Salli Valkama, salli.valkama@tuni.fi

Recipe finder
Documentation

Description

Recipe finder application allows users to search for dish recipes based on dish names, ingredients and dietary preferences. The application retrieves the recipes from a data base using Spoonacular API. The retrieved recipes are displayed in a list and the user can explore the list and open recipes to see more detailed information about them.

Prerequisites

This application is React project and consists of the main component file app, and folders pages, components, theme and services. Public folder also includes necessary files. The key dependencies the project uses are React, Node Package Manager, Node.js, React Router, Material-UI, Styled components, Emotion, React Scripts and HTML React Parser. Also an API key for Spoonacular is needed (provided in the services folder).

Use

The application displays the recipe search fields when starting the program. There is one field for each search criteria: dish name, ingredients and dietary preferences. The dish field takes only one argument that can be set through free typing or selecting an option from an autocompleted list of dishes. The ingredient field takes up to six arguments that can be selected from a list of available ingredients. The diet field takes up to three arguments that can be selected from a list of available diet options. The user must fill at least one of the fields to complete a recipe search. A search is fired on a button click “Search”. The field contents are erased on a button click “Empty”.

The searched results are displayed in a list of recipe cards. Each card holds a recipe title and an image of the meal. The list displays 30 results per page, and navigation buttons along with result page numbers are provided to reveal more results. A button “Back to top” returns the screen view from the end of the list to the top of the page, when needed. The user can scroll through the results and select a certain recipe for further exploring by clicking a recipe card.

The detailed information of the selected recipe is displayed in a new view. The information page holds the recipe title, serving size and cooking time information, an image of the meal, the list of needed ingredients and detailed instructions, in readable form. The application returns to the search view on a button click “Back to results” and displays the last searched results again. The user can keep scrolling the results or start a new search.

Structure overview

The application consists of the main component App that holds page components Home and RecipeDetails (Chart 1). The search and result listing functionalities are located to Home page, that is also the default view of the application when starting the program. SearchBar component holds the search field components for the user input. ResultList component displays the retrieved recipes in a list after a search is performed. RecipeDetails page displays the detailed information of the selected recipe result and provides the access back to Home page.

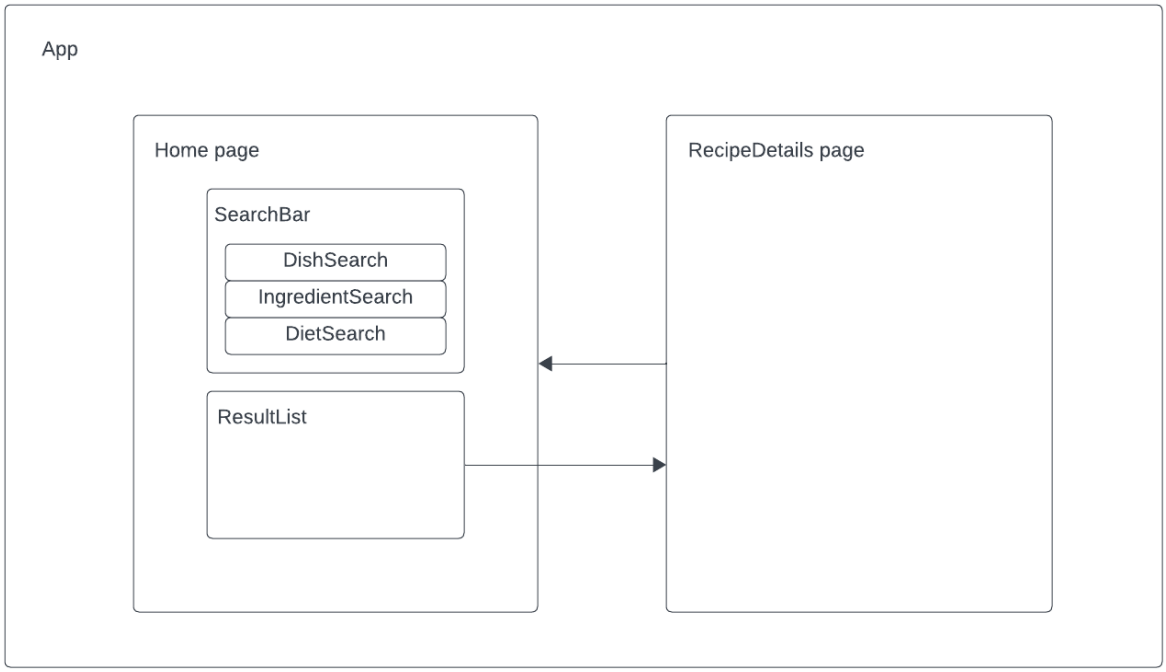

Chart 1. Structure of the application and navigating between page components.

Components

App.js
The main component App holds the page components. Sets the style for the components using ThemeProvider. Provides routing between the pages using BrowserRouter.

Theme.js
Contains MUI material style definitions for texts, colors and for some of the MUI components.

Home.js
Home page component for holding and placing the components responsible of the search (SearcjBar.js) and result listing (ResultList.js) functionalities.

SearchBar.js
Holds and places the search field components (DishSearch.js, DietSearch.js, IngredientSearch.js) and buttons for managing the search. Handles search parameters given by the user.

DishSearch.js
MUI Autocomplete component that allows user to type a value (dish name) or to select a value from the autocompleted list.

DietSearch.js
MUI Select component that allows the user to select multiple values from the diet option list.

IngredientSearch.js
MUI Autocomplete component that creates the options (ingredients) from CSV file content. The needed file is in public folder. Takes multiple user given values.

ResultList.js
Performs a search through an API call based on the search parameters received from SearchBar component. Assigns the retrieved recipes into a list of clickable Cards. Arranges results to several pages using MUI Pagination component. Transmits the recipe ID to Router when a Card is clicked. Takes care of session storage management by storing and setting search parameters between navigations.

RecipeDetails.js
Page component that shows the detailed information of the selected recipe. Takes recipe ID as an argument from Router and performs an API call to fetch recipe details. Arranges the response content into a recipe like format. Provides navigation back to Home page.

api.js
Contains the API key needed for retrieving recipes from Spoonacular API.

About API

The application uses Spoonacular API for retrieving recipes from a database. Author's personal API key is included in the project files. The application uses complex search and recipe information end points for retrieving a list of recipes and more information about a certain recipe. The documentation can be found in spoonacular.com/food-api/docs.

Error handling

The user might make mistakes when filling the search fields. The program checks if the allowed number of arguments have been tried to exceed and guides user with a notification on the screen to correct the argument number. In cases where no arguments are given, the API call phase is prevented until the user gives some arguments. The program makes an API call only with correct amount of search arguments.

Search field for the dish name allows user to type text freely. EncodeURIcomponent method takes care of possible characters not allowed in API calls.

The program throws an error in case of unsuccessful API call and displays an error message on the screen.

Limitations

No tests have been made for the program, so the robustness isn't defined or guaranteed.

Some responsiveness solutions have been implemented for the list features, but they might be inadequate. For example, not all screen sizes have been considered in the implementation. Component sizes might be unpredictable with different screen sizes.

API call points are limited in the free use of the Spoonacular API, so the application can handle only a limited number of searches at a time. It could have been a more efficient solution to use the autocomplete

endpoint with the dish name query, compared to a fixed list of dish names, but the points quota restricts the number of API calls.

The application displays the recipe information as it is when retrieved from the database. No text formatting has been applied to the recipe information. For example, instructions might contain issues such as duplicate numeration and poorly formatted text.