Contents

List of Figures

List of Tables

Acronyms

A

D

J

N

O

R

U

1. Introduction

1.1. Motivation

Awareness and interest in healthy nutrition is on the rise. The demand for organic and regional food is larger than ever. Ethical reasons like climate change or animal welfare aren't the only driving factors of this development. And this development isn't restricted to athletes and wealthy households either. There has never been a larger variety of dietary plans and advice.

Technical innovations helped to shorten the time spent in the kitchen and enabled passionate amateur chefs to prepare dishes that used to be time consuming, in a convenient and fast way. The availability of ingredients increased immensely. Due to globalization and the industrialization of agriculture, the majority of food items became affordable for most people. The product range for vegans, vegetarians and eaters suffering from allergies and intolerances has also broadened.

With all these positive developments, one could come to the conclusion that it has never been easier to have a healthy, personalized diet. Unfortunately, for some reason not everyone seems to use this opportunity and sticks to old habits instead.

1.2. Problem statement

When we were collecting opinions and practices regarding eating habits in our circle of friends and acquaintances, as well as online forums, we often came across similar statements.

A surprising majority of the respondents were quite aware of their eating behavior. Moreover, most of them had a rough understanding what constitutes an unhealthy diet and were already trying to avoid fast food. Therefore, we decided to target this particular group of people that is motivated to improve their nutrition.

By cooking at home, instead of eating out or ordering food, they already took the first step towards a healthier lifestyle but even then it's important what ends up on the plate. At that point, there is often an obstacle, hindering those people to further improve their diet, leading us to their pains.

Pains

According to a study of the Techniker Krankenkasse from 2017 (Katja Wohlers 2017), only 2% of respondents don't have interest at all in healthy nutrition. But on the other hand, only 10% claim to eat healthy enough.

The rest is stuck in the middle, seeing themselves faced with obstacles. This can be the lack of time (56%), motivation (45%), money (30%), or knowledge (25%). The latter is understandable, since the wide variety of contradicting opinions in the complex subject of nutrition can have a discouraging effect. Gathering new recipes is also quite time consuming. Without enough cooking skills this won't feel rewarding either, because trying out something new might end up in a failure. This inconvenience paired with personal habits let the potential users stick to the same 5-7 dishes they know.

Gains

Our persona would like to live a healthier lifestyle but it shouldn't feel like a sacrifice. Suggested alternatives and changes should feel equally good and not inferior. It wishes for convenience instead of the usual handiwork, commonly going into keeping track of one's diet. Additionally, there is a need for clear, trustworthy guidelines rooted in scientific research.

2. Description of Idea

Foodo - Your Loyal Companion on Your Way to a Healthier Lifestyle

Foodo offers the user a collection of standard recipes. A subset of these standard recipes represents the user's recipe stock, namely the dishes the user is used to cook regularly. The main goal of Foodo is to iteratively extend and to improve the healthiness of the user's recipe stock in a personal way. This is achieved by enabling the user to substitute the recipes' ingredients with suggested healthy substitutes leading to healthier variations of the user's recipes. Moreover, Foodo's collection of standard recipes offers the user the opportunity to extend his recipe stock by trying out new recipes.

As nutrition is very individual, Foodo is a personal companion and adjusts its suggestions to the characteristics of the user. The user can set personal preferences. These include his lifestyle, e.g. Vegetarian, Vegan or Low Carb, his nutrition goal, e.g. eating healthy, reducing weight or gaining muscles, his dislikes and his allergies, e.g. Gluten, Lactose or Fructose. After setting these personal preferences, Foodo takes them into consideration and finds the best personal suggestions for the user's recipe stock. If the user accepts a suggested substitution, the recipe gets updated and becomes part of his personal recipe stock. Moreover, the user can give feedback on the substitution after he has cooked the recipe which has been modified with the substitution. Foodo will remember his feedback and incorporate it in its future suggestions. Like this, the suggestions get even more personalized over time.

Foodo offers different channels to interact with the user. These are a webapp and an Alexa skill. The webapp offers detailed information on the available recipes, the user's recipe stock, his preferences, the recipes, their nutrition values and statistics on achieved improvements by using Foodo. This channel targets the sophisticated user and is additionally used for fulfilling administrative tasks like inserting the user's preferences. The Alexa skill focuses on the substitution process and on supporting the cooking activity. Therefore, it does not offer statistics or too detailed information but it guides the user through the core functionalities in a supportive way. This channel targets the casual user and also allows the usage of Foodo while cooking as it only requires voice for the interaction and thus does not interfere with the cooking activity.

2.1. Pain Relievers

Foodo relieves several user pains. Firstly, Foodo relies on a user-centric approach which is compared to a recipe-centric approach more personal and less frustrating for the user. One aspect of the user-centric approach is that the required user input is reduced to a minimum.

Another aspect is the Alexa integration which allows an engaging user interaction through voice commands which reduces the time overhead. Furthermore, Foodo lowers the complexity of finding the right nutrition for the user. Through specifying a nutrition goal, the user does not need to think about how to achieve this goal but rather just think about what goal the user wants to achieve. Foodo also reduces the complexity of finding fitting substitutions by suggesting the best suitable substitutes to the user. Apart from that, the user does not have to search for new recipes anymore if he wants to extend his recipe stock because Foodo suggests the user new recipes from its standard recipe collection. This saves the user a lot of time. Moreover, Foodo enables guided cooking experiments and thereby takes the user's fear of bad results which can occur when the user varies the ingredients of recipes without guidance.

2.2. Gain Creators

Foodo offers several gain creators to the user. The most important gain creator for the user is that its recipe stock overall gets continuously healthier and gets extended over time. The incremental improvements of the amount and the variety of the user's recipe stock establish an awareness for a healthy nutrition for the user. By cooking healthy and getting suggestions for substitutions tailored to the user's preferences, the user also gains knowledge about healthy nutrition. Moreover, Foodo improves the user's cooking experience, as the Alexa integration supports the cooking activity and offers a more convenient interaction than having to use a smartphone which might be difficult while handling ingredients and cooking.

3. Features

About us

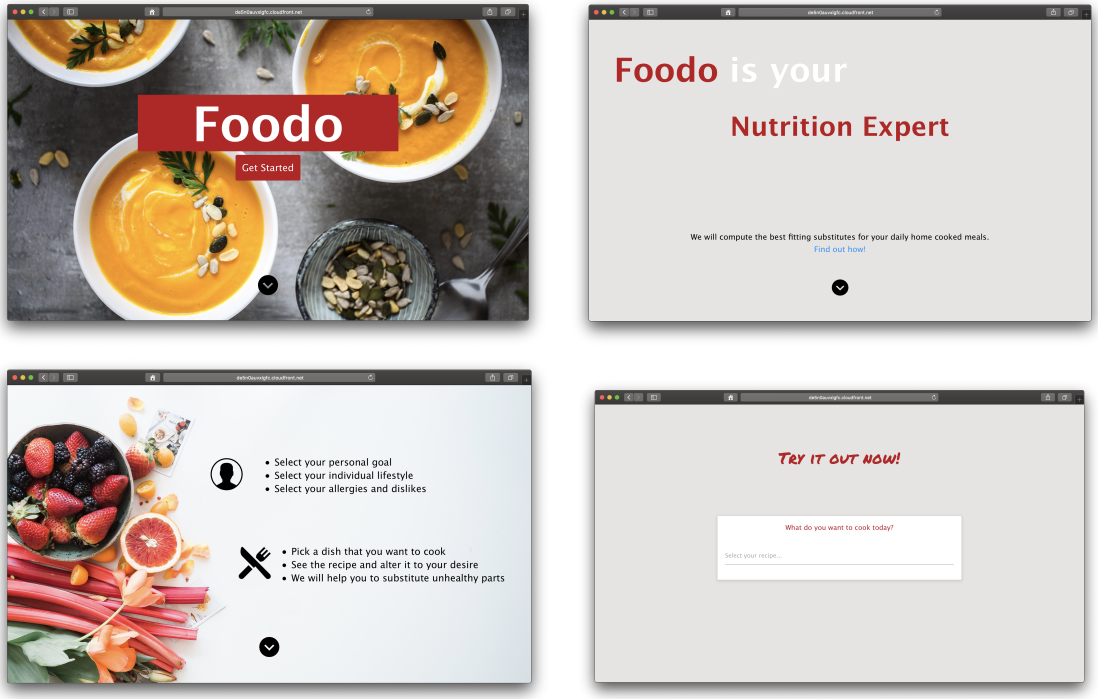

Figure 1 About-Us

Initially, a page visitor starts on our `About-Us` page which is also accessible by clicking on the `About-Us` index tab in the navigation menu. This page is separated into four subpages and mainly serves the purpose of informing the user about Foodo's services. If a user decides to give Foodo a try by either clicking on the `Get Started` button on the first subpage or by selecting a recipe he wants to cook on the last subpage (see figure 1), he will be forwarded to our `Login/Registration-Page` (see next section).

Login/Registration

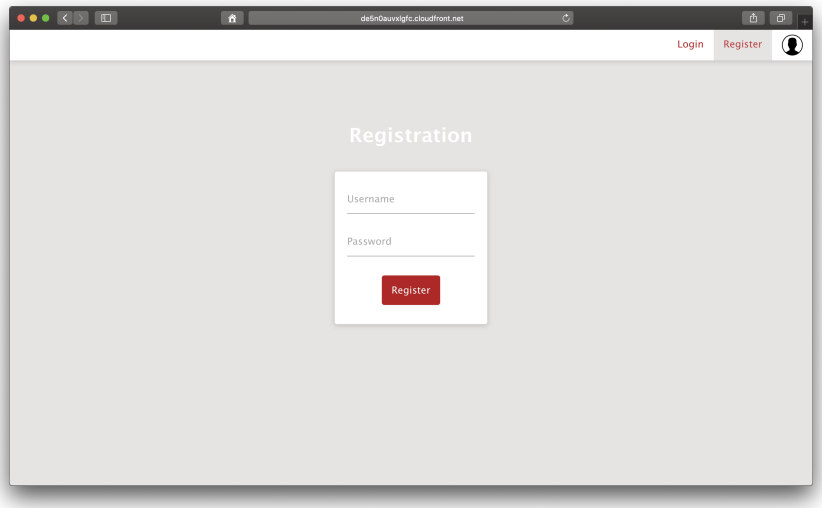

Figure 2 Registration

Before a user can use Foodo's services, he first needs to create an account. Therefore, the user will always be redirected to the `Login/Registration-Page` (see figure 2) when the user tries to call a service URL or clicks the mentioned button or recipe selector in the previous section (`About-Us page`). By inserting an individual username, which has not been taken already, and a hopefully secure password, the user can create a new account. The user also has the option to login with an already registered account by clicking on `Login` in the right upper corner and inserting his previously created credentials in the following form.

Navigation bar

Figure 3 Navbar

In the previous sections, there was no navigation bar at the top of the screen, as one could see in the figures. This was mainly a aesthetic design decision but also hides unavailable options for non-authenticated users. Now, after a user has logged into his account, the navigation bar is visible, shown in figure 3. By clicking on the profile icon in the upper right corner, a menu panel will be shown with additional navigation options for the user. Overall, he has the following navigation options:

- **Home**: The main page of Foodo, described in the next section
- **My Profile**: The user's settings and preferences regarding food items and nutrition; described in section Profile Page.
- **Statistics**: The user's improvements depicted with graphics and values; described in section `Statistics`.
- **Preferences**: This will open another small menu panel with two submenus. Clicking on `Languages` will again open another small menu panel, where the user can choose between the offered languages (currently only English and German). Clicking on `Password` will direct the user to a form, where he can insert his new password.
- **About**: This will direct the user to the `About-Page` again.
- **Logout**: This is quite self explanatory and will logout the user from his current account.

Home page

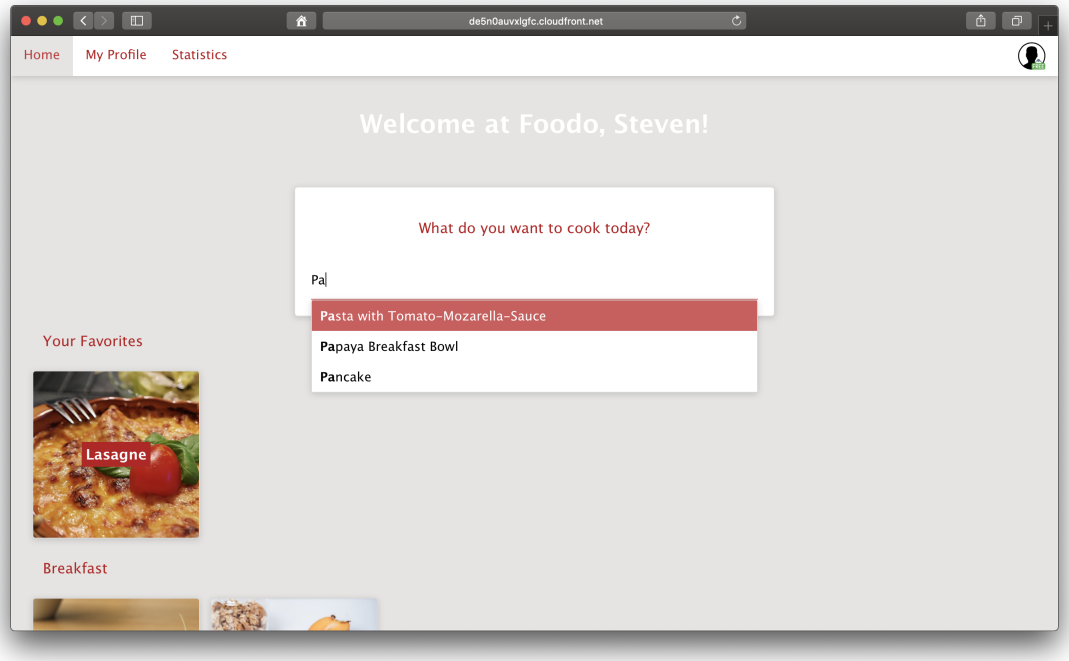

Figure 4 Home page recipe selector

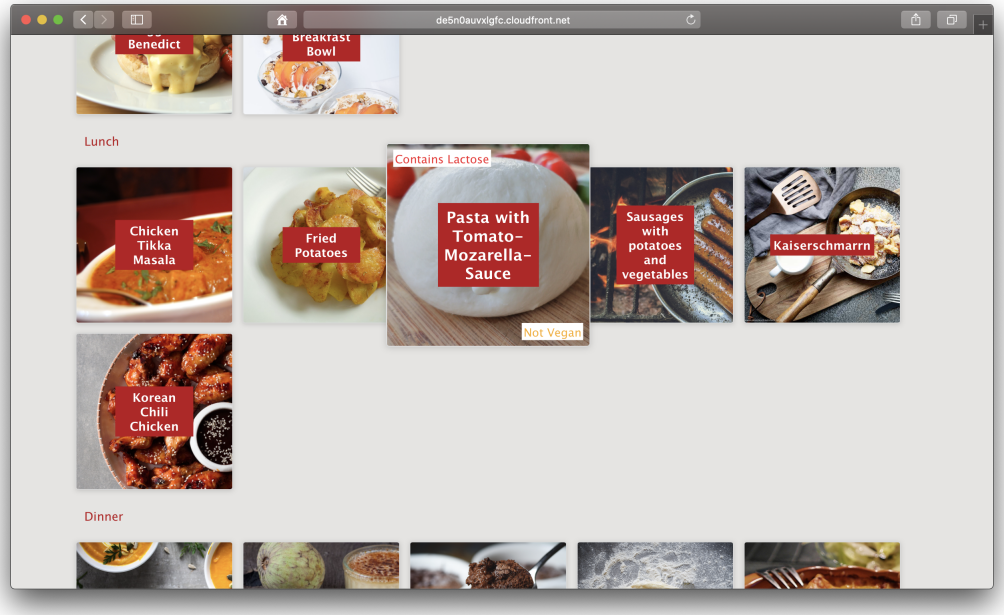

Figure 5 Home page recipe tiles

This is the main page of our application where the user can browse through a variety of recipes. Browsing the available recipes and eventually choosing one recipe to cook can be achieved either by clicking on the `'Select your recipe...'` input field and selecting a recipe out of the appearing list (see figure 4) or by scrolling through the displayed recipe boxes below (see figure 5). The different recipe boxes have been designed to appear visual appealing by showing preview pictures of the dishes. Furthermore, each box shows useful

personalized information about supported lifestyle and allergies on hover. To improve the user experience, the boxes are also categorized into four standard groups and two categories that match the user's lifestyle and allergies:

- Your Favorites
- Breakfast
- Lunch
- Dinner
- Tolerated recipes (allergy-based)
- Lifestyle (e.g. Vegan, Vegetarian)

However, before the user can start the substitution process, it is advised to go to the personal preferences page and select at least a personal nutrition goal. This instruction is displayed in a yellow box, as shown in figure 6.

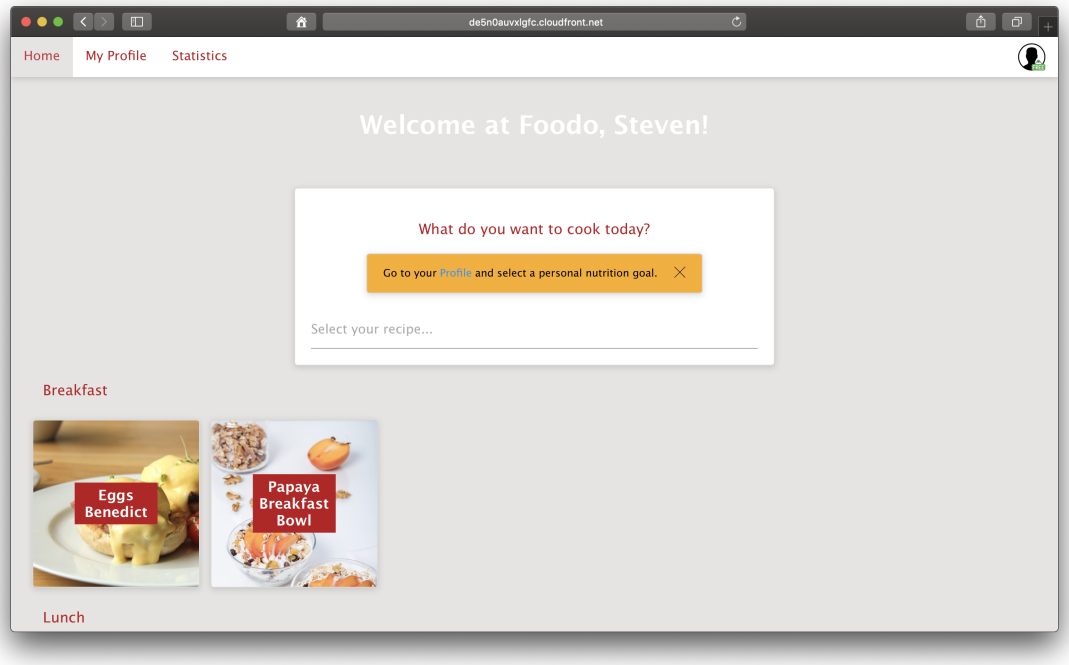

Figure 6 Profile settings for user

Recipe detail view

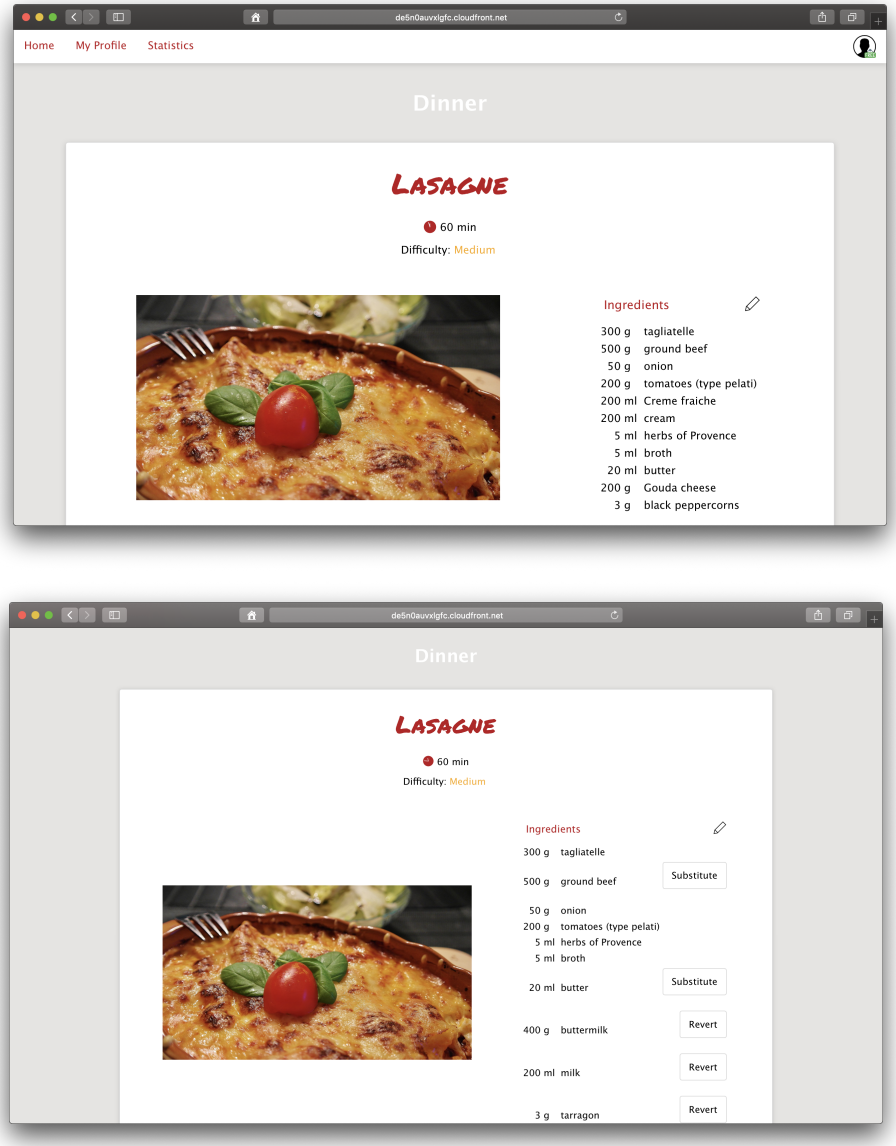

Figure 7 Recipe detail view without and with selected user goal

The first part of the detailed recipe overview consists of a picture and a list of ingredients. Depending on whether the user already selected his personal goal, the user either sees the first or second view depicted in figure 7. For those ingredients where there exists a better substitute in our database based on the nutriscore, a `Substitute` button will be displayed. By clicking on it, the user can select a suitable substitute as shown in figure 8. (You can find more information regarding the substitution algorithm in chapter *4.5 Algorithm*). In the newly opened window the user can see the current NutriScore and possible alternatives for the selected ingredients together with their individual NutriScore. If the user already substituted an ingredient and wishes to revert the substitution, this can be achieveed by clicking on the meanwhile displayed `Revert` button next to the substitute. There is also the option to personalize the recipe by clicking on the pencil icon on top of the ingredients list. This will open the window shown in figure 9 where the user can add or remove current ingredients.

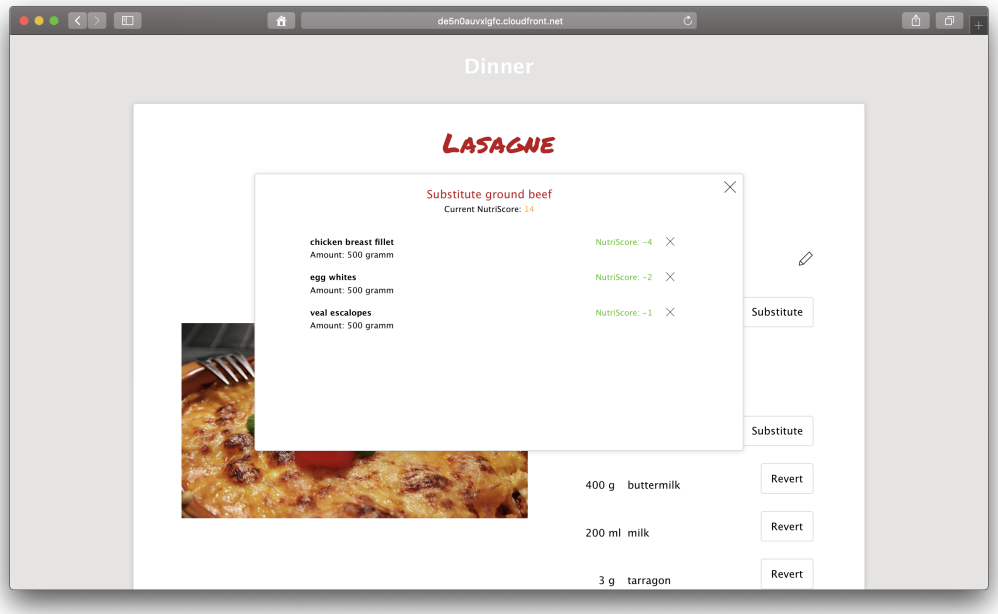

Figure 8 Potential substitutes for beef

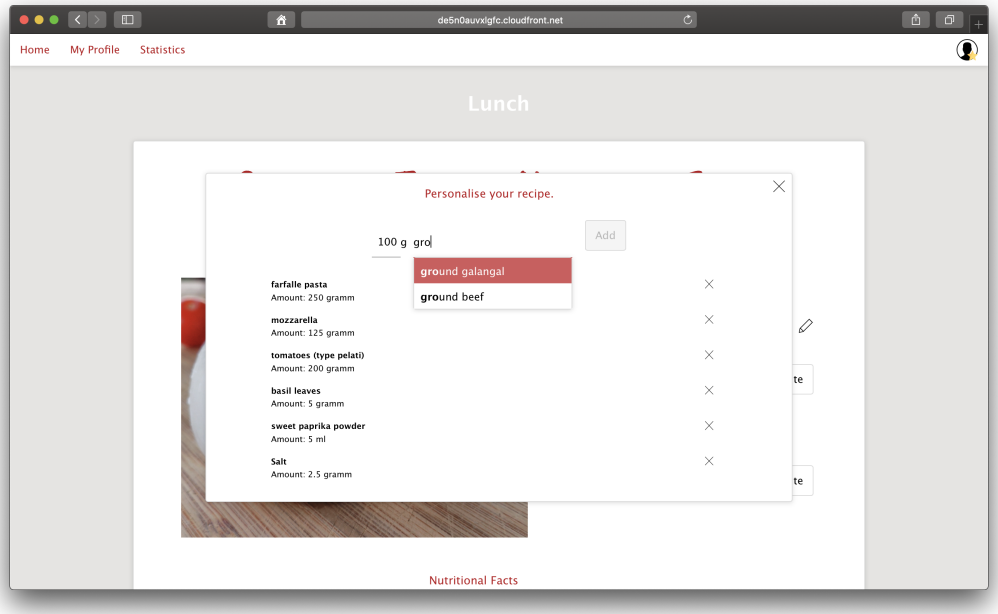

Figure 9 Recipe personalisation

Below the recipe details, the user can also find some nutritional facts consisting of graphs and statistics regarding the (modified) recipe. The user can click through three different representations of the nutritional composition of the currently shown recipe:

- **Pie chart**: This is the initially shown diagram which merely displays the ratio of carbohydrates, fats and proteins without additional numbers (see figure 10 - first screenshot).
- **Table**: Here, the user can find the most important components of the nutrition table (see figure 10 - second screenshot).

- **Bar graph**: This graph shows the most important nutrition components in percentage of the optimal daily intake of an adult based on the EU nutrition guidelines (Regulation (EU) No 1169/2011). It also shows how much percent the user improved or deteriorated its personalized recipe with the current substitutions (see figure 10 - third screenshot).

Figure 10 Pie chart, table and bar graph for a recipe

Profile page

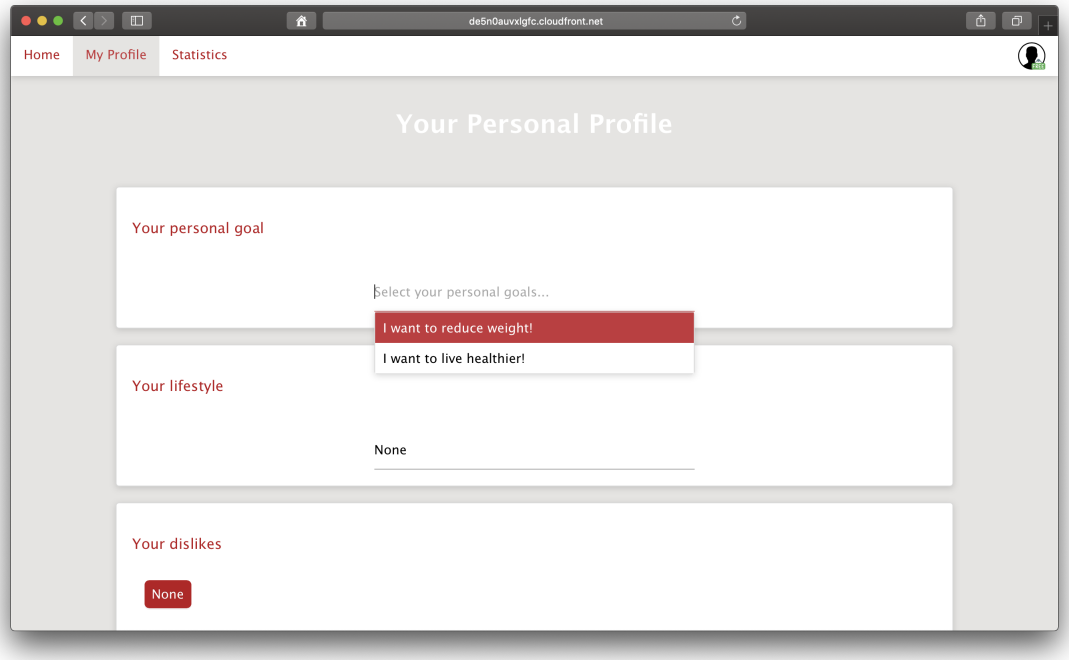

Figure 11 Profile page goal and lifestyle

As already mentioned above, the user first has to select a personal goal before the substitution process becomes available. To improve the received substitutes from the substitution algorithm, it is advised for the user to also insert additional personal information, currently consisting of:

- **Goal**: Reduce weight or live healthier.
- **Lifestyle**: Vegetarian, low carb or vegan
- **Dislikes**: A list of ingredients he does not like and therefore wishes to not be shown as substitutes.
- **Allergies**: A list of ingredients he can not eat and therefore should not be shown as substitutes.

Changes to a user's profile will be saved automatically and can always be changed again.

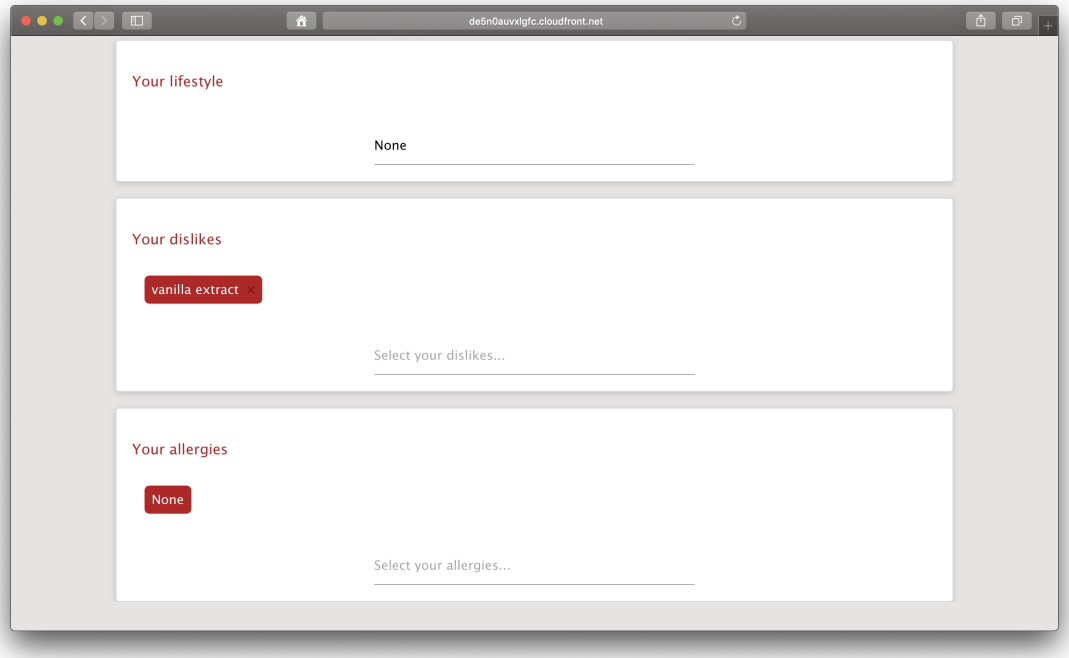

Figure 12 Profile page dislikes and allergies

Statistics page

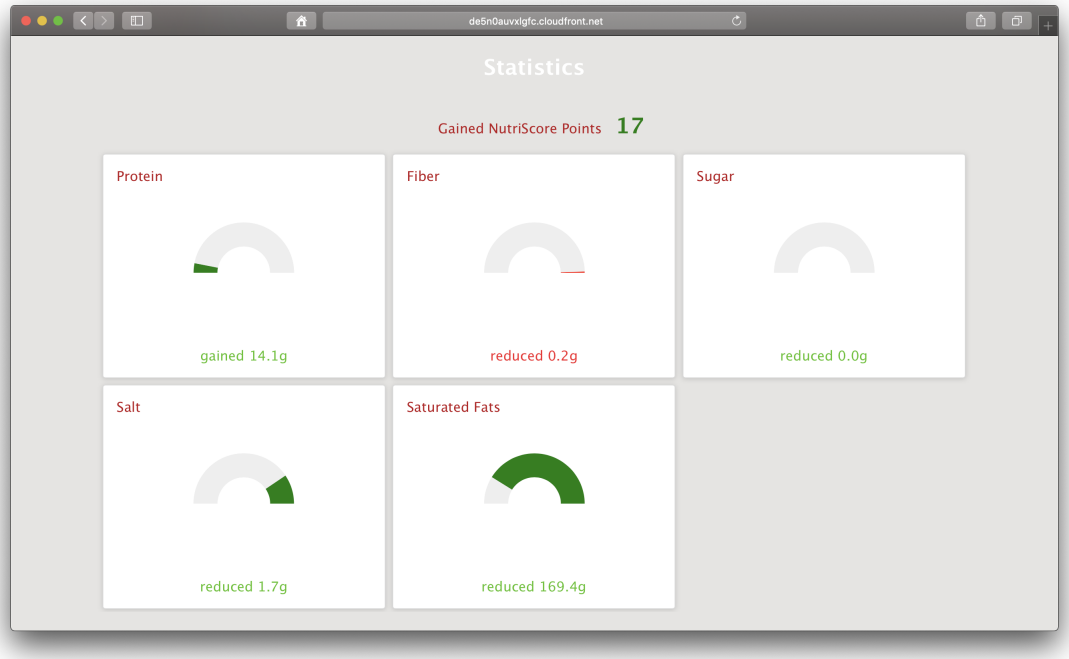

Figure 13 User statistics

By clicking on `Statistics` in the navbar, a user can get an overview about its nutritional career with Foodo. Throughout the substitution history, some personal nutritional data will be

gathered and stored for each user. Through the statistics, one can see how many NutriScore points the user has improved over all recipes in total as well as the amount of the five main nutritional values. Improvements, such as gaining protein or fiber and reducing sugar, salt or saturated fats, will be highlighted in green while negative changes will be highlighted in red.

PayPal Premium

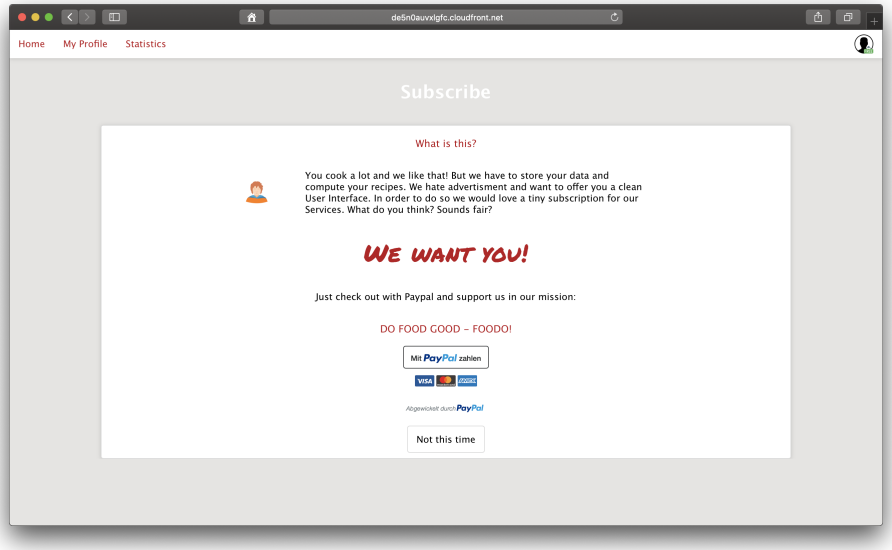

Figure 14 Premium opportunity

After a specified free-to-use period, the user will see the popup shown in figure 14 where he can acquire a premium status through subscribing to the Foodo Paypal subscription plan. This will change the `Free` label next to the profile icon in the upper right corner to a star and will help the developer team to implement additional features. Currently, the free-to-use period allows the user to have up to five personalized recipes and the popup will show when the user tries to view more recipes. It is also to mention that the popup is currently skippable and the subscription plan is not mandatory but voluntary.

Mobile version

Foodo is also available for all mobile devices with access to a web browser. Some examples are shown in the figures below.

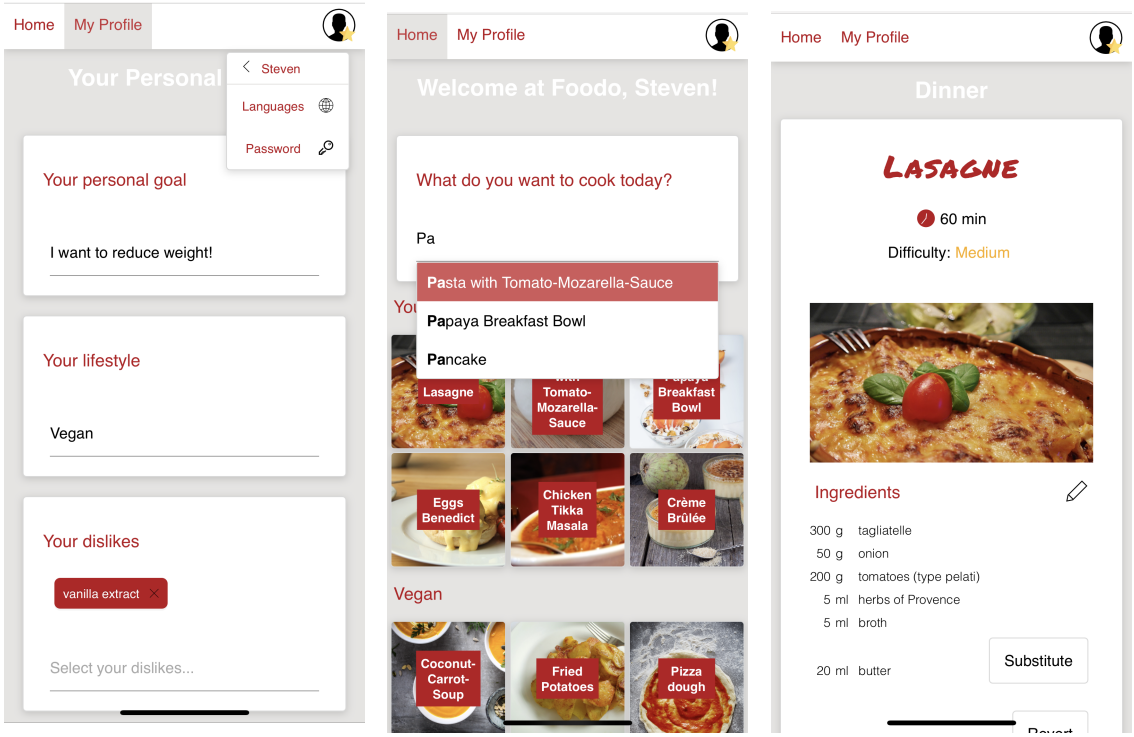

Figure 15 Mobile screenshots

4. Architecture

4.1. Foodo Software System

Overview

The Foodo software system is a distributed web based application that follows state-of-the-art architecture techniques. In the following, the architecture of the Foodo application will be described in a top-down manner.

The software systems consists of six different components that have been organised in four different code repositories. Those six components are:

1. React web application (frontend)
2. HTTP server with REST API (backend)
3. Functional substitution algorithm (algorithm)
4. Lambda function for Alexa intents (lambda function)
5. Alexa skill on the Alexa device (Alexa skill)
6. MongoDB document store (database)

Foodo follows the state-of-the-art design decision to separate the corresponding code into separate repositories. The lambda function and the Alexa skill however share one repository as recommended by the Amazon Development Documentation.

The algorithm is furthermore integrated into the backend as a git submodule to support interoperability and loosely coupling which allows to plug-in different substitution algorithms in the future. The components frontend, backend, lambda function and Alexa skill furthermore communicate through HTTP or related protocols.

In conclusion, the Foodo application can be described as a microservice-oriented application that follows the client-server model. The REST backend and the lambda function can be considered as servers for the Alexa skill and frontend which act as clients. In addition, the backend also serves the lambda function in order to query the database. The backend abstracts the database layer from all other components to support separation of concerns and to reduce code redundancy.

Infrastructure

As for the infrastructure, the Foodo software system is hosted on the AWS (Amazon Web Services) cloud and can therefore be described as a cloud-native application. This design decision allows us to scale, load-balance, and secure the application based on state-of-the-art standards set by big cloud providers. Furthermore, by using AWS, we also profit from low to zero costs through the pay-as-you-go pricing model. Following AWS services are utilized for the different software components:

1. Frontend is hosted as a static webpage on a S3 bucket
2. Frontend is distributed with HTTPS support via Cloudfront
3. Backend is hosted on a Elastic Beanstalk Container
4. Lambda function is hosted on AWS lambda
5. Database is hosted on mLab.com (subsidiary of MongoDB Inc.)
6. Alexa skill is distributed via the Amazon Alexa Appstore
7. Alexa skill is hosted on the Amazon Alexa Developer Console

Code flow

We incorporated HTTPS for our front- and backend to follow state-of-the-art security guidelines that are also mandatory (enforced by Amazon) for the account linking of Alexa skills. On top of HTTPS, the Foodo application uses REST (Representational State Transfer) as a communication paradigm based on JSON (JavaScript Object Notation).

Authorization and authentication are essential parts of a distributed and secure microservice architecture and require a state-of-the-art protocol implementation. The Foodo software system implements the oAuth2 (Open Authorization 2) framework (as defined in RFC 6749) to offer its users a sophisticated authorization and authentication flow, as described below.

Authentication flow

The oAuth2 framework is based on issuing tokens to give a user permission to access restricted parts of the application. Foodo incorporates three different grant types to provide an easy way for the user to login and fulfill all requirements of the Alexa account linking.

The first grant type *Resource Owner Password Credentials* handles the login and sign-up process in the React web app. At first, the user enters his username and password into a webform and submits them to the backend server. Once the backend successfully validated the credentials, it responds with a newly generated access token, which is then securely stored by the React web app.

The second grant type *Authorization Code* is required to link the Foodo user profile to the Amazon account of the Alexa user. This procedure has to be done only once before the initial start of the Alexa skill. After the user has installed the Foodo skill through the Alexa App

(Amazon skill store), the user gets prompted to link the Foodo profile to the Alexa account. On click on the linking button, the user gets redirected to the Foodo web app displaying an authorization form. The React app passes the request to the Foodo backend which validates the access token and responds with an authorization code. Once the React web app has received this code, the user gets redirected to a predefined Amazon URL. Thereafter, an Amazon service sends an access token request to the backend containing the authorization code. As soon as the backend confirms the validity of the authorization code, it sends an access token in response, which will be stored by Amazon Alexa for further use.

The last grant type *Refresh Token* provides a convenient way to renew an expired access token. All issued access tokens contain the following information:

- access-token: string that represents the access token
- access-token-expires-in: datetime after which the access token is invalid
- refresh-token: string that represents the refresh token
- refresh-token-expires-in: datetime after which the refresh token is invalid

When a request attempts to access a protected resource with an expired access token, the backend denies this access. The React web app includes an automatic code flow to identify a failed attempt through the specified error code and requests a new access token using the locally stored refresh token. The backend validates the refresh token and issues a new access token.

Authorization flow

Each request for a protected resource requires authorization. The oAuth2 server implementation for Node.js checks if the HTTP header of a request contains the Authorization property. This property has to include the token type (*Bearer*) as well as the access token which was provided during the authentication process. For example:

```
Authorization:  Bearer b0279b5a647d68241d0e9a3852bfc694a487b989
```

Internationalization

We used the third-party library `i18n` to utilize internationalization based on locales. The Foodo application currently supports English and German but could very easily be expanded to any language. `i18n` is used in the lambda function and frontend simultaneously.

Environment handling

All repositories utilize `.env` files to manage environment variables to enable test, development and production environments. We utilize these logics to enable logging, switching home paths and other environment based information. Furthermore, the `.env` files allow us to encapsulate sensible information like passwords or private keys from the code repositories.

4.2. Frontend architecture

The frontend of the Foodo software system is implemented as a web app that runs in the user's browser. The JavaScript library *React* was used to implement a state-of-the-art UI (User Interface). Furthermore, the following technologies have been used:

- HTML5
- SCSS (CSS-Preprocessor)
- JavaScript and JSX
- Webpack
- NPM
- Eslint

We used the package manager NPM (Node Package Manager) to organize third party libraries and frameworks. We built our application on top of `create-react-app`, an official React skeleton application that comes with pre-configured `Webpack`, `Babel`, `Eslint` and `SCSS` support. To further configure the coding environment, we ejected out of `create-react-app`.

React web app

React is not opinionated about folder structure and code structure but we followed a well adapted pattern of component-container abstraction. React uses components to isolate the frontend code into separate entities. With the component-container abstraction, following naming convention is used:

Components are pure components without state and containers are stateful components that handle the application state. In addition, the frontend utilizes many of the latest React features like context (global state management), custom hooks (functional separation-of-concern pattern to share state or functionality between components) and the use of functional components instead of classes using the new hook API.

This enabled us to build highly performant frontend code that shares API data across the application and manages user state efficiently while also providing a straightforward scalable and reusable component infrastructure.

Design choices

The Foodo frontend UI follows the design principles of Google's *Material Design*[1] and is based on the design language of *Ant Design*[2]. This enables Foodo to have a consistent design and UI across all pages. It relies heavily on the SCSS features of extending css-classes and reusing css-values which allows to predefine design choices like font, coloring,

[1] Material Design - https://material.io
[2] Ant Design - https://ant.design

padding, margin, box shadows and border boxes, and content alignment once for the whole application. This also enables to switch between design prototypes very quickly. The `_variables.scss` file was used to store the global variables of the SCSS designs.

```scss
/* inspired by ant design: https://ant.design/docs/react/introduce */
$primary-color: #BA121F; // primary color for all components
$secondary-color: #c7343b;
$hover-primary-color: #d4595b;
$background-color: $platinum;
$link-color: #1890ff; // link color
$info-color: #1890ff; // info state color
$success-color: #52c41a; // success state color
$warning-color: #faad14; // warning state color
$error-color: #f5222d; // error state color
$border-radius-base: 4px; // major border radius
$border-color-base: #d9d9d9; // major border color
$box-shadow-base: 0 2px 8px rgba(0, 0, 0, 0.15); // major shadow for layers

$success-background-color: $success-color;
$error-background-color: $error-color;
$warning-background-color: $warning-color;
$info-background-color: $info-color;
```

Figure 16 Global styling in `_variables.scss`

Folder structure

The frontend code is furthermore structured as follows:

```
foodo-frontend/
├── public
├── scripts
└── src
    ├── img
    ├── js
    │   ├── components
    │   ├── containers
    │   ├── hooks
    │   ├── provider
    │   ├── services
    │   ├── utilities
    │   └── views
    └── scss
        └── includes
```

As mentioned above, the React components are divided into stateful components (`container`) and pure functional components (`components`). An example for a pure component would be the `button` component that offers an abstraction layer for a predefined button with corresponding styling to reuse across the application. As you can derive from the code example (see figure 17), the Foodo frontend utilizes latest JavaScript syntactic sugar notations like arrow functions and JSX (JavaScript Syntax Extension).

```jsx
/**
 * Standard button component
 * @param classes further css classes
 * @param label a label for the button
 * @param onClick function
 * @param primary if its a primary button (styling)
 * @param disabled if the button should be disabled (styling)
 */
const Button = ( {
    classes, label, onClick, primary, disabled,
} ) => (
    <button
        className={`btn ${ primary ? 'btn-primary' : '' } ${ classes }`}
        type="button"
        onClick={onClick}
        disabled={disabled}
    >
        {label}
    </button>
);
```

Figure 17 Pure basic button component (`button.js`)

The `App.js` container on the other hand, handles the application routing (based on `react-router-dom`), renders the `navbar` and the content of the application, while also rendering all context providers with the global state on mounting of the application. The app container won't be used more than once within the application tree but it is used as a container for content and smaller containers to wrap functionality for the user.

The `hooks` folder is used to abstract functionality with the new hook API. Those functions offer sharable functionality for the different components. For instance, we created the custom hook `useDeviceState.js` (see figure 18) to validate if the current device `isMobile`, a boolean based on `eventListeners` of the browser API that refreshes all dependent components if `isMobile` changes. Furthermore, hooks (in contrast to global state) reduce the risk of side-effects as different components inject the hook's state without sharing it application-wide.

The `provider` folder comprises the context and context-providers. The context API is used to share global (API-)data across different components. For instance, the `RecipesProvider` (see figure 19) holds and shares the standard recipes with the full application scope.

As most of the context providers get initialised as children of the application container component, they have to be rendered only once on application startup. This enables Foodo to get the application state from the backend (e.g. all recipes, ingredients, user recipes) on initial rendering of the application. By that, the visible loading time for the user is reduced significantly. Moreover, as the application container only renders once, the context-provider will stay within the React DOM (Document Object Model) until the user closes the browser tab.

```
import { useState, useEffect } from 'react';

const MOBILE_WIDTH_BREAKPOINT = 750;

const useDeviceState = () => {
    const [ isMobile, setIsMobile ] = useState( MOBILE_WIDTH_BREAKPOINT >= window.innerWidth
 );
    const [ innerWidth, setInnerWidth ] = useState( window.innerWidth );

    useEffect( () => {
        const handleResize = () => {
            const widthIsMobile = MOBILE_WIDTH_BREAKPOINT >= window.innerWidth;
            setIsMobile( widthIsMobile );
            setInnerWidth( window.innerWidth );
        };

        window.addEventListener( 'resize', handleResize, true );
        window.addEventListener( 'deviceorientation', handleResize, true );

        return () => {
            window.removeEventListener( 'resize', handleResize );
            window.removeEventListener( 'deviceorientation', handleResize );
        };
    }, [] );

    return { isMobile, innerWidth };
};

export default useDeviceState;
```

Figure 18 Device state in custom hook (`useDeviceState.js`)

This allows the Foodo frontend to retrieve the data only once during the user session which saves network data and further reduces lag times.

The `service` folder is used to abstract the HTTP functionality which is not part of the React library. All services utilize the third-party library *axios* to implement the AJAX (Asynchronous JavaScript and XML) requests.

The basic HTTP functionality (`GET`, `POST`, `PUT`, `DELETE`, header creation) is implemented in `foodo-api/httpService.js`. Building on top of it, `oAuthService.js` handles the oAuth2 authorization and authentication of the user. The actual REST endpoints get called by specific functions in the different subfolders e.g. `user/userService.js` offers the functions to login and register a user while `user/profile Service.js` offers AJAX calls to post and put profile information of a user.

```
const RecipesContext = React.createContext( {
    recipes: [],
    status: '',
} );

function RecipesProvider( { children } ) {
    const [ recipes, setRecipes ] = useState( [] );
    const [ status, setStatus ] = useState( LOADING_STATUS.IS_IDLE );

    useEffect( () => {
        setStatus( LOADING_STATUS.IS_LOADING );
        getRecipes().then( ( r ) => {
            setRecipes( r );
            setStatus( LOADING_STATUS.HAS_SUCCEEDED );
        } );
    }, [] );

    const context = {
        recipes,
        status,
    };

    return (
        <RecipesContext.Provider value={context}>
            {children}
        </RecipesContext.Provider>
    );
}
```

Figure 19 Global state shared by context-provider (`RecipeProvider.js`)

The `utilities` folder offers a variety of utility classes and helper functions that do not fit in any of the other folders. The internationalizion folder for example holds the `i18n` implementation to support global string management as a lightweight language module. The `localStorage` of the browser API is also abstracted as helper functions in the utility folder and used to store the user token, redirect urls, user locale and other information into the persistent session and local storage.

4.3. Lambda function architecture

The lambda function provides Alexa intent handlers using Node.js and the `ask-sdk` (Software Development Kit for Alexa skill development).The folder structure of the repository incorporates the Alexa skill definition (`skill.json` and `/models`), the lambda function (`/lambda/custom`) as well as utilities like the deployment pipeline (`.ask`) provided by the ASK CLI (Alexa Skills Kit Command Line Interface). The folder structure of the lambda function was designed as follows and is based on skeleton applications provided by the Alexa GitHub[3] repository:

```
lambda/custom
├── intentHandlers
├── internationalization
├── services
│   └── foodo-api
├── .env
├── index.js
└── package.json
```

The lambda function starts on usage of the Alexa skill through an Alexa device or the Alexa Developer Console and runs the `index.js` script. Within `index.js` the `ask-sdk` skill builder utility is called to load the intent handlers from the corresponding folder and build the Alexa skill. The `internationalization` folder is used as a middleware (interceptor) and builds on the same `i18n` package used in the frontend.

Each intent handler provides two functions. `canHandle` defines a boolean return value if the intent handler can process the intent from the Alexa skill that is currently activated by the user and `handle` is used to implement the actual functionality of the lambda function. For instance, the `RecipeHandler` is responsible for `RecipeIntents` and when called, `handle` will output a list of recipes that Foodo offers to the user.

The `CookingHandler` can handle the `CookingIntent` and offers a complex dialogue to navigate the user through the cooking process. The `CookingHandler` therefore utilizes `requestEnvelope` variables that can be filled with values by the user and are passed to the lambda function by the Alexa device. The current dialogue step gets identified by the intent handler by checking which variables have been filled already.

At first the handler checks if the recipe variable has been filled by the user. If not, we output a prompt to the user to ask what he wants to cook. If the recipe value is filled already, we move on and make an AJAX call to the backend to retrieve the user recipe of the currently authorized user by the recipe name. We further prompt if the user wants to substitute the worst ingredient. If the user replies with *yes*, Alexa will insert the value yes into the requestEnvelope

[3] Alexa Github - https://github.com/alexa

`yesNoValue` and we can further retrieve possible substitutes for the user recipe and prompt the different substitution options to the user. If the user answers with a valid option (one, two, or three as defined in the dialogue model), we use the named index to identify the selected substitute and call the backend again to alter the user recipe.

This step is needed as it is currently not supported in Alexa intents to dynamically offer `requestEnvelope` values for a variable that the user can select. As a workaround, the user can pick indices (hard coded `requestEnvelope` values 1, 2, 3) that refer to dynamic values of the substitute names that we output to the user through made AJAX calls. Finally, we inform the user about the change and terminate the interaction.

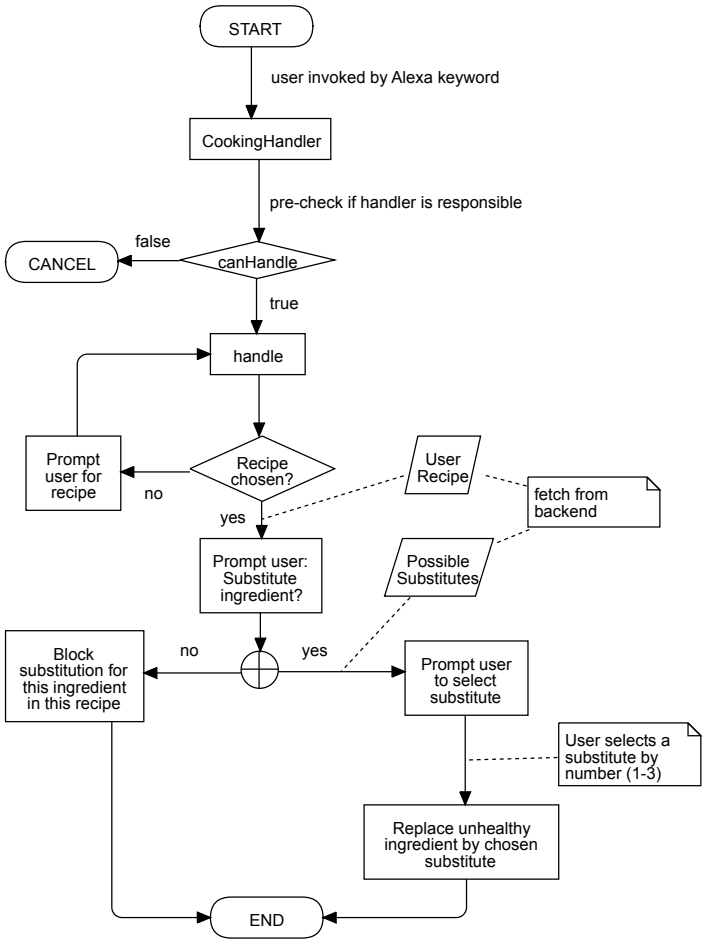

Figure 20 Example of Foodo Alexa skill flow

The different intent handlers define the possible user interactions and facilitate to split different features into separate functions. The possible commands for the user to trigger the intents are defined in the model folder. Here, each intent is defined in all supported languages (German and English) to allow internationalization. The HTTP functionality is implemented within the `services/foodo-api` folder and follows the same design decisions as the frontend services. Again, `axios` is used as a third-party package to implement the requests.

4.4. Backend Architecture

The backend of Foodo provides a REST API using the Node.js web application framework Express. Built on Chrome's JavaScript runtime, Node.js facilitates building lightweight and scalable network applications, as it uses an event-driven, non-blocking I/O model. Express creates a thin layer of basic web application features on top of Node.js which can be extended by middleware modules. The abstraction of complex HTTP methods, extending the API by middleware modules and the lightweight architecture significantly reduce the overhead of building a robust and flexible REST API.

Code structure

The folder structure is designed as follows:

```
foodo-backend
├── index.js
└── src
    ├── controllers
    ├── models
    ├── routes
    ├── logger.js
    ├── middlewares.js
    └── oauth.js
```

We utilize the third-party package mongoose to organize and access the MongoDB database. The `model` folder is used to define the mongoose models that are used to structure data models based on a simple and straightforward JSON object. The `recipe.js` model for instance defines a schema with the following attributes:

```
{
        "_id": {
                "$oid": "5d1f7a5a17a0f81bea0cc696"
        },
        "directions": [],
        "imgUrl": "...",
        "ingredients": [
                {
                        "ingredient": {
                                "$oid": "5d0934799a8ab5830d9edb25"
                        },
                        "amount": 3
                }, ...
        ],
        "name": "Lasagne",
        "preparationTime": 60,
        "meal": "Dinner",
        "difficulty": "Medium",
        "servings": []
}
```

Listing 4.1: Example of recipe "Lasagne"

Thereby, the backend enforces a basic structure that every standard recipe has to follow. Every standard recipe holds a `name`, a `preparation time`, a `meal` definition, a `difficulty`, an image URL and an array of ingredients and some further attributes. The `ingredients array` further encapsulates the two attributes `amount` and `ingredient`. `ingredient` is implemented as a foreign key `_id` which points to a ingredient of the ingredient schema. Those foreign keys can further be populated on REST request. This allows the backend to specify precise response values for the different REST routes. As a conclusion, mongoose allows to design relationships between data objects and we utilized this to implement the food taxonomy (see section 4.6).

The `controllers` folder is used to implement the REST endpoints that are defined in the `routes` folder. Routes define the structure of the express REST API (how) whereas the controllers implement the actual functionality (what) of the different endpoints. We used the state-of-the-art naming convention of a REST api. The backend offers the following REST routes and HTTP functions (Endpoints marked with 🔒[4] require a valid access token to be sent within the request).

Endpoint URI	Method	Parameters	Returns	Description
/auth/token	POST	username, password	bearer token	password grant type
/auth/authorize	GET	-	bearer token	for auth. code & refresh token
/auth/register	POST	username, password	created user object	registering new user

Table 1 Authentication (oAuth2) endpoints in Foodo backend

Endpoint URI	Method	Parameters	Returns	Description
/cooking/start 🔒	POST	recipe name string	[cookingEvent object]	Starts cooking process
/cooking/subsitutes 🔒	GET	-	[ingredient object]	Uses algorithm to calculate top 3 subsitutes
/cooking/subsitute/:selectedNr 🔒	GET	URL param: subsitute #	custom object	Substitutes the unhealthy ingredient with the selected ingredient
/cooking/block 🔒	POST	-	custom object	Blocks a subsitution for an unhealthy ingredient

Table 2 Cooking endpoints in Foodo backend (Alexa only)

The different endpoints can be accessed by the lambda function and the frontend. The subscription route further gets called by PayPal webhooks and allows the backend to verify user subscriptions. All endpoints operate in the same way, retrieving the request data out of the request object, operating on the MongoDB database through mongoose and sending a response back to the client. The Express middleware handles the user authentication via the oAuth2 flow and passes the user object to all the restricted API endpoints fencing the actual REST endpoints from the authorization layer.

[4] Icon made by Freepik www.flaticon.com licensed by CC 3.0 BY

Endpoint URI	Method	Parameters	Returns	Description
/user/me 🔒	GET	-	[user object]	-
/user/allergies 🔒	PUT, POST, DELETE	allergy object	updated user object	update, insert & remove allergy
/user/dislikes 🔒	PUT, POST, DELETE	ingredient object	updated user object	update, insert & remove dislike
/user/lifestyle 🔒	POST	lifestyle object	updated user object	overwrites current lifestyle
/user/goal 🔒	POST	goal object	updated user object	overwrites current goal
/user/level 🔒	PUT	level string	updated user object	sets the user level (e.g. subscribed)
/user/recipes 🔒	GET	-	[pers. recipe object]	-
/user/recipes 🔒	PUT, POST	pers. recipe object	updated pers. recipe object	update & insert pers. recipe
/user/recipes/substitute 🔒	PUT, DELETE	custom object	updated pers. recipe object	substitutes ingredient in recipe & reverts substitution

Table 3 User endpoints in Foodo backend

Endpoint URI	Method	Parameters	Returns	Description
/ingredients	GET	-	[ingredients object]	-
/ingredients/categories/:id	GET	URL param: category id	[ingredients object]	Filters all ingredients by selected category
/recipes	GET	-	[recipes object]	-
/recipes/:id/substitutes	GET	URL param: recipe id	custom object	Uses algorithm to calculate subsitutes for unhealthy ingredients

Table 4 Ingredient & recipe related endpoints in Foodo backend

Endpoint URI	Method	Parameters	Returns	Description
/subscription/create 🔒	POST	subscription object	custom object	-
/subscription/cancel 🔒	POST	subscription object	custom object	Removes subscription and sets user to 'free'
/subscription/subscribed 🔒	PUT	subscription object	custom object	Activates subscription and sets user to 'subscribed'

Table 5 Subscription endpoints in Foodo backend

Utilities

The backend implements some utilities besides the express application. The `script` folder in the root folder contains utility scripts. The `data-import.js` script was used to import and convert the initial dataset that we received for the Foodo project into our data schema. The `internationalization.js` script was used to translate the ingredients names of the initial ingredient data to english to support internationalization. We utilized the Google Cloud API for that matter. The file `src/logger.js` implements a sophisticated logger based on the third-party package `winston` which allowed us to define log levels both for localhost and for the production system to ease debugging and error handling.

4.5. Algorithm

The endpoint `/recipes/substitutes/` is used during the substitution process and calls the substitution algorithm (git submodule). The substitution algorithm searches for substitutes to improve the given recipe according to the user's preferences. The user's preferences consist of:

- a nutrition goal ("Eating healthy" or "Reducing healthily weight"),
- a lifestyle ("Low carb", "Vegetarian", "Vegan" or "None"),
- dislikes (list of disliked ingredients or "None"), and
- allergies ("Gluten", "Lactose", "Fructose" or "None").

For determining how healthy an ingredient is, the algorithm relies on the NutriScore (Julia and Hercberg 2017). This score reaches from 'A' (healthy) to 'E' (unhealthy) and was developed by nutrition scientists for rating the healthiness of dishes. It is mandatory to print it on products sold in France.

If the user's goal is "Reducing healthily weight", the algorithm calculates the energy density of ingredients, as a low energy density of the nutrition is found to be linked to weight reduction (Ello-Martin, Ledikwe, and Rolls 2005). Moreover, the algorithm relies on a food taxonomy to find potential substitutes.

Our food taxonomy consists of twelve categories, namely: 'Dairy Product', 'Oils, fats and shortenings', 'Meat and poultry', 'Fish and seafood', 'Vegetables', 'Fruits', 'Breads, cereals and grains', 'Soups (canned and diluted)', 'Desserts and sweets', 'Nuts and seeds', 'Beverages' and 'Spices'. The algorithm always searches for substitutes within the same food category of the ingredient to be substituted.

Steps of the algorithm

The algorithm consists of two major steps. In the first step the algorithm iterates through the recipe's ingredients and identifies bad ingredients. An ingredient is bad if it fulfills at least one of the following requirements:

- the ingredient's NutriScore is C or worse,
- it has a high energy density and the user's nutrition goal is "Reducing healthily weight",
- it is not suitable for the user's lifestyle,
- it is contained in the user's set of dislikes, or
- it triggers one of the user's allergies.

After having identified the bad ingredients the algorithm tries to find suitable substitutes within the same ingredient category. A potential substitute is suitable if it fulfills the following requirements:

- the substitute's NutriScore is lower than the bad ingredient's one or if the bad ingredient's one is already A or B then A or B is sufficient for the substitute's NutriScore,
- the substitute is not in conflict with the user's nutrition goal,
- it is suitable for the user's lifestyle,
- it does not trigger the user's allergies, and
- it is not contained in the user's set of dislikes, except for the case that there are not any other potential substitutes available which fulfill all of the preceding requirements and are not dislikes of the user.

When the algorithm has identified a list of suitable substitutes for each of the bad ingredients, it sorts these lists ascendingly based on the NutriScore, so that the healthiest substitute is in the first place. The results, containing the ranked substitutes per bad ingredient, are returned to the calling endpoint.

Additionally, the algorithm offers a function for the Alexa skill which identifies the worst ingredient of a recipe based on the NutriScore and returns the three best substitutes for it in an ordered list.

4.6. Database

The MongoDB document store is abstracted by the backend and organised by the mongoose package. Following schemas have been designed through the backend and organize the Foodo data layer:

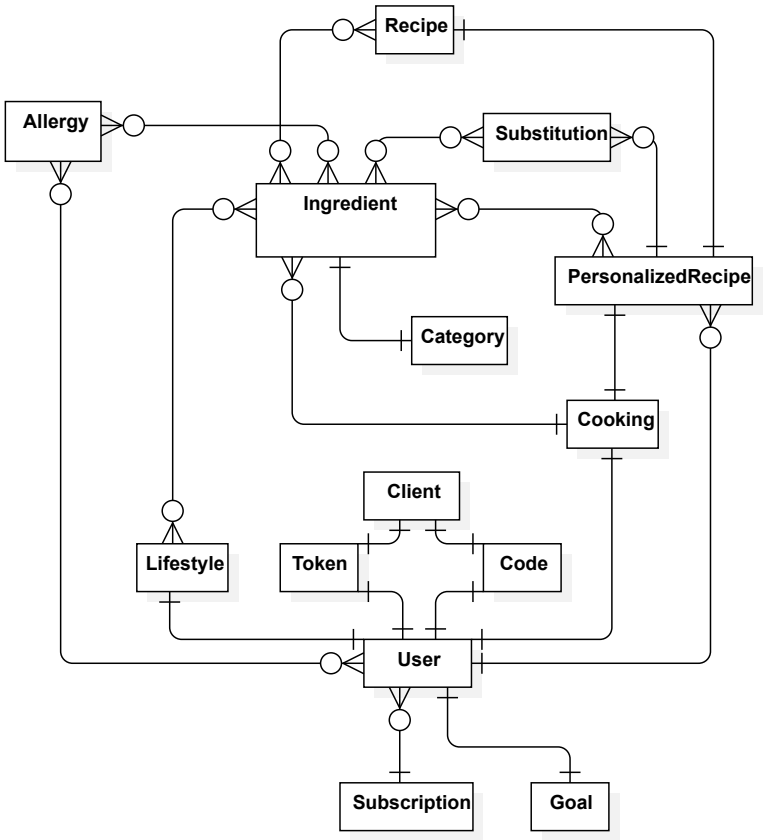

Figure 21 Database schema of Foodo

Data model relationships

The recipe describes a basic recipe that holds a set of ingredients. Each ingredient has a category and a set of allergies (gluten, lactose, fructose) and lifestyles (vegan, vegetarian, none, or low carbohydrates). The user model is connected to the authorization flow data objects client, token, and code. Tokens are used to handle the authorization of the user, whereas clients define the different clients (frontend and lambda function) that the user can use to authenticate. The code schema is used to offer authorization codes for the user to authorize clients (Alexa) to act in behalf of the user. The user also holds subscriptions. We defined a simple access level hierarchy (admin, subscribed, free) that is implemented via the subscription model. In addition, the user holds profile information about his lifestyle, goal (eat healthy, reduce weight), allergies and dislikes (ingredients). The user is also linked to its personalized recipes that contain a reference to the original standard recipe and a new set of current ingredients and a list of blocked substitutes that the user does not wanted to change. The cooking schema is used as a state management for the Alexa cooking intent. If the user

has indicated within the dialogue flow that he wants to substitute, the possible substitutes for the current recipe and user are stored in the cooking object and later accessed if the user decides which substitute he wants to use in the substitution process (see section 4.3).

- Dairy products
- Oils, fats and shortenings
- Meat and poultry
- Fish and seafood
- Vegetables
- Fruits
- Breads, cereals and grains
- Soups (canned and diluted)
- Desserts and sweets
- Nuts and seeds
- Beverages
- Spices

Food taxonomy

One important task of implementing the Foodo software system was designing a food taxonomy that would allow to implement a sophisticated substitution algorithm. From our perspective, personalization was one key aspect that the food taxonomy had to offer to the Foodo user. Therefore, as mentioned above, the user object holds attributes for disliked ingredients, allergies, a lifestyle and a personal goal .

Furthermore, through the cooking process (frontend cooking page or Alexa cooking intent), the selected standard recipe is used as a starting point for the current user's personalized recipe (`personalized recipe` mongoose model). Once the user has a personalized user recipe for a corresponding standard recipe, the user recipe will be used instead.

The user recipe holds further attributes for personalization. Every user recipe holds an array to block possible substitutes. If the user denies a substitute, the Foodo system will remember this decision. To offer fitting and related substitutes for each ingredient (see section 4.5) each ingredient also holds information about supported lifestyles, allergies, NutriScore information and a category (see figure **??**). The definition of categories allows the substitution algorithm to filter for fitting ingredients.

Finally, all past substitutions are saved in the user's personalized recipe and logged in the `substitution` mongoose model. This would allow the Foodo software system to learn and analyze most utilized substitutes for each recipe and ingredient and implement a substitution recommender system or even machine learning based substitution mechanisms.

Test and demo data

The Foodo data records are based on ingredients data that we received as a starting point. The transfer process included internationalization, elimination of redundant and messy data entries, converting amount information to standardized units (gram and milliliter), and transferring the data records to the Foodo `ingredient` mongoose model.

Currently, the MongoDB document store contains 304 ingredients. After transferring the data to the Foodo database, we further needed to add information to each ingredient data entry. For that purpose, the Foodo frontend implements four features for administrators. For each ingredient, we used the `SetAllergies`, `SetCategory` and the `SetLifestyles` administrator pages of the Foodo frontend to add allergies, categories and lifestyle information. To support the data preparation process, we also utilized helper scripts. For instance, we were able to automatically set lactose and fructose allergies based on already set categories. We furthermore created 15 standard recipes for testing and demo purposes through the `CreateRecipe` administrator page.

5. Evaluation

We conducted two big evaluation rounds. The first one targeted the first prototype and the second one the final prototype. The interviews were conducted either in a personal talk or via a messenger. Due to our setup, in which the most current version of Foodo is deployed on AWS and accessible for the public, we just handed the link to our webapp to the participants and they could explore Foodo on their own with their own device. That way, we could also test how good Foodo performs on different devices with different browsers which strengthens our cross-platform and multi-device approach. The two big evaluation rounds each took up two to three weeks. This enabled us to incorporate the first gathered feedbacks and to test the new changes within the same evaluation round. This was only feasible because we developed Foodo with a focus on agile development and relied on continuous integration and continuous delivery so that even small changes could get delivered and deployed to the production system as soon as possible.

Overall, eleven participants tested Foodo and provided us feedback. They liked Foodo a lot and most of them would use Foodo when it is market-ready. Specifically, they praised Foodo's clean design, its cross-platform multi-device approach, the Alexa integration, and the idea of getting recommendations for substitutions which improve the healthiness of recipes. Their feedback or suggestions for improvement can be categorized in three areas, namely functional, visual and error feedback, and it is condensed into the following paragraphs.

5.1. Functional Feedback

Implemented Feedback

Regarding the functional feedback which we already implemented, the participants' feedback can be summarized into seven main points. Firstly, they mentioned that the substitution decision is final and suggested that the recipe and the corresponding substitutions should be editable. Secondly, the participants highlighted that it is common to display the difficulty of a recipe and therefore they would expect that of an app offering recipes. Thirdly, they would love to see the recipes in a box grid overview with nice pictures so that they can scroll through them when searching for inspiration on what to cook. Fourthly, some of the participants did not understand what Foodo exactly does and therefore suggested that there should be an explanatory page on what Foodo is and how it works. Fifthly, they asked for a list of the dishes they cooked last. Sixthly, the participants mentioned that it would be motivating if they could see statistics on how their nutrition improved through the usage of Foodo. Lastly, they praised the possibility to select dislikes on the ingredient level, because some of them eat all kinds of meat except porc meat. With our dislikes functionality, we can incorporate this specific taste.

Not Yet Implemented Feedback

The participants suggested a whole bunch of new features and wished extensions. These can be categorized in extension or refinement of current features, community functionalities, offering recommendations and new other features. The suggestions for extension of current feature focuses on offering "more", meaning more allergies, e.g. nuts, more lifestyles, like "Paleo", more recipe categories, like "fast", "cheap", or "holiday", and more recipes. The asked refinements are mainly focused on improving the quality of the substitution suggestions. In regard of community functionalities, the participants wished to be able to rate recipes and substitutions with stars, to comment and to see comments on substitutions and to post what they have eaten or substituted so that other can see it. Another wished feature are the services offered by a recommender system. The participants would like to get recommendations for recipes based on the ones they like or based on the time of the day. They also would love to get informed about recipes similar to the one they are looking at and about starters and deserts fitting to the current recipe if it is a main dish. During the feedback sessions, the participants suggested even more potential new features. Among these are that they would like to use a portion calculator which also updates the fact table and the diagrams accordingly, to insert own recipes (which we already implemented but just for the admin panel), to get an automatically generated grocery list and to bookmark recipes. Additionally, they suggested that Foodo could offer challenges in which they can participate, offer an integrated timer for supporting the cooking activity, and that Foodo should explain the reason why a certain subsititution is offered so that they can also educate themselves on healthy nutrition by cooking with Foodo. Furthermore, one participant mentioned that Foodo should also try to experiment with substitutions which seem to be a bit crazy at first glance. An example is that starting from a Cafe Latte Foodo could suggest through substitution a Red Beet Latte which is actually a completely different drink.

5.2. Visual Feedback

Implemented Feedback

The visual feedback which we already implemented mainly consists of four remarks. The first one is that not every part in the intermediate prototypes was internationalized. Secondly, at the beginning Foodo did not use standardized abbreviations for ingredient amounts, like "ml" and "g". Thirdly, Foodo should put more visual emphasis on the NutriScore. This is now achieved by displaying the NutriScore of every substitute and by coloring them appropriately. Fourthly, the diagrams were praised for nicely conveying the important nutritional information.

Not Yet Implemented Feedback

The participants noted several times that Foodo should use common units, like a pinch of salt instead of 0.02 g. This would improve the usability a lot. Another important remark is that one

of the diagrams cannot convey its message if the user suffers a red-green color blindness because the diagram is colored in red and green.

5.3. Error Feedback

Implemented Feedback

The participants reported some errors regarding the results of the substitution algorithm, e.g. they selected "Lactose" as allergy and the the algorithm still recommended them to use yogurt as a substitute. This was due to the fact that in the first tests the algorithm was mocked by using randomized categories. The other recognized errors were that it was not possible to select "None" in allergies and some alignment errors on some of the user's devices, like that a button was moved outside the screen.

Not Yet Implemented Feedback

In our current version, some of the participants experienced that on their device with their browser that there are sometimes still some displaying errors. One participant experienced that in his setup the keyboard overlays the input field so that inserting the required input can be quite cumbersome. Another participant discovered in her setup that the alignments in the nutrition table get destroyed by one word that seems to be too long. Apart from that they did not discover any other errors.

6. Limitations

Please find the limitations of the Foodo project divided in obstacles we faced during the development of Foodo and in limitations of the application in its current state.

6.1. Obstacles

Pre-existing standards

One of the main limitations we have been faced with, was the lack of a database with substitutes, one could build upon. Even if we would have decided to implement a learning algorithm computing substitutes, we would have still be missing a test set to verify the results. Information about possible replacements is scattered around the internet in cooking blogs and forums. These are mostly directed at vegans, vegetarians and people suffering from allergies. Additionally, these recommendations are often linked to a certain recipe, making it harder to generalize possible relationships. The same applies to algorithms or even guidelines and rules of thumb.

Context

When it comes to assessing how well an ingredient performs as a substitute, there are multiple properties that have to be considered. Namely the consistency, texture, taste, look and in our case especially the nutritional value. The impact the ingredient has on the consistency and texture of the dish has to be taken into account, otherwise a stew might get turned into a soup. The toughest nut to crack is the adjustment of the list of possible substitutes to the context of a dish. With context being the symbiosis between taste and texture and therefore something hard to assess in digital terms. Applesauce as an egg replacement can work in the context of a cake but not in an omelette. Furthermore, an egg can have multiple purposes or functions within a dish. It can work as a binding or loosening agent and add taste, color and moisture. A fully fetched substitution-algorithm would have to infer the purpose of the ingredient as well as the context of the dish based on the provided recipe. Additionally, substitutes which do not fit for the preparation conditions (e.g. heat) have to be filtered beforehand.

6.2. Current limits of the programm

Every limit demands actions from the user, to compensate for the lack of automation. The more the app evolves, the less input is necessary.

How good is the substitute

As mentioned in 5.1.2, there are multiple properties influencing the suitability of a substitute. At the moment, we are only considering the category of a food item as well as its nutritional value. If it fits in regards of taste and texture has to be decided by the user.

Preparation steps

Currently, we do not provide preparation steps to guide a user. Because a major issues (besides gathering the data) is to assess how the preparation changes through a substitute. The time a dish needs to stay in the oven might differ or even the way the substitute is prepared (sliced, smashed, peeled). This can often be easily inferred by the user but computing those properties is not as easy.

How much should be substituted

Currently we provide information based on the assumption that an ingredient is completely substituted. But there are cases, in which one would only like to substitute a certain percentage or use a mixture out of different alternatives. Further, some substitutes only work as a 'thinner'. Next to this it always has to be considered, how much has been substituted already. Applesauce for example, can be used as an replacement for eggs, butter or as a natural sweetener. But it should not substitute all of them.

Cross-substitute effects

The applesauce-example from above illustrates an additional detail that should be considered in the future. Substitutes provide other properties than their respective counterparts. If we use applesauce as an egg replacement, the whole pastry is getting sweeter which in return should lead to a decrease in the amount of sugar needed. The effect that one substitute choice has on the amount of other ingredients used, is mainly relevant for sugar, salt and fats/oil.

7. Outlook

More of current features

The easiest step to add more value to the current application is by extending existing features.

New nutritional goals
New goals can broaden the target audience. One could open up to the medical market by introducing a strict reduction of sugar for diabetics or lowering salt and cholesterol for people suffering from heart diseases. The implementation would be fairly simply. The algorithm could filter possible substitutes by their content of salt instead of their NutriScore.

Add ingredients and recipes
Foodo should know about as much ingredients as possible, to consider them in the substitution algorithm. This directly improves the proposed substitutes because there are more alternatives to choose from. New recipes on the other hand are essential to attract and keep users. We claimed to provide users with their favorite recipes to spare them from creating them manually. By delivering on this promise, we can embrace Foodos convenience.

Community

The task of adding recipes and ingredients to the database could be transferred to the user. With a rating system, popular recipes could rise to the top and bad ones would be sorted out. This system could also be applied to the substitutes where users like or dislike existing ones and can propose new ones. The data which substitute a user picks and if the user liked it or not is already collected and just has to be aggregated and turned into a popularity ranking. With features like comment sections, users could exchange more tips regarding certain substitutes (how good is it, how to prepare it, how much of the original ingredient should be replaced with it). That way, a first collection of substitutes could emerge, supporting further research in this field. Moreover, the focus on the community would get users more involved and keep them attached to foodo.

Recommendations

Once there is a stable user base, regularly interacting with foodo, the arising data can be used for analytical purposes. Besides optimizing substitution proposals, new recipes could be recommended to users, based on what they have already tried and liked. Adapting to a users preferences would lead to a more personalized experience.

Complements

An easier approach to recipe improvement is to use complementary ingredients instead of substitutes. Complements only have to fit well to the existing dish, but don't have to replace an ingredient in multiple regards (binding/loosening properties, moisture, taste, color). For example peppers fit well to a quiche, stir-fried vegetables or ratatouille but are in no regard a substitute for one of the ingredients in these dishes. The same applies to adding a handful of peas to pork with noodles. An addition that can't be considered a replacement but is still an improvement, at least indirectly. Because the peas increase the total weight and therefore the relative composition of the dish. This can be directly seen in the nutritional information of a single serving (example/ figure below).

Nutrient	**Pork**	**Noodles**	**Peas**
Calories	271 kcal	137 kcal	81 kcal
Fat	17g	2g	0,5g
Carbs	0g	25g	14,5g
Protein	27g	4,5g	5,5g

Table 6 Ingredients and their nutritional values

Nutrients	**Without Peas**	**With Peas**	**Difference**
Calories	754 kcal	674 kcal	81 kcal
Fat	31g	25g	6g
Carbs	62g	61g	1g
Protein	53g	47g	6g

Table 7 Improvement of one serving (400g)

Of course this doesn't work out if still the same amount of pork and pasta is eaten and the peas just come on top. But in theory one would reduce either the pork, the noodles or both, to maintain the same size for a single serving.

Take less

The aforementioned idea of complements builds upon reducing the amount of (unhealthy) ingredients using supplements instead. But there are also use cases in which neither sub-

stitutes nor supplements are needed to improve a recipe. Especially in the context of baking there are often huge amounts of sugar used, which can be reduced without having any noticeable impact on the pastries. The dispensable amount of the unhealthy ingredient could either be provided and verified by the community or automatically inferred by comparing other recipes of the same dish.

Business model

In the future it's also worth thinking of new income streams, besides the current premium membership on a voluntary basis. One source could be advertisements, like banner ads or product placements. Kitchen gadgets or ingredients by a certain brand could be mentioned within recipes or their preparation steps. Partnerships with food retailers would provide a stable income and additional input for the ingredient database. Edeka, Rewe and their competitors are already storing nutritional facts, geographical information (where the product comes from) as well as the current price and could grant us access to them. With these information optimizing a recipe in terms of overall costs or eco-friendliness (prefer regional or seasonal products) could become a new feature. Further, this would make it easier to implement a shopping list, directly linked to a partners delivery service.

Bibliography

Ello-Martin, Julia A, Jenny H Ledikwe, and Barbara J Rolls (2005). “The Influence of Food Portion Size and Energy Density on Energy Intake: Implications for Weight Management”. en. In: *The American Journal of Clinical Nutrition* 82.1, pp. 236–241. ISSN: 0002-9165, 1938-3207. DOI: 10.1093/ajcn/82.1.236S. URL: https://academic.oup.com/ajcn/article/82/1/236S/4863399 (visited on 08/25/2019).

Julia, Chantal and Serge Hercberg (2017). “Nutri-Score: Evidence of the Effectiveness of the French Front-of-Pack Nutrition Label”. In: *Ernahrungs Umschau* 64.12, pp. 181–187. DOI: 10.4455/eu.2017.048.

Katja Wohlers, Michaela Hombrecher (2017). *Iss was, Deutschland. TK-Studie zur Ernährung 2017*. German. Tech. rep. Bramfelder Straße 140, 22305 Hamburg: Techniker Krankenkasse, p. 64. URL: https://www.tk.de/resource/blob/2033596/0208f5f5844c04abbbcbb1389872ee01/iss-was-deutschland-data.pdf.