# METEOR FINAL EXAM REVIEW

## File Structure

- Eager loading
- Lazy loading
- Directory Names Importance
  - *imports, public, server, client, etc.*

## Imports, Exports

- Imports
  - *import './../imports/utils.js' ;*
  - *all imports need to start with ./ - this tells import that you are trying to import a local file as opposed to a third party dependency*

# Imports, Exports

## utils.js

```
export let sayHello = function () { return ‘hello! From imports/utils.js’; };
export let name = ‘michael whitney’;
```

## server/main.js

```
import { sayHello } from ‘./../imports/utils.js’;
console.log(sayHello);
```

## client/main.js

```
import { sayHello, name } from ‘./../imports/utils.js’;
console.log(sayHello);
console.log(name);
```

## React and JSX (javaScript XML)

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Meteor} from 'meteor/meteor';

Meteor.startup(function () { // This takes a function as its one and only argument
    let name = 'newman';
    let jsx = <p>Hello {name}!</p>// jsx gets converted
    ReactDOM.render(jsx, document.getElementById('content'));
});
```

## Render Static

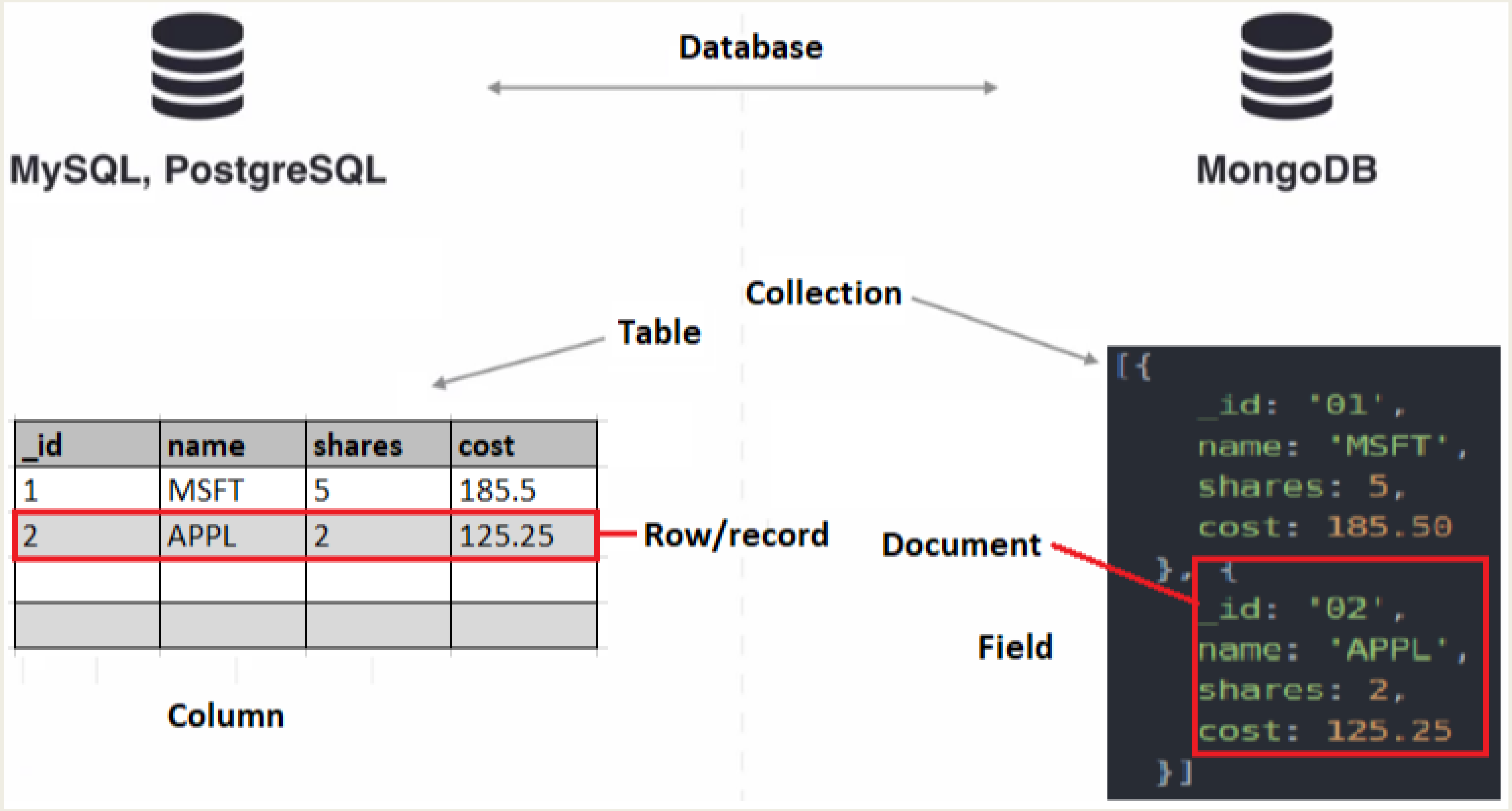
```
import React from 'react';
import ReactDOM from 'react-dom';
import {meteor} from 'meteor/meteor';

const renderCandidates = function (candidateObject) {
  let candidateInfo = candidateObject.map(function(candidate){
    return (
              <p key={candidate._id}>
                {candidate.name} has {candidate.vote} vote[s]
              </p>
             );
  });
  return candidateInfo;
};
```

```
Meteor.startup(function () {
 const candidates = [{
     _id: '01',
     name: 'pat,
     votes: 5,
   }, {
     _id: '02',
     name: 'chris',
     votes: 2,
 ];

  let title = 'The big election;
  let jsx = (
    <div>
      <h1>{title}</h1>
      {renderCandidates (candidates)}
    </div>
  );
  ReactDOM.render(jsx, document.getElementById('content'));
});
```

# Database Comparisons



| _id | name | shares | cost |
|---|---|---|---|
| 1 | MSFT | 5 | 185.5 |
| 2 | APPL | 2 | 125.25 |
| | | | |
| | | | |

```
[{
      _id: '01',
      name: 'MSFT',
      shares: 5,
      cost: 185.50
   }, {
      _id: '02',
      name: 'APPL',
      shares: 2,
      cost: 125.25
   }]
```

## MiniMongo

Meteor gets the database communication job done with **miniMongo**

- client side implementation of mongoDB
- has a set of functions that manipulate a set of arrays and objects designed to work identically to how mongoDB works
- difference is that it is in-memory written in javascript which means that instead of making calls to a db server, we use a synchronous set of functions to change the data behind the scenes

## DDP - Distributed Data Protocol

DDP: A client-server protocol for querying and updating a server-side database and for synchronizing such updates among clients. It uses the publish-subscribe messaging pattern.

publish–subscribe: a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize published messages into classes without knowledge of which subscribers, if any, there may be. Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are.

# Mongo: Query db and Insert

**imports/api/candidates.js**

```
import {Mongo} from 'meteor/mongo';

export const Candidates = new Mongo.Collection('candidates');
```

**server/main.js**

```
import {Meteor} from 'meteor/meteor';
import {Candidates} from './../imports/api/candidates.js';
Meteor.startup(function(){ //the function will run as soon as the server process is finished starting
    Candidates.insert({
        name: 'Pat',
        votes: 5,
    });
    console.log(Candidates.find().fetch());  // .find returns everything  .fetch returns an array
});
```

## update, remove, sort

```
<button onClick={() => { Candidates.update(_id: candidate._id, {$inc: {votes:1}})  }}>1</button>


<button onClick={() => { Candidates.remove(_id: candidate._id) }}>X</button>


let dbCandidates = Candidates.find({/*this empty obj gets all the candidates},
                                                    {sort: {votes: -1}} // this is the second argument
                                          ).fetch();
```

## Tracker

Meteor has a simple dependency tracking system which allows it to automatically rerun templates and other computations whenever Session variables, database queries, and other data sources change.

```
import {Meteor} from 'meteor/meteor';
import { Tracker } from 'meteor/tracker'
Meteor.startup(function(){
  Tracker.autorun(function(){
```

## Arrow Functions

In simple terms: An arrow function expression is a syntactically compact alternative to a regular function expression.

```
let add = function(y) {
        return y+y;
}

let square = (y) => y*y;
```

## React

In simple terms: A js library for building user interfaces. Helps with the creation of complex UIs with smaller pieces - Components

# React Components and Props

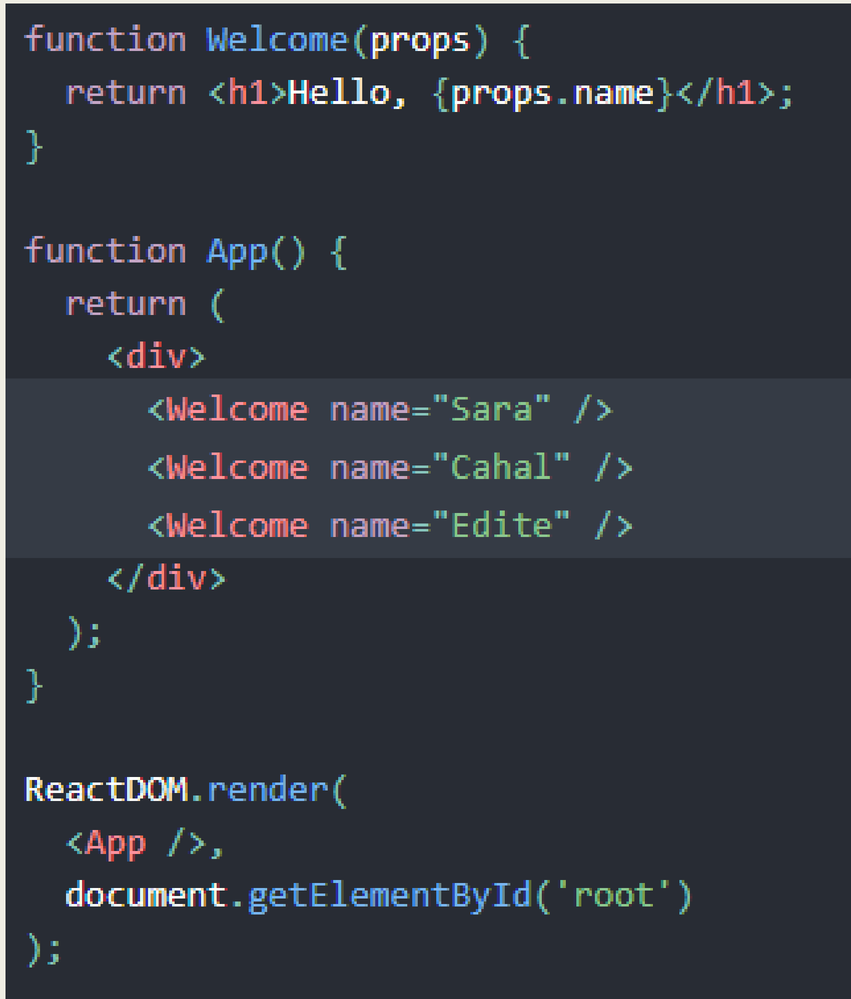
In simple terms: Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object “props”.

# React Components and Props

Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.

For example, we can create an App component that renders Welcome many times:



```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

# React Components and Props

**client/main.js**

```
import CampaignApp from './../imports/ui/CampaignApp.js';


let title = 'The big campaign';

ReactDOM.render(<CampaignApp passedPropTitle={title} document.getElementById('content'));
```

**imports/ui/CampaignApp.js**

```
import TitleBar from './TitleBar.js';

export default class CampaignApp extends
React.Component {

  render(){ // render is used to return jsx markup

   return (

     <div>

        <TitleBar title={this.props.passedPropTitle} />

    </div>

    ); }};
```

**imports/ui/CampaignApp.js**

```
export default class TitleBar extends React.Component {

render(){

    return (

      <div className="title-bar">

        <div className="wrapper">

          <h1>{this.props.title}</h1>        </div>

      </div>

    )

  }

};
```

## propTypes

In simple terms: You can use prop-types to document the intended types of properties passed to components.

```
TitleBar.propTypes = {title: PropTypes.string.isRequired};
```

```
//throws warnings in browser when undefined or not string

//useful if you didn't create the component and if you are tying to use it incorrectly
- do something with an array instead of a string
```

## ES6 Template Strings

In simple terms: designed to make it easy to inject values into a string

- will use the back tick which is to the left of the 1 key
- return 'hi, I am ' + this.name'; will work but template strings are better
- return `hi, I am ${this.name}`;          // this is a js expression

# ES6 Class declarations

In simple terms: To declare a class, you use the class keyword with the name of the class (" Person3 " here).

```
class Person3 {
        constructor(name='anonymous'){
                this.name = name;
        }
        getGreeting(){            // return custom greeting using their name
                return `hello ${this.name}`;      // this is a js expression
        }
 }
let me3 = new Person3();
console.log(me3.getGreeting());
```

## SCSS – Sassy CSS

In simple terms: SCSS is a preprocessor which lets you use features that aren't a part of the wider CSS standard yet, and provides better workflows for maintaining your stylesheets.

Basically the preprocessor takes the scss file with variables, nested selectors, and other features and converts them to css.

## SCSS – Sassy CSS

Similar to Components, it is all about the imports.

Must change client/main.css to main.scss

then going to push styles into imports directory

@import '../imports/client/styles/_main.scss';

_underscore indicates a partial which merely means it is part of the application's styles (there are more than one)

# imports/client/styles/_main.scss

```
* {                // simple reset of css
  margin: 0;
  padding: 0;
}
html {
  font-size: 62.5%;  // sets up base 10
}
body {
  background-color: lightGrey;
  color: grey;
  font-family: Helvetica, Arial, sans-serif;
}
h1 {
  font-weight: 300;
  font-size: 2.4rem;  // this is now 24 px
}
@import './components/_titleBar.scss';  //  ./ indicates to start in current folder
@import './components/_wrapper.scss';
@import './components/_item.scss';
```

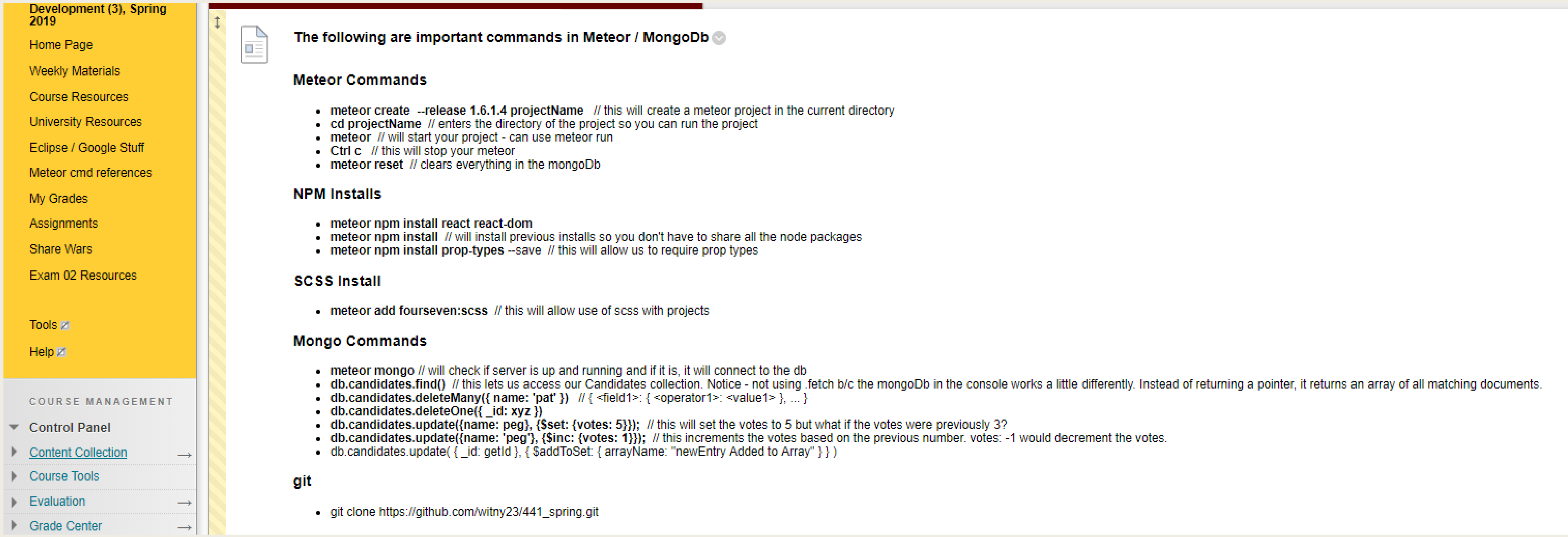
**imports/client/styles/components/_titleBar.scss**

```
.title-bar {
  background-color: grey;
  color: blue;
}
```

**imports/ui/TitleBar.js**

```
<div className="title-bar">
```

# Meteor References



**The following are important commands in Meteor / MongoDb**

**Meteor Commands**

- **meteor create --release 1.6.1.4 projectName** // this will create a meteor project in the current directory
- **cd projectName** // enters the directory of the project so you can run the project
- **meteor** // will start your project - can use meteor run
- **Ctrl c** // this will stop your meteor
- **meteor reset** // clears everything in the mongoDb

**NPM Installs**

- **meteor npm install react react-dom**
- **meteor npm install** // will install previous installs so you don't have to share all the node packages
- **meteor npm install prop-types** --save // this will allow us to require prop types

**SCSS Install**

- **meteor add fourseven:scss** // this will allow use of scss with projects

**Mongo Commands**

- **meteor mongo** // will check if server is up and running and if it is, it will connect to the db
- **db.candidates.find()** // this lets us access our Candidates collection. Notice - not using .fetch b/c the mongoDb in the console works a little differently. Instead of returning a pointer, it returns an array of all matching documents.
- **db.candidates.deleteMany({ name: 'pat' })** // { <field1>: { <operator1>: <value1> }, ... }
- **db.candidates.deleteOne({ _id: xyz })**
- **db.candidates.update({name: peg}, {$set: {votes: 5}});** // this will set the votes to 5 but what if the votes were previously 3?
- **db.candidates.update({name: 'peg'}, {$inc: {votes: 1}});** // this increments the votes based on the previous number. votes: -1 would decrement the votes.
- db.candidates.update( { _id: getId }, { $addToSet: { arrayName: "newEntry Added to Array" } } )

**git**

- git clone https://github.com/witny23/441_spring.git

## That's All Folks!!