CSC309H1S

Programming on the Web

Winter 2023

Lecture 11: React Hooks, Context, and Router

Instructor: Kuei (Jack) Sun

Department of Computer Science
University of Toronto

v1.01

Hooks

History

- Functional components used to be “dumb”
  - Used for *presentation* only; cannot track internal state
  - Does not have access to lifecycle methods (more on this later)
- Class components are difficult to work with
  - Verbose syntax
  - Hard to reuse and share component logic
- Hooks
  - Introduced in React 16.8 (2019)
  - Make functional components much more versatile
  - Now the de-facto way for write clear and concise components

Hooks

- A set of functions that you can call inside a functional component
- E.g., `useState(`*`initialState`*`)`
  - Defines a single state variable within the component
  - Returns the variable and its update function
    - By convention, should be stored using array destructuring
  - Component re-rendered when update is called to change the variable

```
import React, {useState} from 'react';
const Status = (props) => {
    const [status, setStatus] = useState("good");
    const toggleStatus = () => setStatus(status === "good" ? "bad" : "good");
    return <>
        <h3>Situation is {status}</h3>
        <button onClick={toggleStatus}>Toggle</button>
    </>;
};
```

Using Hooks

- Rules of hooks
  1. Only call hooks at the top level
     - Required to ensure deterministic call ordering
  2. Only call hooks from React functions
- Further reading
  - https://medium.com/@ryardley/react-hooks-not-magic-just-arrays-cd4f1857236e
- Benefits
  - Supports multiple state variables
  - Easy to share state(s) with child elements
  - Easier to use compared to class components
- Quercus Exercise Q1

Lifecycle

- So far, we only run code when render is called
  - For both class and functional components
- However, we don't want to run expensive operation on every re-render
  - E.g., sending an ajax request only when component is first loaded
- Lifecycle methods
  - Executes when something happens to a component
  - Class components
    - `componentWillMount()`: before loading a component
    - `componentDidMount()`: after loading a component
    - `componentDidUpdate()`: after updating a component (except initial load)
    - `componentWillUnmount()`: before unloading a component

useEffect

- Replaces lifecycle methods

```
import React, {useState, useEffect} from 'react';
```

- Takes two parameters, a callback and an array of *dependencies*
  - If dependency is empty, callback only occurs on load.
  - Otherwise, callback occurs whenever a dependency changes

```
useEffect(() => {
    console.log("This is called when component mounts");
}, []);
```

- Subscription

```
useEffect(() => {
    console.log("props size or status has changed");
}, [status, props.length]);
```

Tip: Should have one `useEffect` per concern

Function vs. Class Component

```
function ShowCount(props) {
    const [count, setCount] = useState();

    useEffect(() => {
        setCount(props.count);
    }, [props.count]);

    return <div>
        <h1>Count : {count}</h1>
    </div>;
}
```

Function components is much more concise and readable.

```
class ShowCount extends React.Component {
    constructor(props) {
        super(props);
        this.state = { count : 0 };
    }

    componentDidMount() {
        this.setState({
            count : this.props.count
        });
    }

    render() {
        return <div><h1>Count :
            {this.state.count}</h1>
        </div>;
    }
}
```

useEffect Notes

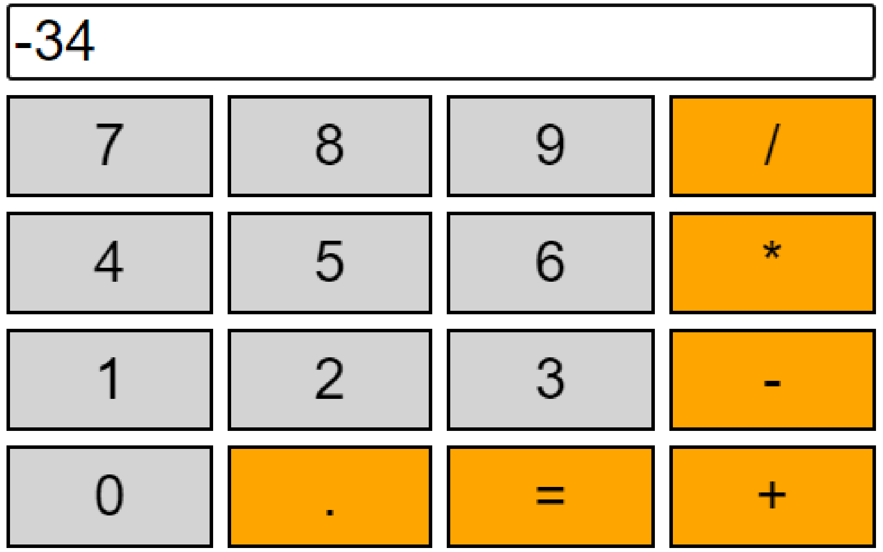
- If dependency is missing, effect would run at every re-render
  - Typically, this is not what you want, except…

```
function Counter() {
    const [count, setCount] = useState(0);

    useEffect(() => {
        document.title = `You clicked ${count} times`;
    });

    return <button onClick={() => setCount(count + 1)}>+1</button>;
}
```

- Dependency array should include all variables used in the effect
  - Otherwise it might use stale values at re-render
    - React sometimes caches values for optimization

Quercus Exercises

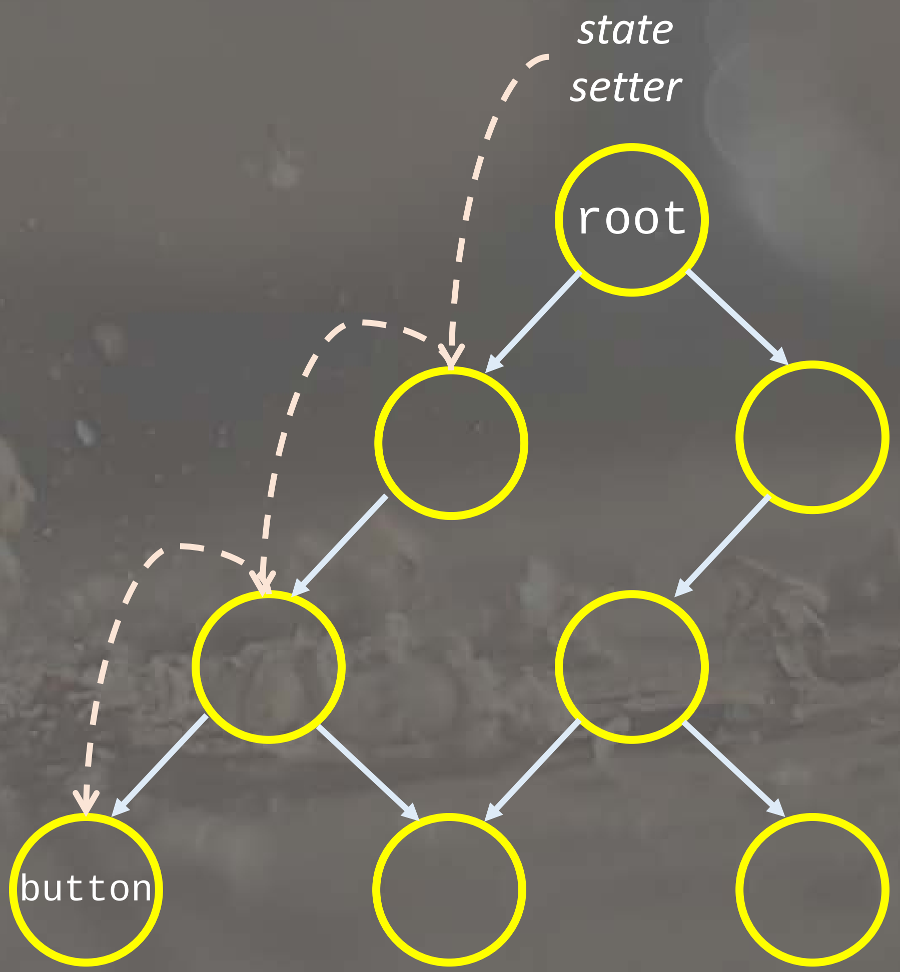
- Question 2
  - Build a simple calculator
  - When a button is clicked, update the display, until = is clicked.
  - Tip: use the eval() built-in function to evaluate an arbitrary JavaScript expression
- Question 3
  - Generate a table of baseball players
    - Using Fetch API
    - https://www.balldontlie.io/api/v1/players
  - Hint: do this on load and not on re-render!
  - Add autocomplete search feature and pagination



Global State

Prop Drilling

- Passing state(s) down to descendants components can be cumbersome
- Example:
    - The subcomponent that fires the request is a deeply nested button
    - You need to pass both the state and its setter function *all the way* down to the button
- Solution?



Global State

- A global state can be a great alternative
  - Accessible everywhere
  - No need to pass states all the way down
- Like global variables, don’t use them for everything!
  - Makes your code dirty and harder to understand
  - Makes component harder to reuse
- Context
  - React’s solution to support global state
  - Create a state variable and its setter, and put them in a context
  - Everything inside the context is accessible within its provider

Context

- Convention
  - Create a contexts folder under src and put all context files inside
- `createContext`
  - Creates a context that can be later used

```
import { createContext } from "react";

export const APIContext = createContext({
    players: [],
    setPlayers: () => {},
});
```

- Advice
  - Put default initial values for every variable that you will include in the context

Provider

- Creates an environment where the context is available
  1. With `useState`, create the state(s) and their setters
  2. Put a provider around the parent component and initialize it

```
function App() {
  const [players, setPlayers] = useState([]);

  return <APIContext.Provider value={{players, setPlayers}}>
      <Players />
    </APIContext.Provider>;
}
```

  3. Any descendant components can access the context with `useContext`

```
const { players } = useContext(APIContext);
```

Benefits

- Context enables you to handle API data easily
- Many components need to access them
  - E.g., username, profile data, etc.
- Various components can call APIs to fetch data
- Advice
  - For each Django app, create a context in React
  - Then, write a function that sets up relevant values and their setters
    - Name of this function should start with “use”
- Further reading
  - https://dmitripavlutin.com/react-context-and-usecontext/

Context Example

- “use” function

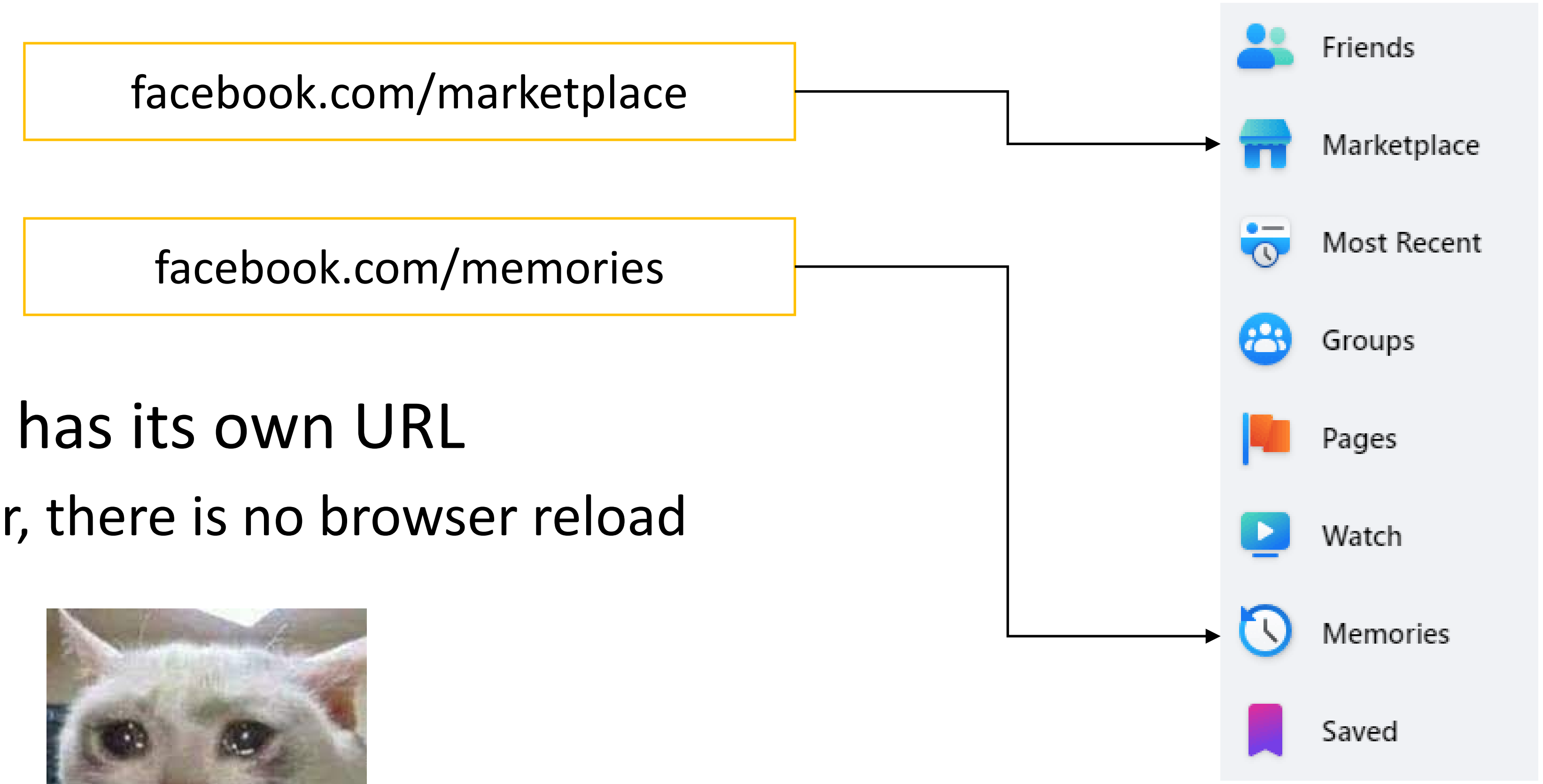
```
export function useAPIContext() {
    const [deployment, setDeployment] = useState([]);
    const [servers, setServers] = useState([]);
    const [applications, setApplications] = useState([]);
    const [applicationStatus, setApplicationStatus] = useState([]);
    const [availableLogDates, setAvailableLogDates] = useState([]);

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

Inside the Provider

```
<APIContext.Provider
    value={useAPIContext()}>
    <ControlPanel />
</APIContext.Provider>
```

Multi-Page React App

Pages



- Each page has its own URL
  - However, there is no browser reload

Naïve Approach

- Using what we know so far…

Even with this solution, we still do not have individual URLs for each component, making it difficult to return to a specific page quickly.

Wants

- Access to specific component via an URL
- Changing URL does not result in browser reload

```
function Facebook() {
    const [page, setPage] = useState("");
    const Navbar = () => <nav>
        <a onClick={() => setPage("watch")}>Watch</a>
        <a onClick={() => setPage("groups")}>Groups</a>
        <a onClick={() => setPage("marketplace")}>Marketplace</a>
    </nav>;

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

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

Router

- Installation
  - Run this in your project directory

    `npm install react-router-dom`

- Convention
  - Create a pages folder inside src
  - Put each page's component in a separate file or directory (preferred)
- Further reading
  - https://reactrouter.com/en/6.9.0/start/overview
  - https://www.w3schools.com/react/react_router.asp

Example organization

- `src`
  - `pages`
    - `Groups`
      - `index.jsx`
    - `Marketplace`
      - `index.jsx`
    - `Watch`
      - `index.jsx`

Routes and Links

- Set up the routes in App.js
  - Same idea as setting up urls.py in Django

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

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

Root path

Link

- Similar to <a>, but without a browser reload

```
import { Link, useParams } from "react-router-dom";
```

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

- URL arguments
  - Specified as part of the route definition, using : before parameter name

  ```
  <Route path="groups/:groupID" element={<Groups />} />
  ```

  - Can be accessed via a hook

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

  - Same way to link to the page

  ```
  <Link to="/groups/42">Groups</Link>
  ```

Query Parameters

- Can be accessed via another hook

```
import { useSearchParams } from "react-router-dom";

// By convention, underscore in front of a name means "don't care".
const [searchParams, _setSearchParams] = useSearchParams();
```

- To extract a specific key:

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

- Use query parameters in an URL:

```
<Link to="/groups/42?name=kia">Groups</Link>
```

Navigation

- Sometimes, you need a URL change via code
- Example
  - When Response is 401, redirect to the login page
- Vanilla JavaScript
  - This causes the browser to reload!

```
window.location.replace("/marketplace");
```

- React Router

```
import { useNavigate } from "react-router-dom";

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

Outlet

- We need a navbar to navigate through pages
  - Bad idea to copy it to all the pages
- What happens when we specify an element for root URL?
  - Only that element will be rendered and all child elements will be ignored.
- In nested routes, React renders the first component that *partially matches* the URL and has an element
- However, it continues matching the remaining URL and returns the matching child components as `<Outlet />`
- Convention
  - Root element is used to specify layout; child components are rendered within.

Using Outlet

- In App.js

```
<Route path="/" element={<Layout />}>
        <Route index element={<Home />} />
        …
```

- In Layout.jsx

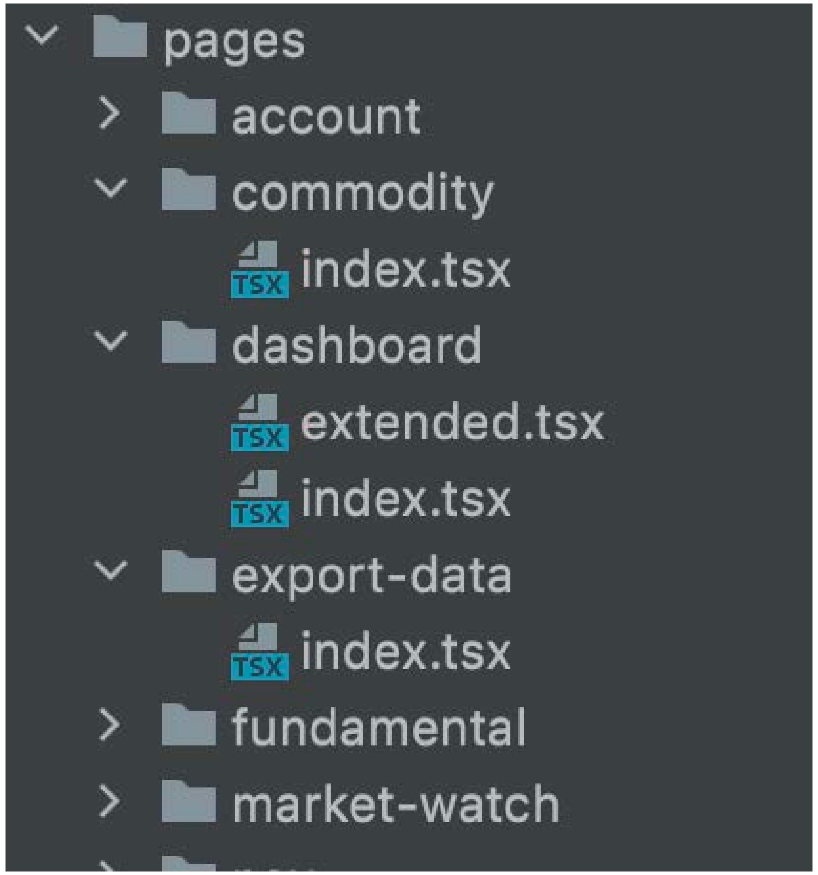
```
const Layout = () => {
    return <>
        <header>
            <Link to="/watch">Watch</Link>
            <Link to="/groups/88/?name=joe">Groups</Link>
            <Link to="/marketplace">Marketplace</Link>
        </header>
        <Outlet />
    </>;
}
```

Child components will be rendered where `<Outlet />` is.

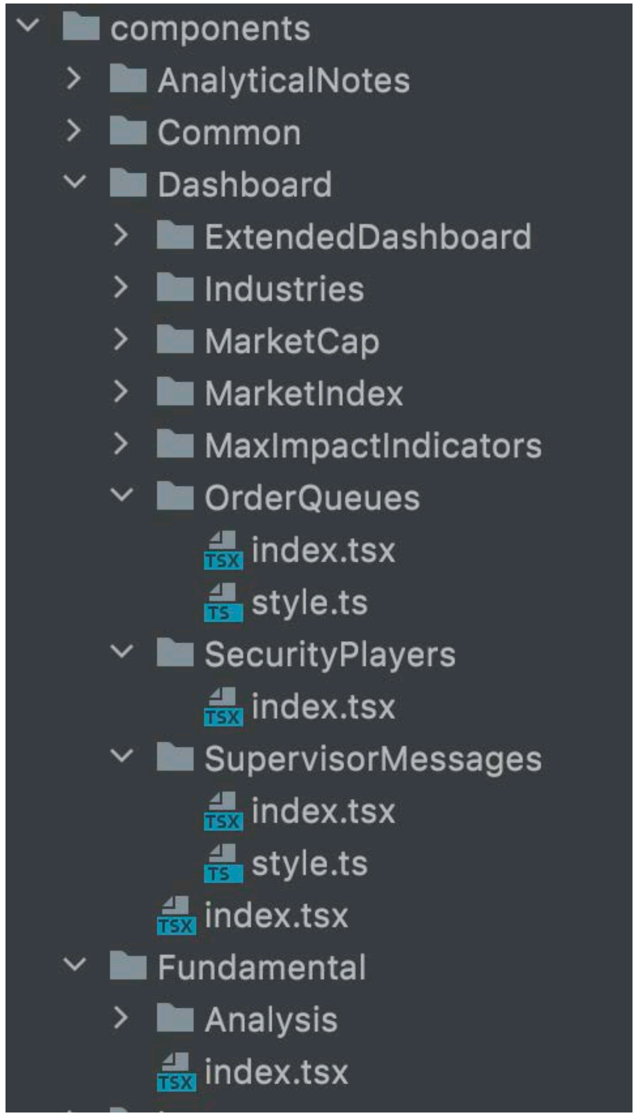
Term Project

- File structures for React project varies
- Good practice to separate pages from reusable components
  - E.g., inputs, tables, forms, buttons, etc.
- Do not let a component become too big (in LOC)
  - Refactor by extracting child components
- Expect most components to have multiple children
  - Thus, each component/page should have its own directory, not just file
  - You can put child components in a subfolder of the parent component
- Dedicate a page to login, signup, forms, and navbar items
- Use function components and hooks instead of class components

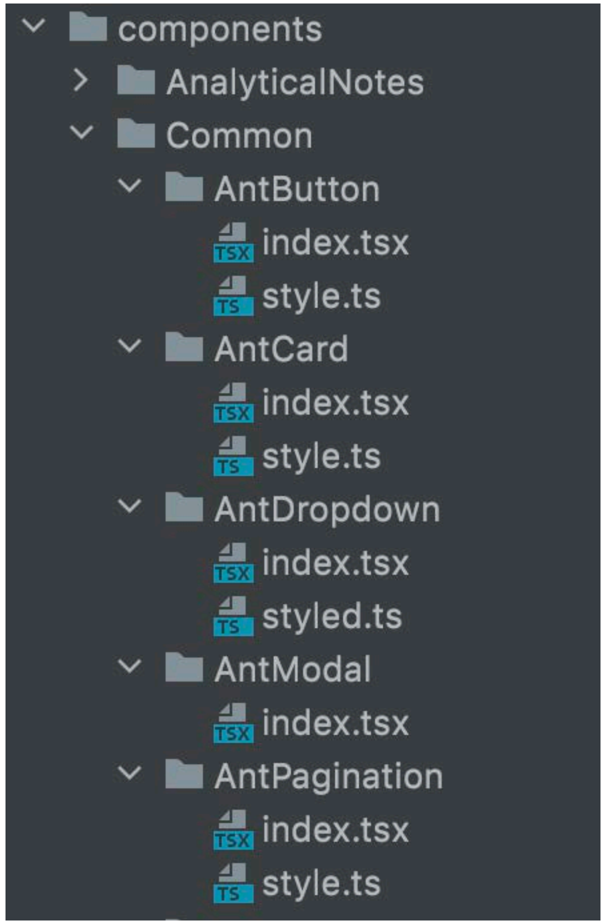
Example File Structure



```
pages
  account
  commodity
    index.tsx
  dashboard
    extended.tsx
    index.tsx
  export-data
    index.tsx
  fundamental
  market-watch
```



```
components
  AnalyticalNotes
  Common
  Dashboard
    ExtendedDashboard
    Industries
    MarketCap
    MarketIndex
    MaxImpactIndicators
    OrderQueues
      index.tsx
      style.ts
    SecurityPlayers
      index.tsx
    SupervisorMessages
      index.tsx
      style.ts
    index.tsx
  Fundamental
    Analysis
    index.tsx
```



```
components
  AnalyticalNotes
  Common
    AntButton
      index.tsx
      style.ts
    AntCard
      index.tsx
      style.ts
    AntDropdown
      index.tsx
      styled.ts
    AntModal
      index.tsx
    AntPagination
      index.tsx
      style.ts
```

Final notes

- Important announcement
    - Class cancellation notice
    - Classes on March 22, March 24, and March 27 are *cancelled*
    - Please spend the extra time on timely completion of A3 and P3
- Exercise 9 + 10
    - Due next Sunday, Apr 2nd
- Midterm Results
    - Should be released on Wednesday
        - Some TAs are not done yet