**Professional Task 3**

**Field of expertise :** Software  
**Task :** Implement

**Level :** 3

**Description of task :** *Build and make available a software system in line with existing systems and on the basis of the designed architecture, using existing frameworks.*

*Using test automation when performing tests.*

**Description of your activity with reference to your prove that demonstrates you have reached the desired level:**

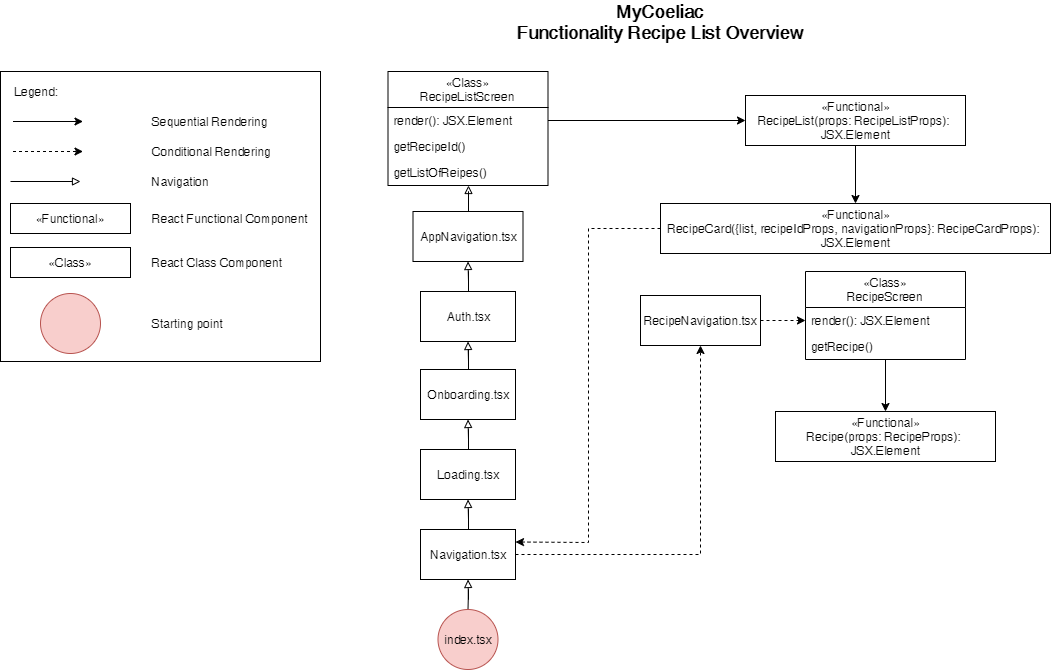
For the project I implemented the component that displays a list of recipes. The list contains cards that shows a preview of a recipe. When a recipe is clicked on, a detailed page containing the instructions to prepare the recipe will appear.

For technology we used React Native with Typescript. The structure of a project in React (Native) differs from a basic web app in HTML/Javascript/CSS. Instead of separating the view (HTML) from the logic (Javascript), in React the view is rendered with Javascript. React has the concept of a ‘component’ which is usually a Javascript class or function that returns a HTML element.

There are two types of components: Functional components and Class components. Functional components are just plain HTML elements and so they are stateless. Then there are Class components which are broader than functional components. It can have a state and props. It is a class so it can be initialized and call a constructor.

A React component can have a State and Props. Only a Class Component can have a State because Functional Components are supposed to be stateless. When a component has a State, it means that the component can change or be updated. Both a Class Component and Functional Component can have Props. Props stands for properties and can be used to receive or pass data from one component to another.

A big difference in using React with Javascript and Typescript is that Typescript enforces the coding standard more strictly than Javascript. For example, when passing parameters or defining objects and variables, Typescript will always require you to define the type of data you are passing. This is not the case with Javascript.



The starting point of the application is the index file. This index file calls the Navigation file which uses the React Navigation package to route the app navigation. The root navigation of the app is built with the createSwitchNavigator() function. This navigator only renders one screen at a time, which is exactly what we want for the first part of the app up until the Home screen. It does not deal with back navigation, so it is useful in cases that going back to a previous screen is not necessary.

Onboarding and Loading are only shown once. Not only can screens be loaded, but navigation files can also be loaded, which turns it into a nested navigator. A nested navigator is used in instances that you have to switch to another type of navigator or you’re dealing with multiple navigators. From line 14 to line 17 shows how a navigator is passed instead of a screen.

The Recipe List component is the first component shown right after the app navigated from Login to the Home screen. The Recipe List Screen is a Class component. We consider the ‘screen’ to be a container, so that is why all of our screens are Class components. These containers can be made up out of multiple Functional components and are always the parent of components. A Functional component is never rendered directly as a screen. A Functional Component is just a plain Javascript function or HTML element and is why we consider it individual HTML elements.

So, the Recipe List component renders the list of recipes. This list is retrieved from the server with the function getListOfRecipes() like on line 108 in the parent of Recipe List which is the Recipe List Screen component. Because the list of recipes can change, it is saved in a State called ‘recipes’.

The Recipe List component has a prop called ‘recipes’ in order to receive data, in this case, a list of recipes, from the parent. You can see this on line 174 in Recipe List Screen component. Now that the Recipe List component has the list of recipes, it’ll then use that data and display it with a Recipe Card component. The Recipe List component itself uses a Flatlist (React Native component) that has a predesigned mobile-friendly table or list. This list has properties like ‘data’ and ‘renderItem’ to render an array of data, which in this case are Recipe Card components, as shown on line 25 in Recipe List component.

The data, list of recipes, that was passed from Recipe List Screen component to Recipe List, is again passed to the Recipe Card component in the prop ‘list’ like on line 26 in Recipe List component. The Recipe Card component uses this data to display the properties of each recipe.

The Recipe Cards are clickable and when clicked, it’ll navigate to the detail page of that particular recipe. In order to display the right recipe, the Recipe Card needs to pass the right ID to the Recipe Screen component, so that it knows which recipe it has to pass on to the Recipe component. This is done with the function getRecipeId(id: number) on line 28 in Recipe Card component.

Earlier was explained how the list of recipes get passed to Recipe Card component, which means that an ID belonging to a particular recipe can be retrieved. On line 81 in Recipe Card component can be seen how that data is passed into the parameter of the getRecipeId(id: number) function. The ID is then saved in the prop recipeIdProps.

On line 81 you can also see the conditional function that if an ID exists and has been passed to the getRecipeId(id: number) function, that the navigateToRecipe() function gets called. Via navigation is also how the recipe ID gets passed to the Recipe Screen component, which is the container or the parent of an recipe detail page.

On line 39 is shown how you navigate from the Home screen to a Recipe detail page. It shows how to route via a nested navigator. First it routes to Recipe, which is the route name of the Recipe Navigator as shown on line 17 in Navigation file. Then to actually use the Recipe Navigator and navigate to a screen defined within this navigator, in the ‘action’ parameter of the function on line 39 of Recipe Card component, you can refer to the navigator route with ‘route name’ parameter. According to the documentation, the action parameter runs a particular action in the child router, if the screen or parent is a navigator, so this is exactly what you need if you wish to navigate within a nested navigator.

To be able to use React Navigation properties in React with Typescript, you have to explicitly define the type of navigation properties you are using. This is passed down from the parent to all of its child components, because the type of property cannot be optional and therefore has to be passed to every linked component. On line 21, 26 and 39 in Recipe Card component is shown how a React Navigation can be defined and used.

To send the recipe ID through navigation to the Recipe Screen component, the params parameter in the navigate function is used. With this parameter you can pass any data along to the next screen or component. So, the recipe Id is passed as recipeId.

Once navigated to the Recipe Screen component, you can access this parameter with the getParam() function of React Navigation. Once the recipe ID is received, it is then used in the findRecipe(recipe: any) function to find the right recipe that is to be displayed on the detail page.

The Recipe Screen component fetches its own list of recipes and searches for the recipe that was clicked on, with the recipe ID sent by the Recipe Card component. As shown on line 59 in Recipe Screen component, it uses the Recipe component to pass the right recipe to. The Recipe component uses the recipe data that was passed to render the detail page.

Evidence:

Recipe List Screen Component: <https://github.com/DESQOL/MyCoeliac/blob/master/src/screens/home/RecipeListScreen.tsx>

Recipe List Component: <https://github.com/DESQOL/MyCoeliac/blob/master/src/components/organisms/RecipeList.tsx>

Recipe Card Component: <https://github.com/DESQOL/MyCoeliac/blob/master/src/components/molecules/RecipeCard.tsx>

Recipe Screen Component: <https://github.com/DESQOL/MyCoeliac/blob/master/src/screens/recipe/RecipeScreen.tsx>

Recipe Component: <https://github.com/DESQOL/MyCoeliac/blob/master/src/components/molecules/Recipe.tsx>

Recipe Navigation Component: <https://github.com/DESQOL/MyCoeliac/blob/master/src/navigations/RecipeNavigation.tsx>