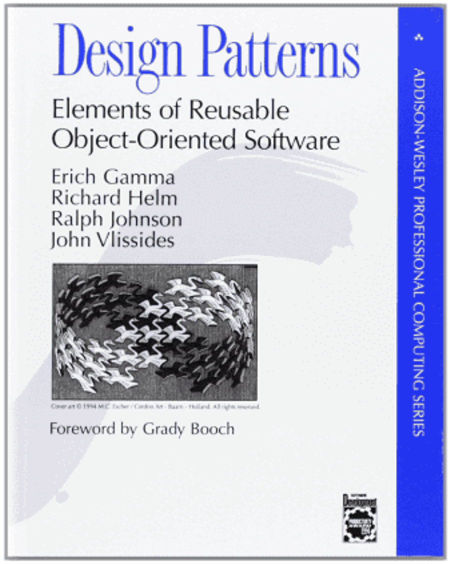

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

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. **Containers.**
  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>
```

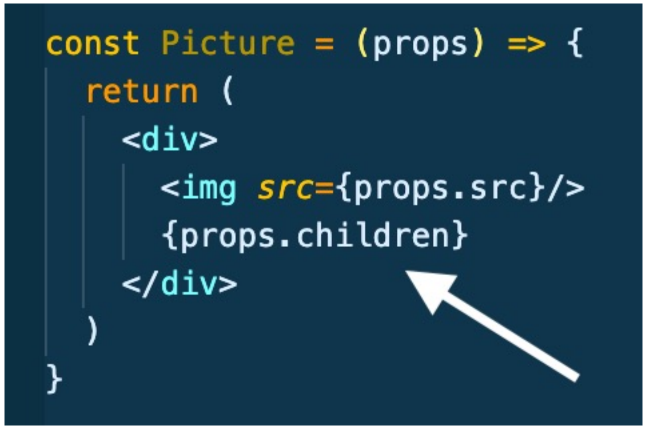
- **<div> has two children; <ul> has three children**

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

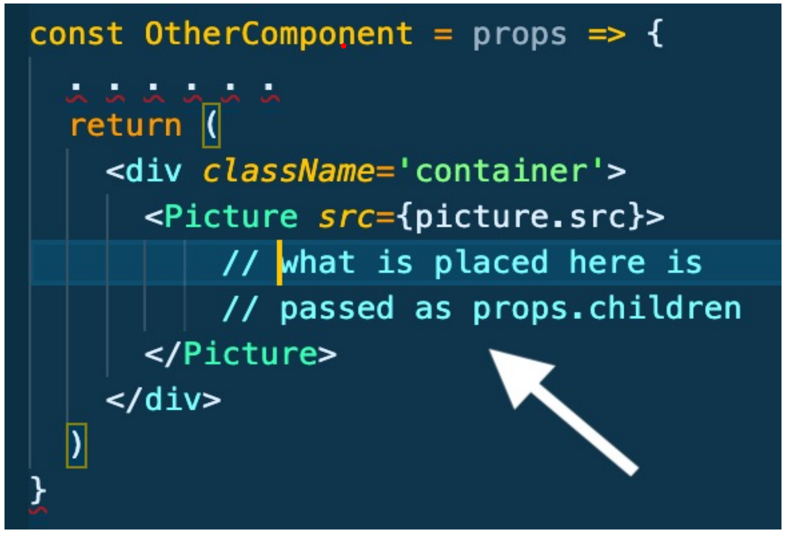
<div> has three children.

The Container pattern.

All React components have a special <u>children</u> prop so that consumers can pass components directly by nesting them inside the jsx.

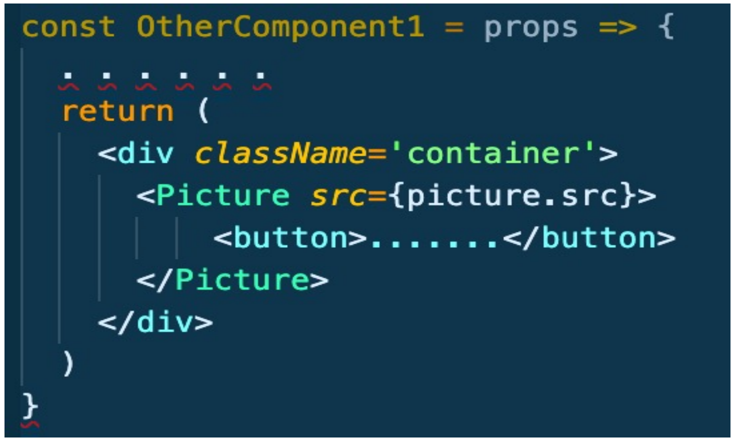


```
const Picture = (props) => {
  return (
    <div>
      <img src={props.src}/>
      {props.children}
    </div>
  )
}
```

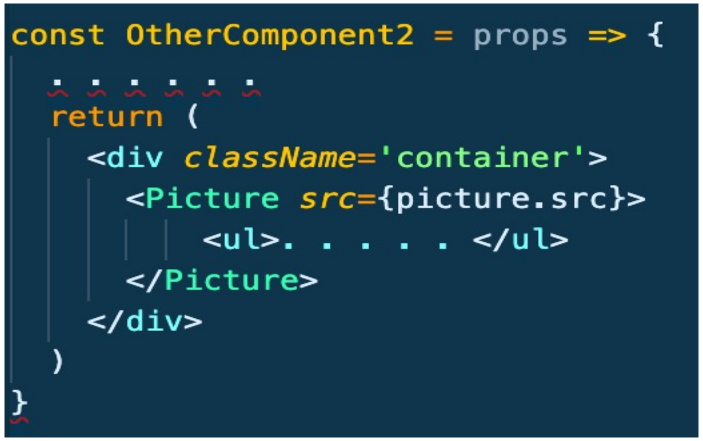


```
const OtherComponent = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
          // what is placed here is
          // passed as props.children
      </Picture>
    </div>
  )
}
```

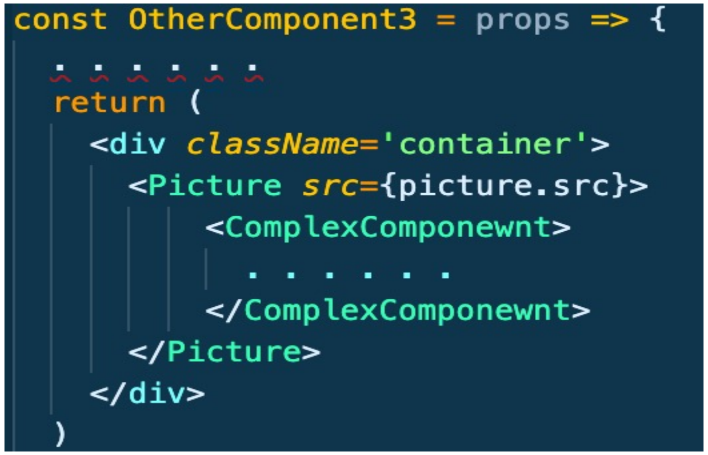
- **When the** Picture **component renders, its** props.children **will <u>display what the consumer places</u> between the opening and closing tags of Picture.**
- **This <u>de-couples</u> the** Picture **component from its content and makes it** reusable**.**



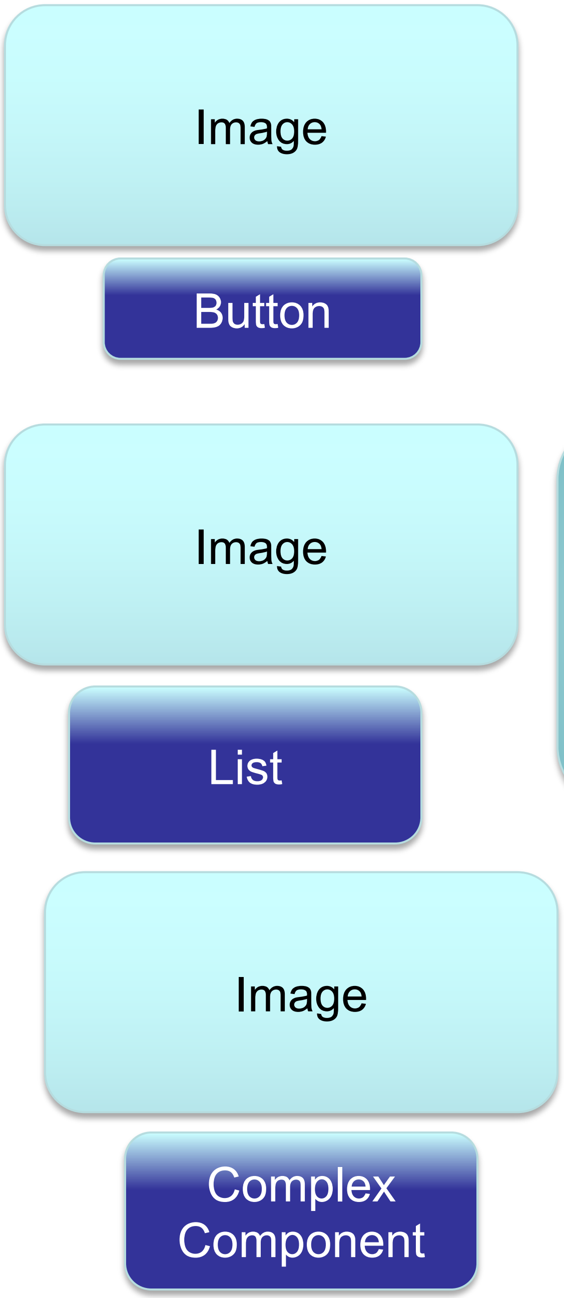
```
const OtherComponent1 = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
          <button>.......</button>
      </Picture>
    </div>
  )
}
```



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



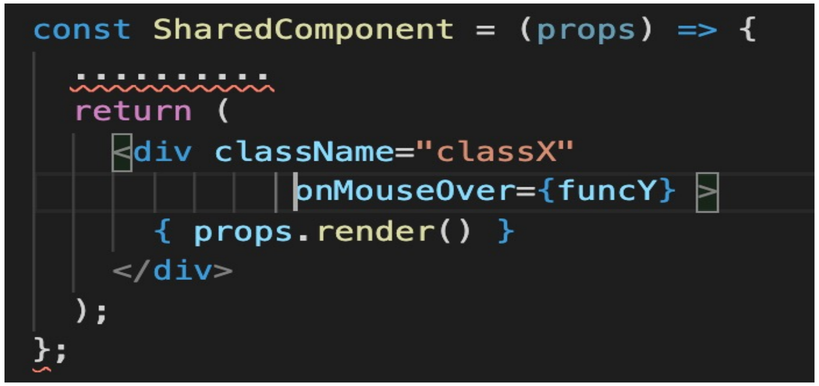
```
const OtherComponent3 = props => {
  . . . . . .
  return (
    <div className='container'>
      <Picture src={picture.src}>
          <ComplexComponewnt>
            . . . . . .
          </ComplexComponewnt>
      </Picture>
    </div>
  )
```



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

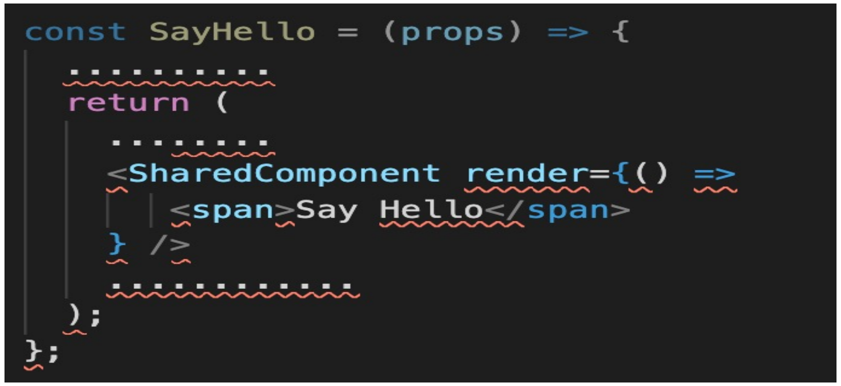
The Render Prop pattern

- **Use the pattern to share logic between components.**
- **Dfn.**: A render prop is a function prop that a component uses to know what to render.

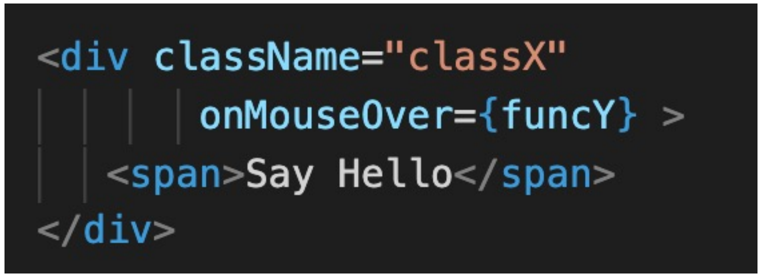


```
const SharedComponent = (props) => {
  ..........
  return (
    <div className="classX"
          onMouseOver={funcY} >
      { props.render() }
    </div>
  );
};
```

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

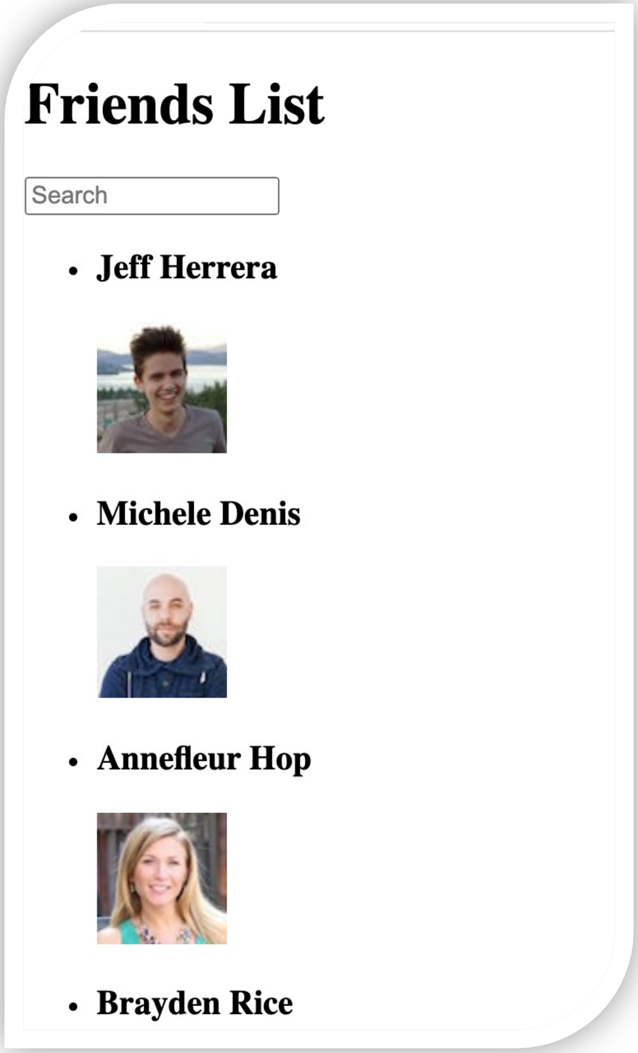


```
const SayHello = (props) => {
  ..........
  return (
    ........
    <SharedComponent render={() =>
      <span>Say Hello</span>
    } />
    .............
  );
};
```



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

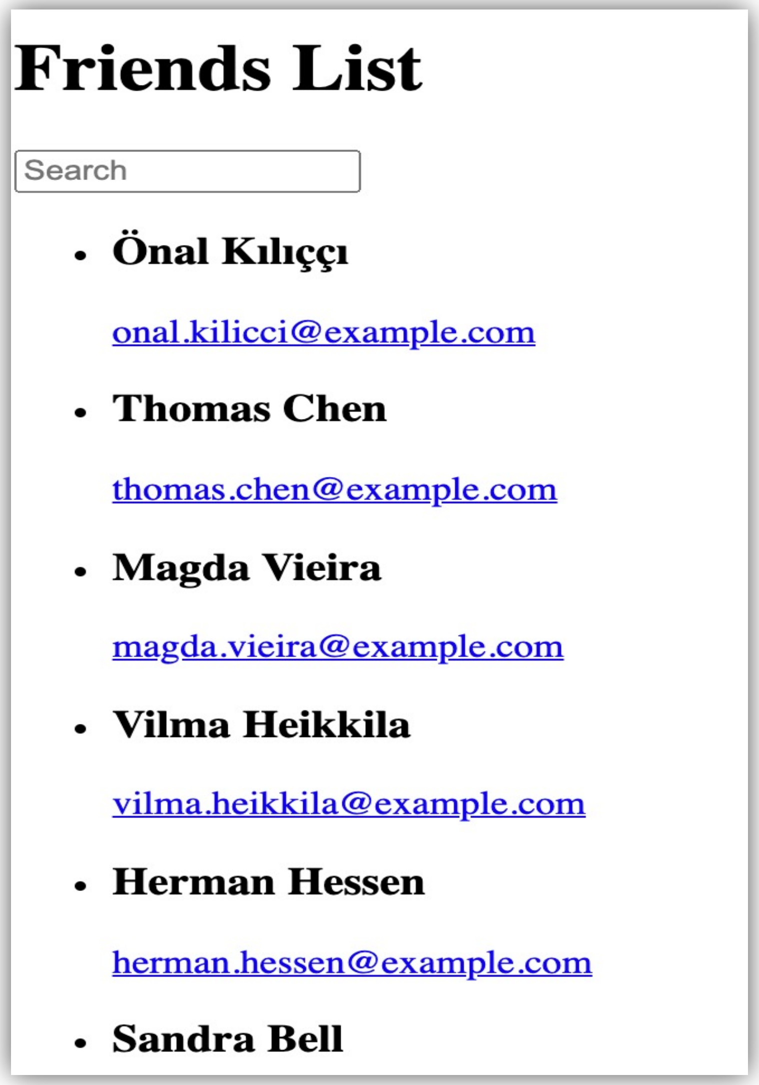
The Render Prop - Sample App.



Friends List

Search

- Jeff Herrera
- Michele Denis
- Annefleur Hop
- Brayden Rice



Friends List

Search

- Önal Kılıççı

  onal.kilicci@example.com
- Thomas Chen

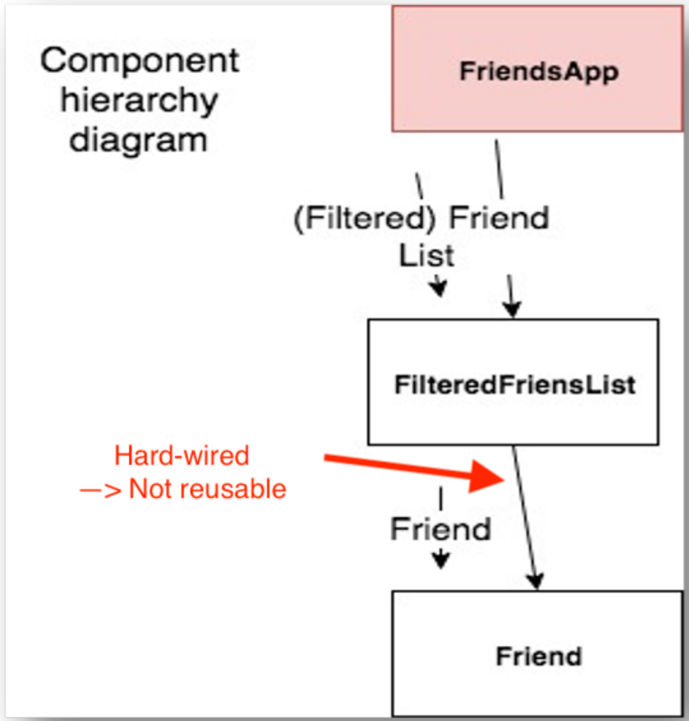
  thomas.chen@example.com
- Magda Vieira

  magda.vieira@example.com
- Vilma Heikkila

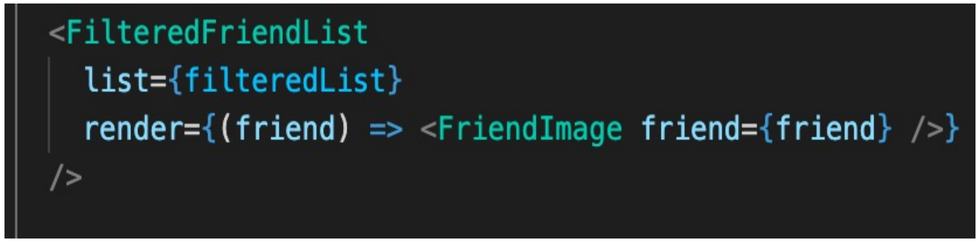
  vilma.heikkila@example.com
- Herman Hessen

  herman.hessen@example.com
- Sandra Bell

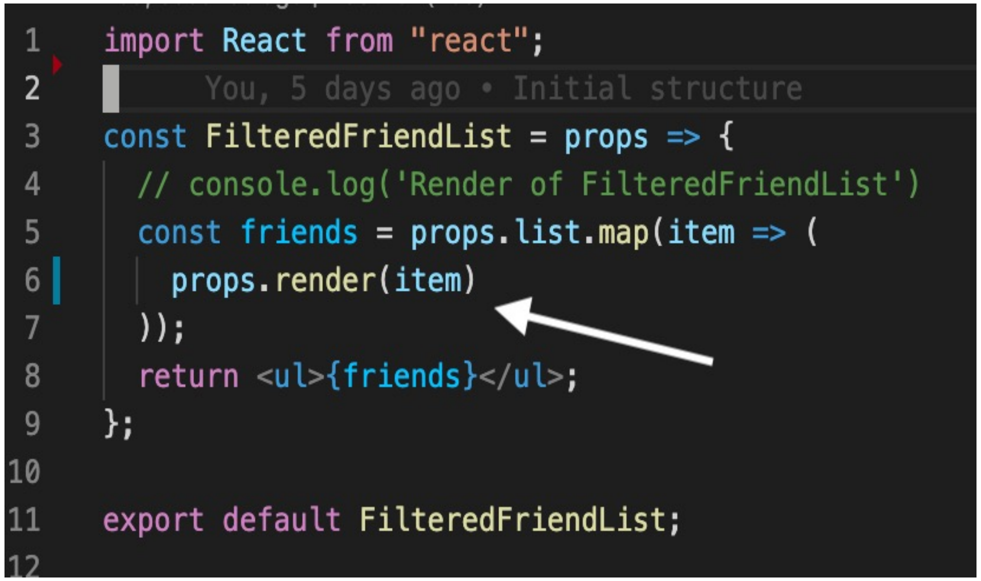
The Render Props - Sample App.



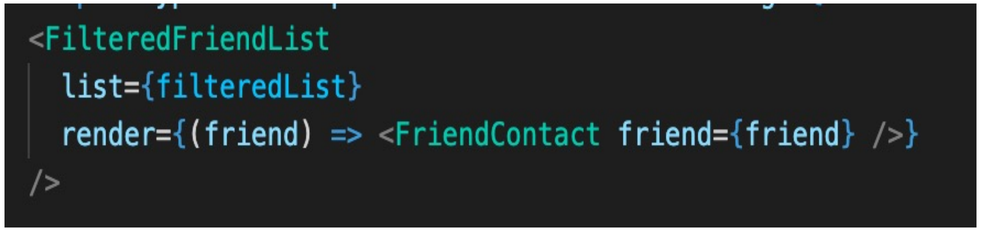
- **Solution: As well as passing the list of matching friends to**
- **, we also tell it <u>how to render a friend</u>**
- **Use a prop to communicate the 'how', i.e. a render prop**



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



```
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} />}
/>
```

- FilteredFriendList **is no longer statically importing the component for rendering a friend.**
- **It receives this via the render prop.**
- **The friends array elements will be** Friend **components, e.g.** FriendContact, FriendImage

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

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

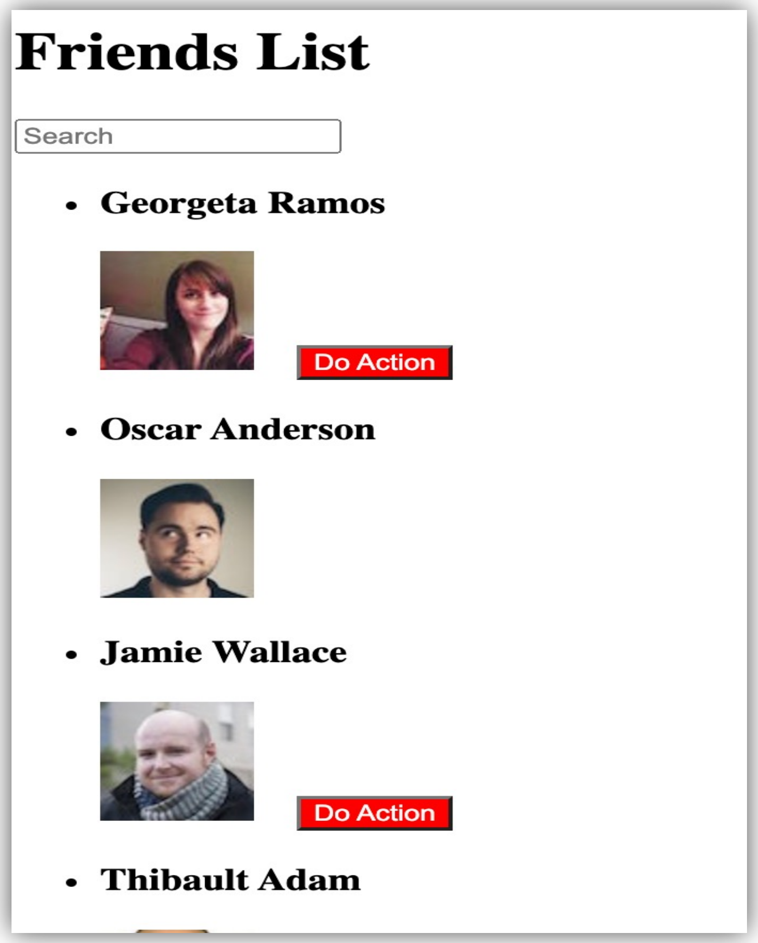
Reusability.

- **Core React composition Patterns:**
  1. **Containers**
  2. **Render Props**
  3. **Higher Order Components.**

- **HOC is a function that takes a component and returns an enhanced version of it.**
  - **Enhancements could include:**
    - **Statefulness**
    - **Props**
    - **UI**
- **Ex –** withRouter **function.**
- **Naming convention:** withXXXXXXXX**()**

Higher Order Component - Sample App.

- **Objective: Add a click handler to multiple components.**



Friends List

Search

- Georgeta Ramos

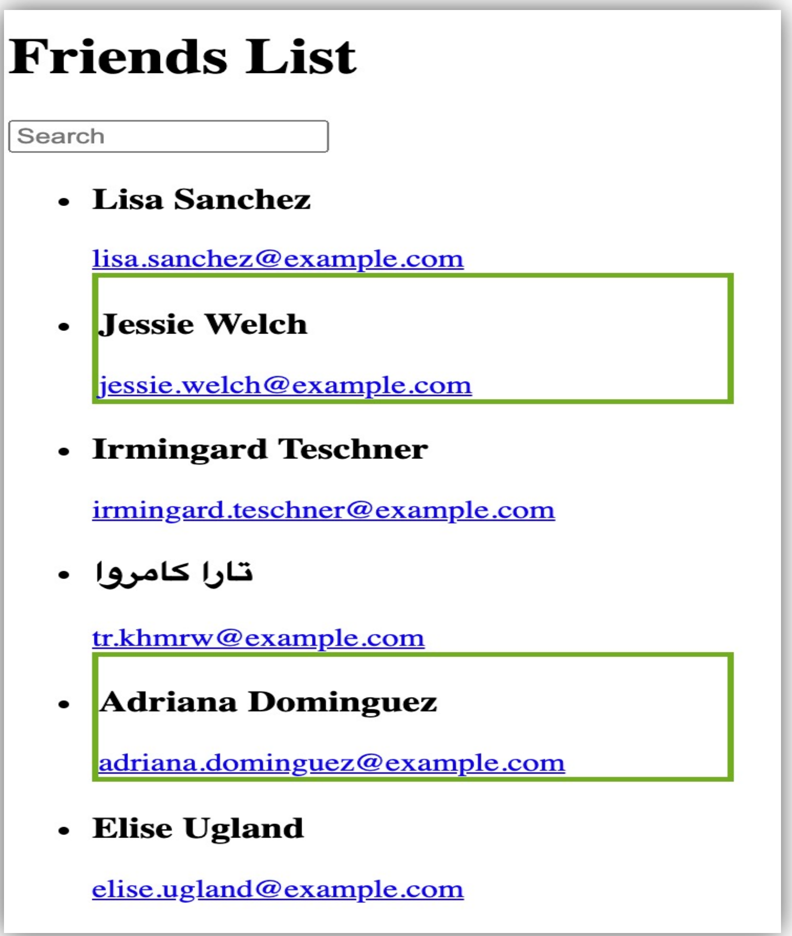
  Do Action

- Oscar Anderson

- Jamie Wallace

  Do Action

- Thibault Adam



Friends List

Search

- Lisa Sanchez

  lisa.sanchez@example.com

- Jessie Welch

  jessie.welch@example.com

- Irmingard Teschner

  irmingard.teschner@example.com

- تارا کامروا

  tr.khmrw@example.com

- Adriana Dominguez

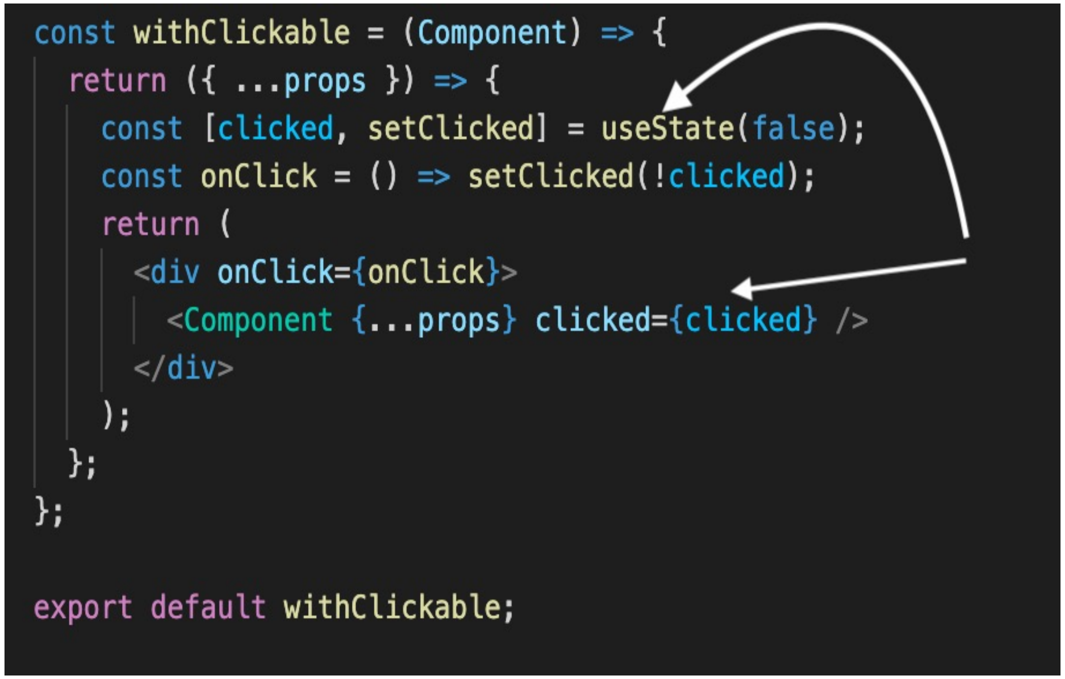
  adriana.dominguez@example.com

- Elise Ugland

  elise.ugland@example.com

Higher Order Component - Sample App.

```
const withNothing = (Component) => {
        return ({ ...props }) => ( <Component {...props} /> );
}
```



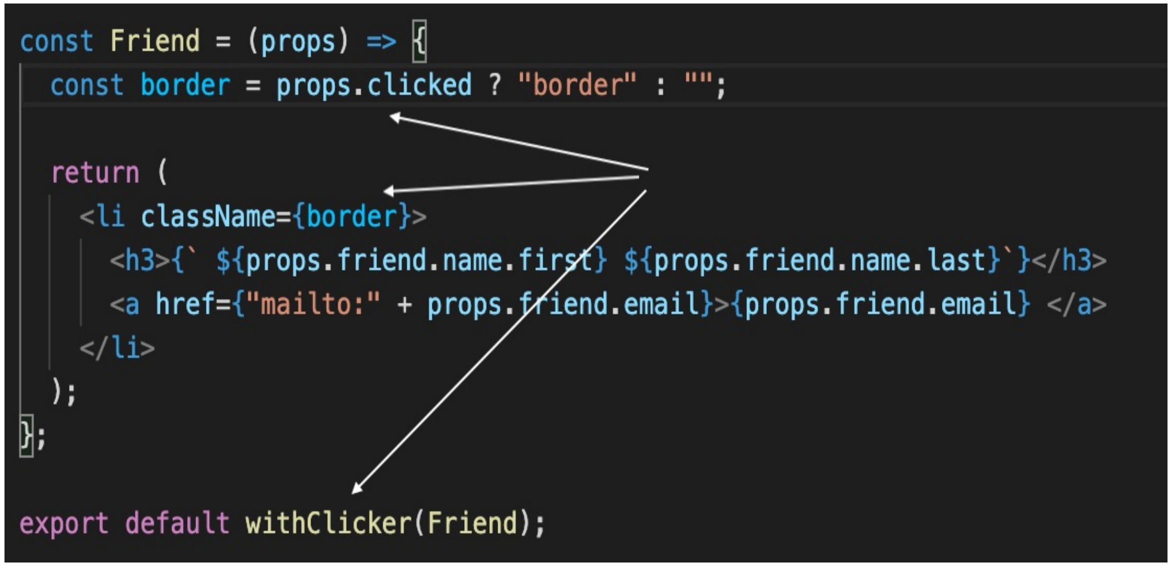
```
const withClickable = (Component) => {
  return ({ ...props }) => {
    const [clicked, setClicked] = useState(false);
    const onClick = () => setClicked(!clicked);
    return (
      <div onClick={onClick}>
        <Component {...props} clicked={clicked} />
      </div>
    );
  };
};

export default withClickable;
```

This HOC enhances a custom component by:

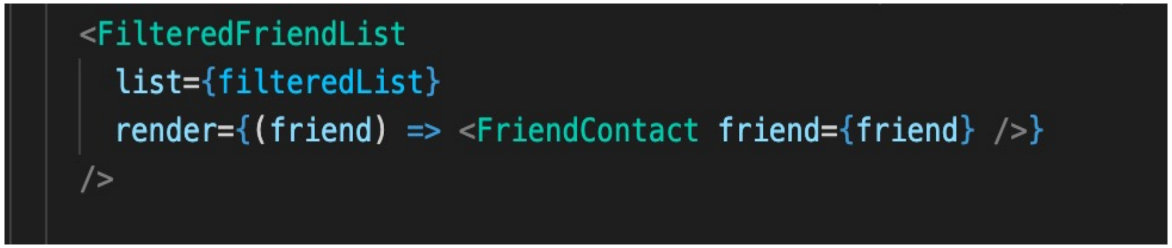
1.Adding state **that is controlled by the onClick event.**

2.Passing it an additional prop

Higher Order Component - Sample App.



```
const Friend = (props) => {
  const border = props.clicked ? "border" : "";

  return (
    <li className={border}>
      <h3>{` ${props.friend.name.first} ${props.friend.name.last}`}</h3>
      <a href={"mailto:" + props.friend.email}>{props.friend.email} </a>
    </li>
  );
};

export default withClicker(Friend);
```



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

- **The** clicked **prop does not appear in the invocation of our custom component (FriendContact); its supplied by the HOC**
- **The custom component decides how to react to the state change caused by the onClick event**

Summary.

- **Objectives – Reusability, Separation of Concerns (Single Responsibility), DRY.**
- **Benefits - Maintainability, Understandability, Extendability, Adaptability.**
- **Means – Apply design patterns.**

- **React App.**
  - **Composition.**
  - **Patterns – Container, Render Prop, Higher Order Component.**

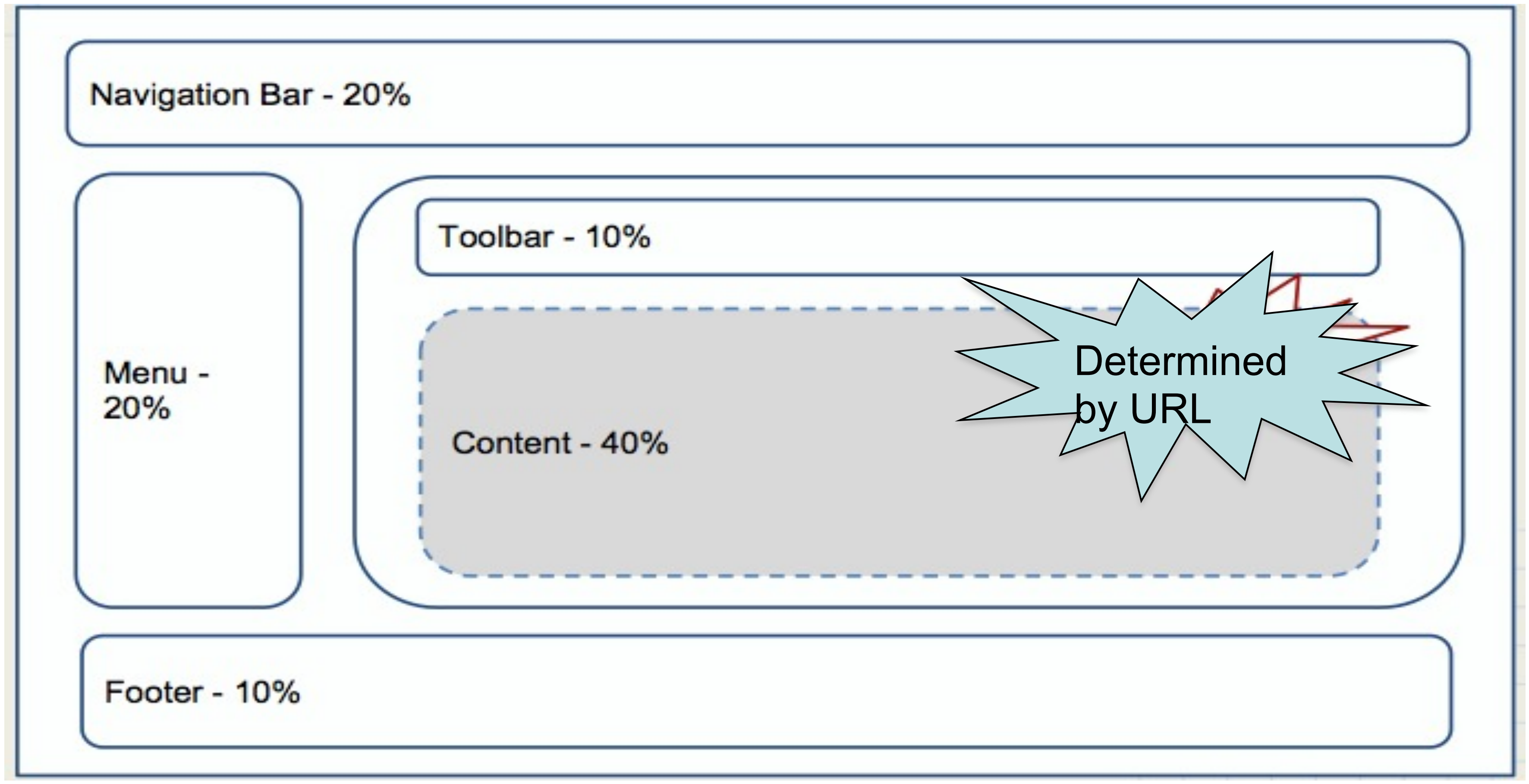
- **More patterns later**

Navigation

(Continued)

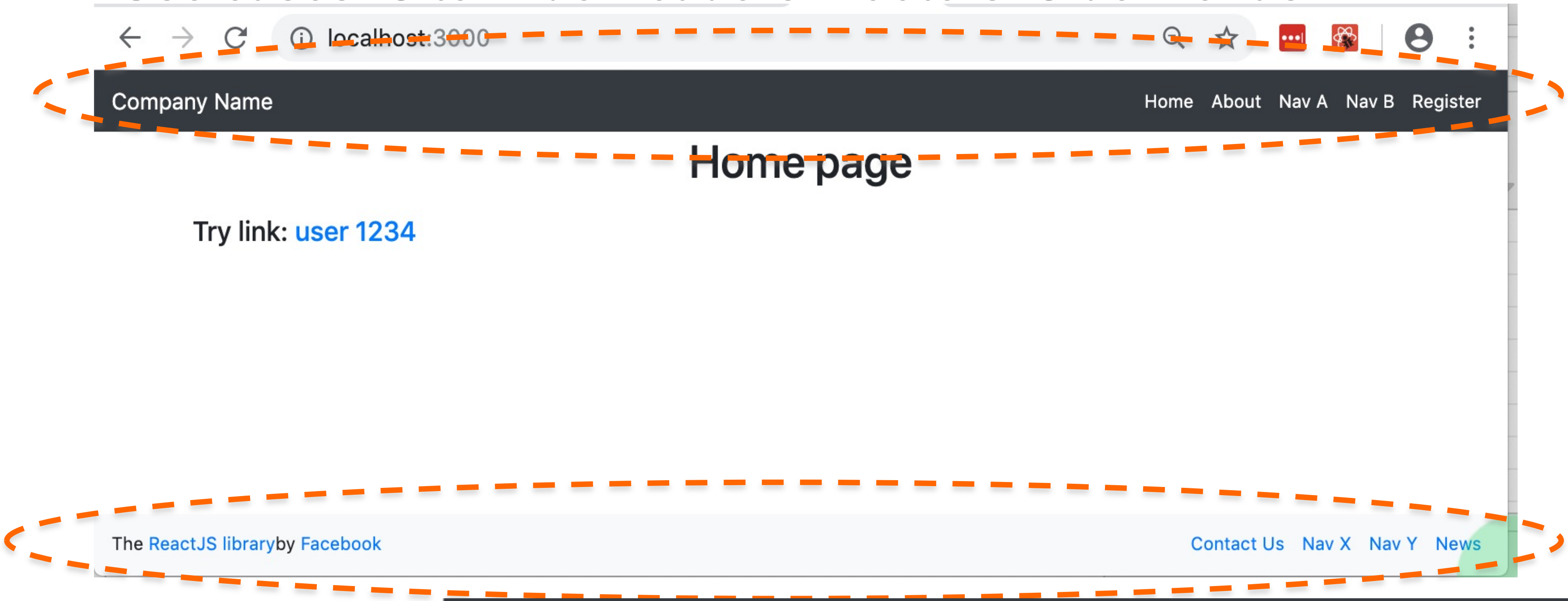
(See Archive from earlier lecture for code samples.)

Typical Web app layout



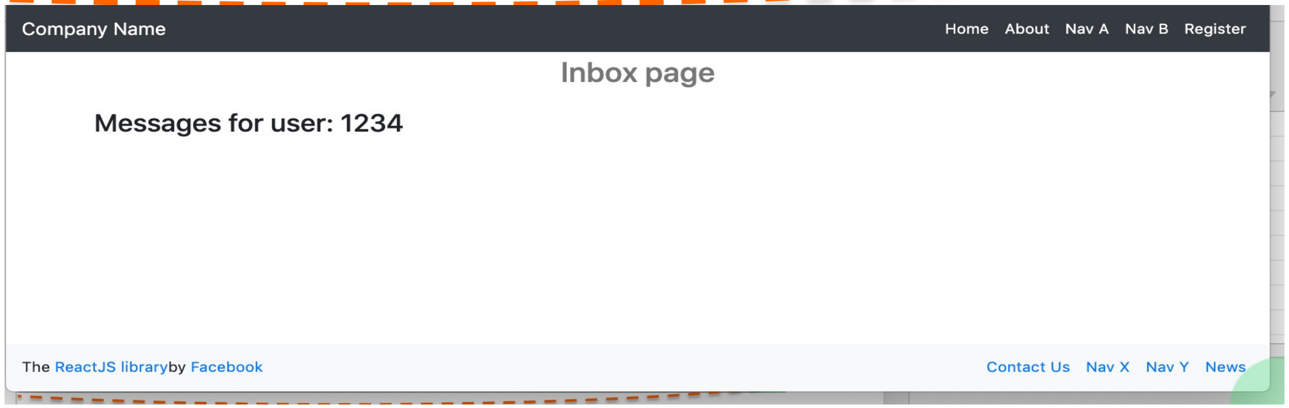
Persistent elements/components

- **Use cases: Site-wide Headers. Footers. Side menus.**



Home page

Try link: user 1234

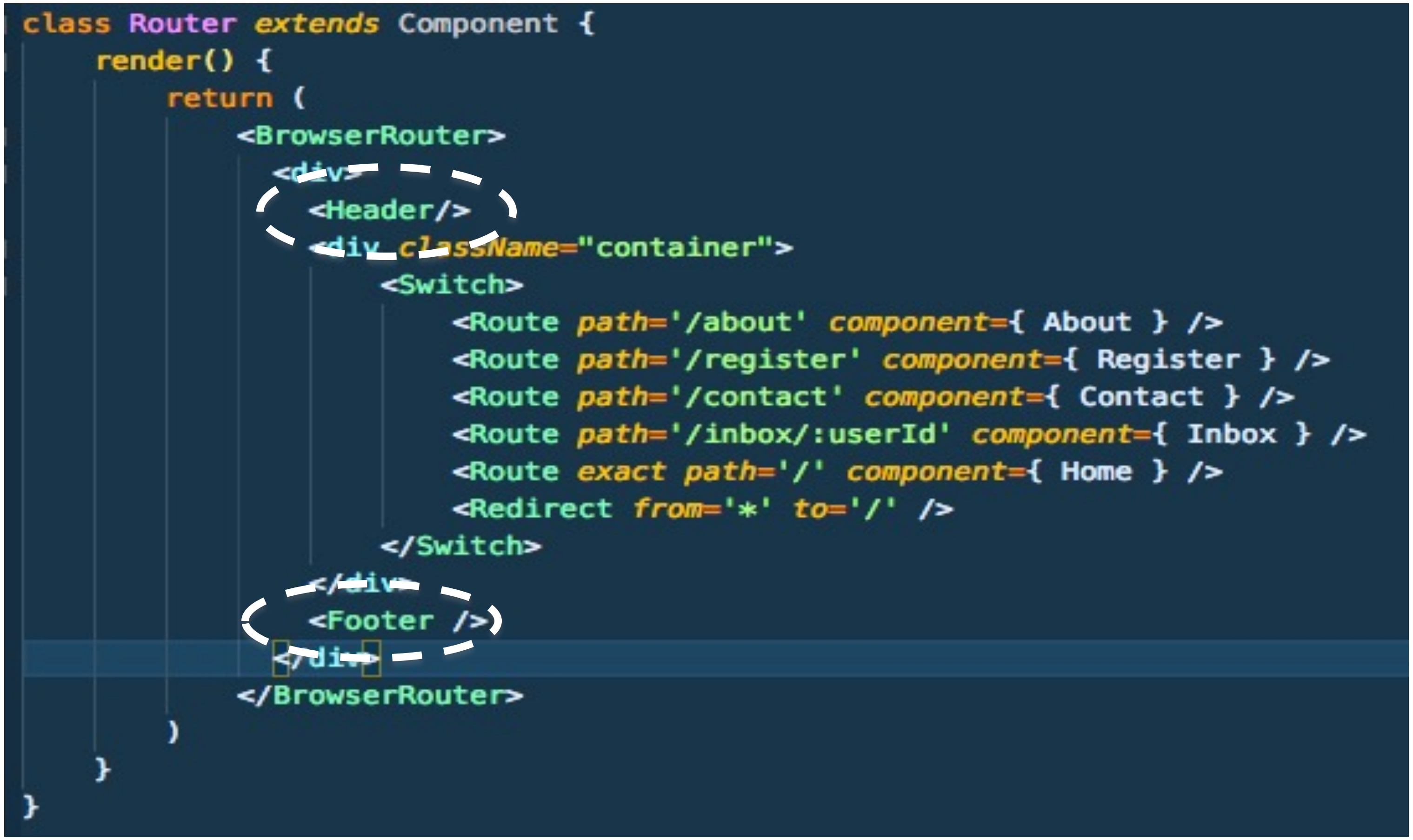


Inbox page

Messages for user: 1234

Persistent elements/components

- **Ref.** src/sample6



```
class Router extends Component {
    render() {
        return (
            <BrowserRouter>
              <div>
                <Header/>
                <div className="container">
                    <Switch>
                        <Route path='/about' component={ About } />
                        <Route path='/register' component={ Register } />
                        <Route path='/contact' component={ Contact } />
                        <Route path='/inbox/:userId' component={ Inbox } />
                        <Route exact path='/' component={ Home } />
                        <Redirect from='*' to='/' />
                    </Switch>
                </div>
                <Footer />
              </div>
            </BrowserRouter>
        )
    }
}
```

Alternative <Route> API.

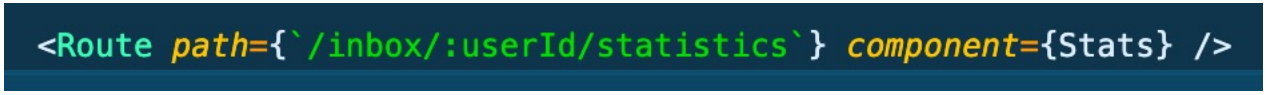
- **To-date:** *<Route path={…URL path…} component={ ComponentX} />*
  - **Mounted component always gets a default prop object.**
- **Disadv.: We cannot pass custom props to the mounted component.**

- **Alternative:**

  <Route path={…URL path…} render={…function….}>

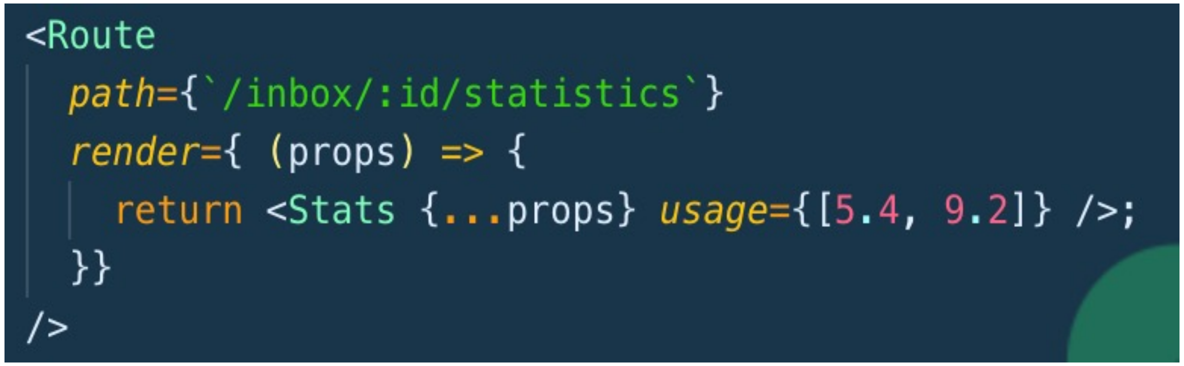
  – **where** *function* **return the mounted component.**
- **EX.: See** /src/sample7/**.**

  Objective: Pass usage data to the <Stats> **component from** sample4's nested Route.



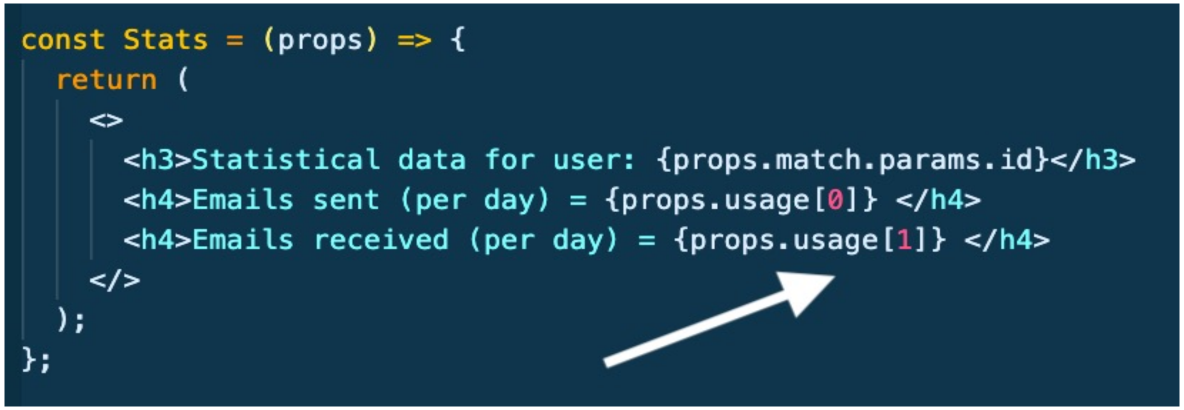
```
<Route path={`/inbox/:userId/statistics`} component={Stats} />
```

Alternative <Route> API.



```
<Route
  path={`/inbox/:id/statistics`}
  render={ (props) => {
    return <Stats {...props} usage={[5.4, 9.2]} />;
  }}
/>
```

- **The** render **prop function argument is the inherited props object.**

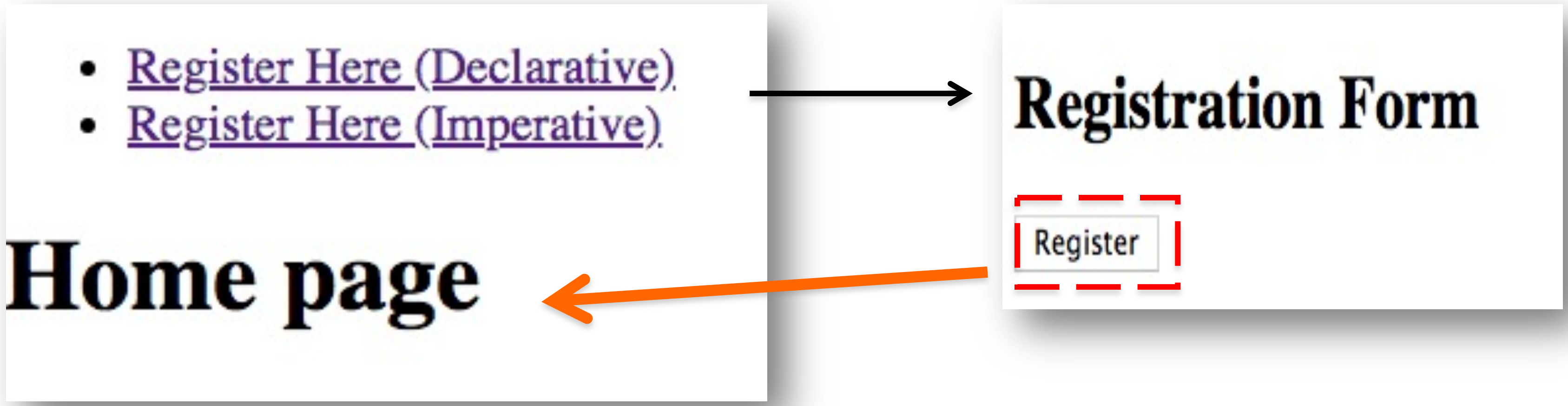


```
const Stats = (props) => {
  return (
    <>
      <h3>Statistical data for user: {props.match.params.id}</h3>
      <h4>Emails sent (per day) = {props.usage[0]} </h4>
      <h4>Emails received (per day) = {props.usage[1]} </h4>
    </>
  );
};
```

Programmatic Navigation.

- **Performing navigation in JavaScript.**
- **Two options:**
  1. **Declarative – requires state; use** <Redirect />.
  2. **Imperative – requires** withRouter**() ; use** props.history

- **EX.: See** /src/sample8/**.**



Summary

- **React Router package adheres to React principles:**
  - **Declarative.**
  - **Component composition.**
  - **The event → state change → re-render**
- **Package's main components - <BrowserRouter>, <Route>, <Redirect>, <Link>.**
- **The** withRouter() **higher order component.**
  - **Additional props:**
    - **props.match.params; props.history; props.location**

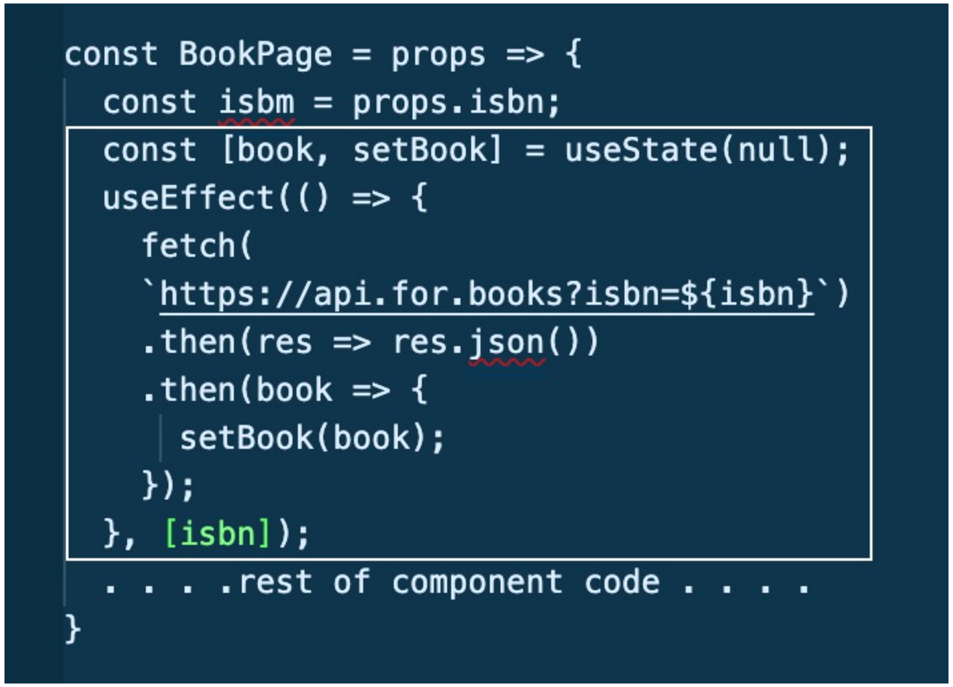
- **Recently add hooks support (alternative to** withRouter**)**
  - **useHistory(), useParams(), useLocation**

Custom Hooks

Custom Hooks.

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

Example:

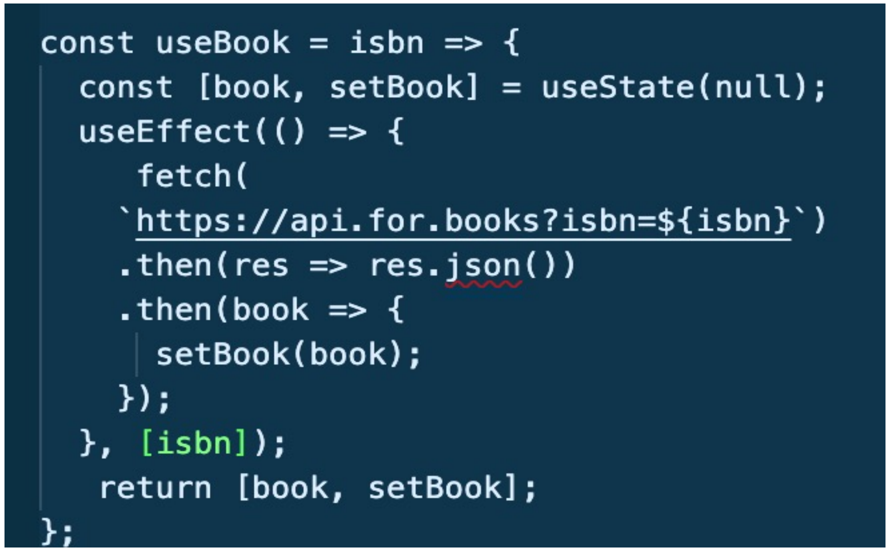


```
const BookPage = props => {
  const isbm = props.isbn;
  const [book, setBook] = useState(null);
  useEffect(() => {
    fetch(
    `https://api.for.books?isbn=${isbn}`)
    .then(res => res.json())
    .then(book => {
      setBook(book);
    });
  }, [isbn]);
  . . . .rest of component code . .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

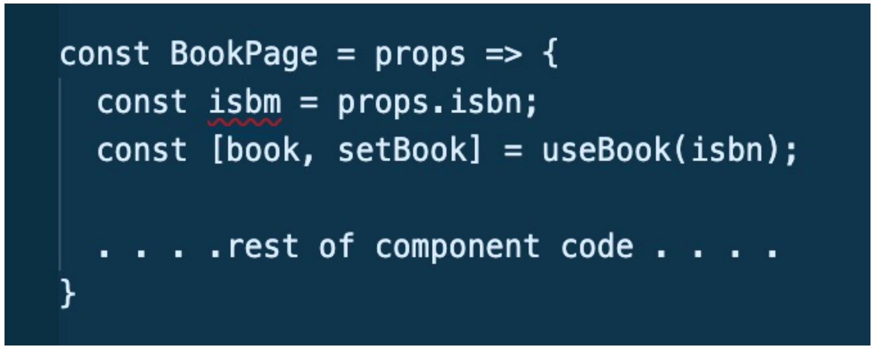
Objective – Extract the book-related state code into a custom hook.

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 can 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.**