Topics

- Routing/Navigation
- Design Patterns
- Custom Hooks

Navigation

The React Router library

Routing - Introduction

- **Allows multiple views in an app.**
  - **But there's only one page (.html) → Single Page App (SPA)**

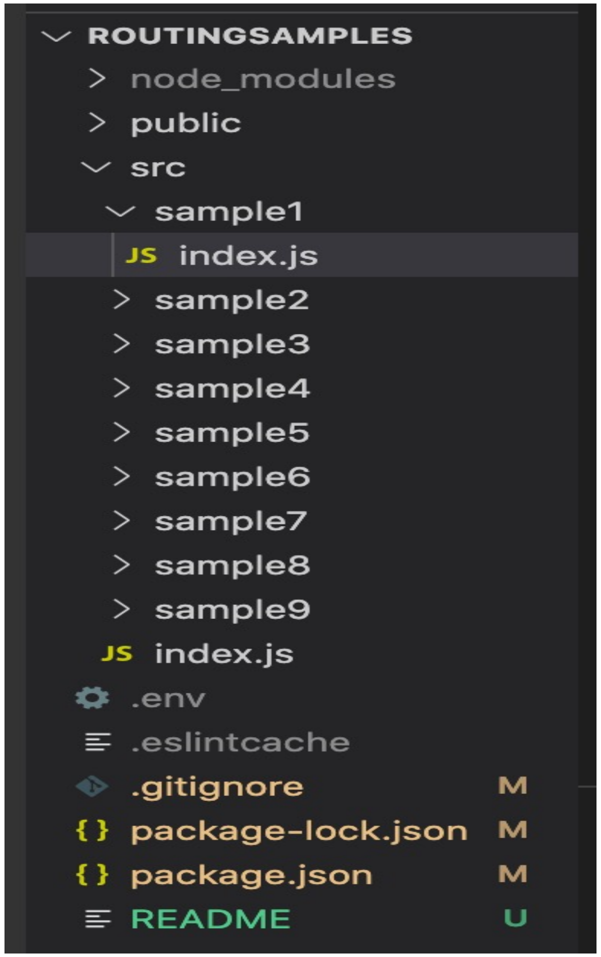
- **Keeps the browser's URL in sync with the UI.**

- **Adheres to traditional web principles:**
  1. **Addressability.**
  2. **Information sharing.**
  3. **Deep linking.**
  - **1st generation client rendering apps violated these principles.**

- **Not supported by the React framework.**
  - **A separate library is required: React Router.**

Demos

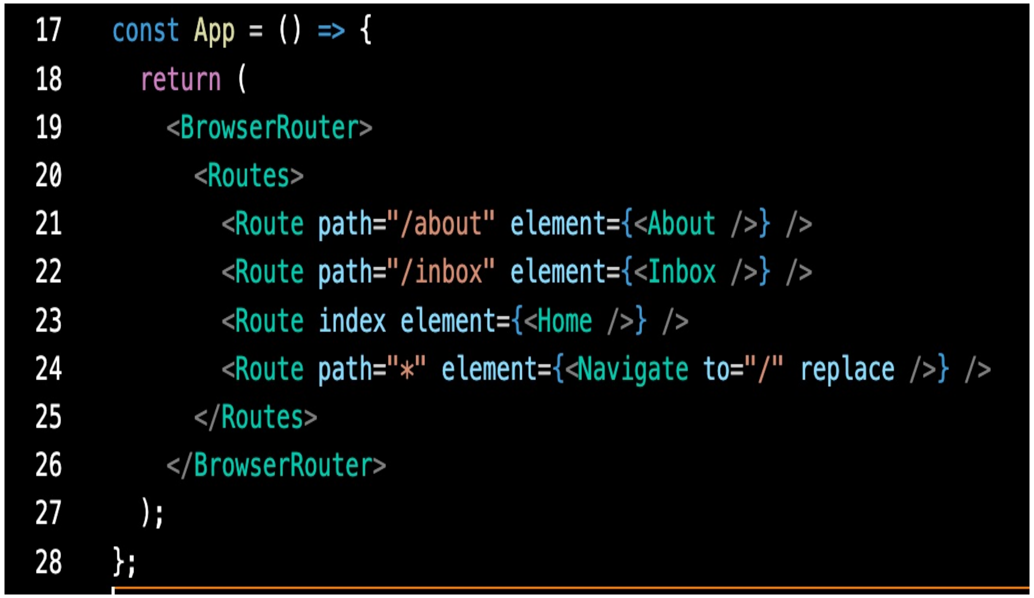
- **See lecture archive.**
- **Each sample demos a routing feature.**



```
ROUTINGSAMPLES
  node_modules
  public
  src
    sample1
      index.js
    sample2
    sample3
    sample4
    sample5
    sample6
    sample7
    sample8
    sample9
    index.js
  .env
  .eslintcache
  .gitignore                M
  package-lock.json         M
  package.json              M
  README                    U
```

Basic routing configuration

	URL	Components
1	/	Home
2	/about	About
3	/inbox	Inbox

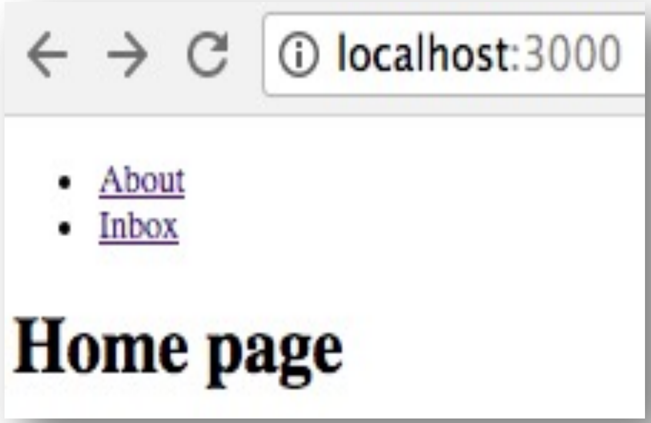


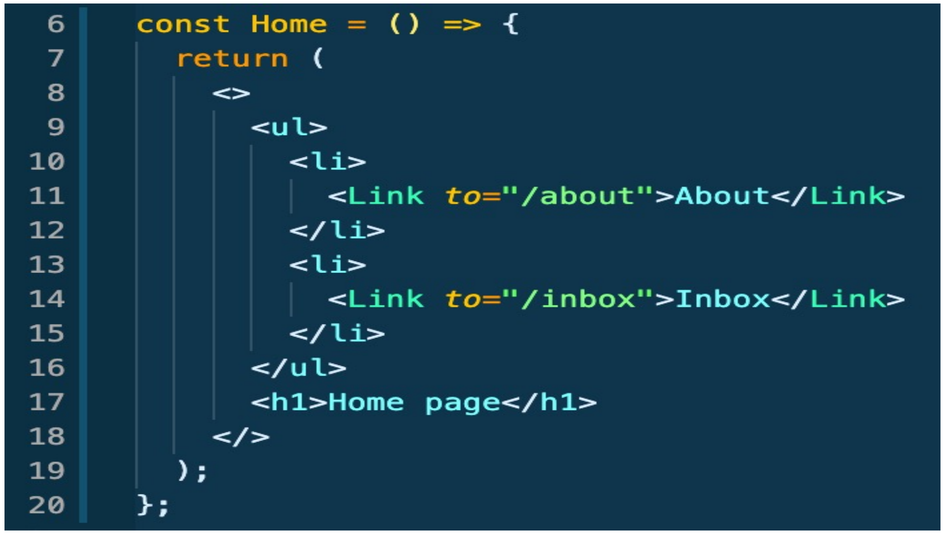
```
const App = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/about" element={<About />} />
        <Route path="/inbox" element={<Inbox />} />
        <Route index element={<Home />} />
        <Route path="*" element={<Navigate to="/" replace />} />
      </Routes>
    </BrowserRouter>
  );
};
```

- **Declarative style.**
- <BrowserRouter> **- matches browser's URL to a <Route>** path**.**
- **Matching Route's>** element **is mounted on the DOM.**
  - element **can take any arbitrary JSX.**
  - **Use** index **for root path case (/).**
  - **Use * path for 404 case.**
  - <Navigate> **changes browser's URL address.**
- **App component termed the** Router **component.**
- **Ref. src/sample1**

Hyperlinks

- **Use the <Link> component for internal links.**
  - **Use anchor tag for external links - <a href . . . . . >**
- **Ref. src/sample2/**





```
const Home = () => {
  return (
    <>
      <ul>
        <li>
          <Link to="/about">About</Link>
        </li>
        <li>
          <Link to="/inbox">Inbox</Link>
        </li>
      </ul>
      <h1>Home page</h1>
    </>
  );
};
```

- **<Link> changes browser's URL address (event)**

→ React Router handles event by consulting its routing configuration

→ Selected Route's elements **mounted on DOM → Browser repaints the screen.**

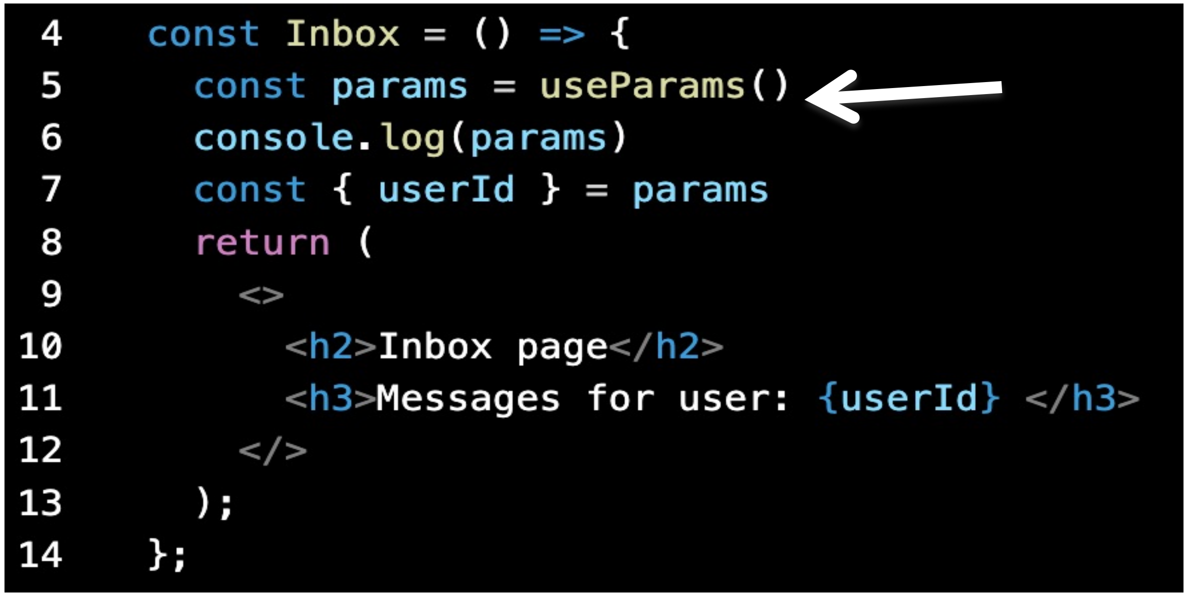
Dynamic segments.

- **Parameterized URLs, e.g. /users/*22*, /users/*12*/purchases**
  - **How to declare a parameterized path in the routing configuration?**
  - **How does a component access the parameter value?**
- **Ex: Suppose the Inbox component shows messages for a specific user, where the user's id is part of the browser URL**

  e.g /inbox/123, where 123 is the user's id.

- **Solution:** <Route path='/inbox/:userId' element={ <Inbox/> } />
  - **The colon (:) prefixes a parameter in the path.**
  - **Parameter name is arbitrary.**
  - **Ref src/sample3**

Dynamic segments.



```
const Inbox = () => {
  const params = useParams()
  console.log(params)
  const { userId } = params
  return (
    <>
      <h2>Inbox page</h2>
      <h3>Messages for user: {userId} </h3>
    </>
  );
};
```

- useParams **hook (React Router library).**
  - **Returns an object containing the parameter value.**
  - **Other useful hooks also provided (see later)**
- **More than one parameter allowed.**

  e.g. /users/:userId/categories/:categoryName

Nested Routes

- Objective**: A component's child is dynamically determined from the browser's URL (Addressability).**

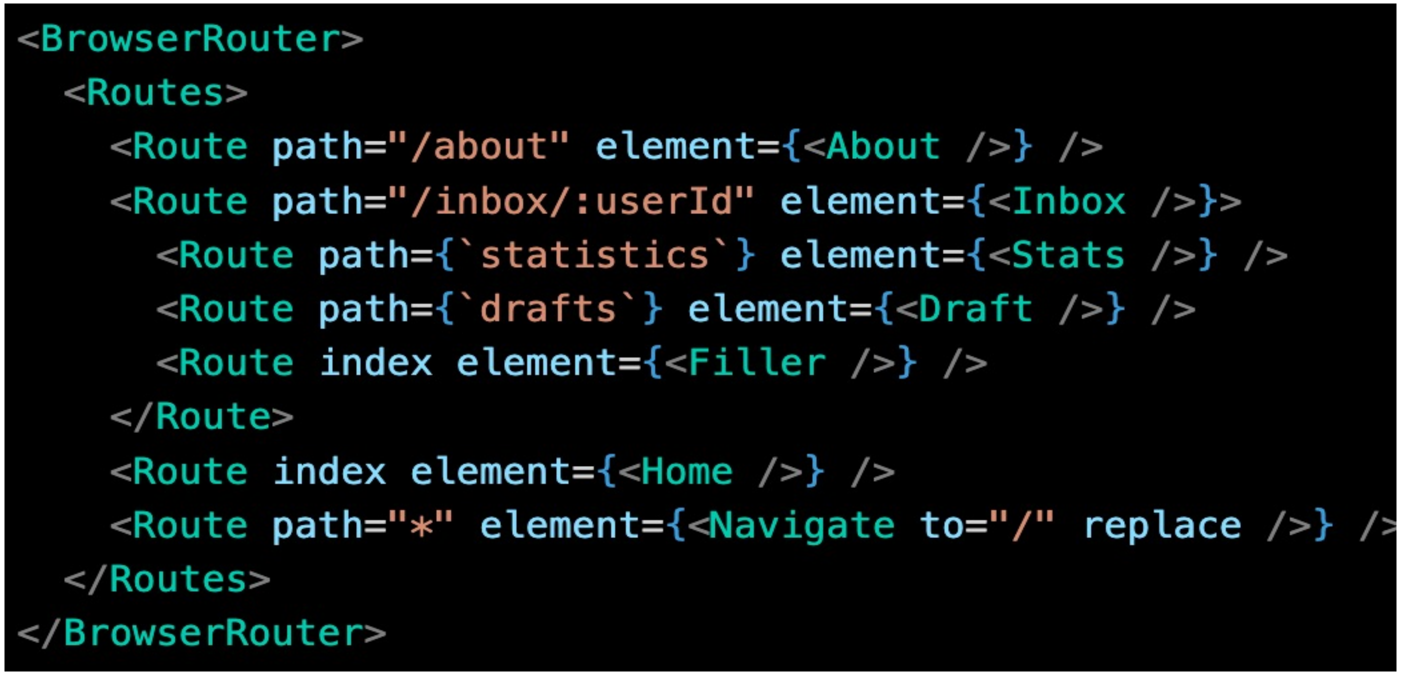
- **EX.: (See src/sample4) Given the route:**

  <Route path='/inbox/:userId' element={ <Inbox /> } />,

  use the following rules to determine a nested component hierarchy:

/inbox/XYZ/statistics	**/inbox/XYZ/draft**
<Inbox>	**<Inbox>**
<Stats/>	**<Drafts/>**
</Inbox>	**</Inbox>**

Nested Routes

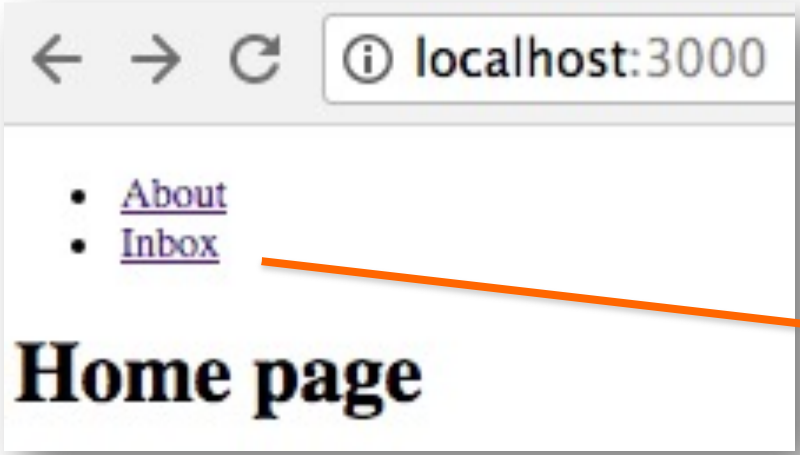


```
<BrowserRouter>
  <Routes>
    <Route path="/about" element={<About />} />
    <Route path="/inbox/:userId" element={<Inbox />}>
      <Route path={`statistics`} element={<Stats />} />
      <Route path={`drafts`} element={<Draft />} />
      <Route index element={<Filler />} />
    </Route>
    <Route index element={<Home />} />
    <Route path="*" element={<Navigate to="/" replace />} />
  </Routes>
</BrowserRouter>
```

- **Use RELATIVE path strings in the nested <Route> entries.**
- **The index <Route> is optional.**
  - **For the default case.**
  - **Avoids a ‘blank’ section on screen.**
- **Use <Outlet/> as a placeholder in the container component**

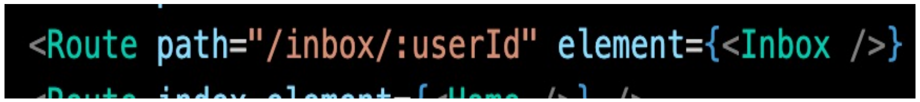
Extended <Link>

- **Objective: Supply data to the component mounted by a <Link>.**
- **EX.: See /src/sample5/.**

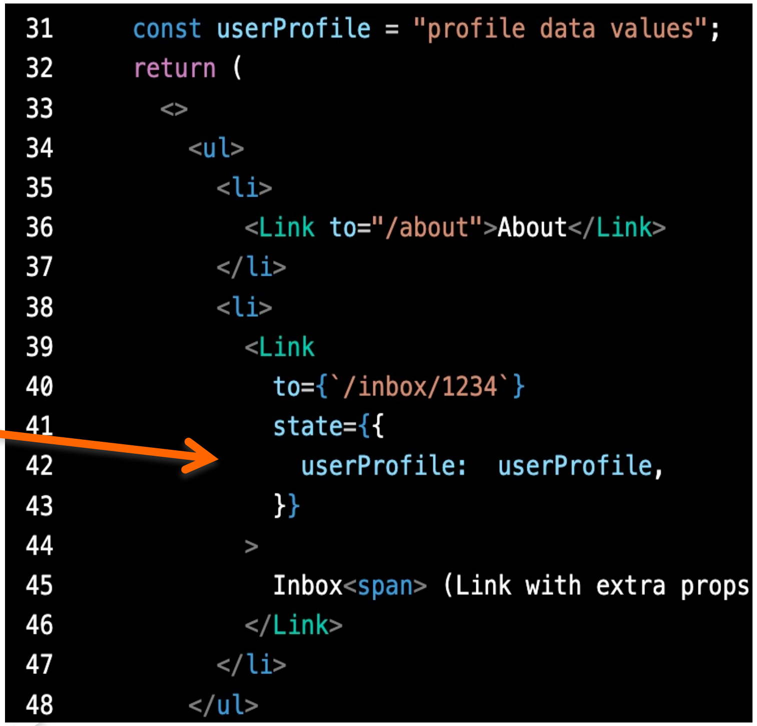


- About
- Inbox

Home page



```
<Route path="/inbox/:userId" element={<Inbox />}
```

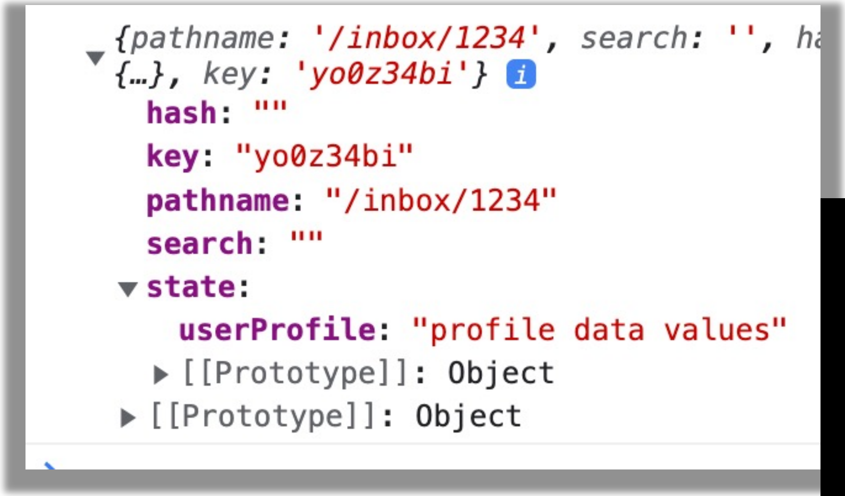


```
    const userProfile = "profile data values";
    return (
      <>
        <ul>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            <Link
              to={`/inbox/1234`}
              state={{
                userProfile:  userProfile,
              }}
            >
              Inbox<span> (Link with extra props
            </Link>
          </li>
        </ul>
```

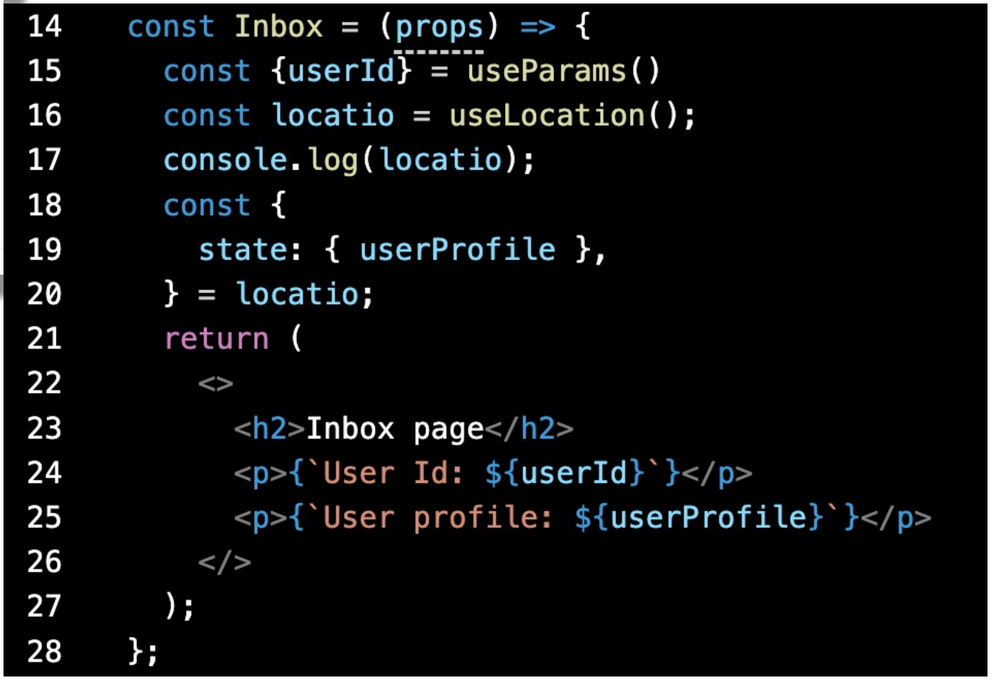
- **How does Inbox access the** userProfile **data included in the hyperlink?**
  - **A.: The** useLocation **hook**

Extended <Link>

- **React Router creates a** location **object each time the URL changes.**



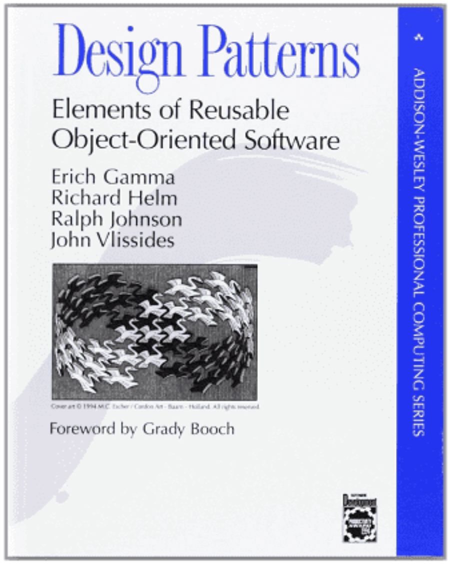
```
▼ {pathname: '/inbox/1234', search: '', ha
  {…}, key: 'yo0z34bi'} i
    hash: ""
    key: "yo0z34bi"
    pathname: "/inbox/1234"
    search: ""
  ▼ state:
      userProfile: "profile data values"
    ▶ [[Prototype]]: Object
  ▶ [[Prototype]]: Object
```



```
const Inbox = (props) => {
  const {userId} = useParams()
  const locatio = useLocation();
  console.log(locatio);
  const {
    state: { userProfile },
  } = locatio;
  return (
    <>
      <h2>Inbox page</h2>
      <p>{`User Id: ${userId}`}</p>
      <p>{`User profile: ${userProfile}`}</p>
    </>
  );
};
```

Routing

- **More later**


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Design Patterns

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software **design**

Reusability & Separation of Concerns.

- The DRY principle – Don’t Repeat Yourself.
- Techniques to improve DRY(ness) (increase reusability):
  1. Inheritance (is-a relationships, e.g. Car is an automabile)
  2. Composition (has-a relationships, e.g. Car has an Engine)

- React favors composition.
- Core React composition Patterns:
  1. Container.
  2. Render Props.
  3. Higher Order Components.

Composition - Children

- HTML is composable

```
 <div>
    <h2>Some Heading</h2>
    <ul>
         <li> . . . . . </li>
         <li> . . . . . </li>
         <li> . . . . . </li>
    </ul>
 </div>
```

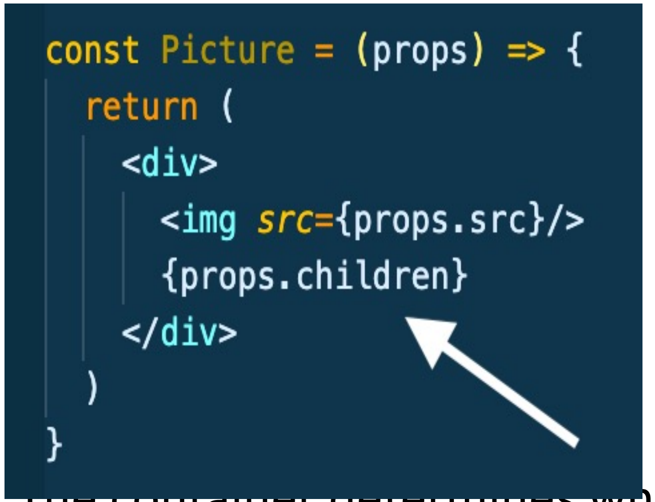
```
<div>
    <p>……….</p>
   <img …………. />
  <a href ………../>
<</div>
```

<div> has three children.

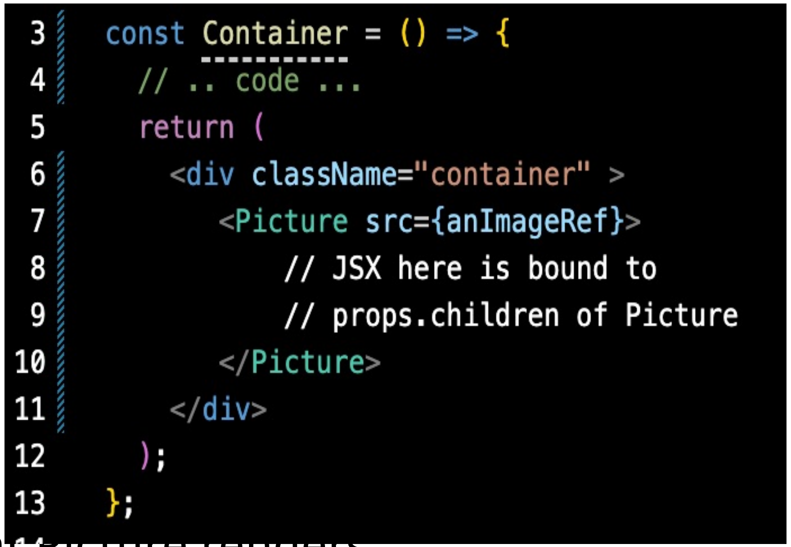
- <div> has two children; <ul> has three children

The Container pattern.

All React components have a special <u>children</u> prop. It allows a consumer (container) to pass other components to it by nesting them inside the jsx.

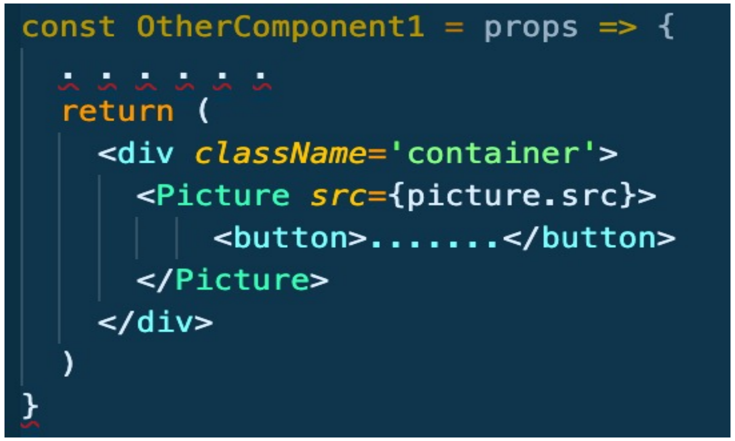


```
const Picture = (props) => {
  return (
    <div>
      <img src={props.src}/>
      {props.children}
    </div>
  )
}
```

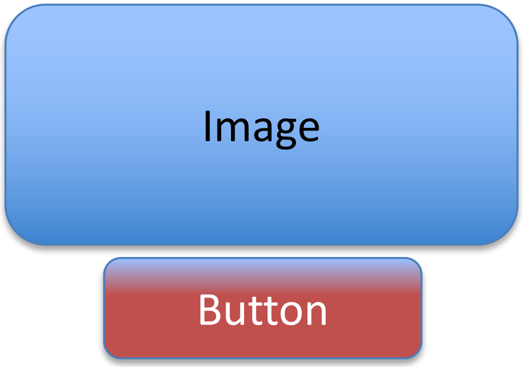


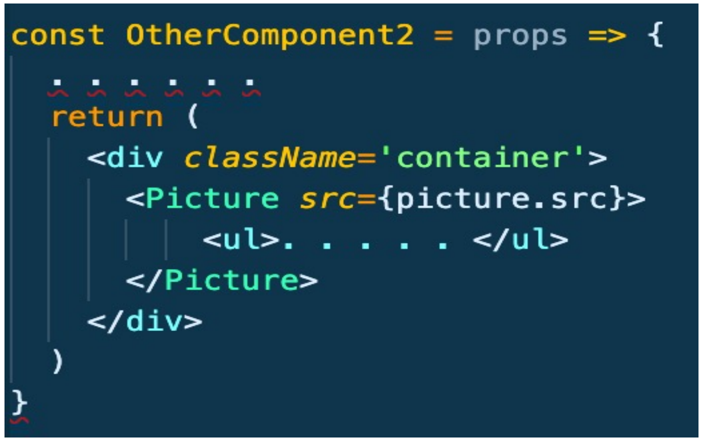
```
const Container = () => {
  // .. code ...
  return (
    <div className="container" >
      <Picture src={anImageRef}>
          // JSX here is bound to
          // props.children of Picture
      </Picture>
    </div>
  );
};
```

- The container determines what Picture renders;
- This <u>de-couples</u> the Picture component from its content and makes it <u>reusable</u>.

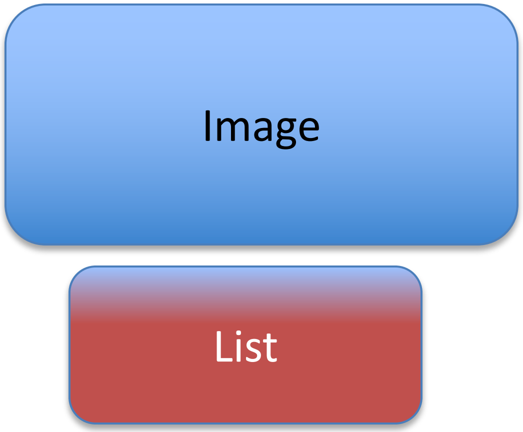


```
const OtherComponent1 = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
          <button>.......</button>
      </Picture>
    </div>
  )
}
```

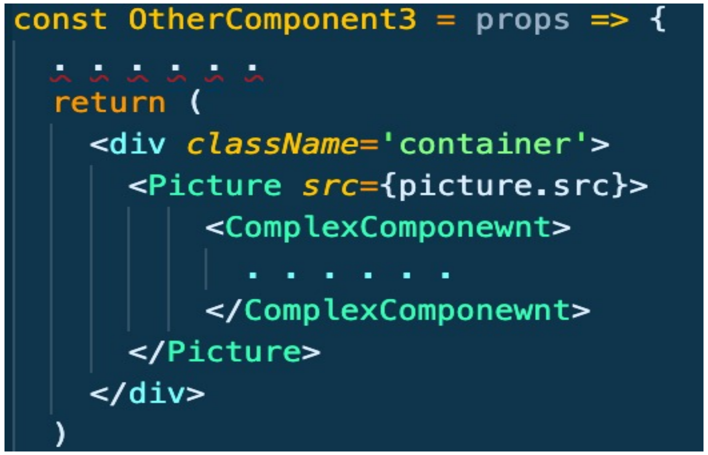




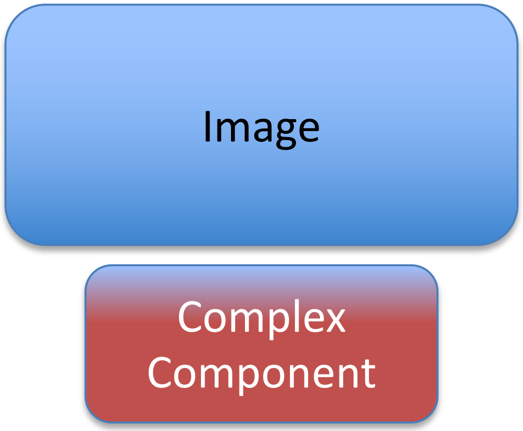
```
const OtherComponent2 = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
          <ul>. . . . . </ul>
      </Picture>
    </div>
  )
}
```



Picture is **composed** with other elements / components

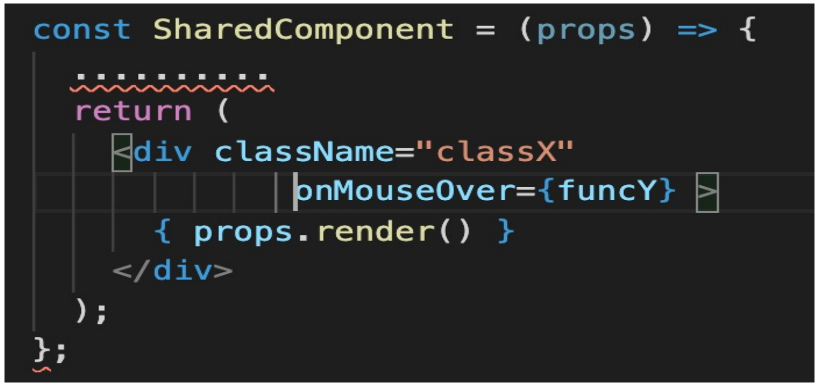


```
const OtherComponent3 = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
          <ComplexComponewnt>
            . . . . . .
          </ComplexComponewnt>
      </Picture>
    </div>
  )
```



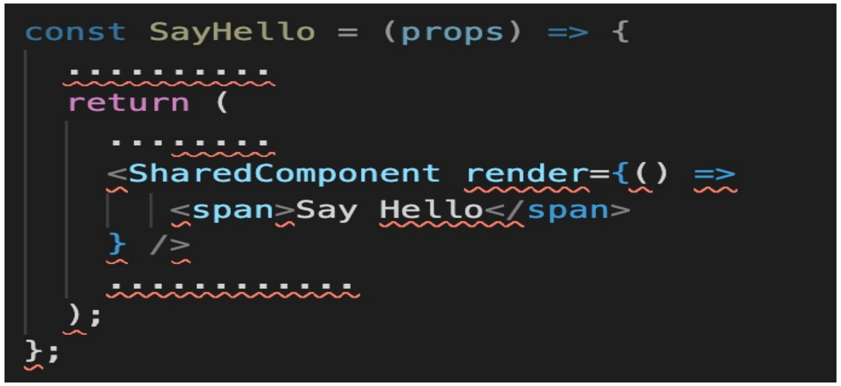
The Render Prop pattern

- Use the pattern to share logic between components.
- Dfn: A render prop is a <u>function prop </u>that a component uses to generate part of its rendered output.

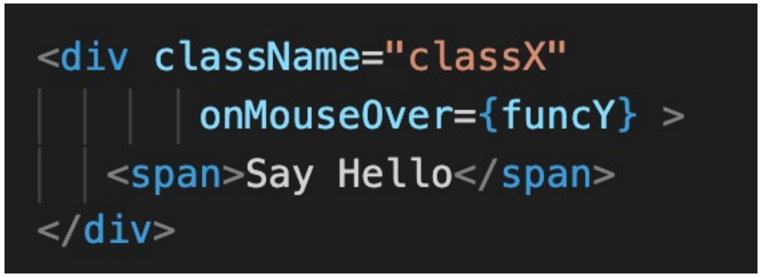


```
const SharedComponent = (props) => {
  ..........
  return (
    <div className="classX"
          onMouseOver={funcY} >
      { props.render() }
    </div>
  );
};
```

- SharedCoomponent **receives its render logic from the consumer, i.e.** SayHello.
- Prop name is arbitrary.

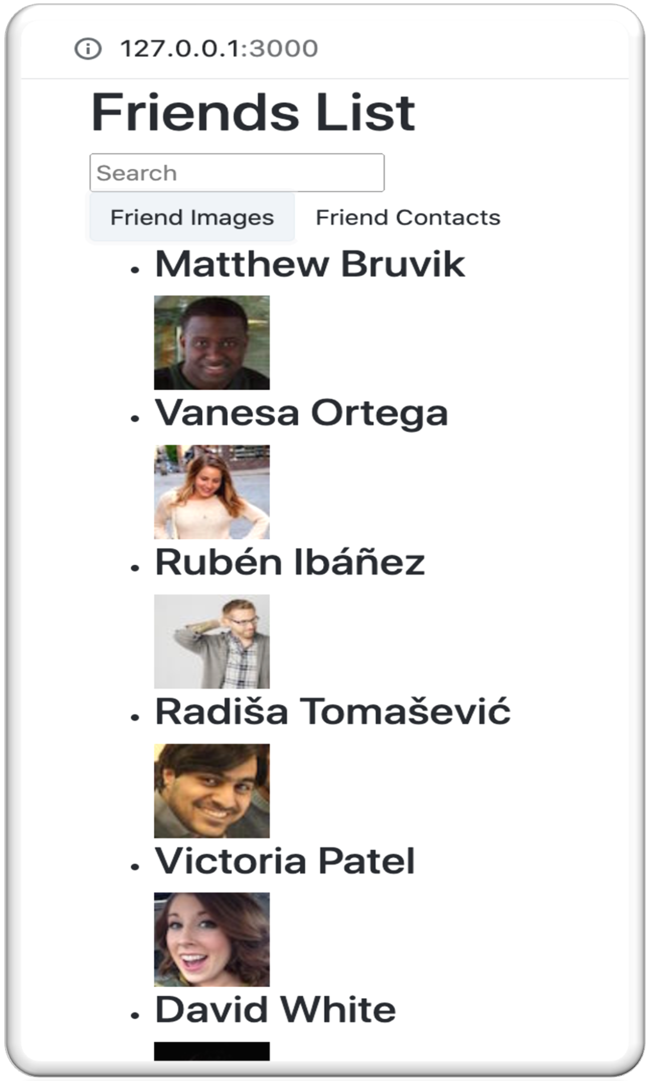


```
const SayHello = (props) => {
  ..........
  return (
    ........
    <SharedComponent render={() =>
      <span>Say Hello</span>
    } />
    .............
  );
};
```



```
<div className="classX"
        onMouseOver={funcY} >
    <span>Say Hello</span>
</div>
```

The Render Prop - Sample App.



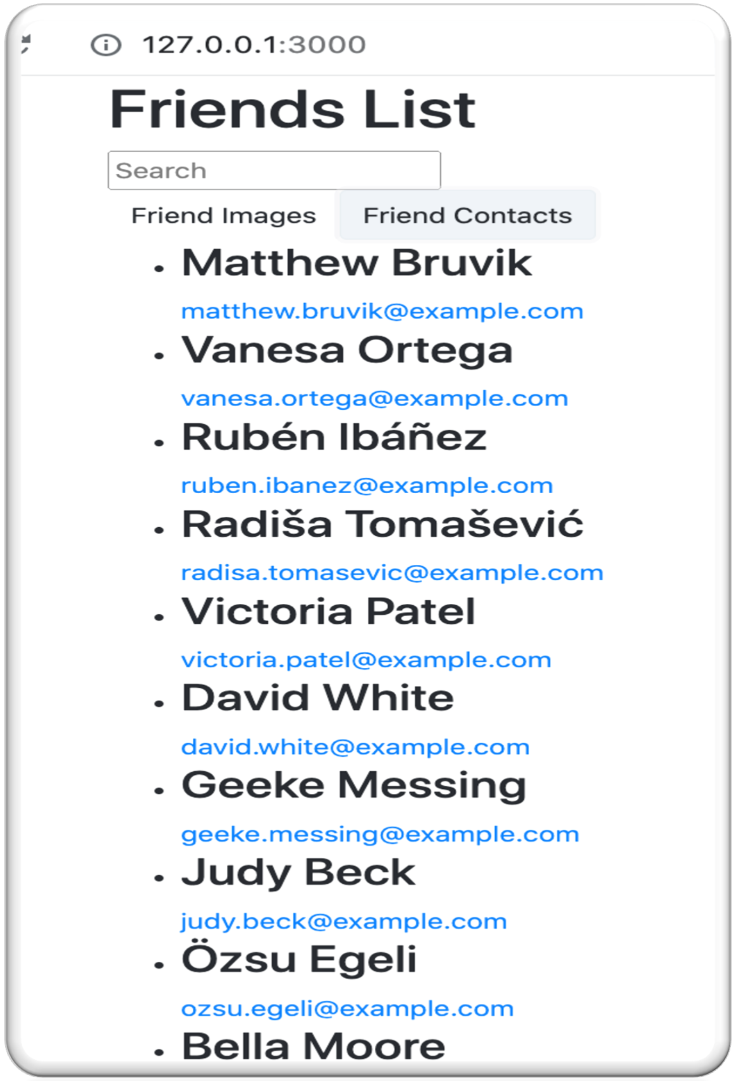
127.0.0.1:3000

Friends List

Search

Friend Images Friend Contacts

- Matthew Bruvik
- Vanesa Ortega
- Rubén Ibáñez
- Radiša Tomašević
- Victoria Patel
- David White



127.0.0.1:3000

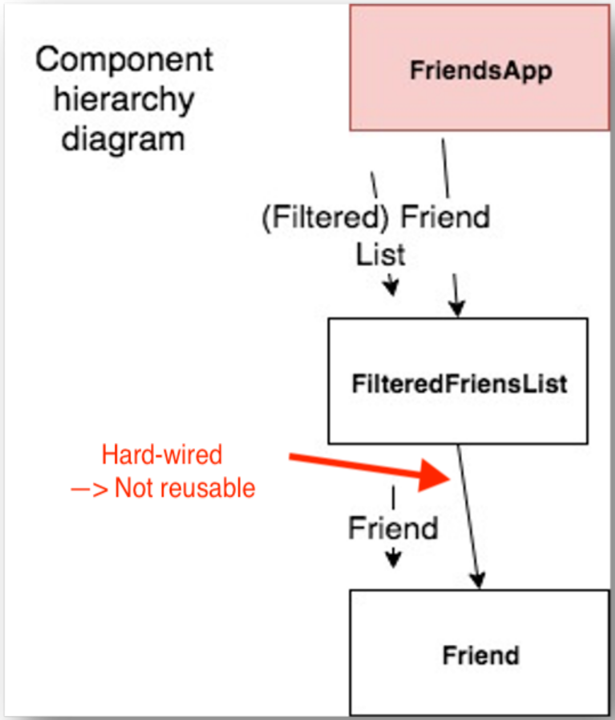
Friends List

Search

Friend Images Friend Contacts

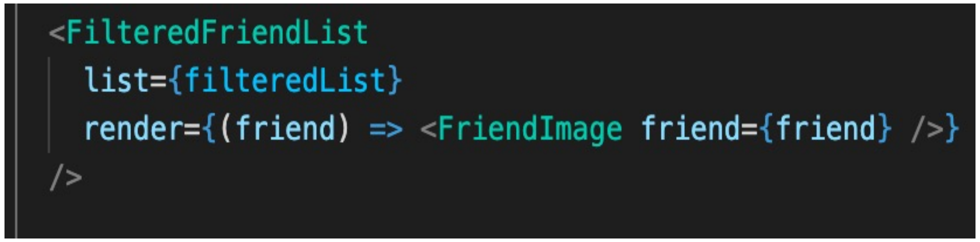
- Matthew Bruvik
  matthew.bruvik@example.com
- Vanesa Ortega
  vanesa.ortega@example.com
- Rubén Ibáñez
  ruben.ibanez@example.com
- Radiša Tomašević
  radisa.tomasevic@example.com
- Victoria Patel
  victoria.patel@example.com
- David White
  david.white@example.com
- Geeke Messing
  geeke.messing@example.com
- Judy Beck
  judy.beck@example.com
- Özsu Egeli
  ozsu.egeli@example.com
- Bella Moore

The Render Props - Sample App.



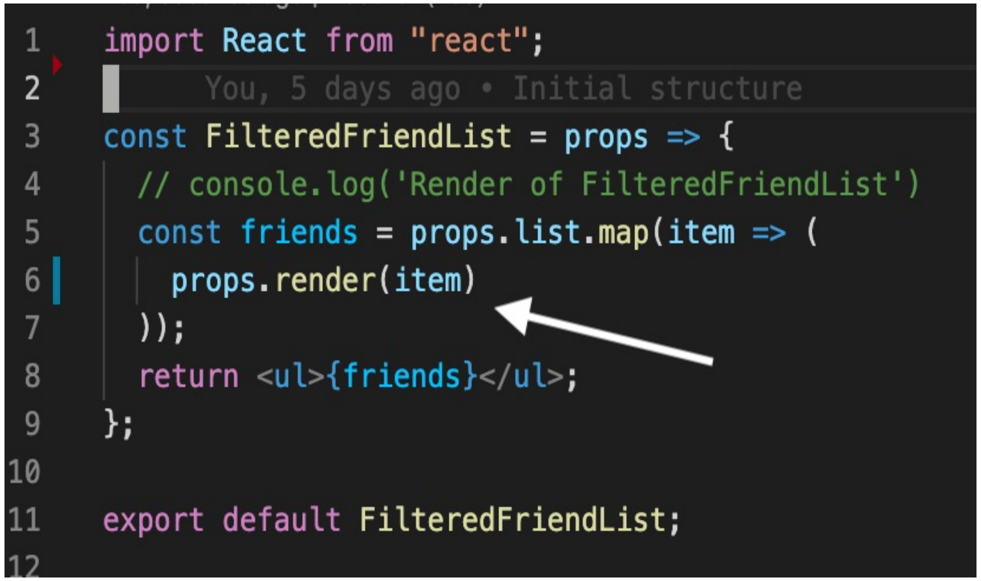
- Updates to design:

1. FriendsApp passes a render-prop to FilteredFriendList, indicating how Friends should be rendered.
2. Remove static import of Friend component type from FilteredFriendList.



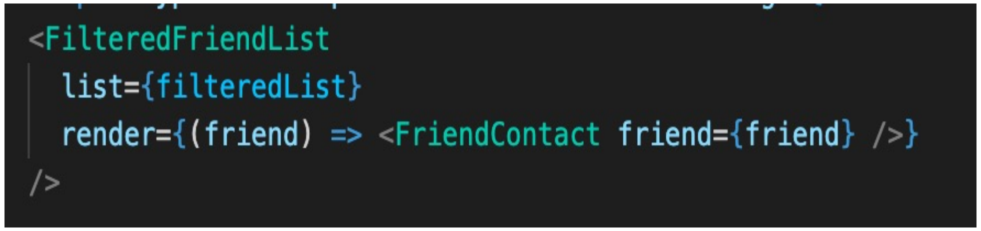
```
<FilteredFriendList
  list={filteredList}
  render={(friend) => <FriendImage friend={friend} />}
/>
```

- **Without this pattern we would need a** FilteredFriendList **component for each use case, thus violating the DRY principle.**



```
import React from "react";
    You, 5 days ago • Initial structure
const FilteredFriendList = props => {
  // console.log('Render of FilteredFriendList')
  const friends = props.list.map(item => (
    props.render(item)
  ));
  return <ul>{friends}</ul>;
};

export default FilteredFriendList;
```



```
<FilteredFriendList
  list={filteredList}
  render={(friend) => <FriendContact friend={friend} />}
/>
```

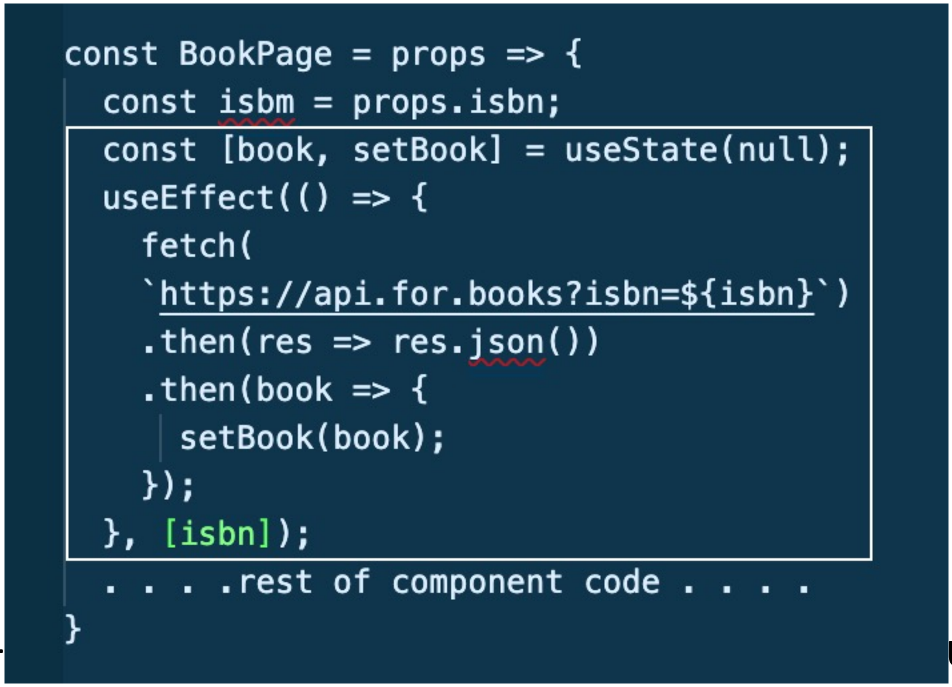
- **The prop name is arbitrary;** render **is a convention.**

Custom Hooks

Custom Hooks.

- Custom Hooks let you extract component logic into reusable functions.
- Improves code readability and modularity.

Example:

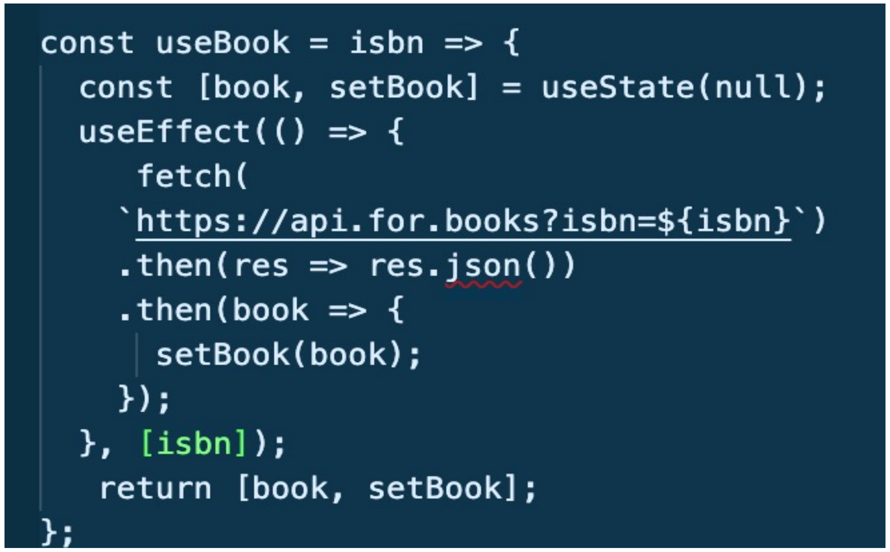


```
const BookPage = props => {
  const isbm = props.isbn;
  const [book, setBook] = useState(null);
  useEffect(() => {
    fetch(
    `https://api.for.books?isbn=${isbn}`)
    .then(res => res.json())
    .then(book => {
      setBook(book);
    });
  }, [isbn]);
  . . . .rest of component code . . . .
}
```

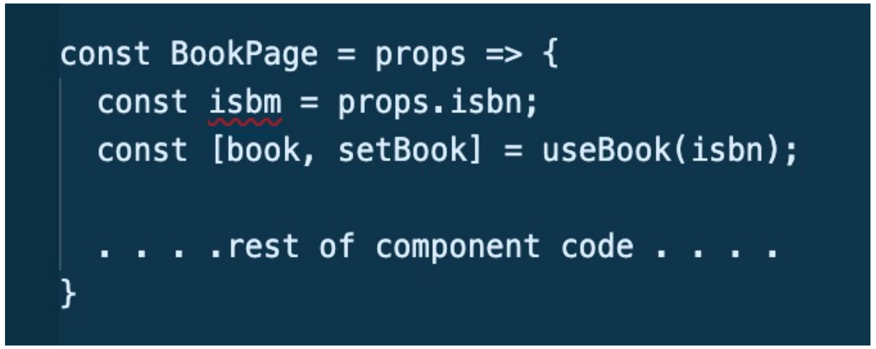
Objective – ustom hook.

Custom Hook Example.

Solution:



```
const useBook = isbn => {
  const [book, setBook] = useState(null);
  useEffect(() => {
     fetch(
    `https://api.for.books?isbn=${isbn}`)
    .then(res => res.json())
    .then(book => {
      setBook(book);
    });
  }, [isbn]);
   return [book, setBook];
};
```



```
const BookPage = props => {
  const isbm = props.isbn;
  const [book, setBook] = useBook(isbn);

  . . . .rest of component code . . . .
}
```

- Custom Hook is an ordinary function BUT should only be called from a React component function.
- Prefix hook function name with use to leverage linting support.
- Function can return any collection type (array, object), with any number of entries.