

# CS142: Section 5

Single Page Apps with React Material UI

# Agenda

- Single Page Applications (SPAs)
  - What is it?
  - How it works
- Problem 1
- Problem 2

# Review: Single Page Application

- Browser sends one request to server
- Server sends back one index.html and lots of JavaScript
- Browser receives, builds view templates, fetches model data

...

- User input -> URL changes
- Browser switches out part of view, fetches new model data
  - React Router

# SPA Pros and Cons

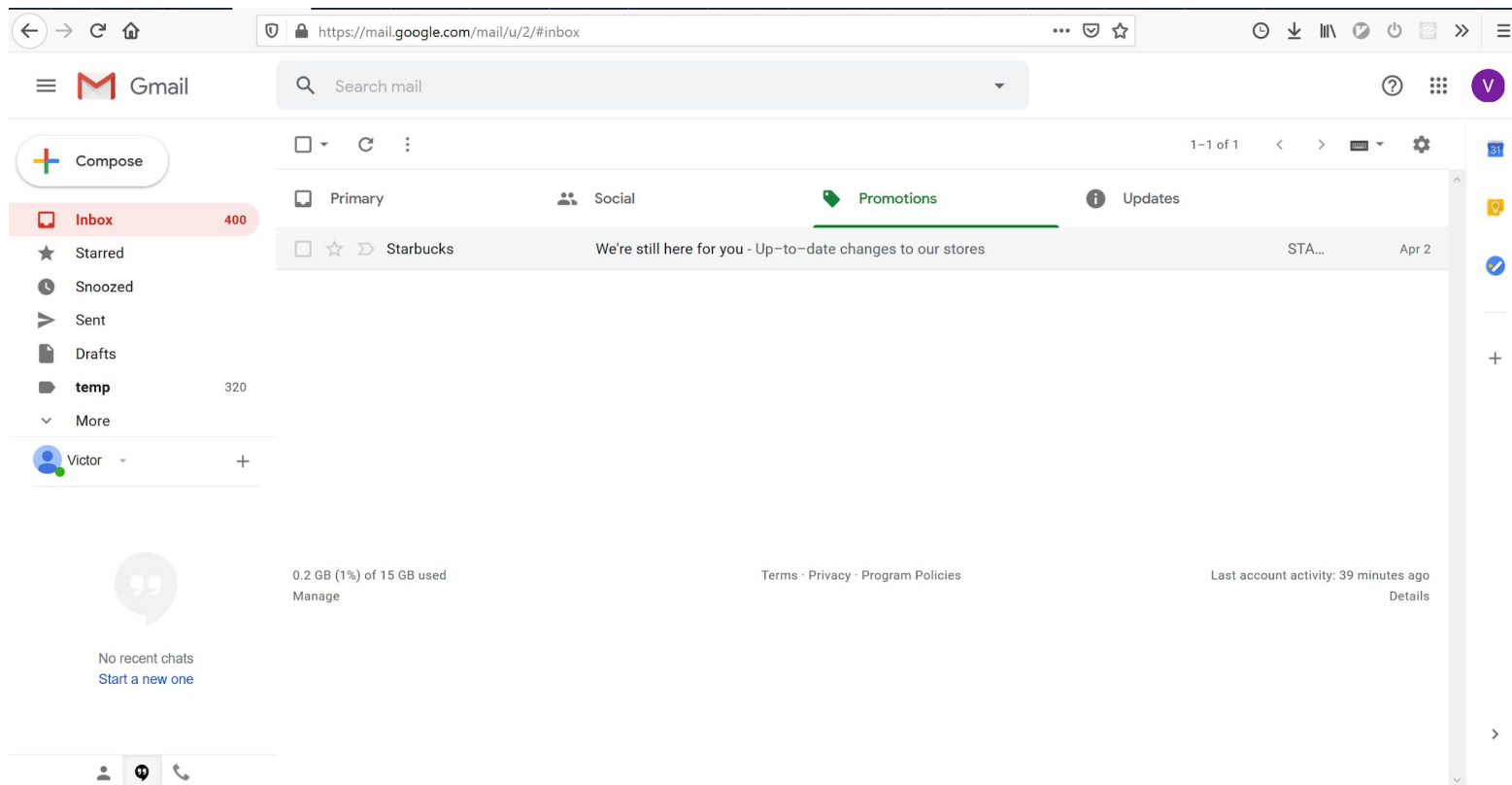
## Pros

- Faster experience for users (usually)
- Simplifies backend server
- Easier to develop and debug

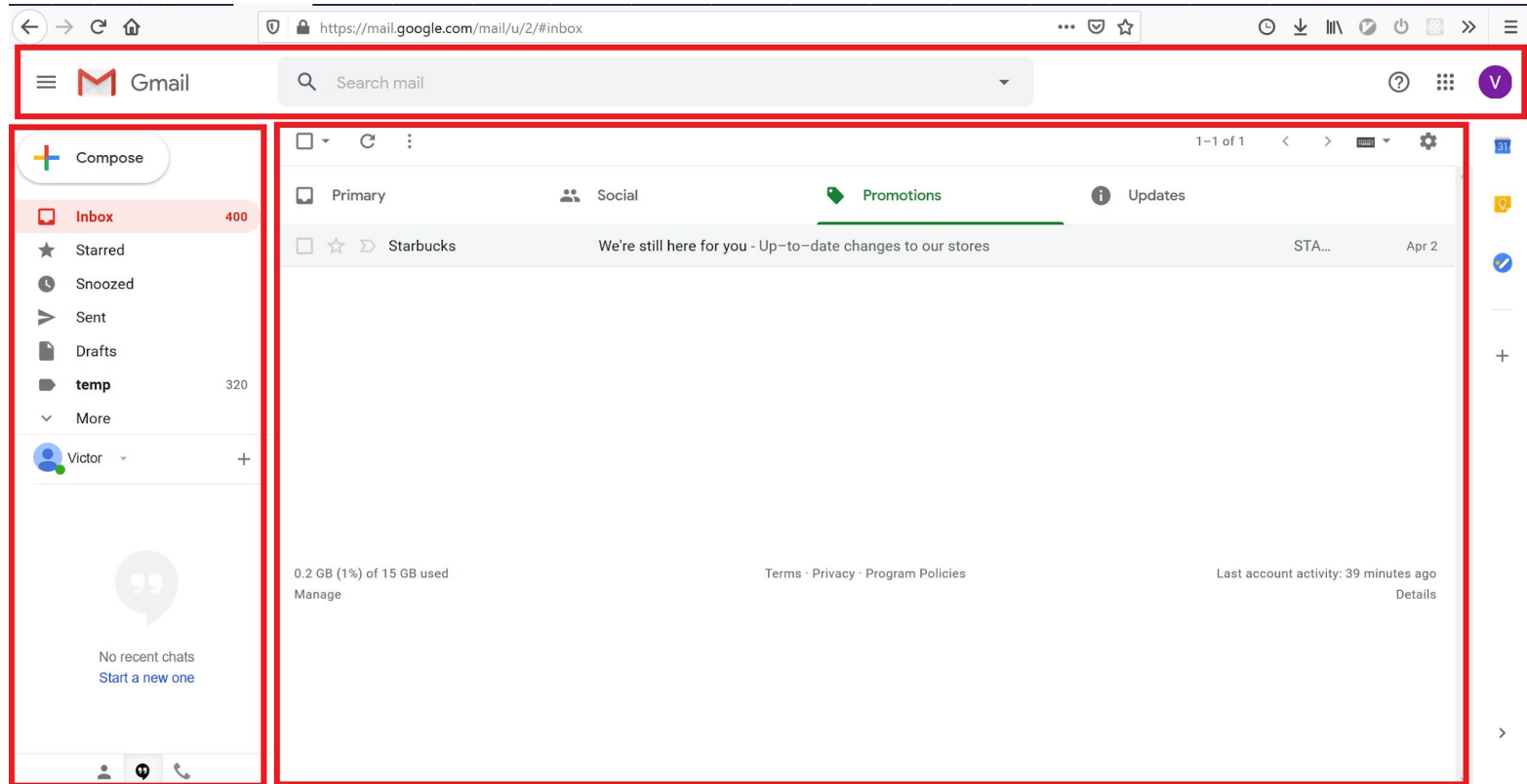
## Cons

- Search Engine Optimization (SEO)
- Requires Javascript and puts most of the workload on the client

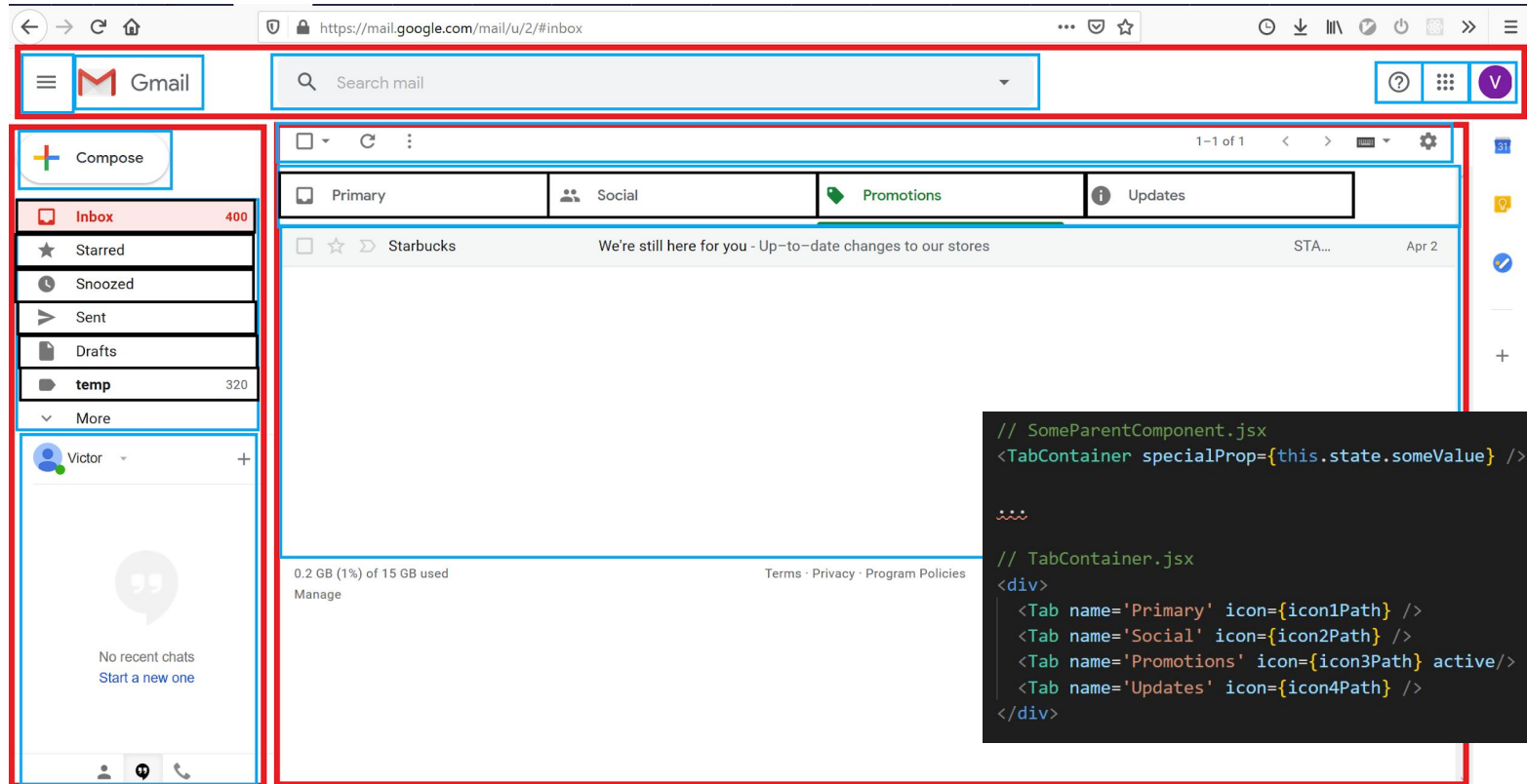
# SPA Example



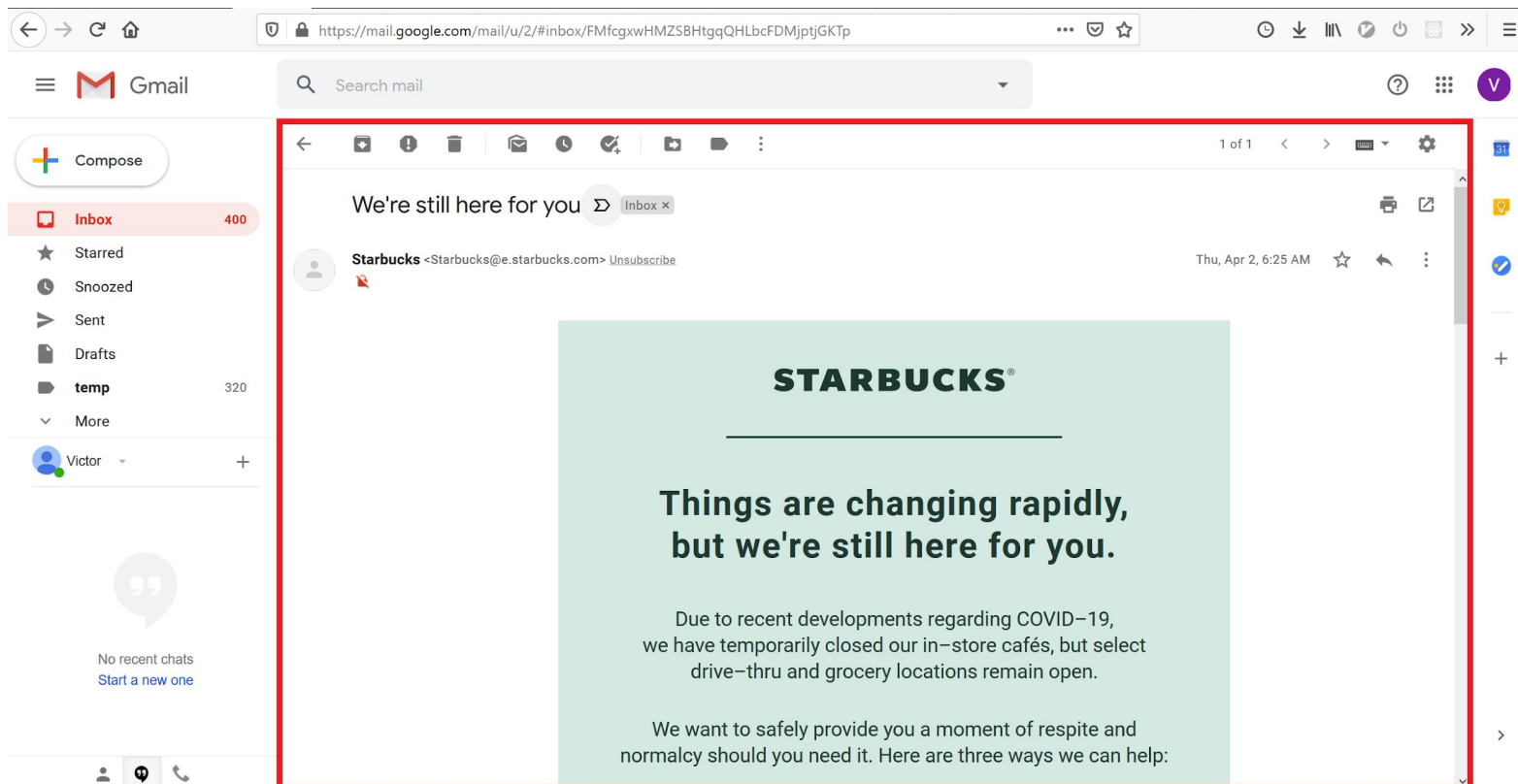
# Views are compositions of smaller building blocks/templates



# Views are compositions of smaller building blocks/templates



# Everything is a component that can be switched out





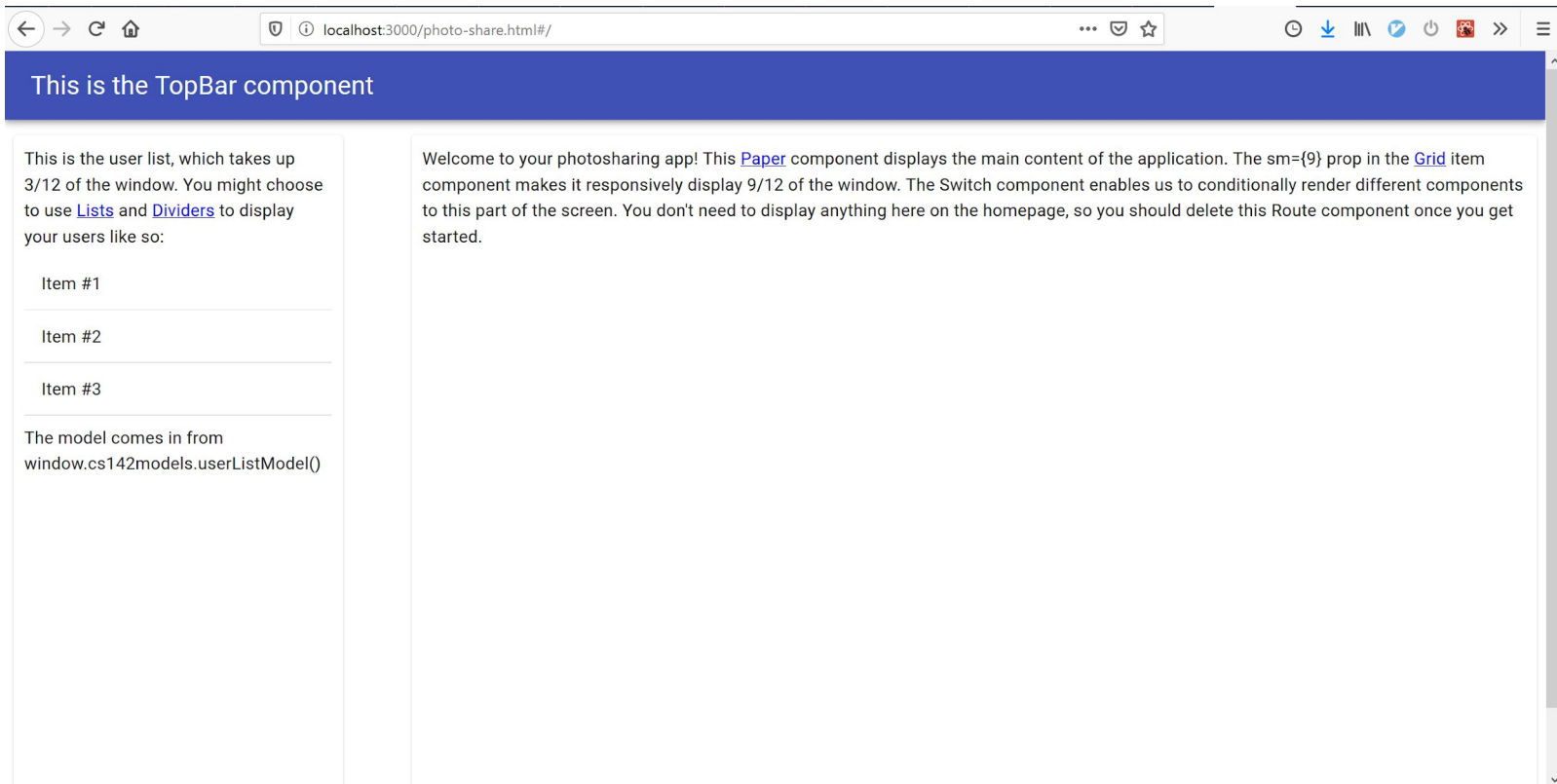
# Problem 1

# Goal

Lay out the framework for your photo sharing app.

Given some hard-coded model data (`window.cs142models`), display the model in a navigable web app format. (Similar to designing the `States` component from Project 4.)

# Base Layout



# Code Walkthrough

```
window.cs142models.userListModel()  
  [User, User, User]
```

## **User**

```
_id: "57231f1a30e4351f4e9f4bd7"  
first_name: "Ian"  
last_name: "Malcolm"  
location: "Austin, TX"  
description: "Should've stayed in the car."  
occupation: "Mathematician"
```

*modelData/photoApp.js*

# Code Walkthrough

```
window.cs142models userModel ("57231f1a30e4351f4e9f4bd9")  
  _id: "57231f1a30e4351f4e9f4bd9"  
  first_name: "Peregrin"  
  last_name: "Took"  
  location: "Gondor"  
  description: "Home is behind, the world ahead... And  
there are many paths to tread. Through shadow, to the edge  
of night, until the stars are all alight... Mist and shadow,  
cloud and shade, all shall fade... all... shall... fade... "  
  occupation: "Thain"
```

*modelData/photoApp.js*

# Code Walkthrough

```
window.cs142models.photoOfUserModel("57231f1a30e4351f4e9f4bd9")  
    [Photo, Photo, Photo]
```

## Photo

```
_id: "57231f1a30e4351f4e9f4be5"  
user_id: "57231f1a30e4351f4e9f4bd9"  
date_time: "2013-12-03 09:02:00"  
file_name: "took1.jpg"  
comments: [Comment, Comment]
```

*modelData/photoApp.js*

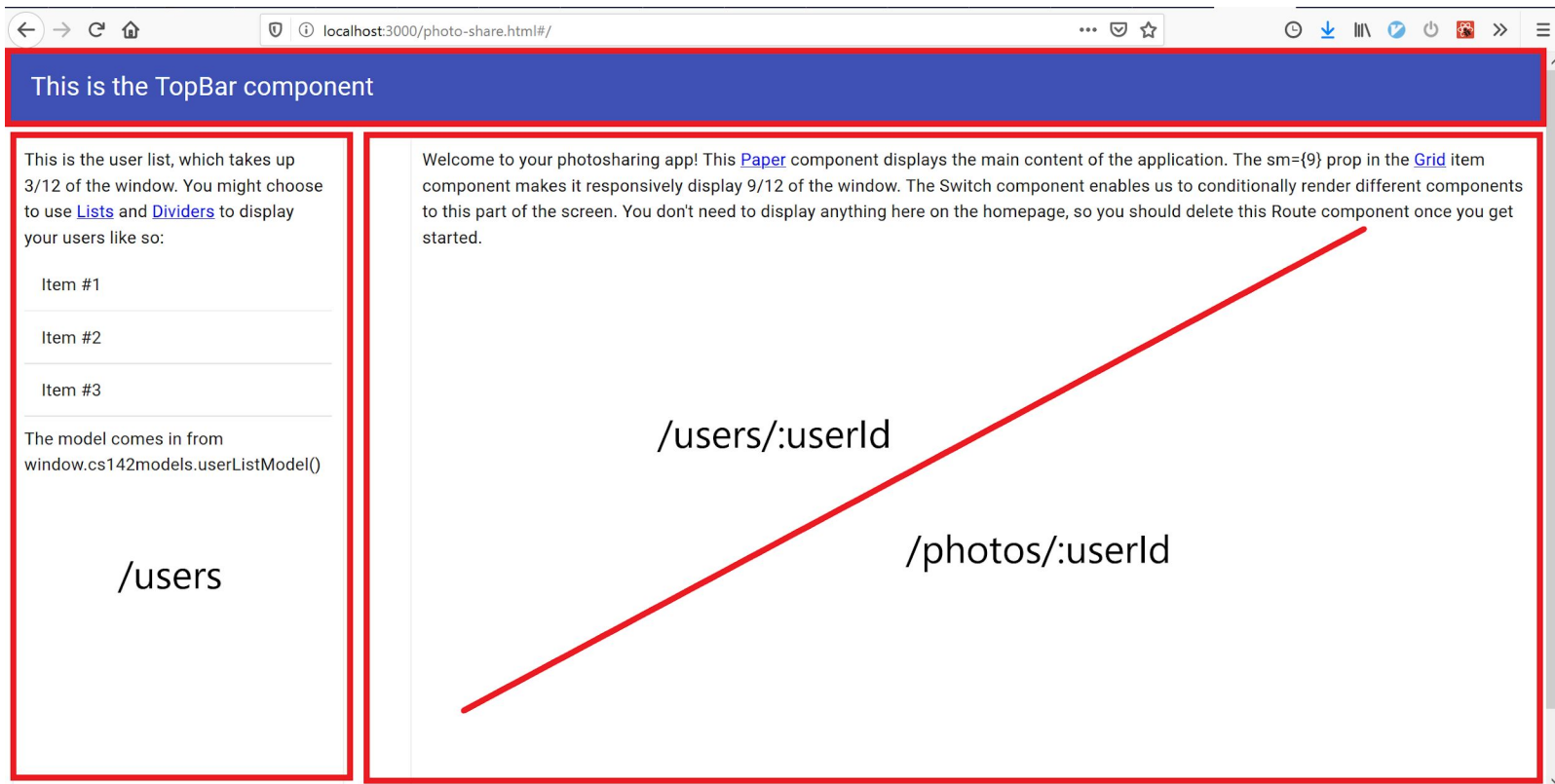
# Code Walkthrough

## Comment

```
_id: "57231f1a30e4351f4e9f4bf4"  
photo_id: "57231f1a30e4351f4e9f4be5"  
user: {user object} // user who posted the comment  
date_time: "2016-01-04 2:00:01"  
comment: "Which one are you?"
```

*modelData/photoApp.js*

# Base layout





# Code Walkthrough

photo-share.html

Your Name	
<a href="#">User A</a>	
<a href="#">User B</a>	
<a href="#">User C</a>	

*components/userList.jsx*

# Code Walkthrough

photo-share.html

Your Name		User A's Detail	
<a href="#"><u>User A</u></a>		first_name last_name	
<a href="#"><u>User B</u></a>		<a href="#"><u>Photos</u></a>	
<a href="#"><u>User C</u></a>			

*/users/:userId*

*components/userDetail.jsx*

# Code Walkthrough

photo-share.html

Your Name		User A's Detail	
<a href="#"><u>User A</u></a>		first_name last_name	
<a href="#"><u>User B</u></a>		<a href="#"><u>Photos</u></a>	
<a href="#"><u>User C</u></a>			

`/users/:userId`

`components/userDetail.jsx`

# Code Walkthrough

photo-share.html

Your Name		Photos of User A	
<a href="#"><u>User A</u></a>		photo1.jpg	
<a href="#"><u>User B</u></a>		<a href="#"><u>User B</u></a> : comment 1	
<a href="#"><u>User C</u></a>		photo2.jpg	

*/photos/:userId*

*components/userPhotos.jsx*

# Passing information between components

- Composition of many React components -> parent components may need to get some new input/info from child components.
- Pass a function down as props to call from the child component

```
class Parent extends Component {
  constructor() {
    this.state= { currentInfo: '' };
  }

  setInfo = (newInfo) => {
    this.setState({currentInfo: newInfo});
  }

  render() {
    return (
      <div>
        <Child onNewInfo={this.setInfo} />
      </div>
    )
  }
}
```

```
class Child extends Component {
  constructor() {}
  handleNewInfo = (evt) => {
    this.props.onNewInfo(evt.target.value);
  }
  render() {
    return (
      <div>
        <input value={...} onChange={this.handleNewInfo}/>
      </div>
    )
  }
}
```

# Material UI



## MATERIAL-UI

React components for faster and easier web development.  
Build your own design system, or start with Material Design.

# Some useful components for design

- [Button](#)
- [Grid](#)
- [Card](#)
- [List](#)

Components from 3rd party libraries only take certain props and values.

Look up the documentation/API to know how you can customize them

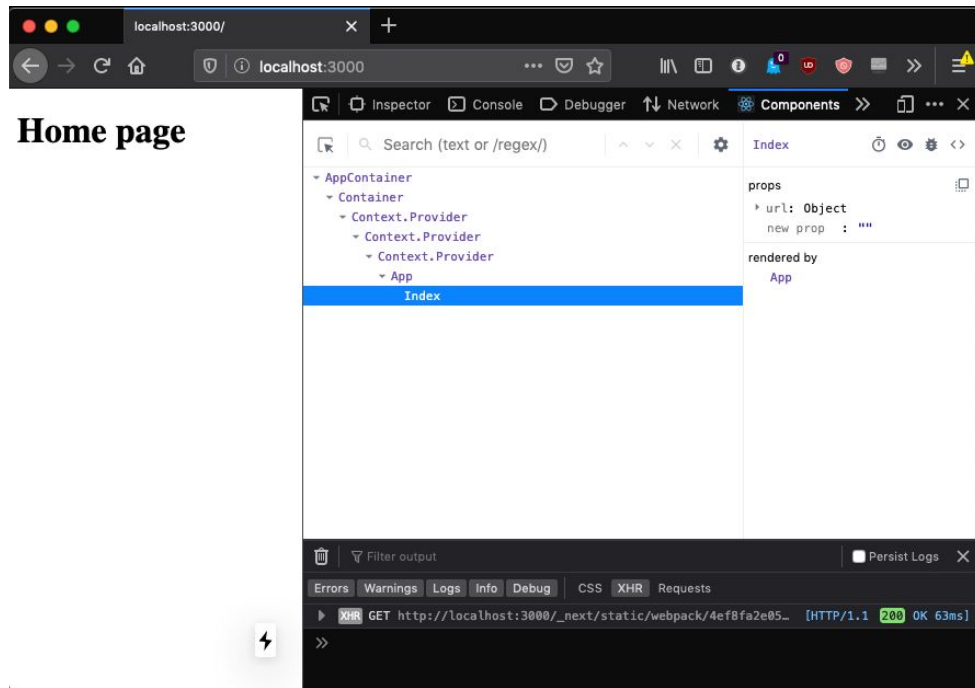


```
<Button variant="contained" color="secondary">  
  Secondary  
</Button>  
<Button variant="contained" disabled>  
  Disabled  
</Button>
```

```
<MenuItem button onClick={() => this.updateSelected(2)} selected={this.state.cur === 2}>  
  <ListItemIcon>  
    <FolderIcon />  
  </ListItemIcon>  
  <ListItemText primary="Archives" />  
</MenuItem>
```

# React Developer Tools

- Common errors come from passing in the wrong props, not updating state correctly, etc.
- Use React Developer Tools browser extension to help debug (similar to the normal browser developer tools)





# Problem 2

# Goal

Instead of getting model data from our global variable (`window.cs142models`), fetch the model data from a server.

You will ultimately replace all calls to `window.cs142models` with calls to `fetchModel()`

# Fetching Models

```
fetchModel(url) .then(doneCallback) ;
```

- fetchModel - Generic method to fetch any model from a server at a specified route (URL)
- url - string - The URL to issue the GET request to

```
function doneCallback () {  
    // Do something  
}
```

- Called with argument (model) when the GET request is done

```
fetchModel(url) .then((res) => {...}) ;
```

# Fetching Models

- Getting model data for our views from a web server takes time.
  - Asynchronous
- AJAX request
  - Asynchronous Javascript and XML
  - Browser prepares and sends HTTP request to server
  - Server processes request and sends back data
  - Browser receives data, processes, and updates view

# XHR Requests

```
xhr = new XMLHttpRequest();  
xhr.onreadystatechange =  
    xhrHandler;  
xhr.open("GET", url);  
xhr.send();
```

```
function xhrHandler() {  
    // Don't do anything if not final state  
    if (this.readyState !== 4) {  
        return;  
    }  
  
    // Final State but status not OK  
    if (this.status !== 200) {  
        console.log(this.statusText);  
        return;  
    }  
  
    // Do something with this.responseText
```

For More: See Server Communications Lecture

# Promise (overview)

- ~~● How do we know when the asynchronous call is over?~~
  - What do we want to do with the return value when the call completes?
- An object that represents an intermediate state of some operation and will eventually produce some useful value.
  - Fulfilled, rejected, or pending

```
new Promise((resolve, reject) => {  
  ...  
  resolve({data: someData});  
  ...  
  reject({error: msg});  
})
```

# Promise (overview)

- How do you get/use the fulfilled Promise return value?

```
let x = myFunc(url).then((res) => {  
    return res;  
})
```

- Similar to just using callback functions for long running operations.

```
processData(params, function(status, data) {...}));
```

# JSON (Javascript Object Notation)

- Server sends data back in JSON
- Need to parse it into Javascript Object to have workable data type

## Helpful methods

- `JSON.parse(someJSONObject)`
- `JSON.stringify(someObject)`

## JSON Example:

```
{  
  "name": "Jane Smith",  
  "age": 40,  
  "job": "doctor"  
}
```



# React Lifecycle Methods (important!)

- Execute logic at critical times of a component's lifecycle
- Components are frequently created, modified, and destroyed
  - `componentDidMount()`
  - `componentDidUpdate()`
  - `componentWillUnmount()`
- Think about when you may want to retrieve model data from the server

```
class CoolComponent extends React.Component {  
  constructor(){}  
  
  componentDidMount() {  
    //process some data  
  
    this.setState({  
      data: newData  
    });  
  }  
  
  componentDidUpdate(prevProps, prevState) {  
    //Compare prevProps to current props  
    //Some logic depending on last previous state  
    this.setState({  
      myValues: someNewValues  
    })  
  }  
  
  render(){}  
}
```

# Homework Tips

Use what you learned in Homework 4! Much of this assignment is applying concepts that you've already learned and used.

Details on React Material: <https://material-ui.com/>

More details on XHR requests:

<https://web.stanford.edu/class/cs142/lectures/ServerCom.pdf>