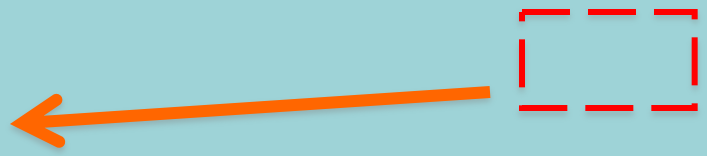
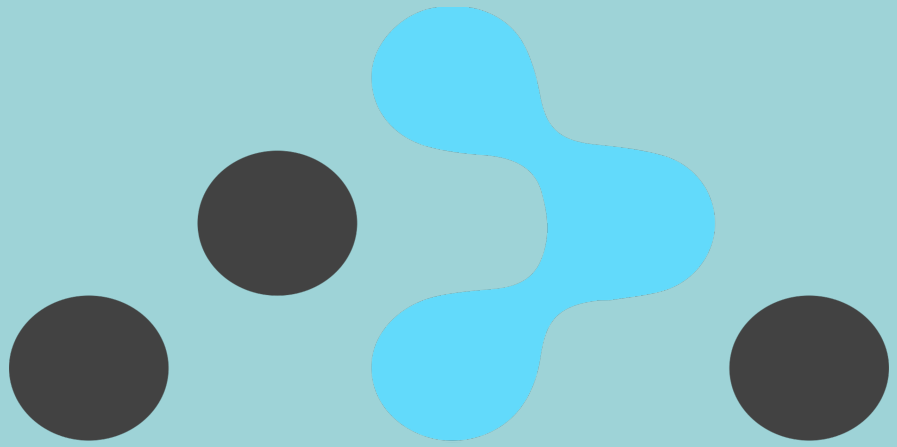
Agenda

- **More Routing samples.**
- **More Design pattern samples.**
- **Custom Hooks.**
- **Assignment 1 specification.**





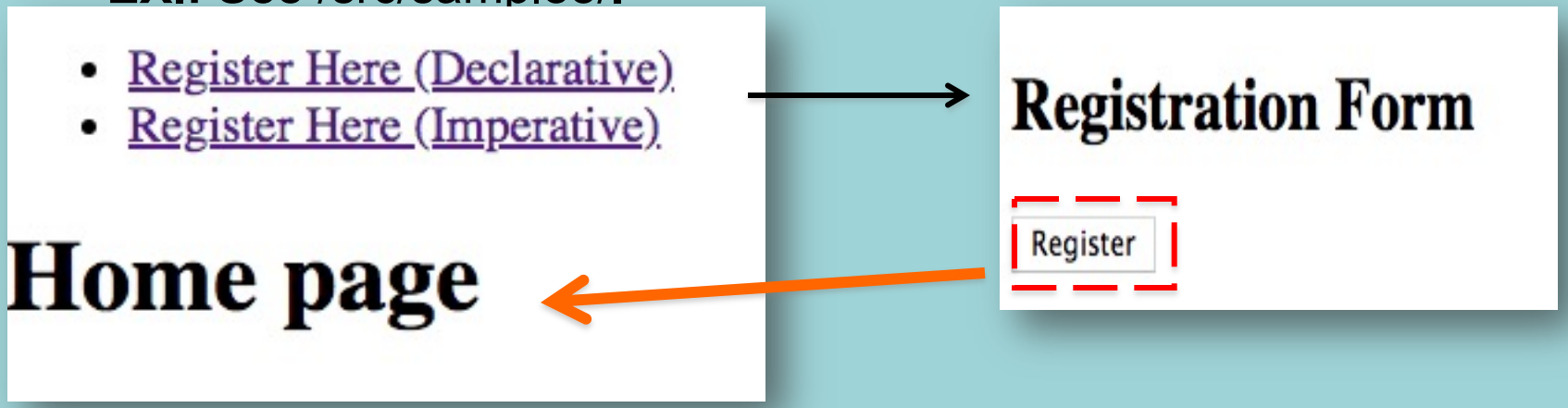
Navigation

(Continued)

Programmatic Navigation.

- **Performing navigation in JavaScript.**
- **Two options:**
  1. **Declarative – requires state; use** <Redirect />.
  2. **Imperative – requires** the useHistory **hook provided by react-router**

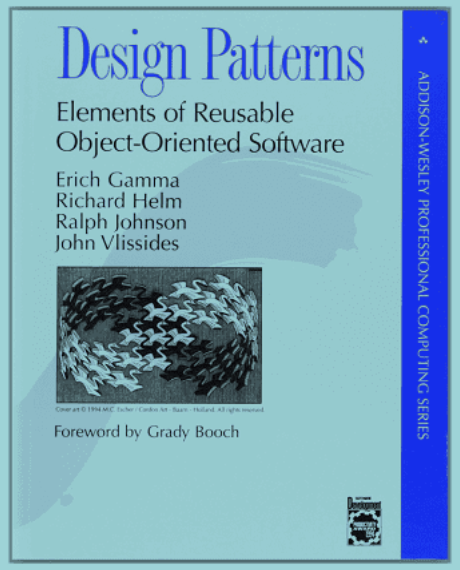
- **EX.: See** /src/sample8/**.**



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>.**

- **Special hooks to allow us access routing data/methods, e.g.** useParams, useHistory, useLocation.



Design Patterns

(Continues)

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.**

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.

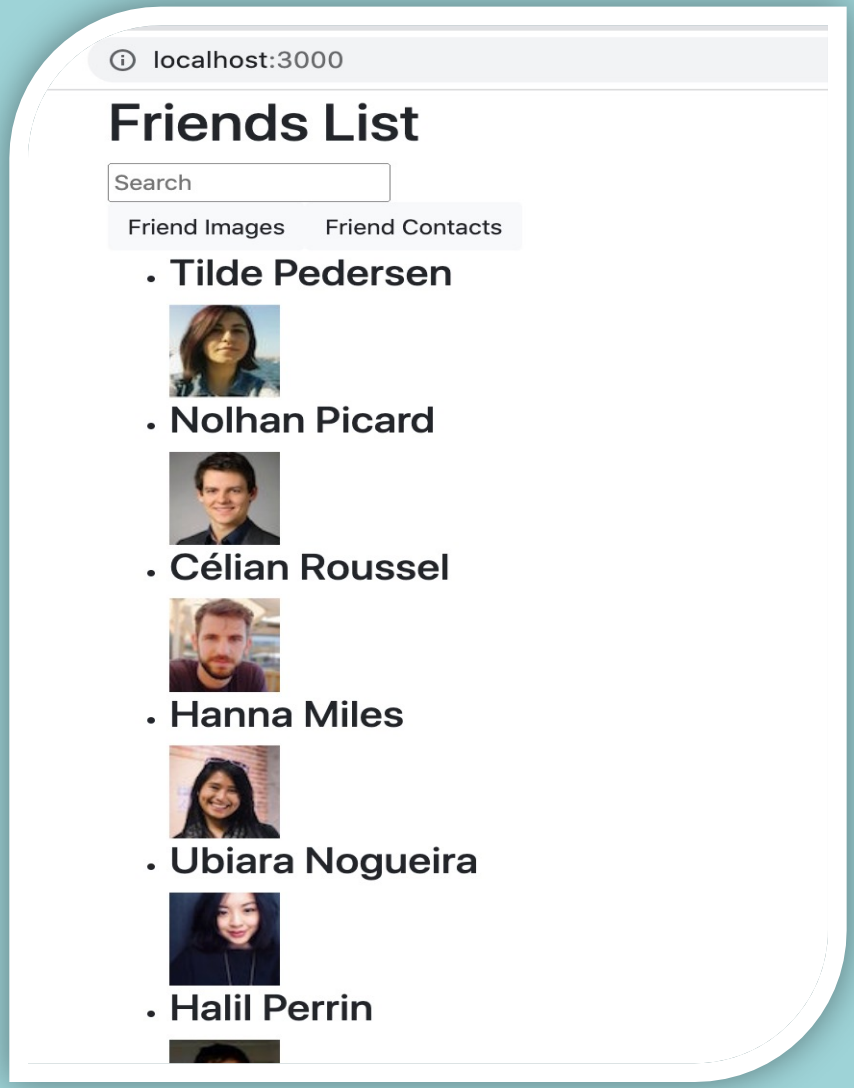
```
const SharedComponent = (props) => {
  ..........
  return (
    <div className="classX"
                onMouseOver={funcY} >
      { props.render() }
    </div>
  );
};
```

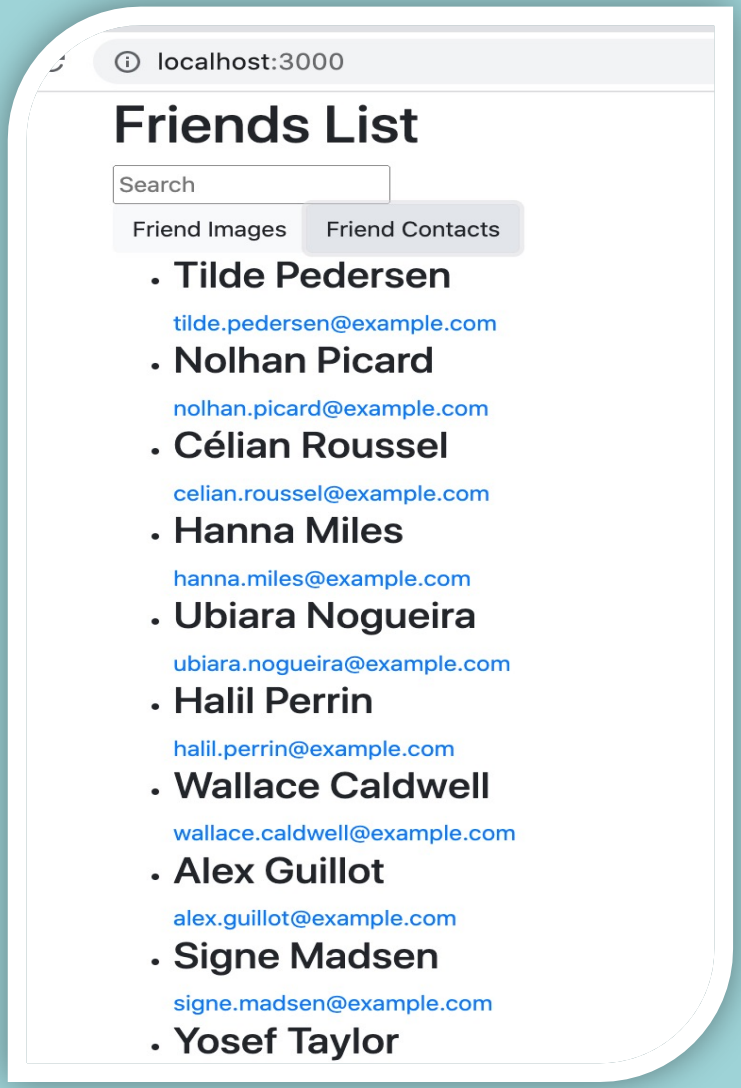
- SharedCoomponent **receives its render logic from the consumer, i.e.** SayHello.
- Prop name is arbitrary.

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

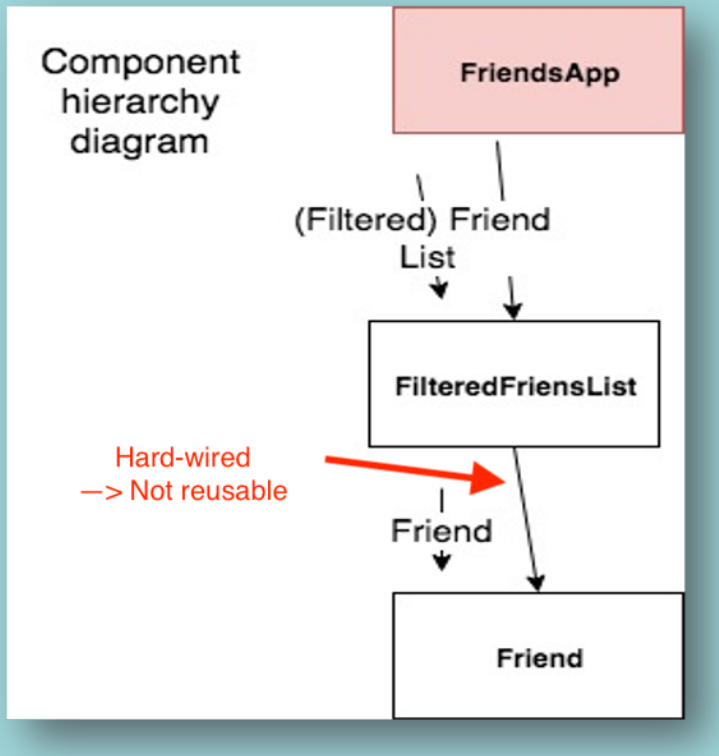
```
<div className="classX"
            onMouseOver={funcY} >
  <span>Say Hello</span>
</div>
```

The Render Prop - Sample App.





The Render Props - Sample App.



- Problem: **The** FilteredFriendList **component is statically bound to a** Friend **component type.**
- **We want to support multiple types of** Friend **components.**
- Solution: T**ell** FilteredFriendList **how to render a Friend, using a render prop.**
- **Promotes the DRY principle.**

```jsx
import React from "react";
// import Friend from ....  STATIC DEPENDENCY
const FilteredFriendList = ({list, render}) => {
  const friends = list.map((item) =>
      render(item)
  );
  return <ul>{friends}</ul>;
};

export default FilteredFriendList;
```

- FilteredFriendList **does not statically import the component for rendering a friend.**
- **It receives this via the render prop instead.**

```jsx
import FilteredFriendList from "./components/filteredFriendList";
import FriendContact from "./components/friendContact";
import FriendImage from "./components/friendImage";

const FriendsApp = () => {
  const [searchText, setSearchText] = useState("");
  const [format, setFormat] = useState("image");
  const [friends, setFriends] = useState([]);

  const Friend = format === "image" ? FriendImage : FriendContact;
```

```jsx
</ul>
<FilteredFriendList
  list={filteredList}
  render={(friend) => <Friend key={friend.email} friend={friend} />}
/>
```

Reusability.

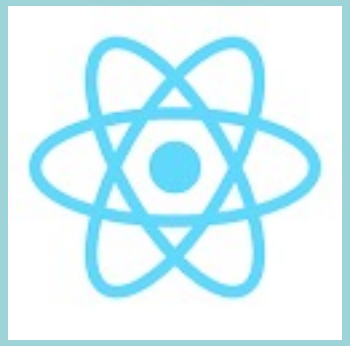
- **Core React composition Patterns:**
  1. **Containers**
  2. **Render Props**
  3. **Higher Order Components.**

- **HOC is a function that takes a component and returns an enhanced version of it.**
  - **Enhancements could include:**
    - **Statefulness**
    - **Props**
    - **UI**

Summary.

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

- **React SPA development:**
  - **Composition.**
  - **Patterns – Container, Render Prop, Higher Order Component.**
  - **Hooks are a new design approach to problems previously solved using patterns.**



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 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.**