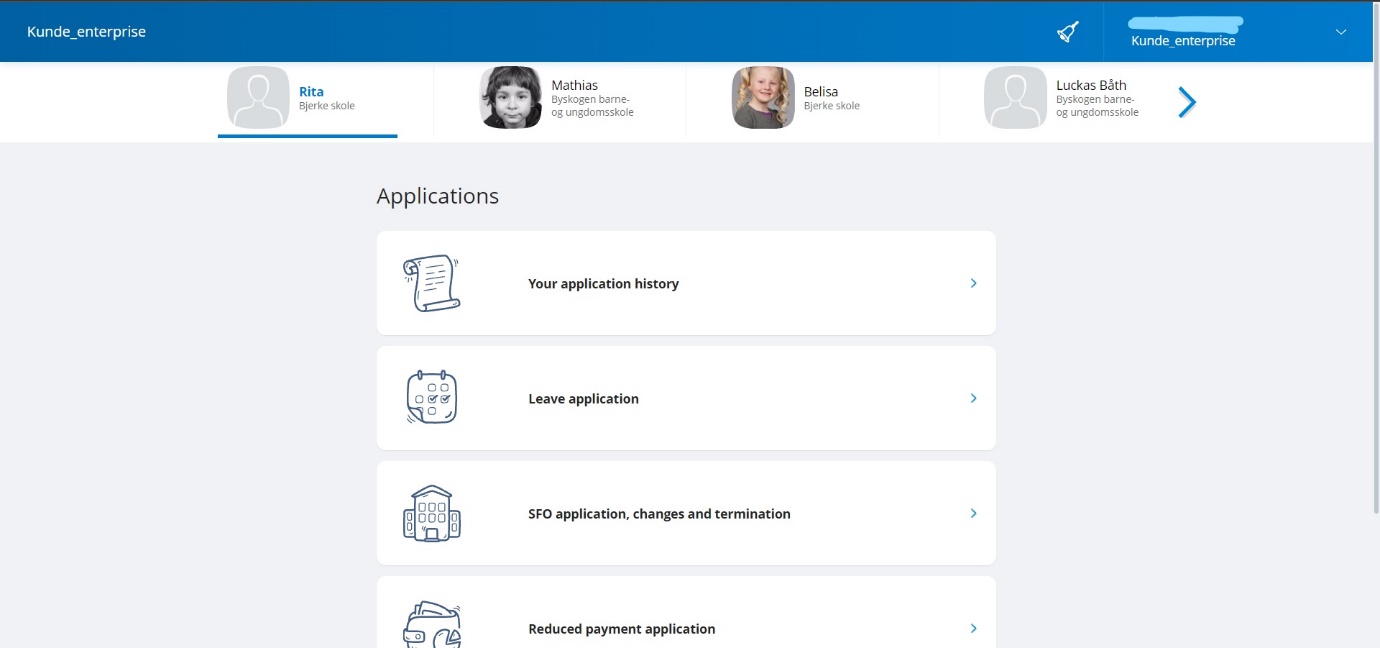


Figure 35 - Children list

### 4.2.4 Should be able to fill out and send forms

When a user wants to fill out a form for specific child, all they got to do, is select a child in the children list and then select the application they want to send. So, in this example, a user has chosen Rita as an active child and proceeded to fill in the form.

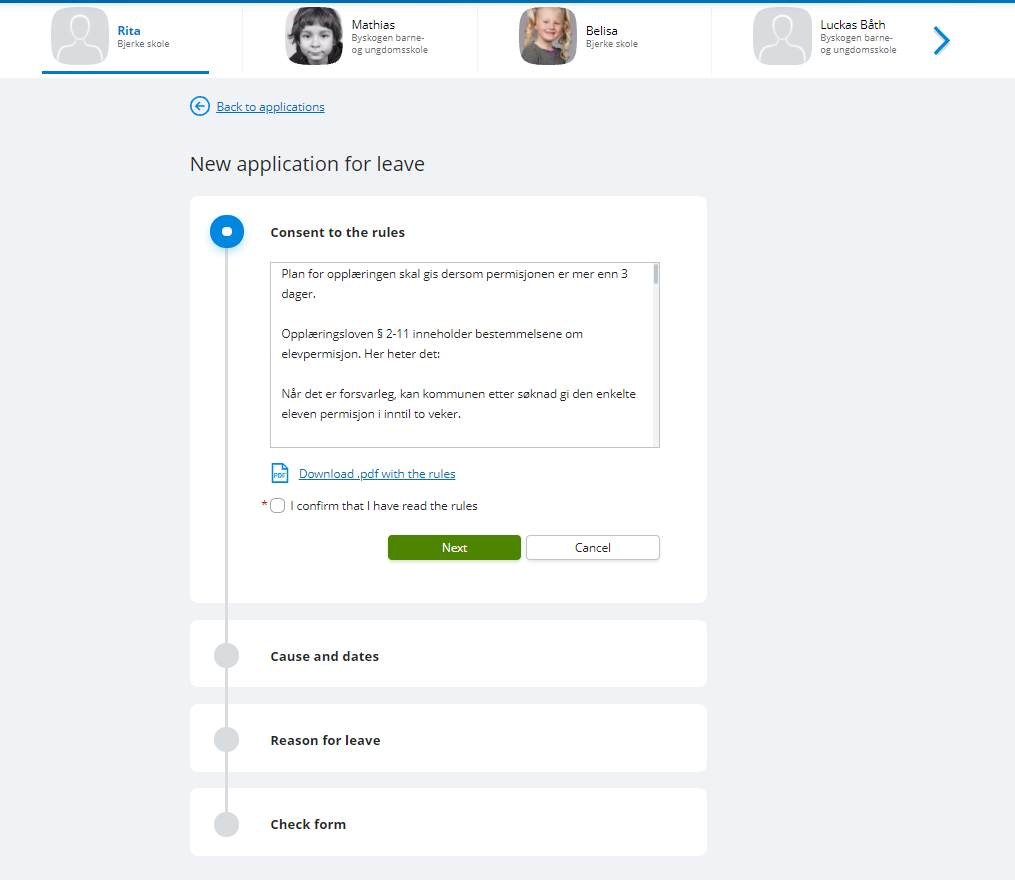


Figure 36 - First step of application form

User can decide to add more children to send the application for few children at a time. So, in this example a user has added more children.

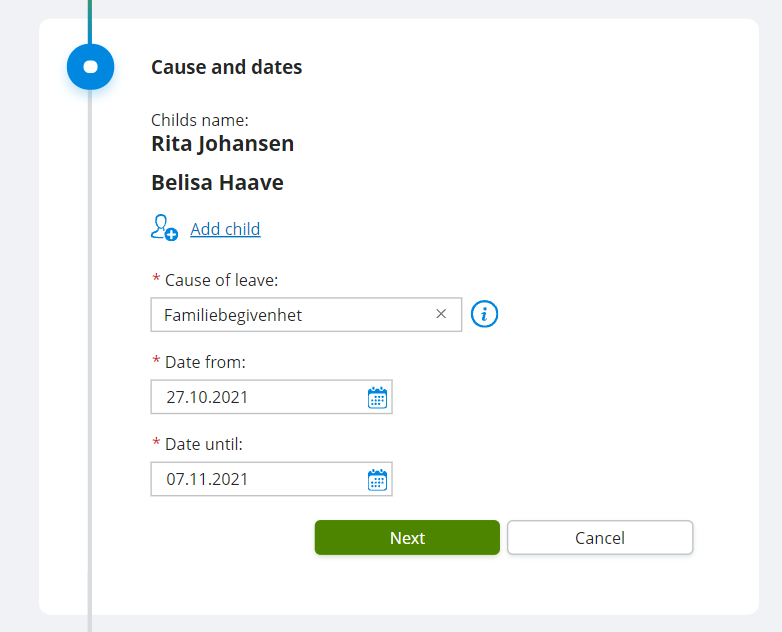


Figure 37 - Second step of application form

A user needs to provide a reason why they are applying for the application and upload necessary files to provide information.

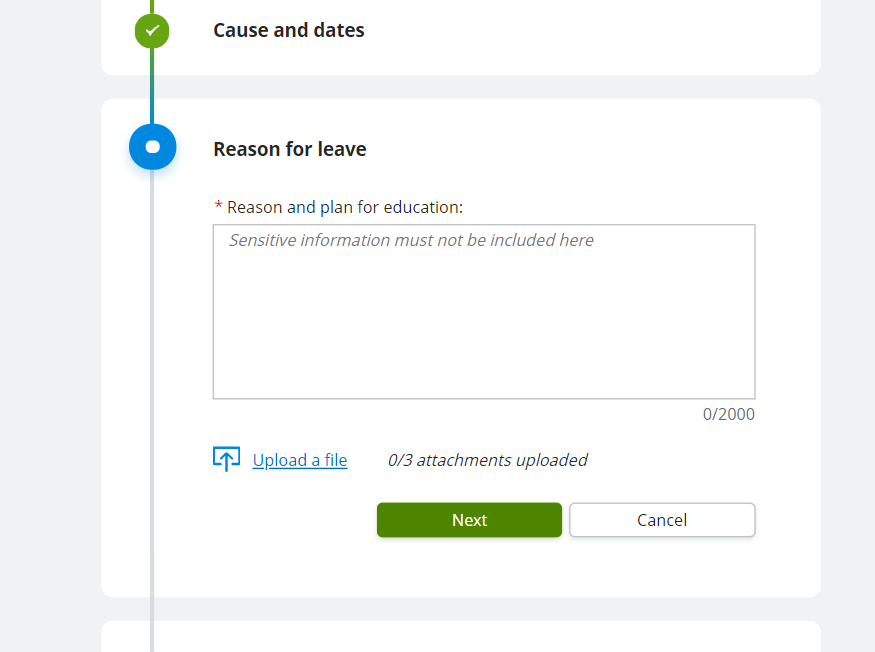


Figure 38 - Third step of application form

The last step of the application provides the user an overview of what information has been entered.

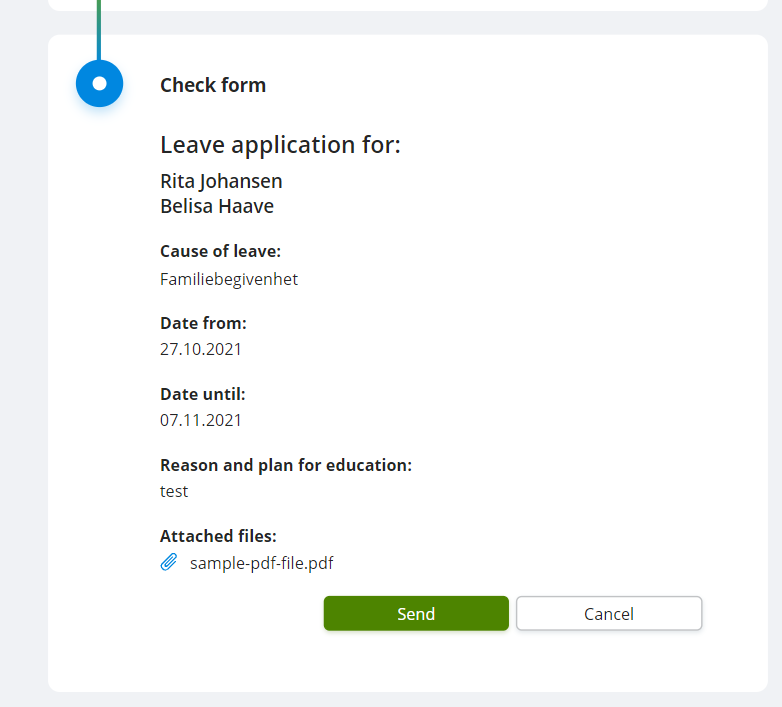


Figure 39 - Final step of application form - Form overview

Once a user confirms the application in the overview, and application has been sent successfully without any issues, user will be redirected to the success page.

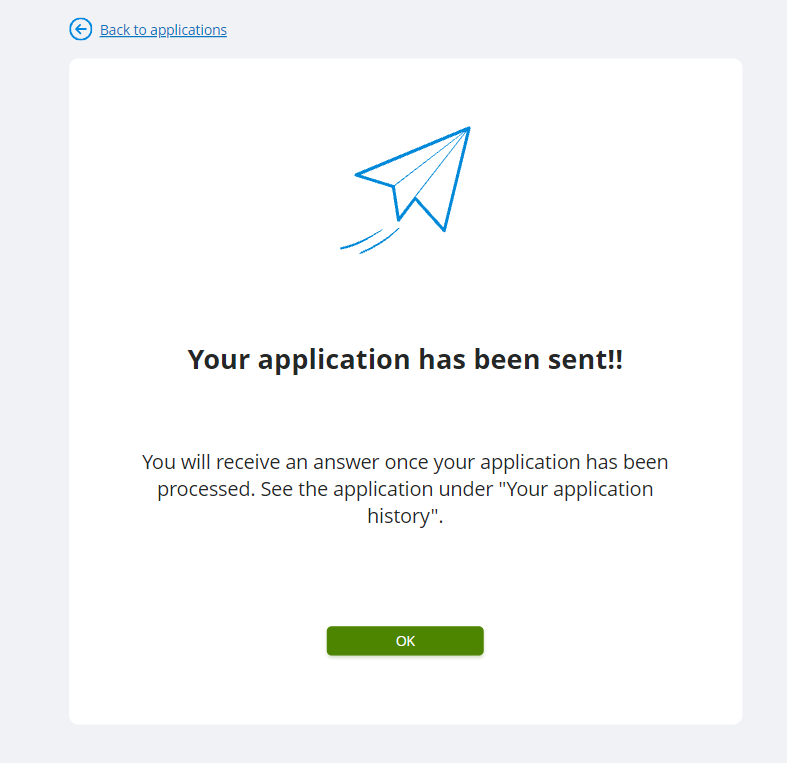


Figure 40 - Application sent success page

### 4.2.5 Should be able to add more children on leave application form to send for more than one child

When a user wants to add more children to send the form for few children at a time, they can click the add child button which will add children to the list, so when the form is going to be sent, it will send the form for a few children at a time.

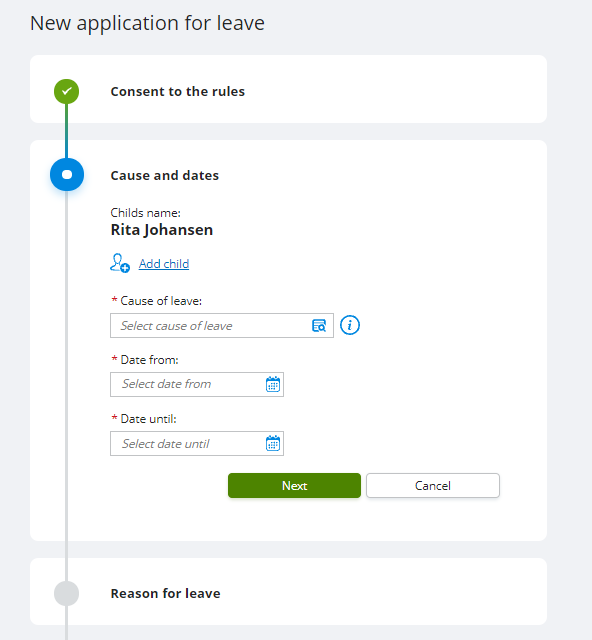


Figure 41 - Add additional children button

When a user clicks the add child button, they will see the children that are available and go to the same school as the active child.

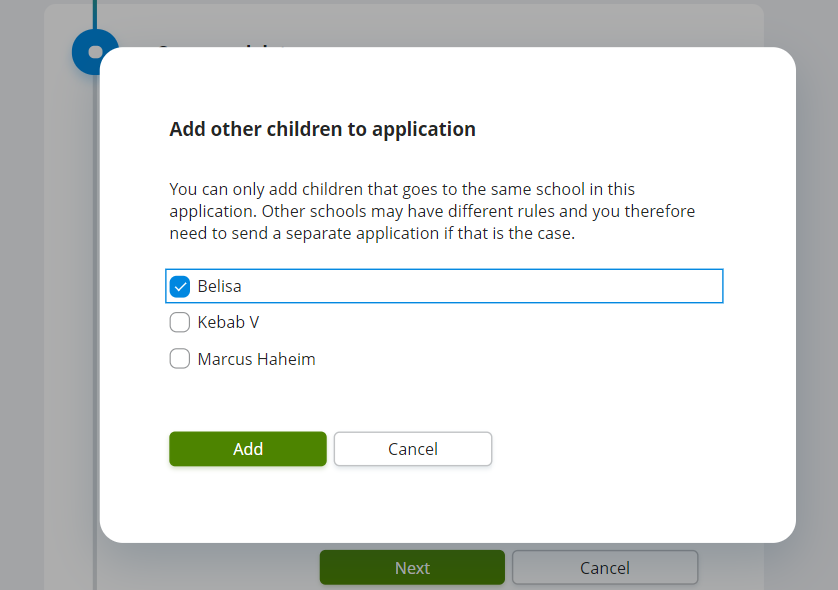


Figure 42 - Adding children popup modal

When a use confirms and adds a child, modal automatically closes, and the system adds a child into the list to be sent.

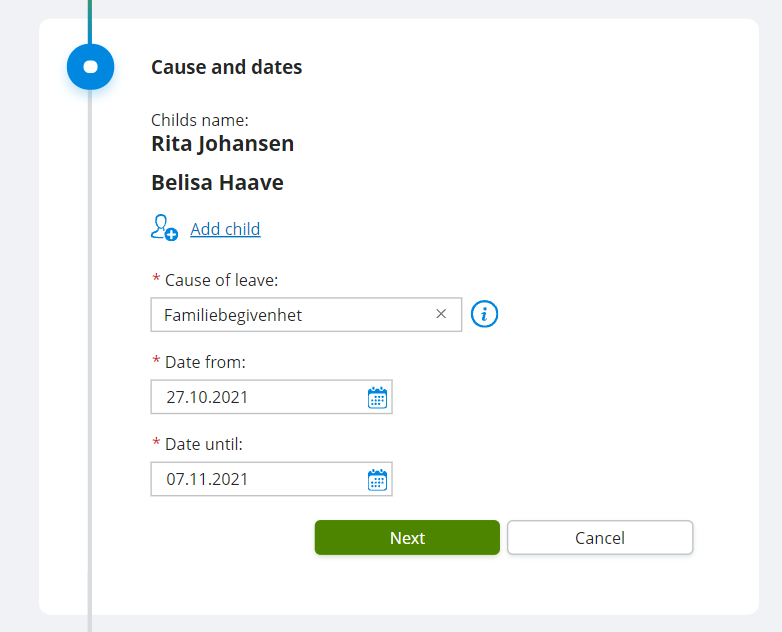


Figure 43 - List with added children

### 4.2.6 Should be able to see sent applications

When a user has sent an application successfully, user will be greeted with a success message.

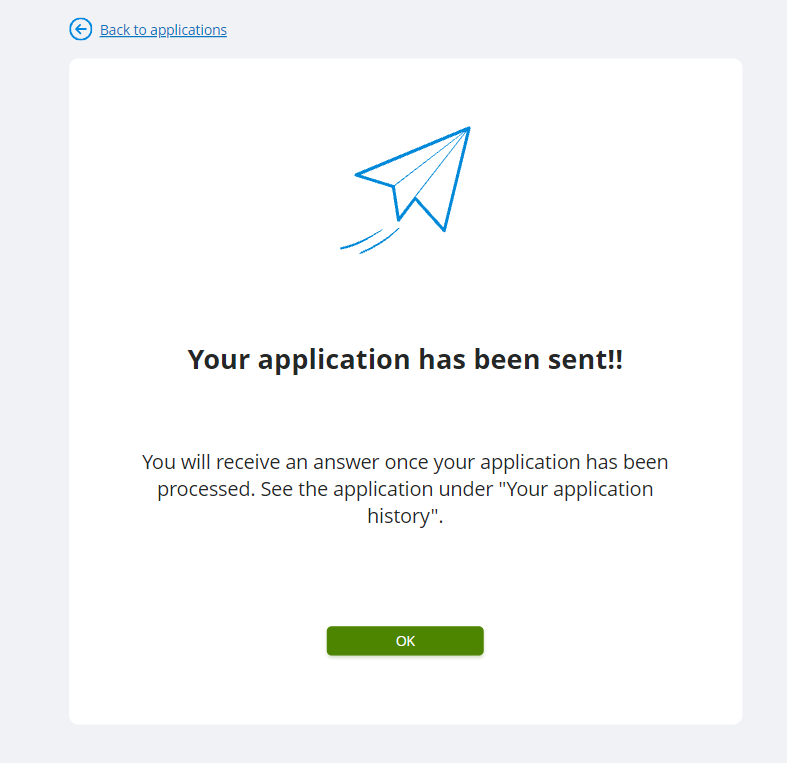


Figure 44 - Application sent success page

Then user can go into applications page and click application history button.

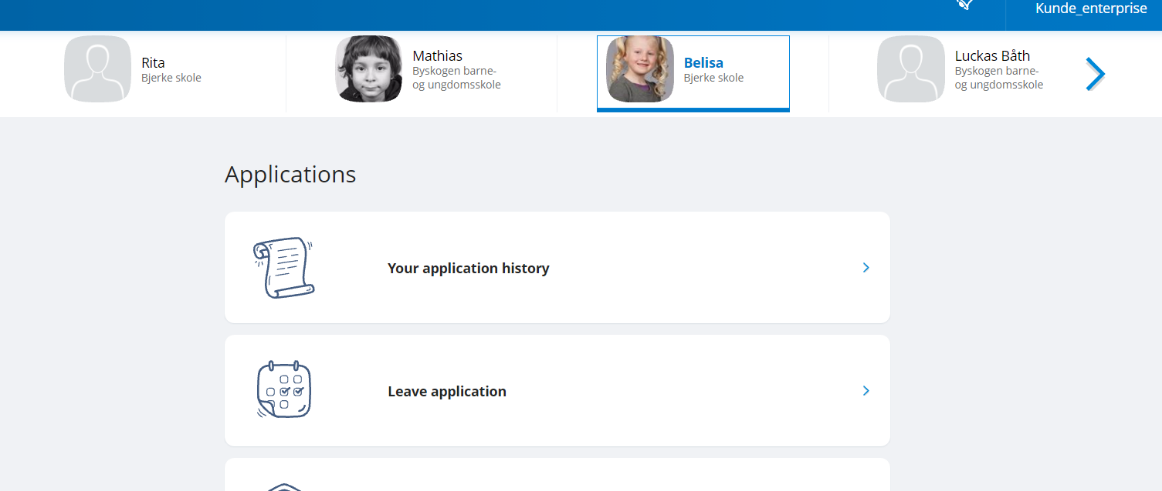


Figure 45 - Application history button

And when user opens an application history button, the system will redirect user to application history page, where user can see all sent applications. The application we just sent can be seen at the top of the page.

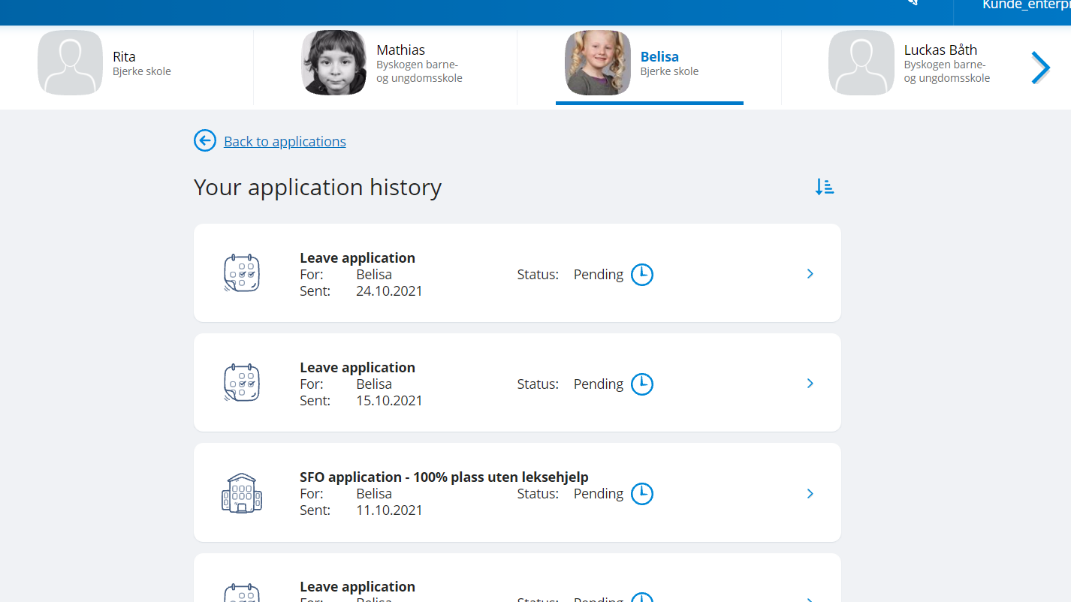


Figure 46 - Application history page

### 4.2.7 Should be able to sort sent applications from newest to oldest and vice versa

User can also sort the applications from newest to oldest by clicking the blue arrow button at the top right above all the applications. Newest applications are shown by default.

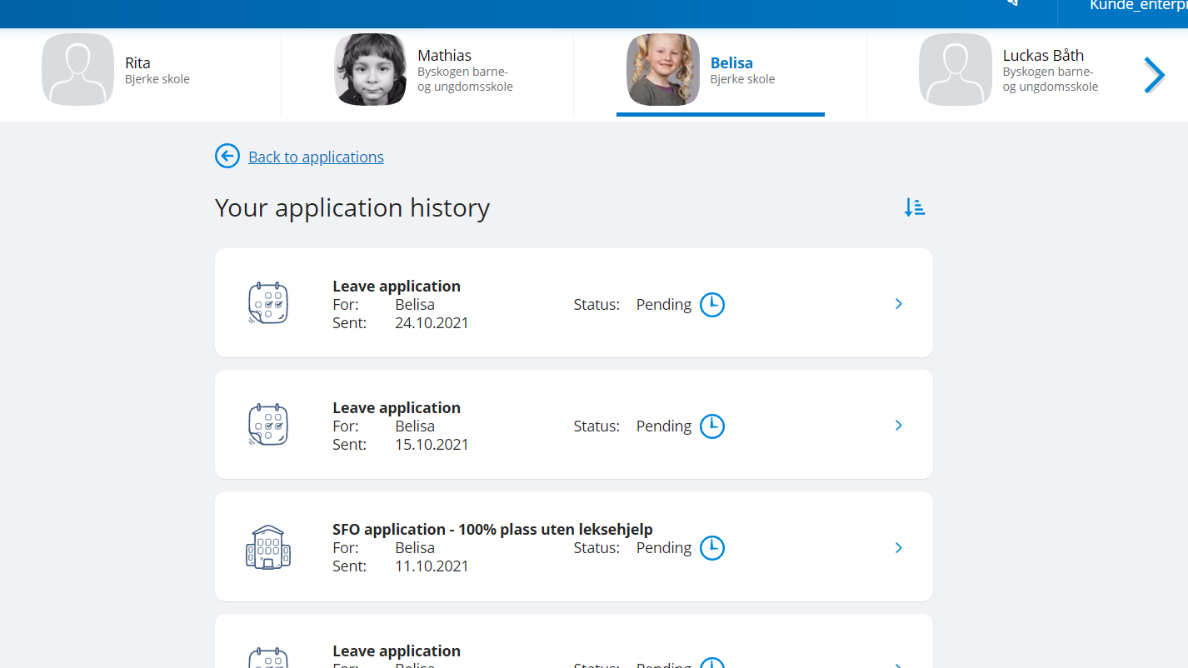


Figure 47 - Application history sort button – display newest

When a user clicks it, the system will show oldest applications at the top.

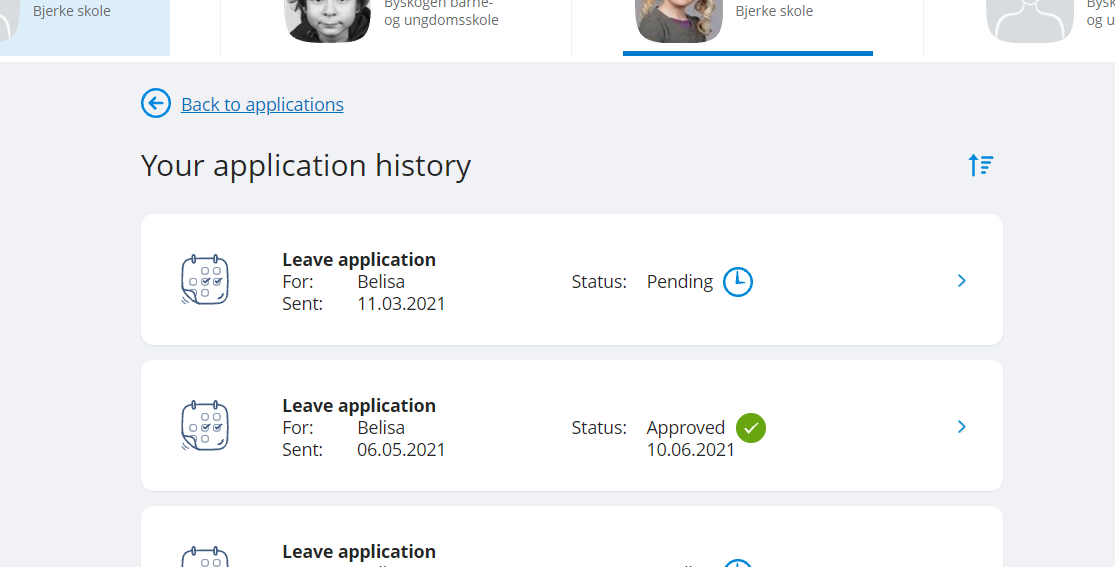


Figure 48 - Application history sort button – display oldest

### 4.2.8 Should show error messages if mandatory fields are left empty

If a user has missed some fields during the application process, user will be provided with error messages below the fields that error has occurred.

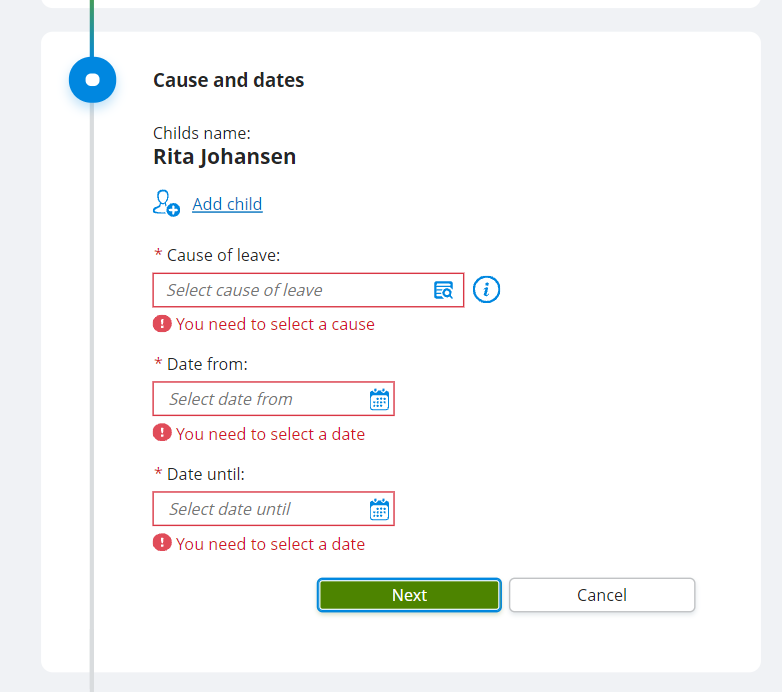


Figure 49 - Error messages - missing fields

### 4.2.9 Should display warning if user tries to leave application process

If a user is trying to leave the application when it has been started but has not yet been finished. Then if the user is trying to leave the page, user will be provided a confirmation dialog with a warning that all entered information will be lost.

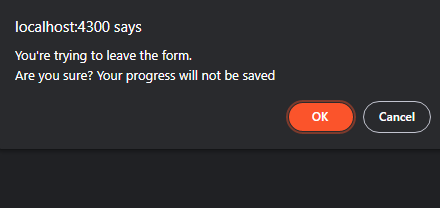


Figure 50 - Warning when trying to leave the form

### 4.2.10 Should display success page if form was sent successfully

When the form has been sent successfully and no internal errors occurred, a user will be redirected to success page after clicking the send button.

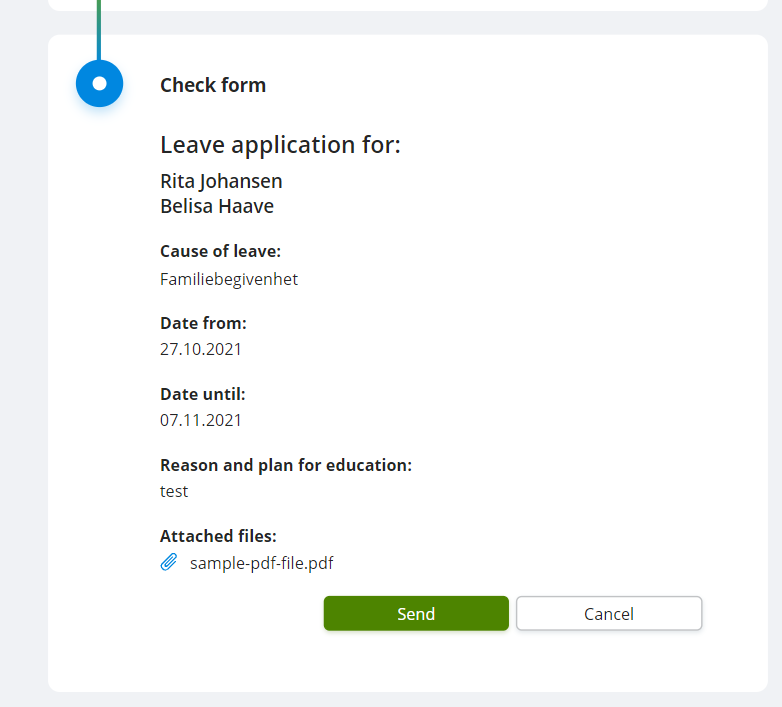


Figure 51 - Application form overview step

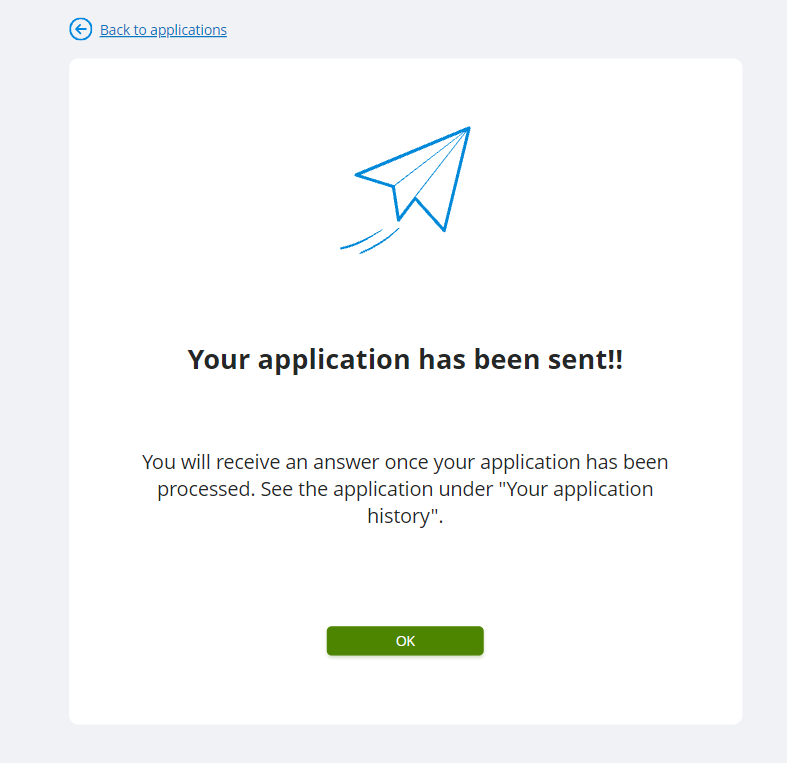


Figure 52 - Application sent success page

### 4.2.11 Should be able to navigate and see contact information for schools and municipality

To navigate to the contacts page, all user has got to do, is click on the user dropdown menu at the top right of the screen and click ‘Contact information’.

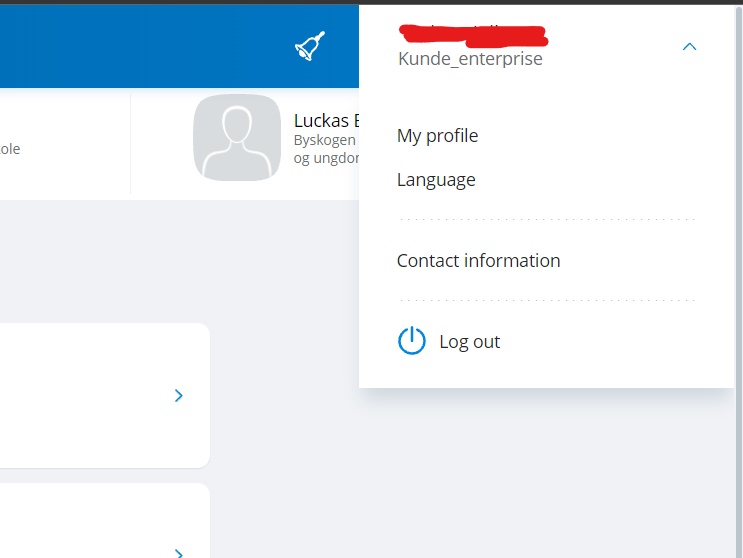


Figure 53 - User dropdown menu

When clicked, the user will be redirected to the contacts page, where the user will be presented will contact information details like municipality and all the school contact data that children go to.

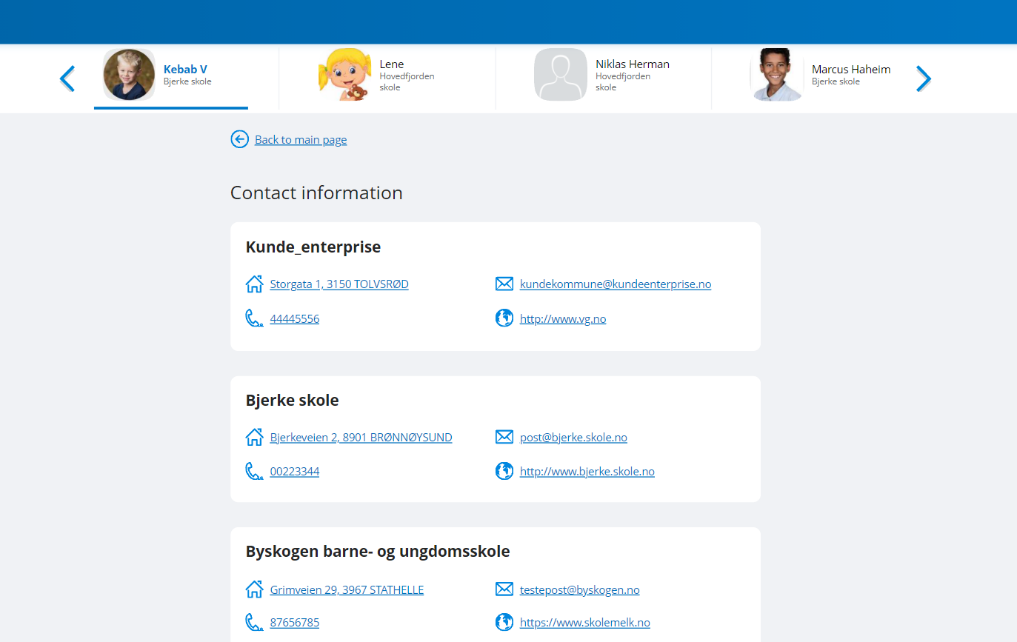


Figure 54 - Contact information page

# CONCLUSIONS AND FUTURE STEPS

For the conclusions, I would say that the practice went by perfectly. All the functional requirements were implemented as planned and now we’re just improving the accessibility of the application itself.

The most difficult part was I’d say learning new tools and how they work. Since all teams use different tools for, their projects depending on their needs, when I began working in a team, I had a lot to learn. From libraries we’re using in our project, to various accessibility nuances and testing tools.

To sum up the practice, after the internship and when I began working on the project with other people in a team, these past couple of months were a success. I got comfortable during this time. Less and less time is spent on tasks, harder tasks don’t seem as daunting as it seemed when I started working in a team. I learnt how the development process goes when working at a company. During this time, I also learnt more best coding practices and various tools I could use in my future projects.

# LITERATURE

1. Visma Nordic Cool web library. [online] Available at: <<https://ux.visma.com/weblibrary/latest/development/documentation/docs/alerts.php>> [Accessed 30 October 2021].

2. RxJS - Overview. [online] Available at: <<https://www.shopify.com/encyclopedia/what-is-ecommerce>> [Accessed 30 October 2021].

3. Angular documentation <<https://angular.io/docs/>/>

4. SCSS <[https://sass-lang.com/](https://sass-lang.com//)/>

5. TypeScript <<https://www.typescriptlang.org/>/>

6. Jira < <https://www.atlassian.com/software/jira//>>

7. Git < <https://git-scm.com//>>

8. GitHub < <https://github.com//>>

9. Wave – Web accessibility evaluation tool < <https://wave.webaim.org//>>

10. axe DevTools < <https://www.deque.com/axe/devtools//>>

11. Cypress < <https://www.cypress.io//>>