CS142: Section 5

Single Page Apps with React Material UI

Agenda

- Single Page Applications (SPAs)
  - What they are
  - How they work

- Project 5
  - Problem 1
  - Problem 2

Review: Single Page Applications (SPAs)

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

...

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

SPA Pros and Cons

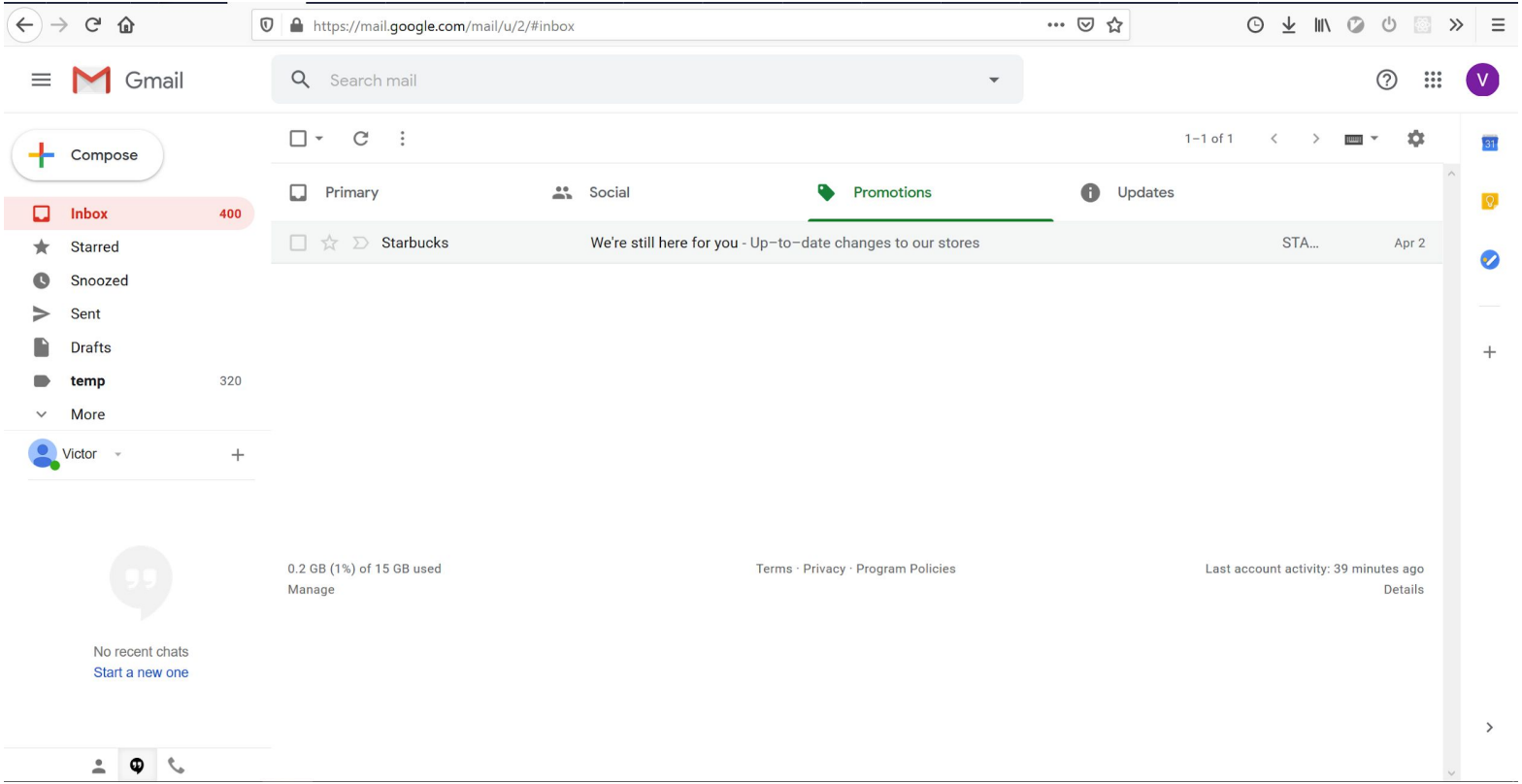
Pros

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

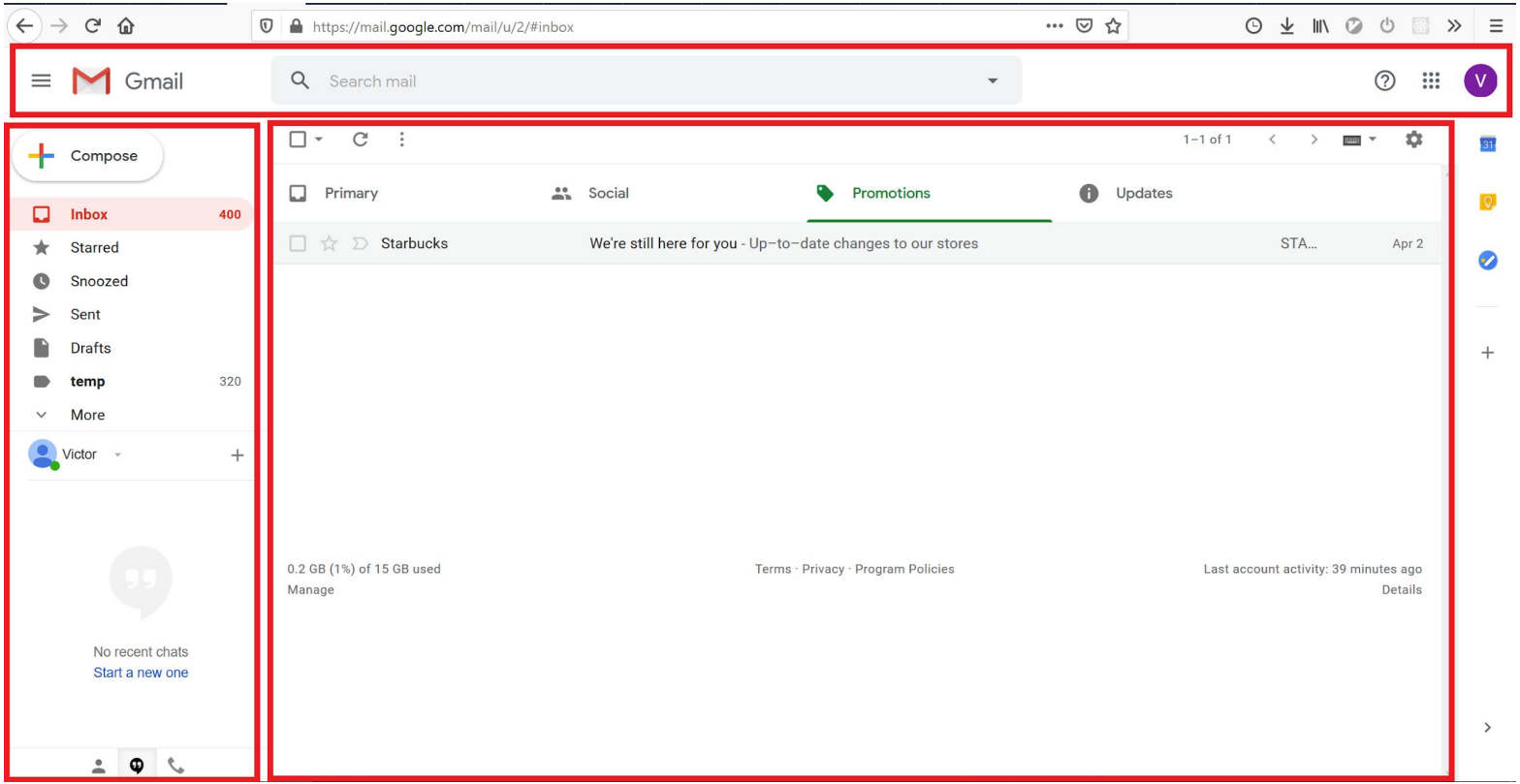
Cons

- Requires Javascript
- Puts most of the workload on the client
- Takes time to load initially

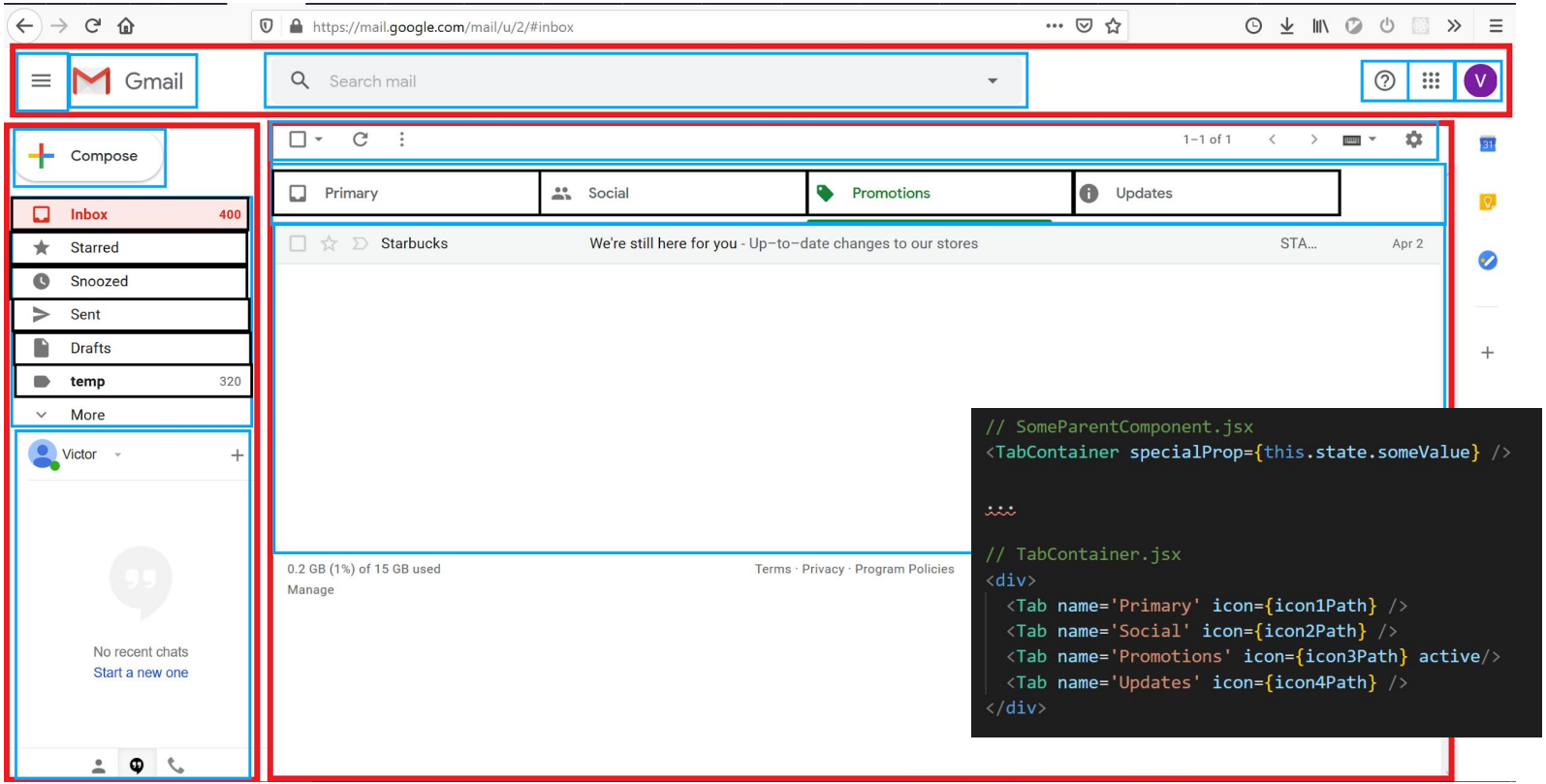
SPA Example


https://mail.google.com/mail/u/2/#inbox
Gmail
Search mail
Compose
Inbox 400
Starred
Snoozed
Sent
Drafts
temp 320
More
Victor
1–1 of 1
Primary
Social
Promotions
Updates
Starbucks
We're still here for you - Up–to–date changes to our stores
STA...
Apr 2
0.2 GB (1%) of 15 GB used
Manage
Terms · Privacy · Program Policies
Last account activity: 39 minutes ago
Details
No recent chats
Start a new one

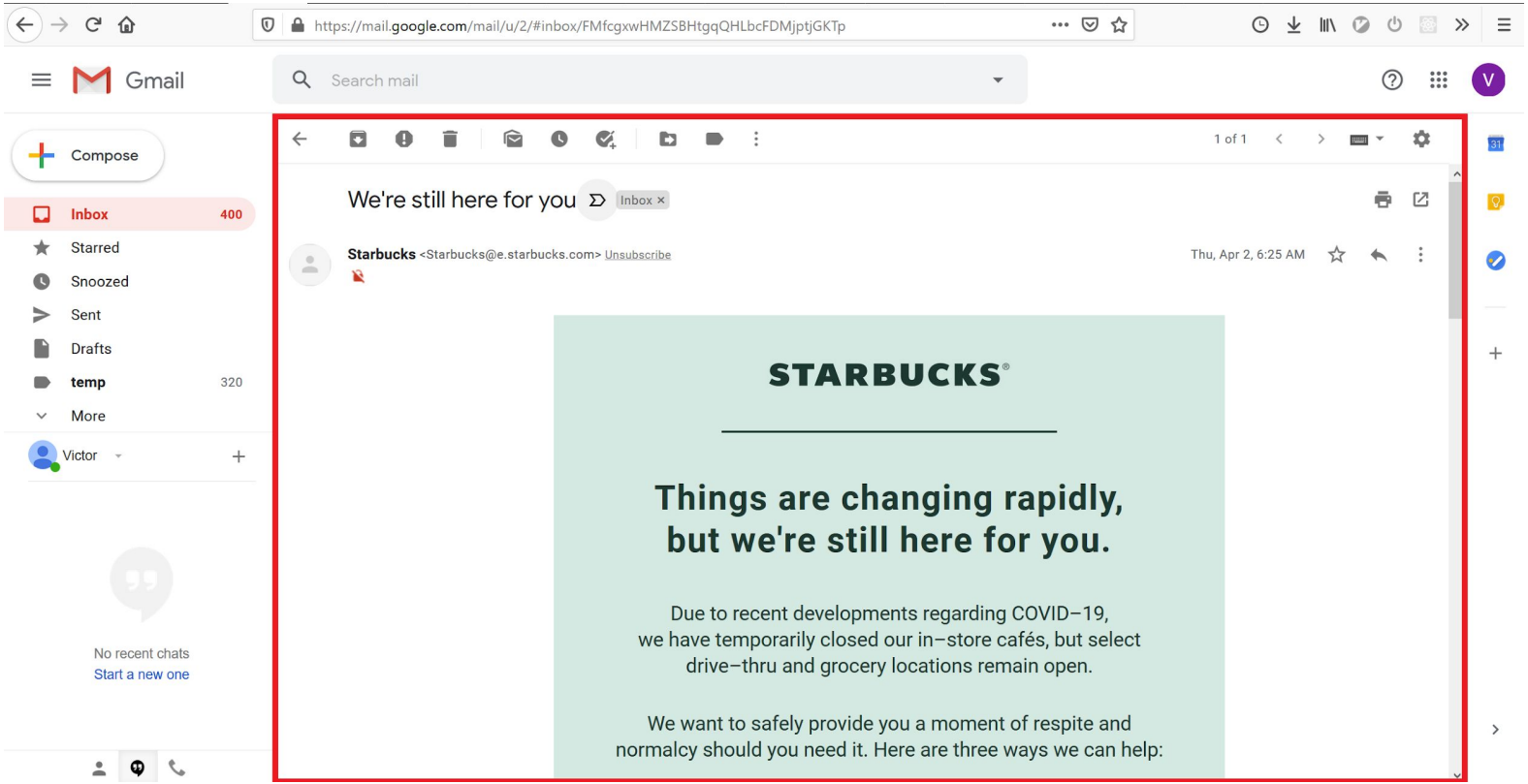
Views are composed of smaller building blocks/templates



Views are composed of smaller building blocks/templates



Everything is a component and can be switched out



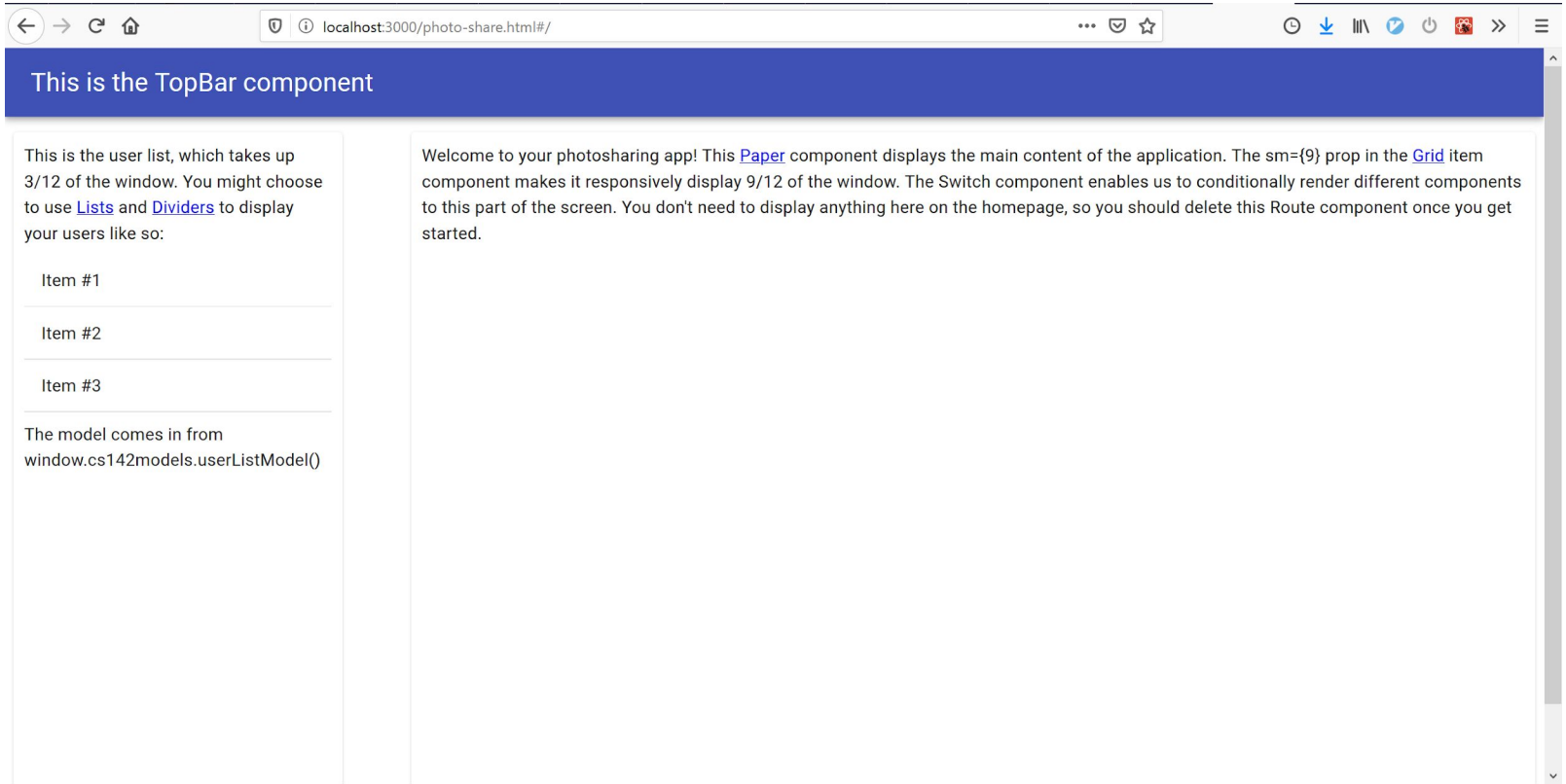
Problem 1

Goal

- Lay out the framework for our 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


localhost:3000/photo-share.html#/
This is the TopBar component
This is the user list, which takes up 3/12 of the window. You might choose to use Lists and Dividers to display your users like so:
Item #1
Item #2
Item #3
The model comes in from window.cs142models.userListModel()
Welcome to your photosharing app! This Paper component displays the main content of the application. The sm={9} prop in the Grid item component makes it responsively display 9/12 of the window. The Switch component enables us to conditionally render different components to this part of the screen. You don't need to display anything here on the homepage, so you should delete this Route component once you get started.

Data Structure - UserList

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

Data Structure - User

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

Data Structure - PhotoOfUser / Photo

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" **<- /images directory**
comments: [*Comment*, *Comment*]

modelData/photoApp.js

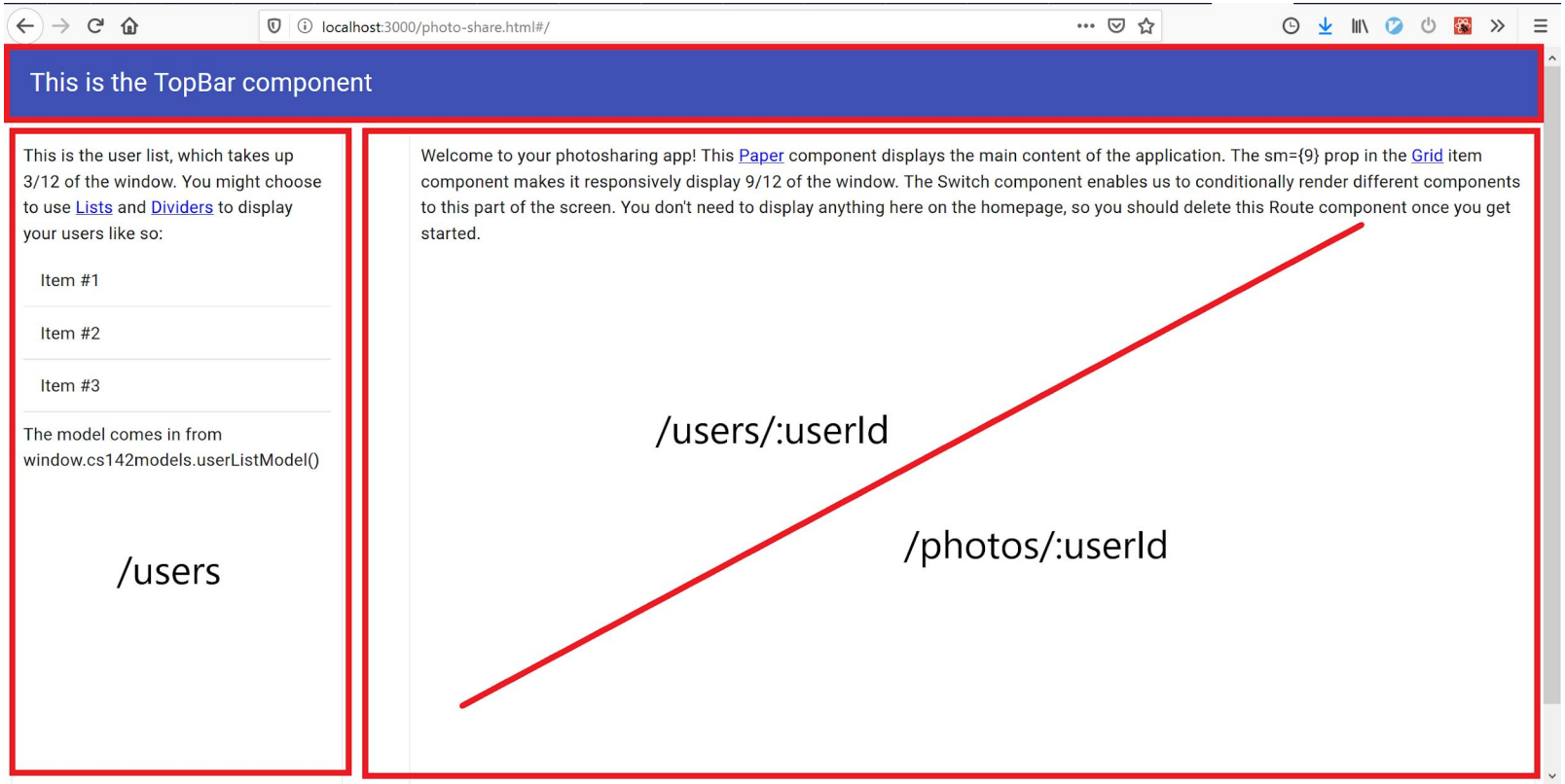
Data Structure - Comment

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


localhost:3000/photo-share.html#/
This is the TopBar component
This is the user list, which takes up 3/12 of the window. You might choose to use Lists and Dividers to display your users like so:
Item #1
Item #2
Item #3
The model comes in from window.cs142models.userListModel()
/users
Welcome to your photosharing app! This Paper component displays the main content of the application. The sm={9} prop in the Grid item component makes it responsively display 9/12 of the window. The Switch component enables us to conditionally render different components to this part of the screen. You don't need to display anything here on the homepage, so you should delete this Route component once you get started.
/users/:userId
/photos/:userId

Requirements - User List

`photo-share.html`

Your Name	
User A User B User C	

`components/userList.jsx`

Requirements - User List / User Detail

`photo-share.html`

Your Name	User A's Detail
User A User B User C	first_name last_name Photos

`/users/:userId`

`components/userDetail.jsx`

Requirements - User Detail

`photo-share.html`

Your Name	User A's Detail
User A User B User C	first_name last_name **Photos**

`/users/:userId`

`components/userDetail.jsx`

Requirements - User Photo / Comment

`photo-share.html`

Your Name	Photos of User A
User A User B User C	photo1.jpg User B: comment 1 photo2.jpg

`/photos/:userId`

`components/userPhotos.jsx`

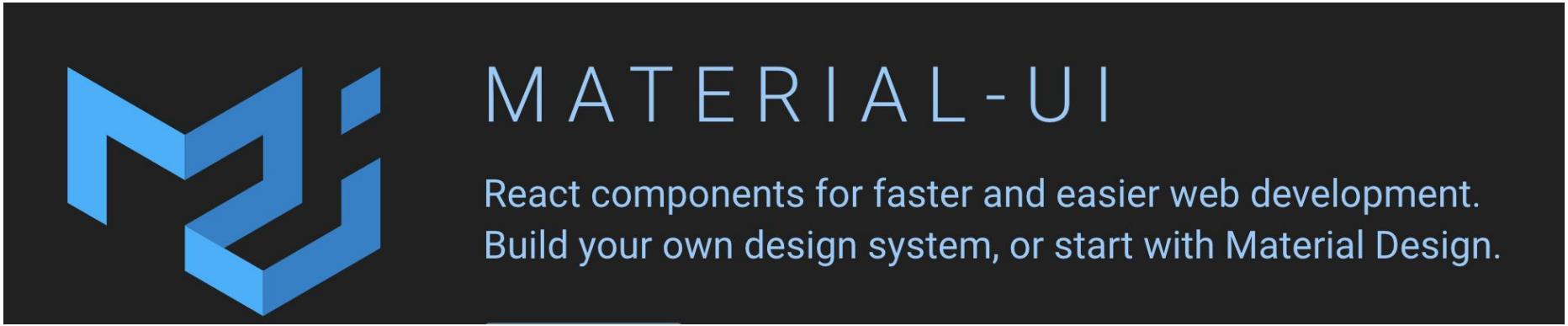
Passing information between components

- 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


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

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

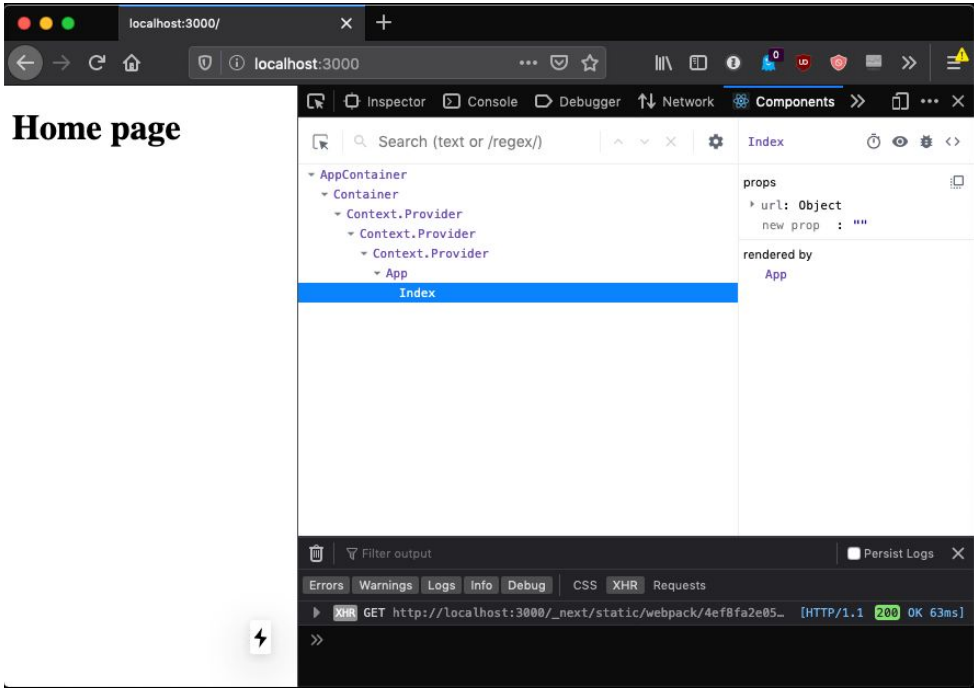
Look up the documentation /API to know how we can customize them

```
<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`), we fetch the model data from a server

- We will ultimately replace all calls to `window.cs142models` with calls to `fetchModel()`, which needs to be implemented in `/lib/fetchModelData.js`

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 (res) {
   // Do something
}
```

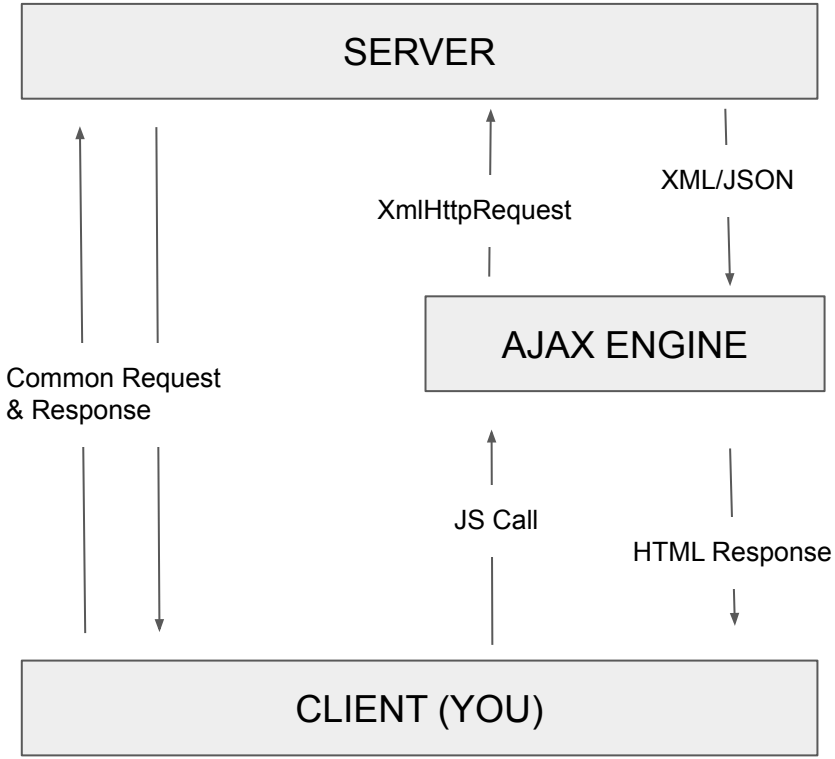
- Called with argument (model) when the GET request is done

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

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

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

Promise (Overview)

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

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

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

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

JSON (Javascript Object Notation)

- Server sends data back in the JSON format
- Need to parse it into a Javascript Object to have a 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 we 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 we learned in Homework 4!

Most of this assignment is applying concepts that we've already learned and used.

Details on React Material: https://mui.com/

More details on XHR requests:
https://web.stanford.edu/class/cs142/lectures/ServerCom.pdf