Aplikacje internetowe oparte o komponenty - React

Zuzanna Maciulewska

Temat projektu - książka kucharska

Aplikacja umożliwia przeglądanie przepisów, tworzenie nowych, edytowanie oraz usuwanie istniejących. Każdy przepis zawiera następujące informacje

- Tytuł (informacja niezbędna, nie więcej niż 20 znaków)
- Listę składników(opcjonalnie)
- Instrukcję przygotowania(informacja niezbędna)
- Czas przygotowania dania (opcjonalnie, liczba większa od 0)
- Link do źródła przepisu(opcjonalnie, Adres URL zaczynający się od http/https)

Architektura komponentów

W aplikacji zastosowano architekturę flux z wykorzystaniem implementacji Redux.
Na zrzucie ekranu poniżej została przedstawiona struktura plików w projekcie. Komponenty zostały podzielone na dwa rodzaje tzw. “Dumb Components”, które zajmują się warstwą prezentacji oraz “Smart Components” (containters), które odpowiadają za komunikację z store dostarczonym dzięki implementacji Redux. W celu obsługi wywołań asynchronicznych zastosowano middleware redux-thunk.
Opis plików:

- src/index.js - główny plik, który jest odpowiedzialny za wyświetlanie wszystkich komponentów. Zdefiniowany w nim jest Redux store oraz dołączony do niego redux-thunk middleware. Wywołana zostaje metoda, która pobiera przepisy z zewnętrzengo API, aby można było wyświetlić je w komponencie RecipeList. Zostaje wyrenderowany komponent App.js
- src/App.js - główny komponent, który jest nadrzędnym dla pozostałych. Zostaje tutaj zdefiniowany routing. W zależności od danego adresu URL są wyświetlane komponenty RecipeList, CreateRecipe lub EditRecipe.

Reducers - implementacja elementów wymaganych przez zastosowanie Redux’a

- src/reducers/index.js - nadpisanie funkcji dostarczonej przez bibliotekę redux. Definiuje się w nim obiekt, który jest nadrzędnym dla wszystkich obiektów zwracanych przez różne funkcje reducera
- src/reducers/recipeReducer.js - funkcja ta pełni rolę store z architektury Flux. Jako parametry przyjmuje state oraz action. W zależności od action.type, które są zaimportowane z package src/actions zostaje zwrócony zaktualizowany stan.

Actions - definicja wywoływanych akcji

- src/actions/types.js - definicja możliwych do wywołania akcji, pełni rolę interfejsu między recuderem, a definicją akcji
- src/actions/index.js - implementacja możliwych do wywołania akcji. Tutaj następuje komunikacja z zewnętrznym API. W zależności od wywołanej akcji zostają przekazywane różne parametry. W efekcie wywołanej akcji następuje odwołanie do reducera

Kontenery (package containers) - odpowiedzialne za komunikację pomiędzy akcjami, a komponentami odpowiedzialnymi za renderowanie widoków.

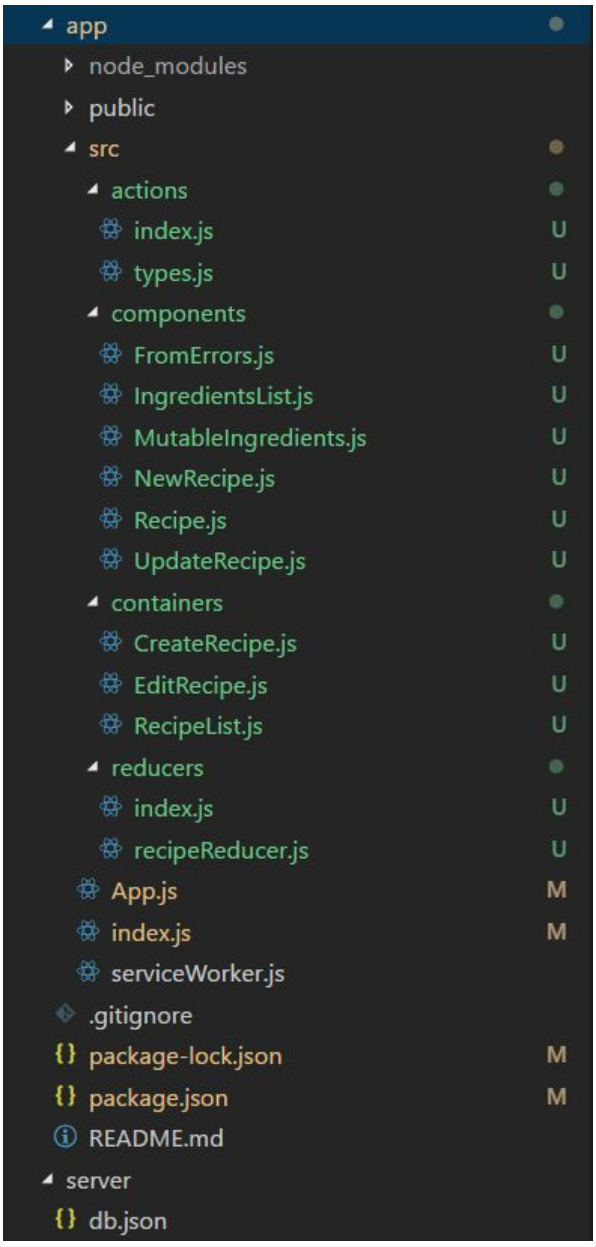
- src/containers/CreateRecipe.js - przekazuje obiekt recipe z komponentu NewRecipe.js do akcji createRecipe z src/actions/index.js obiekt recipe znajduje się w tym kontenerze
- src/containers/RecipeList.js - ten komponent funkcyjny przekierowuje do dwóch akcji. W przypadku gdy zostaje wyświetlana lista to mapuje stan ze store na property recipes. Wyświetla obiekty z listy recipes z wykorzystaniem komponentu Recipe.js. Jako parametr przekazuje obiekt recipe o danym kluczu z listy. Drugą akcją za jaką jest odpowiedzialny jest usuwanie przepisu. W tym przypadku mapuje id jako parametr akcji deleteRecipe.
- src/containers/EditRecipe.js - ten komponent podobnie jak CreateRecipe przekazuje obiekt recipe jako parametr do akcji updateRecipe z pliku src/actions/index.js

Komponenty (components) - ich podstawową funkcjonalnością jest renderowanie widoków z wykorzystaniem przekazanych im obiektów.

- src/components/NewRecipe.js - nie dostaje parametru wejściowego, tworzy nowy obiekt Recipe, który za pośrednictwem CreateRecipe.js przekazuje do akcji createRecipe. Zawiera w sobie komponent MutableIngredients, do którego przekazuje jako parametr pustą listę ingredients, która następnie jest przez ten komponent modyfikowana. Kolejnym komponentem w nim zawartym jest FormErrors.
- src/components/UpdateRecipe.js - ten komponent otrzymuje jako parametr id przepisu, który jest przekazany poprzez id w routingu. Po zatwierdzeniu zostaje przekazany obiekt recipe za pośrednictwem kontenera EditRecipe do akcji updateRecipe. W tym komponencie również zawarty jest komponent MutableIngredients.js. W tym przypadku jako parametr dostaje wypełnioną listę składników które następnie modyfikuje. Kolejnym komponentem w nim zawartym jest FormErrors.
- src/components/Recipe.js - komponent odpowiedzialny za wyświetlanie pojedynczego reordu z listy recipes. Jako parametr otrzymuje pojedynczy obiekt recipe z kontenera RecipeList.js. Zawiera w sobie komponent IngredientList, któremu jako parametr przekazuje listę składników. W tym komponencie istnieje przycisk odpowiadający za usuwanie przepisu, przekierowuje do akcji deleteRecipe za pośrednictwem RecipeList kontenera. W przypadku tej akcji jako parametr przekazuje Id przepisu. Istnieje również opcja edytowania przepisu. Następuje przekierowanie za pomocą routingu do kontenera UpdateRecipe. Jako parametr jest podany w adresie url id przepisu
- src/components/IngredientsList.js - komponent odpowiedzialny za wyświetlanie listy składników. Jako parametr otrzymuje tablicę ingredients.

- src/components/MutableIngredients.js - komponent odpowiedzialny za zarządzanie listą składników. Jako parametr otrzymuję tablicę składników. Następnie ją modyfikuje. Zmiany są widoczne w komponencie rodzicu (czyli NewRecipe.js i EditRecipe.js)
- src/components/FormErrors.js - komponent odpowiedzialny za wyświetlanie listy błędów walidacji. Jako parametr otrzymuje listę takich błędów.

Struktura projektu



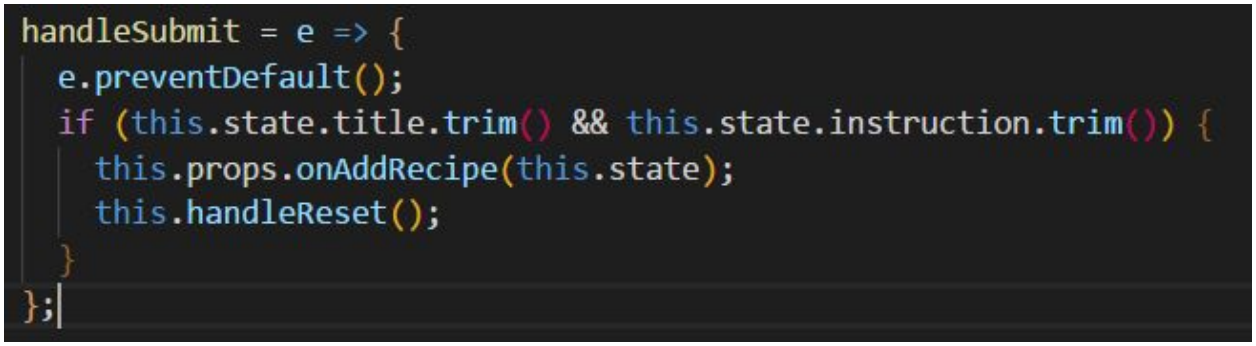
Elementy punktowane

1. Dwupoziomowa struktura danych (lista obiektów, number, string, url, tablica obiektów typu string)

```
{
  "recipes": [
    {
      "id": 1,
      "title": "SZARLOTKA Z BUDYNIOWĄ PIANKĄ",
      "instruction": " Do mąki dodać proszek do pieczenia, cukier oraz pokrojone w kostec
      "sourceUrl": "https://www.kwestiasmaku.com/przepis/szarlotka-z-budyniowa-pianka",
      "mealPrepTime": "120",
      "ingredients": [
        "300 g mąki pszennej",
        "1 łyżeczka proszku do pieczenia",
        "200 g masła (zimnego)",
        "4 jaja",
        "2/3 szklanki cukru",
        "2 budynie waniliowe po 40 g (bez cukru)",
        "1/3 szklanki oleju roślinnego",
        "1 kg prażonych jabłek"
      ]
    },
    {
```

2. Funkcjonalności
   Dodawanie przepisów
   Usuwanie przepisów
   Odczyt przepisów
   Edytowanie przepisów
   Usuwanie/dodawanie składników
3. Walidacja danych przekazanych w formularzach

   Sprawdzenie czy title i instruction nie są puste W UpdateRecipe.js i NewRecipe.js



```
handleSubmit = e => {
  e.preventDefault();
  if (this.state.title.trim() && this.state.instruction.trim()) {
    this.props.onAddRecipe(this.state);
    this.handleReset();
  }
};
```

Sprawdzenie czy title jest dłuższy niż 0 znaków i krótszy niż 20 znaków z wykorzystaniem komponentu FromErrors.js w komponentach NewRecipe.js i UpdateRecipe.js

Sprawdzenie czy mealPrepTime jest większy od 0

```
handleInputChange = e => {
  const name = e.target.name;
  const value = e.target.value;
  this.setState({ [name]: value },
    () => { this.validateField(name, value) });
  this.setState({
    [e.target.name]: e.target.value
  });
};
```

```
<button type="submit" disabled={!this.state.formValid} className="btn btn-primary">Add Recipe</button>
```

```
validateField(fieldName, value) {
  let fieldValidationErrors = this.state.formErrors;
  let titleValid = this.state.titleValid;
  let mealPrepTimeValid = this.state.mealPrepTimeValid;

  switch (fieldName) {
    case 'title':
      //titleValid = value.match(/^([\w.%+-]+)@([\w-]+\.)+([\w]{2,})$/i);
      titleValid = value.length < 20 && value.length>0 ;
      fieldValidationErrors.title = titleValid ? '' : ' must be less than 20 characters, cannot be empty';
      break;
    case 'mealPrepTime':
      mealPrepTimeValid = value > 0;
      fieldValidationErrors.mealPrepTime = mealPrepTimeValid ? '' : ' must be bigger than 0';
      break;
    default:
      break;
  }
  this.setState({
    formErrors: fieldValidationErrors,
    titleValid: titleValid,
    mealPrepTimeValid: mealPrepTimeValid
  }, this.validateForm);
}

validateForm() {
  this.setState({ formValid: this.state.titleValid && this.state.mealPrepTimeValid });
}
```

Ponadto zastosowano walidację uniemożliwiającą w miejscu source url wstawienie czegoś poza adresem URL oraz wstawienie czegoś, co nie jest liczbą w MealPrepTime.

4. Komponenty w tym jeden funkcyjny
   W projekcie jest 6 komponentów odpowiedzialnych za prezentację danyh oraz 3 komponenty odpowiedzialne za komunikację ze store. Komponentem funkcyjnym jest IngredientsList.js

```
import React from 'react';

const IngredientsList = props => (
  <ul>
    {
      props.items.map((item, index) => <li key={index}>{item}</li>)
    }
  </ul>
);

export default IngredientsList;
```

5. Komponenty reużywalne - FromErrors.js i MutebleIngredients.js są używane w komponentach NewRecipe.js oraz UpdateRecipe.js
6. Modyfikacja danych za pomocą 4 typów żądań (POST, PUT, GET, DELETE) w src/actions/index.js
   POST

```
export const createRecipe = ({ title, instruction, ingredients, mealPrepTime, sourceUrl }) => {
  return (dispatch) => {
    return axios.post(`${apiUrl}`, {title, instruction, ingredients, mealPrepTime, sourceUrl})
      .then(response => {
        dispatch(createRecipeSuccess(response.data))
      })
      .catch(error => {
        throw(error);
      });
  };
};
```

   PUT

```
export const updateRecipe = (recipe) => dispatch => {
  return axios.put(`${apiUrl}/${recipe.id}`, recipe)
    .then(response => dispatch(updateRecipeSuccess(response.data)))
    .then(dispatch(fetchAllRecipes()) )
    .catch(error => {
      throw(error);
    });
}
```

   GET

```
export const fetchRecipe = (id) => dispatch => {
  return axios.get(`${apiUrl}/${id}`)
    .then(response => {
      dispatch({
        type: FETCH_RECIPE,
        payload: response.data
      })
    })
    .catch(error => {
      throw(error);
    });
};
```

```
export const fetchAllRecipes = () => {
  return (dispatch) => {
    return axios.get(apiUrl)
      .then(response => {
        dispatch(fetchRecipes(response.data))
      })
      .catch(error => {
        throw(error);
      });
  };
};
```

DELETE

```
export const deleteRecipe = id => {
  return (dispatch) => {
    return axios.delete(`${apiUrl}/${id}`)
      .then(response => {
        dispatch(deleteRecipeSuccess(id))
      })
      .catch(error => {
        throw(error);
      });
  };
};
```

7. Routing
   W pliku src/App.js zdefiniowano router oraz adresy routingu

```
class App extends Component {
  render() {
    return (
      <Router>
      <div className="container">
      <Route exact path="/" component={RecipeList} />
      <Route path="/new" component={CreateRecipe} />
      <Route path="/edit/:id" component={EditRecipe} />
      </div>
      </Router>
    );
  }
}
```

Przykład użycia w komponencie Recipe.js

```
 </button>
 <Link className="btn btn-primary float-right" to={"/edit/" + id}>Edit</Link>
</div>
```

8. Flux (Redux)
   Implementacja architektury została opisana w punkcie dotyczącym architektury aplikacji.
   Definicja store w src/index.js

```
const store = createStore(rootReducer, applyMiddleware(thunk));

store.dispatch(fetchAllRecipes());

ReactDOM.render(
    <Provider store={store}>
        <App />
    </Provider>, document.getElementById('root'));

serviceWorker.register();
```

Reducer zarządzający store src/reducers/recipeReducer.js

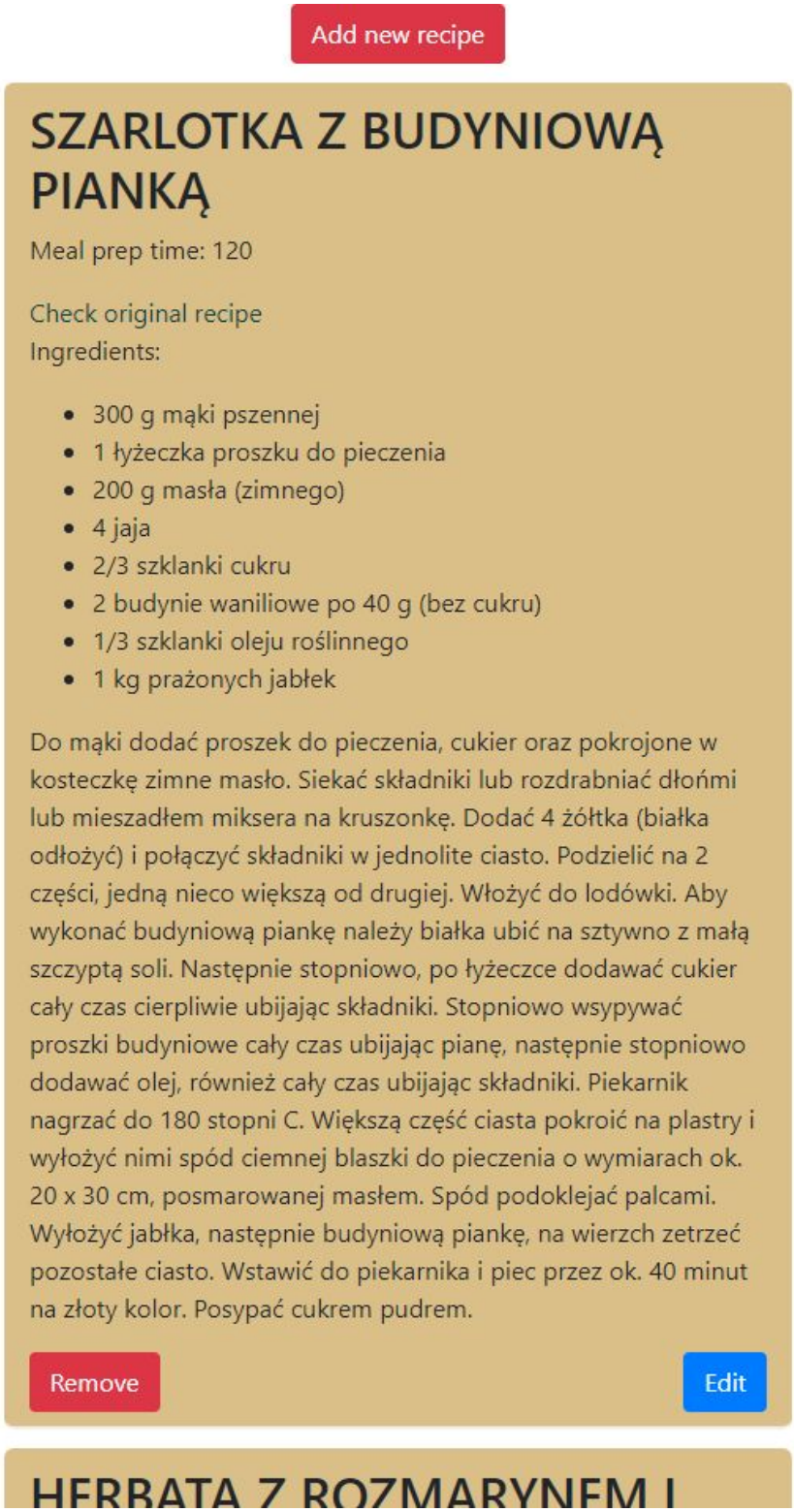
```
import { ADD_RECIPE, DELETE_RECIPE, FETCH_RECIPES, UPDATE_RECIPE, FETCH_RECIPE } from '../actions/types';

export default function recipeReducer(state = [], action) {
    switch (action.type) {
        case ADD_RECIPE:
            return [...state, action.payload];
        case DELETE_RECIPE:
            return state.filter(recipe => recipe.id !== action.payload);
        case FETCH_RECIPES:
            return action.recipes;
        case UPDATE_RECIPE:
            return [ action.payload, ...state];
        case FETCH_RECIPE:
            return {
                ...state,
                recipe: action.payload
            }
        default:
            return state;
    }
}
```

Definicja głównego obiektu w store src/reducers/index.js

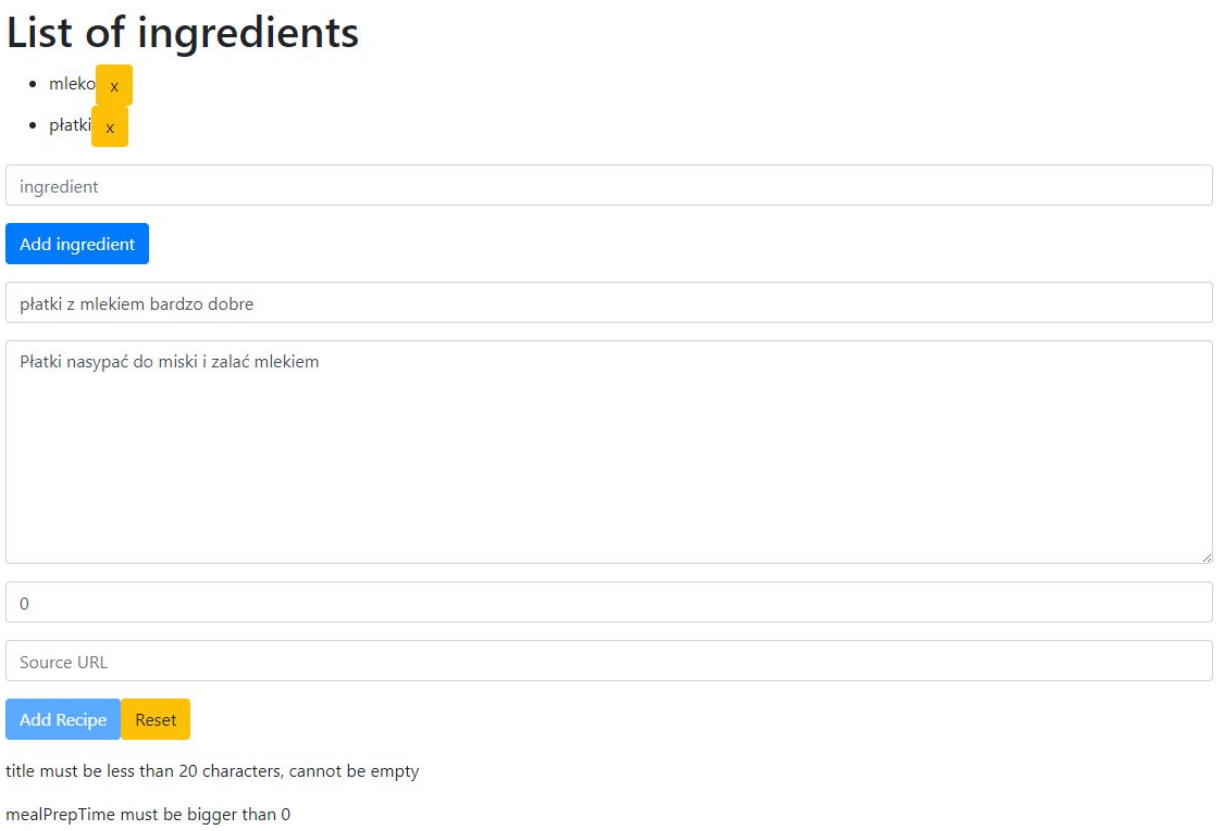
```
import { combineReducers } from 'redux';
import recipes from './recipeReducer';

export default combineReducers({
    recipes: recipes
});
```

Instrukcja użytkownika

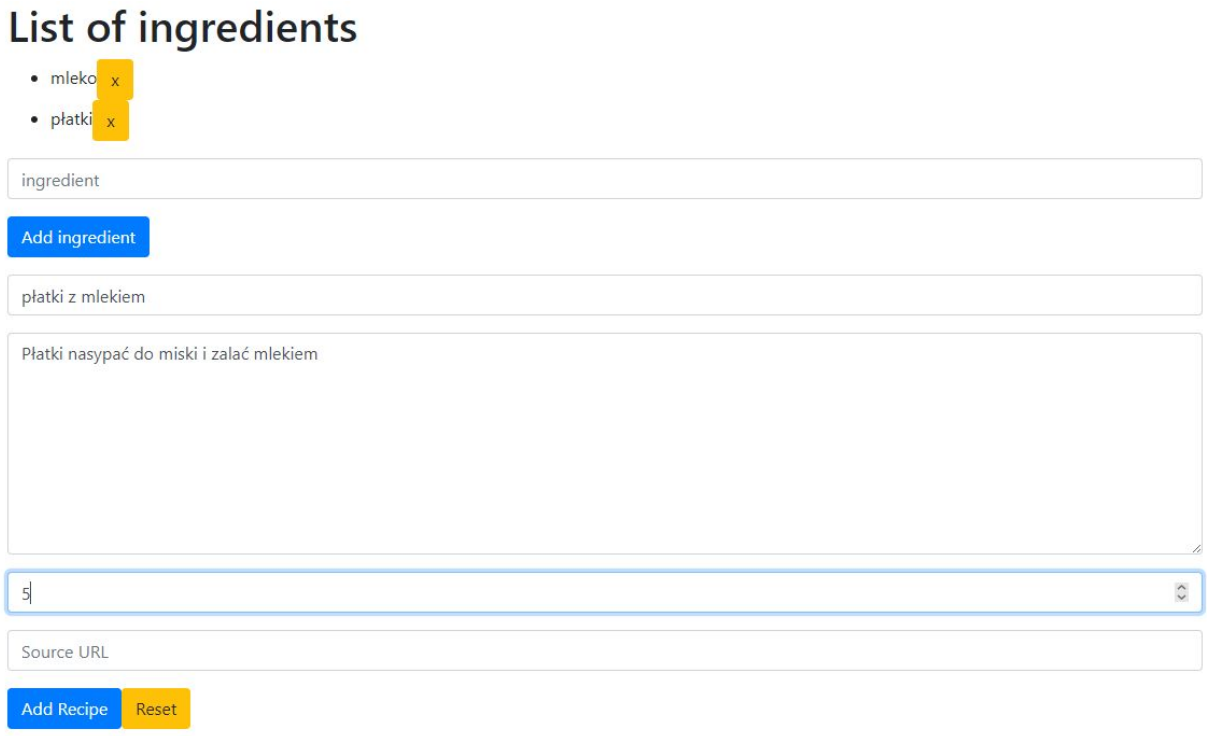
Strona startowa



Po naciśnięciu przycisku Add new recipe przechodzi się do formularza dodawania nowego przepisu

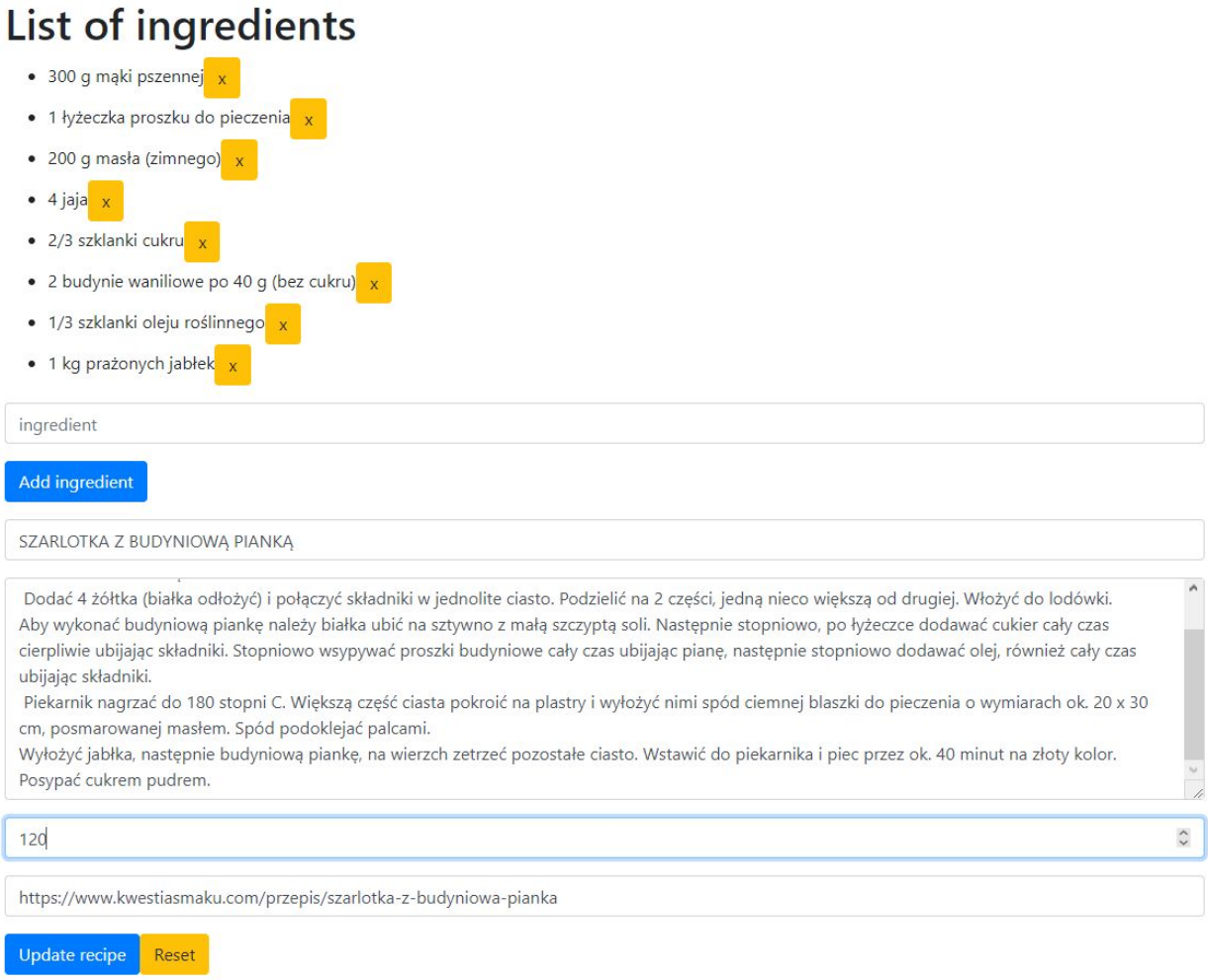


W tym przypadku wyświetliły się błędy walidacji, więc nie można zatwierdzić formularza.



W tym przypadku formularz jest poprawnie wypełniony, można go zatwierdzić i dodać przepis.

Na stronie głównej przy każdym przepisie widnieją dwa przyciski “Delete” oraz “Edit”. W przypadku naciśnięcia pierwszego przepis zostanie usunięty, a strona główna przeładowana. Po naciśnięciu przycisku “Edit” przechodzimy do formularza edycji przepisu. Jest on analogiczny do formularza tworzenia, z tą różnicą, że jest on wypełniony obecnymi danymi. W tym przypadku również formularze są walidowane.



Instalacja

W folderze ReactProject/server należy wpisać komendę json-server -p 4000 db.json
W folderze ReactProject/app należy wpisać komendę npm start