Implementing a Compelling User Interface in React

1. Setting the scene
2. Implementing routing
3. Displaying all destinations
4. Displaying one destination

```
React demo:  demo-full-stack-client
To install: npm install
To run:     npm start
```

Pearson

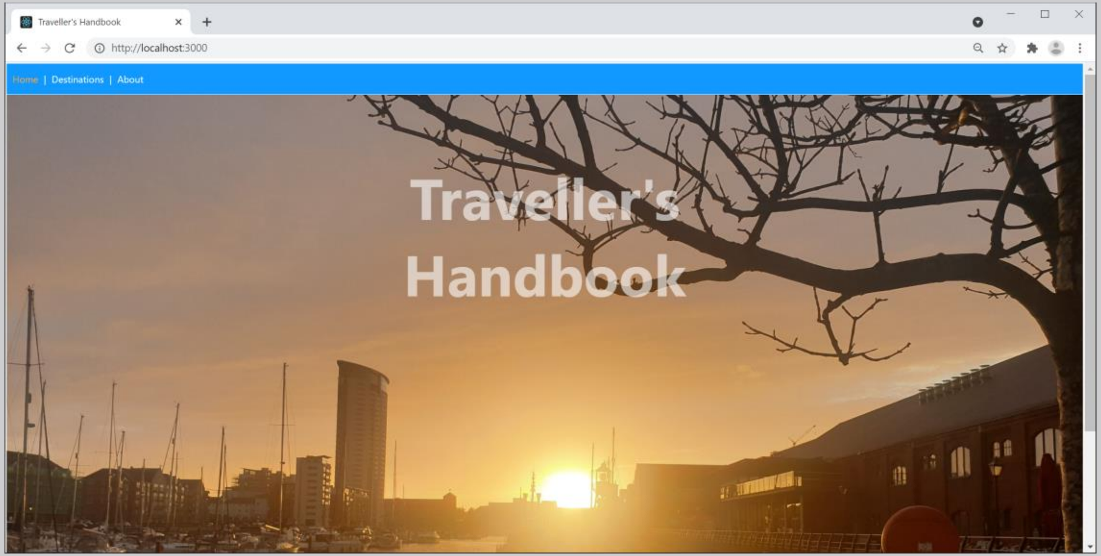
1. Setting the Scene

- Example application
- Application characteristics

Pearson

Example Application

- In this chapter we'll explore a complete UI in React
  - See `demo-full-stack-client`



Application Characteristics

- The application has several key characteristics:
  - Single-page application (SPA)
  - Compelling UI
  - Realistic application structure
  - Utilizes contemporary React patterns and techniques
  - Interacts with Spring Boot back-end via REST

2. Implementing Routing

- React and single-page applications
- Supporting routing in React
- Defining a router table
- Defining a menu

React and Single-Page Applications

- React has excellent support for implementing SPAs
  - Define an `App` component that always remains resident
  - Define multiple sub-components, which can be swapped in and out of the `App` component
- Each sub-component maps to a logical URL
  - This is called "routing"
  - To display a different sub-component in the browser, simply navigate to the URL for that sub-component

Supporting Routing in React

- Add these dependencies in `package.json`:

```
"dependencies": {
  "react-router-dom": "^5.2.0",
  "@types/react-router-dom": "^5.1.7",
  …
}
```

React Router

React Router TypeScript declarations

package.json

- Wrap your `App` component in `<BrowserRouter>`:

```
ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById('root')
)
```

index.tsx

Defining a Router Table

- Define a router table as follows, typically in `App`:

```
export default function App() {
  return ( … … …
    <Switch>

      <Route exact path="/" >
        <Home />
      </Route>

      <Route path="/destinations">
        <Destinations />
      </Route>

      <Route path="/destination/:id">
        <Destination />
      </Route>
      …
      <Route path="*" >
        <PageNotFound />
      </Route>

    </Switch>
  )
}
```

App.tsx

Defining a Menu

- It's common to define some kind of menu component
  - Use `<NavLink>` to create links to your routes

```
export default function Menu() {
  return (
    <nav>
      <NavLink exact to="/">Home</NavLink>&nbsp;|&nbsp;
      <NavLink to="/destinations">Destinations</NavLink>&nbsp;|&nbsp;
      <NavLink to="/about">About</NavLink>
    </nav>
  )
}
```

`Menu.tsx`

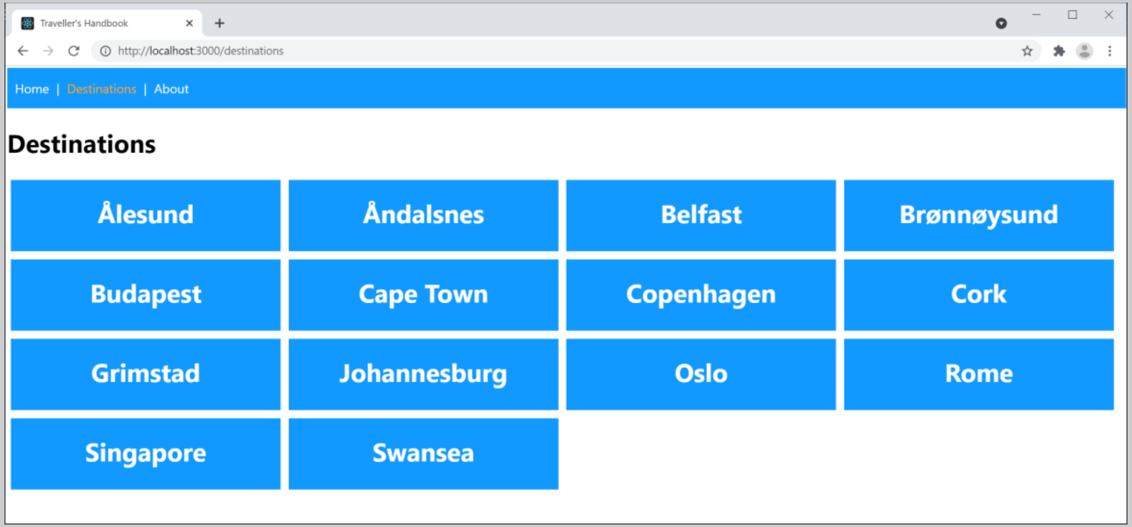
- You typically display the menu in your `App` component
  - See `App.tsx`

3. Displaying all Destinations

- Overview
- How the Destinations component works
- Using React state storage
- Using React effect hooks
- Displaying parameterized route links

Overview

- Browse to the `/destinations` location
  - The `Destinations` component is activated



Traveller's Handbook

http://localhost:3000/destinations

Home | Destinations | About

Destinations

Ålesund	Åndalsnes	Belfast	Brønnøysund
Budapest	Cape Town	Copenhagen	Cork
Grimstad	Johannesburg	Oslo	Rome
Singapore	Swansea		

How the `Destinations` Component Works

- The `Destinations` component gets its destinations data via REST
  - This is asynchronous, so it could take a while…

- When the destinations data finally arrives:
  - Store the data in React state
  - Causes React to automatically re-render the component

Using React State Storage

- You can't store state (data) in a local variable
  - Local variables disappear at the end of the function
- Instead you must use `React.useState()`

```
let [destinations, setDestinations] = React.useState<Array<any>>([])     Destinations.tsx
```

- `React.useState()` returns:
  - A reference to state data maintained by React
  - A function you must call if you change the state (this tells React to re-render your component)

Using React Effect Hooks

- You don't need to get data on every render
  - Just get data after first render, and store it in React state

- If you have work you want to do after a component is rendered, call `React.useEffect()`
  - Pass in a **lambda**, specifying the work you want to do
  - Pass in a **dependency array**

```
React.useEffect(() => {
    RestClient.getDestinations()
              .then(destinations => setDestinations(destinations))
}, [])                                                    Destinations.tsx
```

Displaying Parameterized Route Links

- The `Destinations` component displays hyperlinks for all the destinations, parameterized by id
  - E.g. `destination/1`

```
{destinations.map((dest: any, i: number) =>
  <Link to={`destination/${dest.id}`} key={i} className='blockLink'>{dest.place}</Link>
)}
```

Destinations.tsx

- `index.css` defines styles so the links are uber-cool!

```
.blockLink { … … … }

.blockLink:hover { … … … }
```

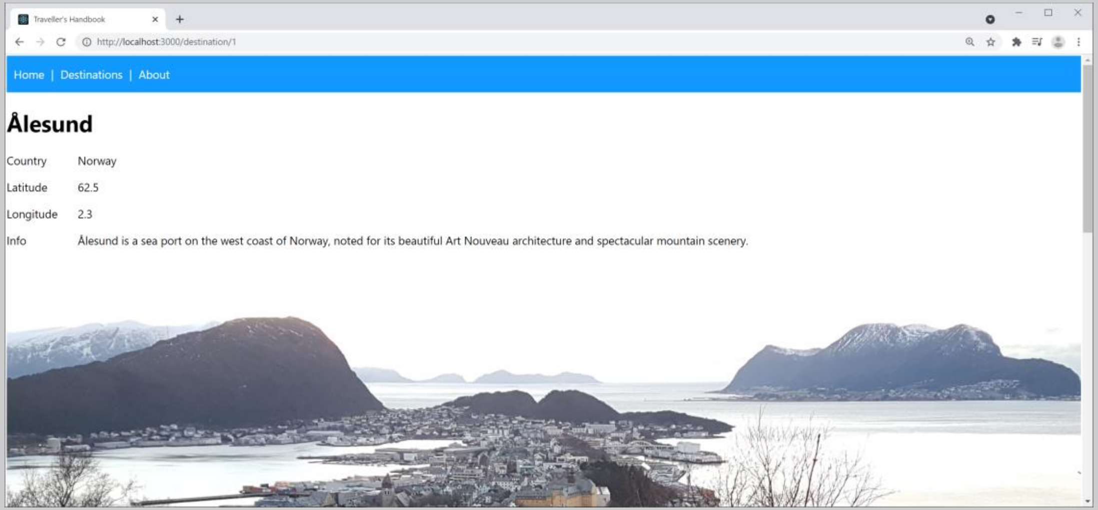
index.css

4. Displaying one Destination

- Overview
- How the Destination component works
- Component modular decomposition
- Defining and using custom hooks

Overview

- Click one of the destination links
  - The `Destination` component is activated, with an id
  - The component displays details for that destination


Traveller's Handbook
http://localhost:3000/destination/1
Home | Destinations | About

Ålesund

Country	Norway
Latitude	62.5
Longitude	2.3
Info	Ålesund is a sea port on the west coast of Norway, noted for its beautiful Art Nouveau architecture and spectacular mountain scenery.

How the `Destination` Component Works

- The `Destination` component gets the id parameter from the location URL (see the routing table)

```
let {id} : any = useParams()                                                  Destination.tsx
```

- The component then gets data for that destination, and stores the data in React state

```
let [destination, setDestination] = React.useState<any>(undefined)

React.useEffect(() => {
  RestClient.getDestination(id)
             .then(destination => setDestination(destination))
             .catch(err => alert(err))
}, [id])                                                                      Destination.tsx
```

Component Modular Decomposition

- The `Destination` component is quite complex
  - So we render it in 2 separate sub-components

```
<React.Fragment>
  <DestinationDetails {...destination} />
  <DestinationReviews {...destination} />
</React.Fragment>                                              Destination.tsx
```

Defining and Using Custom Hooks (1 of 2)

- The `DestinationReviews` component makes use of "custom hooks" to simplify rendering

```
function DestinationReviews(destination: any) {
  return (
    <React.Fragment>
      {useReviewsMarkup(destination)}
      {useAddReviewFormMarkup(destination)}
    </React.Fragment>
  )
}
```

Custom hook

Custom hook

Destination.tsx

- A custom hook is a helper function
  - Contains reusable logic to simplify components
  - Function name conventionally starts with `use`

Defining and Using Custom Hooks (2 of 2)

- The `addReviewFormMarkup()` custom hook displays a `<form>` where the user can add a review

- When the user submits the form, the function:
  - Collates the review details into an object
  - Sends the review object to the server, via a REST API
  - Adds the review to the client-side destination object
  - Causes the component to re-render (via a state change)

Summary

- Setting the scene
- Implementing routing
- Displaying all destinations
- Displaying one destination