Tutorial 12: ReactJS (3)

Objectives

In this tutorial, we will continue with the flashcards-react app & focus on practicing:

- Fetching data
- Routing with react-router-dom

Tutorial Exercises

Download the *tut11_solution: flashcards-react* & complete the exercises below.

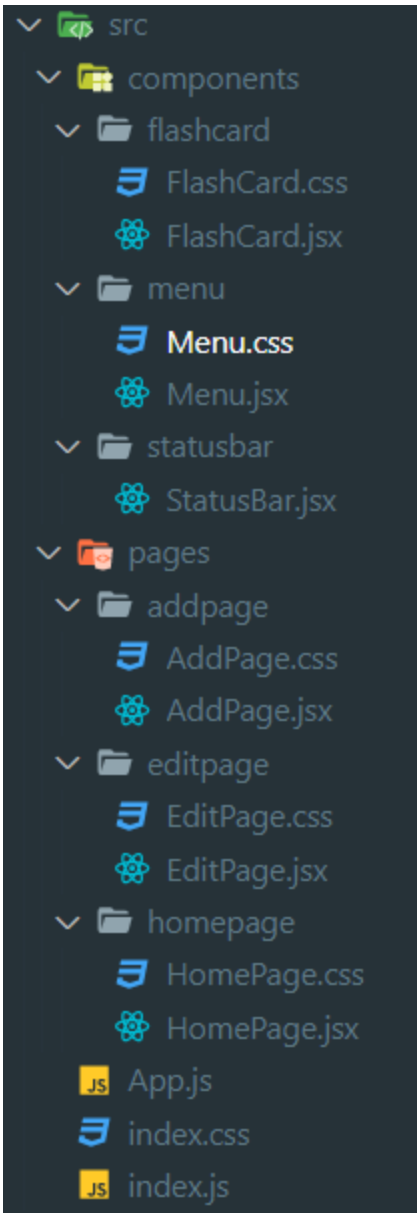
Exercise 1: Your second page (20 mins)

- In this exercise, we add a new UI (page) to input data (word + definition) for creating a new flashcard.

Task 1: refactor App to become a page named `HomePage`

We consider App component – what we are having now, as a page (`HomePage`). This page provides basic function to learn new words (flipping card, navigating cards).

Note: The suggested folder structure is as follow.



- Copy `App` component then rename to become `HomePage`
- Refactor `App` component to use just created `HomePage`

```
render() {
  <HomePage />
}
```

Task 2: create a new page named AddPage

- Create a new page (actually a component) named `AddPage`. This page displays a form that allows user to add a new FlashCard.

Note: The suggested folder structure is as above.

- At first, this page contains ONLY a title “new FlashCard”

- Refactor `App` component to use just created `AddPage`

```
render() {
  return <>
    <HomePage />
    <AddPage />
  </>;
}
```

Exercise 2: Link pages with `<Link>`, `<Switch>`, `<NavLink>` (20 mins)

We are displaying about `HomePage` & `AddPage` at the same time. Let’s update to display corresponding page based on the route:

1) Route: `/` → `HomePage`
2) Route: `/add` → `AddPage`

- Install required package for routing functionality: **react-router-dom@5.3.0**
- **Step 1:** update **`index.js`** to wrap App component with **`<BrowserRouter>`**

```
import {BrowserRouter} from 'react-router-dom';

ReactDOM.render(
    <BrowserRouter>
        <App />
    </BrowserRouter>,
    document.querySelector('#root')
);
```

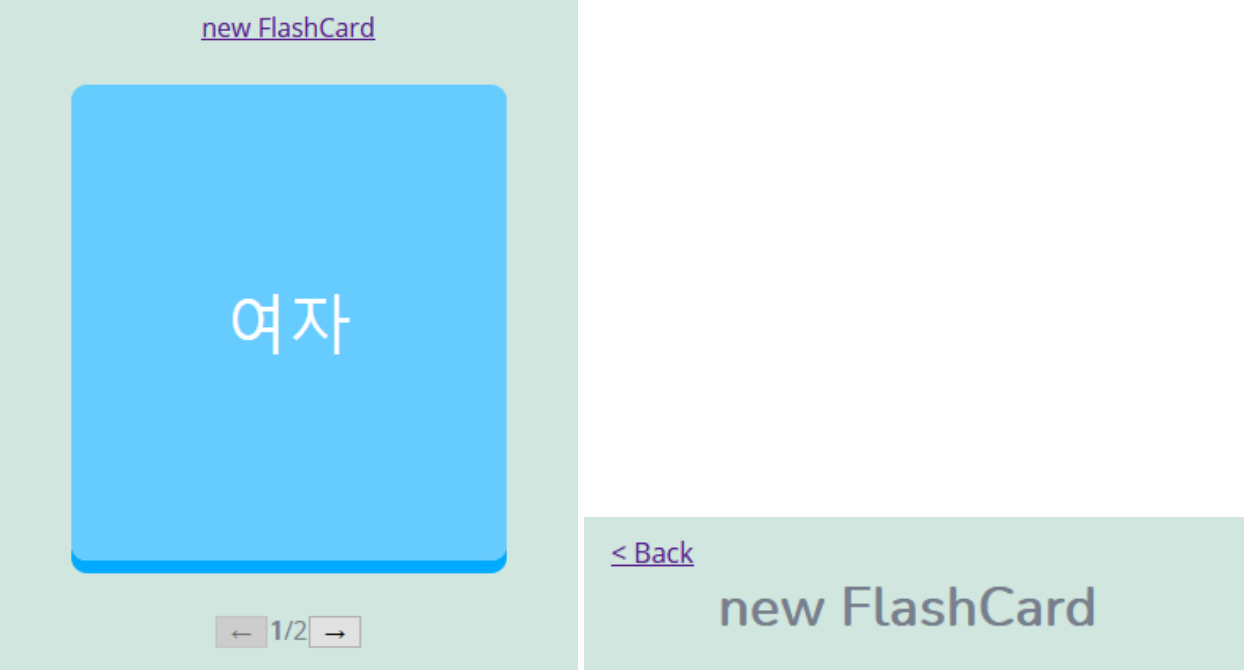
- **Step 2:** update **`App.js`** to display corresponding page based on the route using **`<Route>`** Test displaying different components by manually changing the url on the browser.

```
render() {
    return <div className="App">
      <Switch>
        <Route exact path="/" component={HomePage} />
        <Route exact path="/add" component={<AddPage />} />
      </Switch>
    </div>;
  }
```

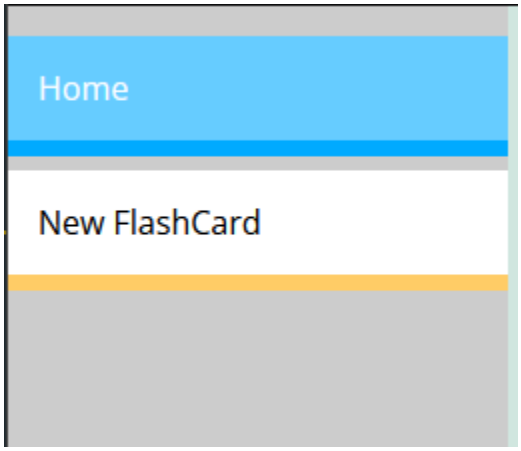
- **Step 3**: update `HomePage` & `AddPage` to add links to each other using **`<Link>`**

```
export default function Card({monster}) {
  return <div className="card-container">
    <img
      alt={monster.name}
      src={`https://robohash.org/${monster.id}?set=set2&size=180x180`}
    />
    <h2>{monster.name}</h2>
    <p>{monster.email}</p>

    <Link to={`/${monster.id}`}>Details</Link>
    <Link to={`/${monster.id}/edit`}>Edit</Link>

  </div>;
}
```

Expected result is as follow:



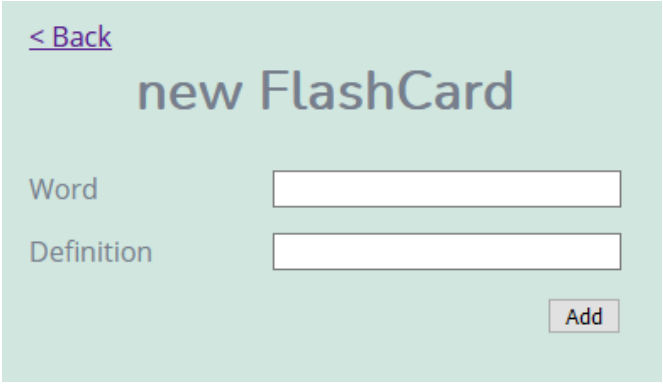
- **Step 4**: The menu



  - Create a new component for the menu with provided style.
  - Use **`<NavLink>`** to active corresponding menu based on route
  - Use menu in the `App` component

Exercise 3: AddPage – Form to create a new FlashCard

Let's use form to capture data (word, definition) & create a new FlashCard. Update the component with the form & use the provided style.



using **`state`** to store form data

- **`handleChange()`:** capture form input data changes into state
- **`handleSubmit()`:** do something with form data (`state`) & redirect back to `HomePage`

Note:

1) to redirect in React, you need to use history API with `withRouter()` from react-router-dom.
2) Using the API: ***POST*** https://wpr-quiz-api.herokuapp.com/cards to add a new card with required body {word, definition}.

OPTIONAL Exercises:

- EditPage – form to update a FlashCard
- Button delete a FlashCard with user confirmation