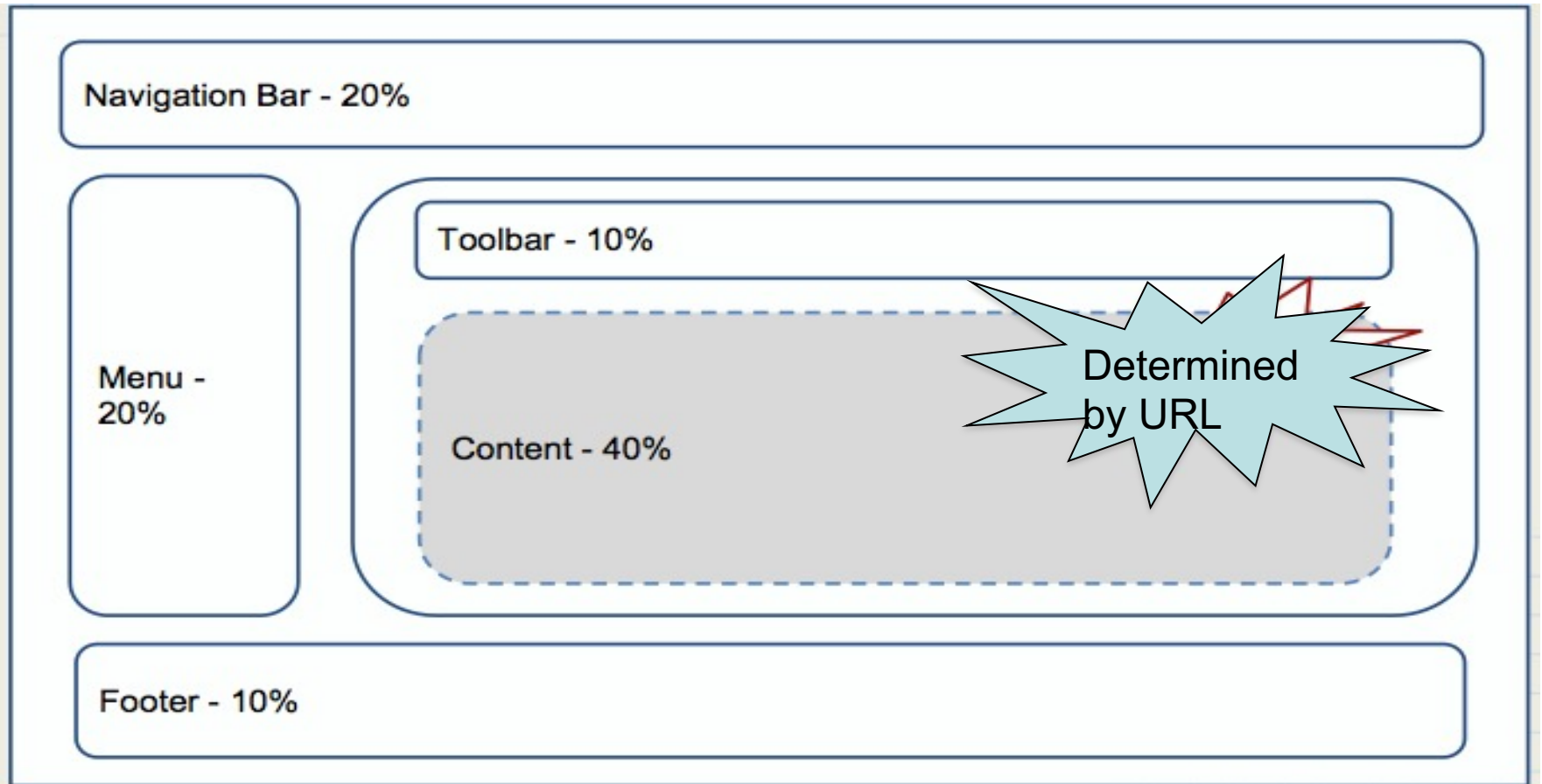


Navigation

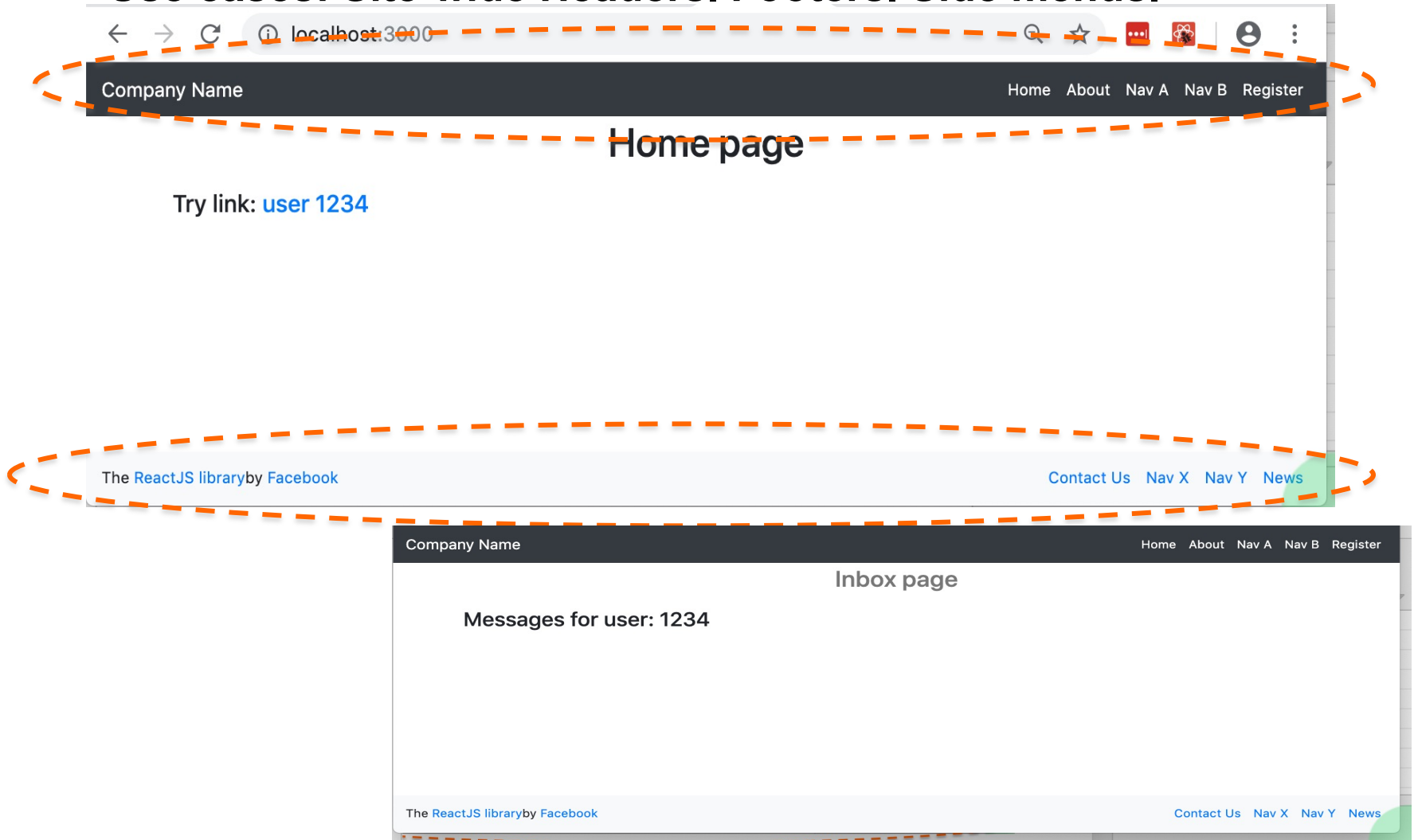
(Continued)

Typical Web app layout



Persistent elements/components

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



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]} />;
  }}
/>
```

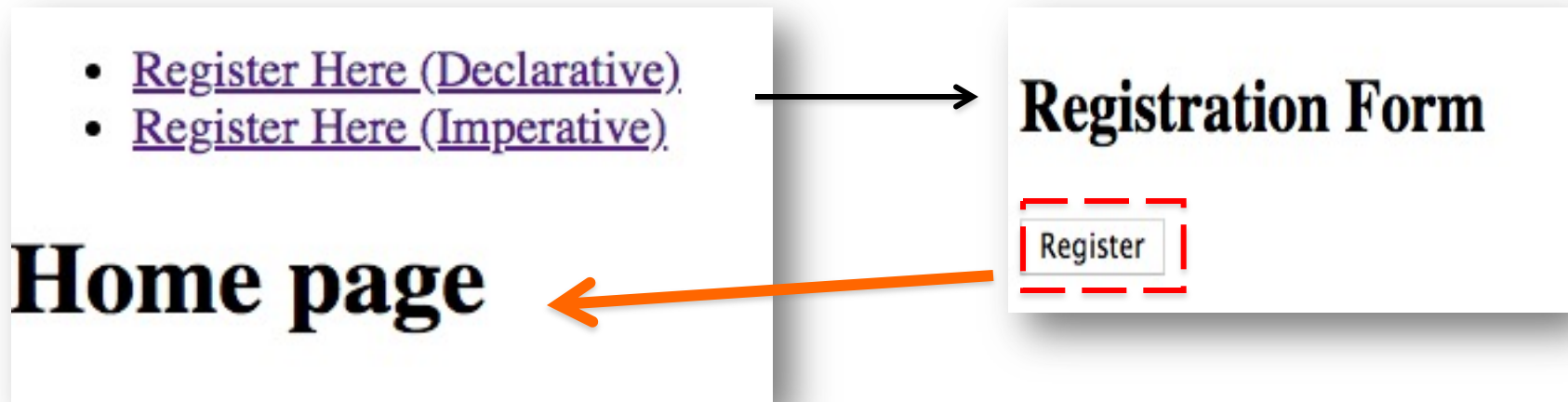
- The render prop function argument is the inherited props object.

```
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>
    </>
  );
};
```



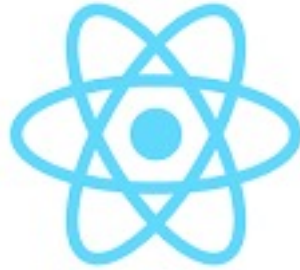
Programmatic Navigation.

- 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 added hook 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 isbn = 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 isbn = 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.

