# React pt. 3: Context, Pages, and Routers

Kianoosh Abbasi

CSC309 Fall 2022

# So far

- HTML, CSS, and Django backend
- JavaScript front-end
  - DOM, jQuery, Ajax
  - Advanced topics: closures, arrow functions, promises
- Single-page applications with React
  - JSX, props, events, state
  - React projects, NodeJS, hooks
  - API calls

# This week

- Global state and Context
- Multi-page React apps
  Routers and Links
- Review of concepts: P3 prep

# Prop drilling

- Passing state down to children can be quite cumbersome

- Example: The component that fires the request is a deep child button

  You need to pass both the state and its setter function all the way down

# Global state

- A global state is can be a great alternative
- Accessible everywhere!
  - No need to pass things all the way down
- Like global variables, don't use them for everything!
  - Makes your code dirty and harder to understand
  - Makes component re-use harder

# Context

- React's way to handle global state
- Create state variables and put them and/or setters in a context
- Everything inside the context is accessible within its provider

# Context

- Create the context (usually in a separate file)

```
export const TestContext = createContext({
    var1: null, var2: null,
});
```

- Put a default initial value for every variable that you will include in your context

# Provider

- Create an object
```
const myObject = { var1: 1, var2: 2 };
```

- Put a provider around the parent component and pass the object
```
<TestContext.Provider value={myObject}>
    …
</TestContext.Provider>
```

- At any descendent, you can access the context object
```
const { var1, var2 } = useContext(TestContext)
```

- More information:
  https://dmitripavlutin.com/react-context-and-usecontext/

# Why context is so great?

- Enables you to handle API data very easily
- Many components need to access them
  Username, profile data, etc.
- Various components can call APIs to fetch data
- For each Django app, create a context that includes the relevant values and their setters
  Its name should start with "use"

# Context example

```
export function useAPIContext() {

    const [deployment, setDeployment] = useState([]);

    const [servers, setServers] = useState([]);

    const [applications, setApplications] = useState([]);

    const [applicationStatus, setApplicationStatus] = useState([]);

    const [availableLogDates, setAvailableLogDates] = useState([]);

    return {
        deployment,
        setDeployment,
        servers,
        setServers,
        applications,
        setApplications,
        applicationStatus,
        setApplicationStatus,
        availableLogDates,
        setAvailableLogDates,
    };
```

```
ReactDOM.render(
  <React.StrictMode>
      <APIContext.Provider value={useAPIContext()}>
          <ControlPanel/>
      </APIContext.Provider>
  </React.StrictMode>,
  document.getElementById('root')
);
```

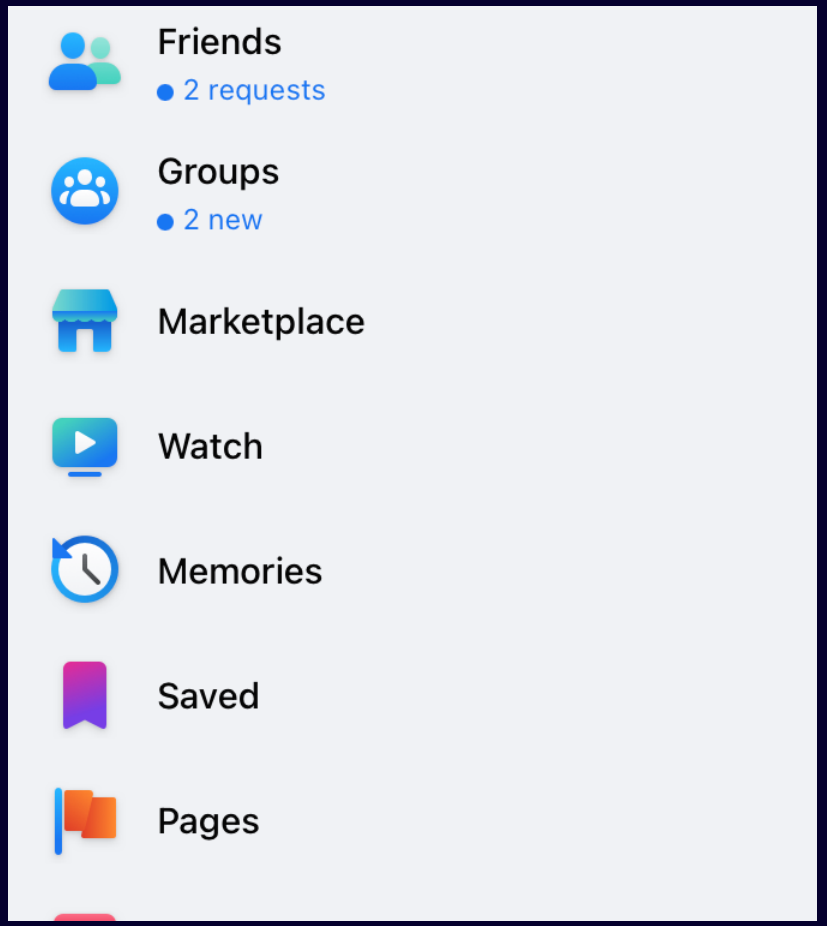
Codes by Myles Thiessen. https://thiessem.ca

# Multi-page React apps

# Pages

facebook.com/marketplace

facebook.com/memories




But there's still no browser reload!!

# Implementation with our current knowledge

```
export const TopLevelComponent = () => {
    const [page, setPage] = useState( initialState: "")

    // Tabs
    const Navbar = () => <nav>
        <a onClick={() => setPage( value: "watch")}> Watch </a>
        <a onClick={() => setPage( value: "groups")}> Groups </a>
        <a onClick={() => setPage( value: "marketplace")}> MarketPlace </a>
    </nav>

    // Render

    const Page = () => {
        switch(page) {
            case "watch":
                return <Watch />
            case "groups":
                return <Groups />
            case "marketplace":
                return <MarketPlace />
            default:
                return <Feed />
        }
    }

    return <Navbar>
        <Page />
    </Navbar>
}
```

# Pages

- Even though it's called single-page, it's good to have pages sometimes
- Example: Tabs
- If the components are very different, why bother with a state variable at the top level?

# Pages

- A key drawback of single-page applications is that there is no URL to copy for a specific part of the website
- Solution: Routers
  https://reactrouter.com/docs/en/v6/getting-started/tutorial
- Changes the URL without a browser reload!
- The specific component is accessible via that URL

# Routers

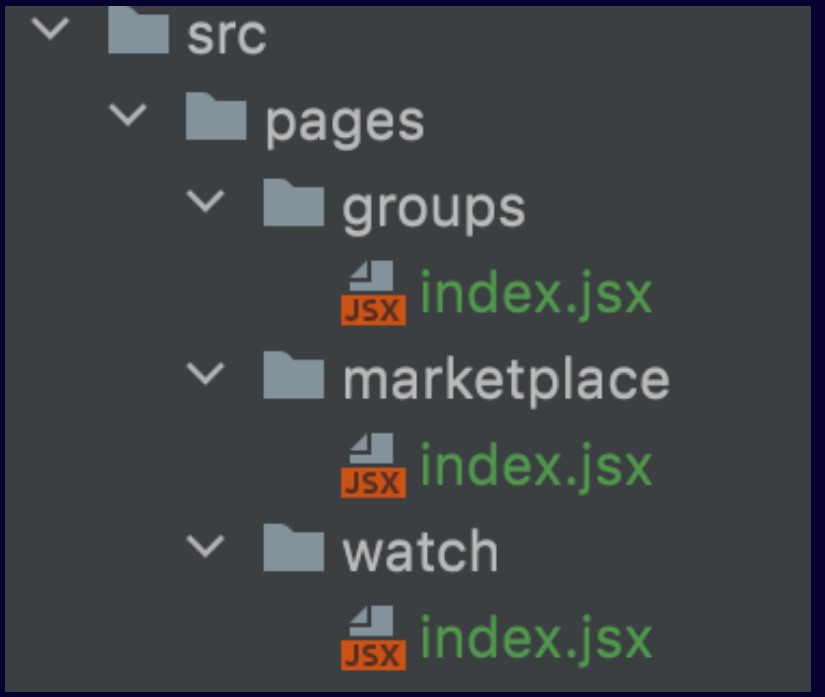
Visit https://www.w3schools.com/react/react_router.asp

- Installation

```
npm install react-router-dom
```

- Create a pages folder inside src

- Put each page's component in a separate file or directory (preferred) there



# Routers

- Introduce the routes in App.js

```
<BrowserRouter>
  <Routes>
    <Route path="/">
      <Route path="groups" element={<Groups />} />
      <Route path="marketplace" element={<MarketPlace />} />
      <Route path="watch" element={<Watch />} />
    </Route>
  </Routes>
</BrowserRouter>
```

- Now, test the URLs on your browser!

  /groups or /marketplace

# Links

- Like the familiar <a> tag, but without a browser reload

- Usage

```
<Link to="/watch">watch</Link>
```

# URL arguments

- Arguments are specified at the route definition

```
<Route path="watch/:watchID" element={<Watch />} />
```

- Can be accessed via a hook

```
const { watchID } = useParams();
```

- Usage is like before

```
<Link to="/watch/128">Watch</Link>
```

# Query parameters

- Accessed via another hook!

```
const [searchParams, setSearchParams] = useSearchParams();
```

- To extract a specific key:

```
searchParams.get('name')
```

- Usage

```
<Link to="/watch/128?name=kia">Watch</Link>
```

# Navigation

- You might need a URL change via code
- Example: If response is 401, redirect to the login page
- Like `window.location.replace()` in regular JS
- Via React router:

```
let navigate = useNavigate();
navigate("/marketplace")
```

# Outlet

- We still want a navbar to navigate through pages
  - It's a very bad idea to copy it at all children
- What if you had an element for root as well?
  - Then, that element will always be rendered!
  - All child elements will be ignored!!
- However, you can always access to a child component named outlet

# Outlet

- In nested routes, React renders the first components that partially matches the URL and has an element
- But it continues matching the rest of the URL and returns the matching child components as Outlet
  - Returns the index element if path is an exact match
- Root component is the layout: navbar, sidebar, header, etc. (like the base template in Django)
- Child components are rendered within that layout

# Index element

- Right now, the root path "/" is empty
  - You can specify an element for it
- But a better practice is to have an index element
  - People start browsing your site at the root path
- Usage

```
<Route index element={<Home />} />
```

# Full example

App.js

```
function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="groups" element={<Groups />} />
          <Route path="marketplace" element={<MarketPlace />} />
          <Route path="watch/:watchID" element={<Watch />} />
        </Route>
      </Routes>
    </BrowserRouter>
  )
```

pages/layout/index.jsx

```
const Layout = () => {
    return (
        <>
            <nav>
                <Link to="/">Home</Link>
                <Link to="/groups">Groups</Link>
                <Link to="/watch/1">Watch</Link>
                <Link to="/marketplace">Marketplace</Link>
            </nav>
            <Outlet />
        </>
    )
}
```

# Your project

- File structure for React projects varies
- A good practice is separating pages from reusable components
- Also, do not let your components become too big
  - Always extract child components in these cases

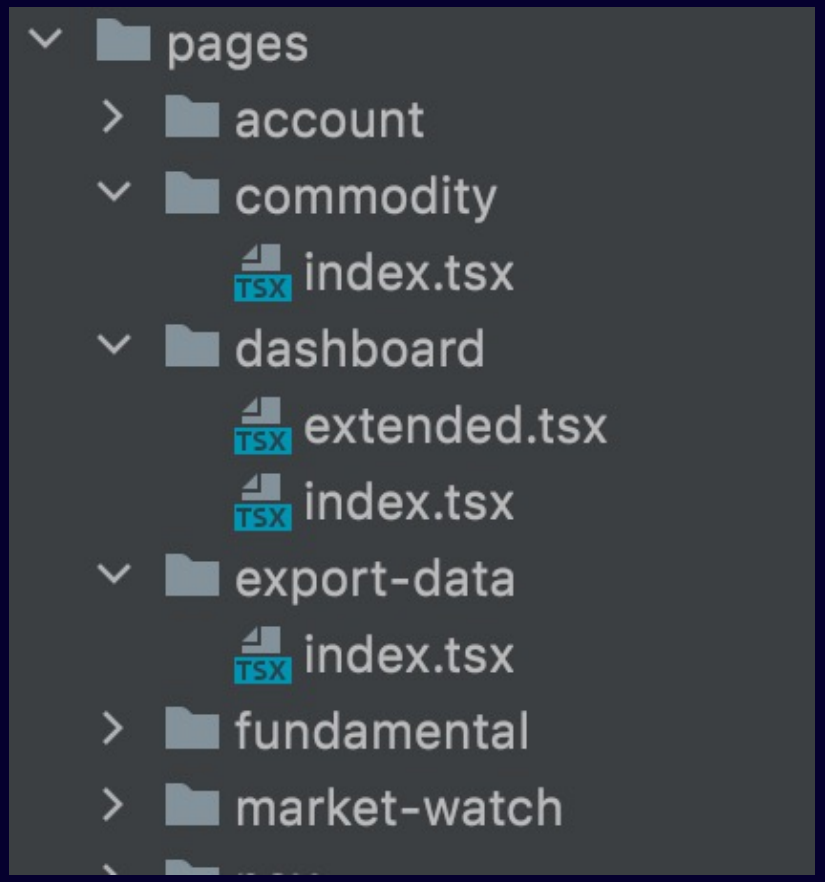
# Your project

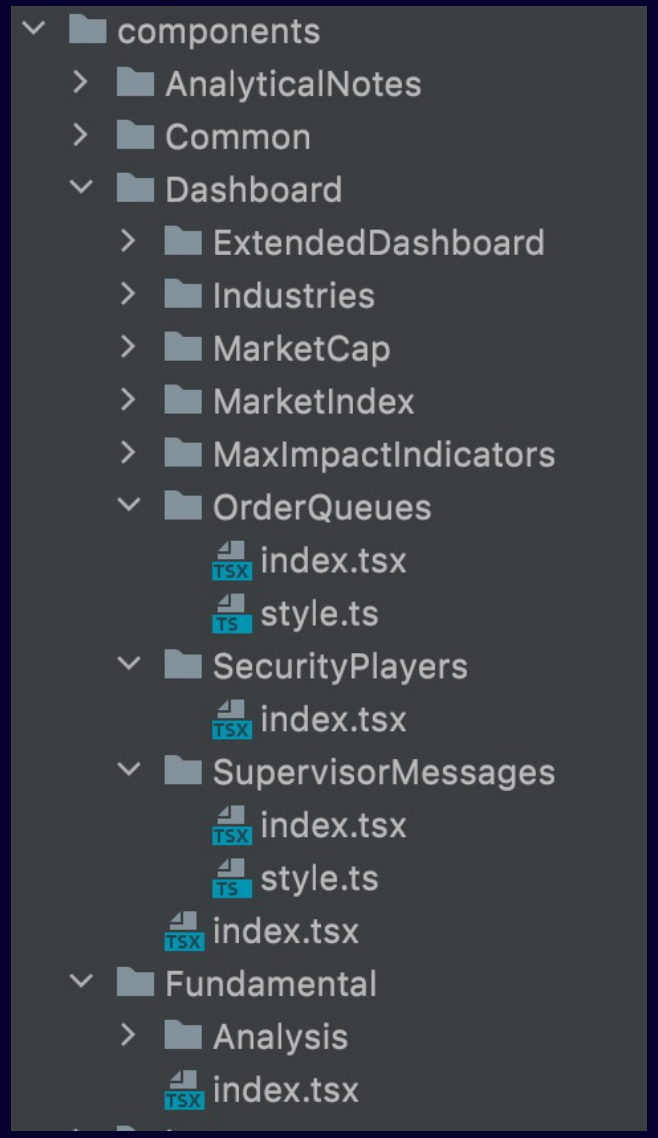
- So, expect your components to have multiple children
- That's why it's important to put every component/page inside a directory, not just a JSX file
- Child components can be the subdirectories

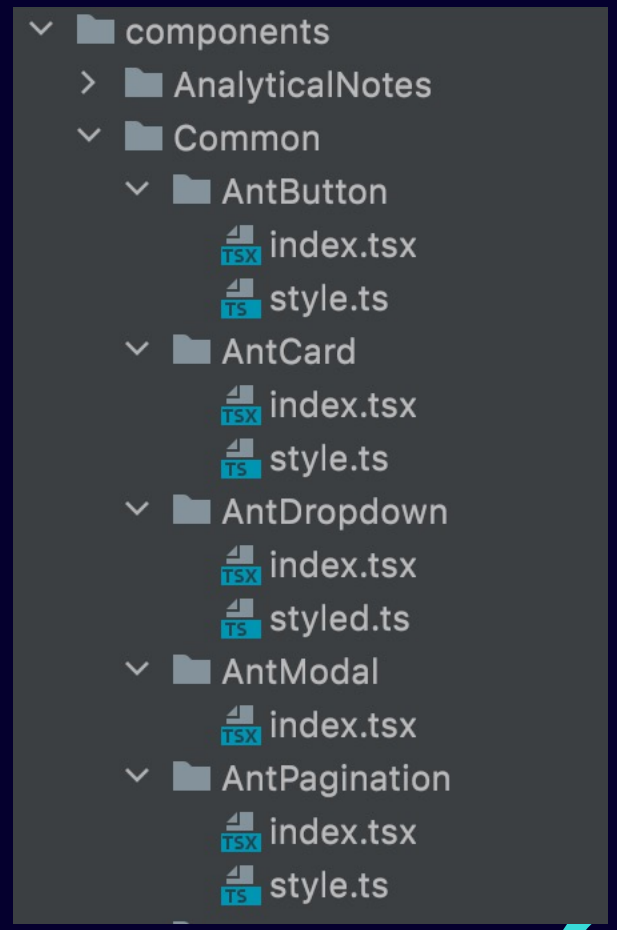
# Your project

- Always separate common/base components from pages
  Examples: inputs, tables, forms, buttons, etc.
- Dedicate a page to login, signup, forms, and navbar items
- Use function components and hooks instead class components

# Example file structure



# Review of Concepts: Q & A

# This week

- Global state and Context
- Multi-page React apps
  Routers and Links
- Review of concepts: P3 prep

# Next week

- Optional content for your interest!

- Backend and frontend deployment
  - DevOps
  - System-administration and Docker

- Course conclusion!