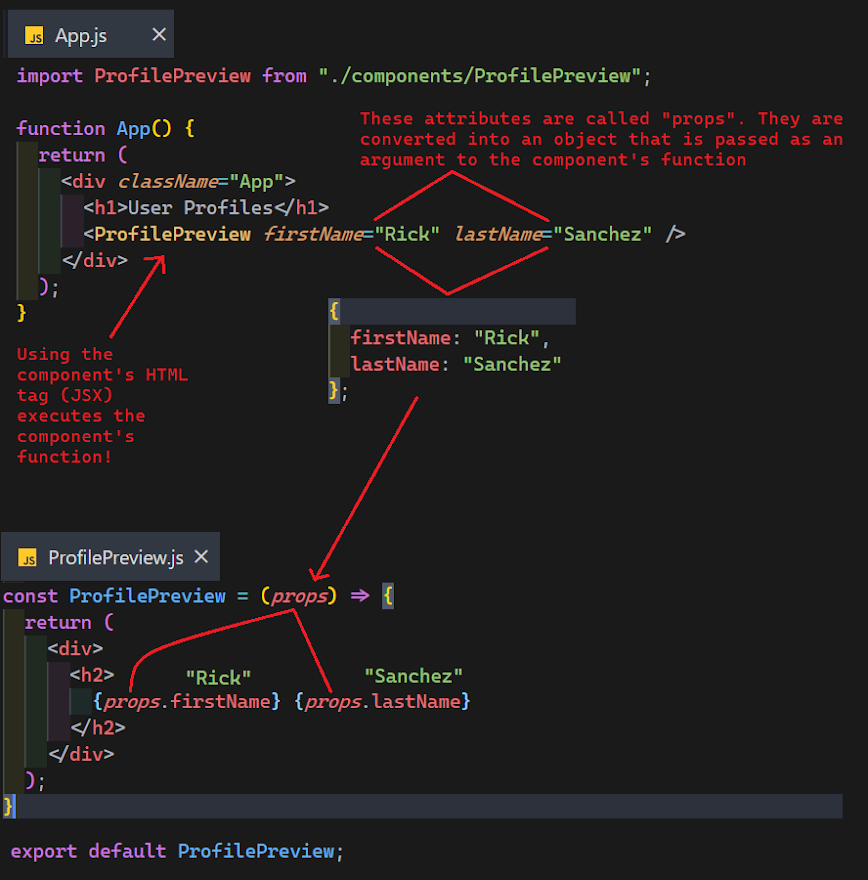
# MERN Cheat Sheet

* [Student Resources Folder](https://drive.google.com/drive/u/1/folders/11zE6KbvIdgr6_KtYGtMRNMja6Q41IAuo)

## Component Diagram



## Full Stack Project Setup

* **Replace {{TEXT}} with your own text**

1. Open terminal to where you want your project to be created
2. **mkdir {{PROJECT\_NAME}}**
3. **cd {{PROJECT\_NAME}}**
4. **code .** to open VSCode to **PROJECT\_NAME** folder
5. In VSCode terminal: **npm init -y** to initialize a new EMPTY back-end node project (your server) with a **package.json**

### Install back-end dependencies

* 1. **npm i express mongoose cors**
  2. Your back-end **package.json** should now show have these dependencies listed

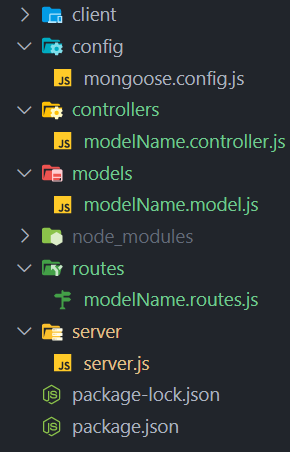
1. Create a **server** folder that we will add to later
2. In VSCode terminal
   1. **npx create-react-app client**
   2. This will create a **client** folder for your front-end react app
3. Open a 2nd VSCode terminal 
   1. **cd client** in the new terminal when the folder has been created

### Install front-end dependencies

* 1. **npm i axios @reach/router**
  2. Your front-end **package.json** that is inside **client** folder should now have these dependencies listed

1. Create the modularized back-end folder structure and files in the **server** folder following the learn platform or lecture video / lecture code
2. **cd server** in the **other** VSCode terminal and run the server (see server.js and mongoose.config.js code below)
   1. **nodemon server.js**
3. **Test all back-end CRUD functionality using postman before you start writing components**
   1. Including sending in invalid lengths or missing keys that are required
4. From **client** folder run react app with **npm start**
   1. You should have two VSCode terminals open, one open to **client** folder running the react app and one open to **server** folder running your **server.js**

## Folder Structure



## **Server.js**

**const express = require("express");**

**const cors = require("cors");**

**const express = require("express");**

**const cors = require("cors");**

***// Environment vars.***

**const port = 8000;**

**const db\_name = "NAME\_YOUR\_DATABASE";**

***// Immediately execute the import mongoose.config.js function.***

**require("../config/mongoose.config")(db\_name);**

**const app = express();**

***// req.body undefined without this!***

**app.use(express.json());**

**app.use(cors());**

**require("../routes/message.routes")(app);**

**app.listen(port, () =>**

**console.log(`Listening on port ${port} for REQuests to RESpond to.`)**

**);**

## 

## 

## 

## **mongoose.config.js**

**const mongoose = require("mongoose");**

***// Export a function to be called in server.js***

**module.exports = (*db\_name*) => {**

**mongoose**

**.connect(`mongodb://localhost/${*db\_name*}`, {**

**useNewUrlParser: true,**

**useUnifiedTopology: true,**

**useFindAndModify: false,**

**})**

**.then(() => {**

**console.log(`Successfully connected to ${*db\_name*}`);**

**})**

**.catch((*err*) => {**

**console.log(`mongoose connection to ${*db\_name*} failed:`, *err*);**

**});**

**};**

## Bootstrap (optional)

* React bootstrap that uses components instead of the class names primarily is technically better, but if you want to just get quick access to the bootstrap class names you can use this.

1. npm install bootstrap
2. in /src/index.js: import 'bootstrap/dist/css/bootstrap.css';

## Full Stack MERN Routing *Request* *Response* Cycle Walkthrough

1. React routing triggered from address bar, click on **<Link>** component, or **navigate("url")** method which comes from **@reach/router**
2. Component is mounted to the DOM to be rendered
3. Component sends API ***request*** from **useEffect**
4. Component renders while waiting for ***response***
   1. This is why you may need to use a condition for rendering before the response has come back
5. Back-end routes file then routes ***request*** to
6. Controller method executed
7. Controller method sends DB query
8. Controller method receives DB ***response*** in **.then**
9. **res.json(dbResponseData);** - ***response***  sent to client (front-end)
10. Component's state updates in the **.then** passing the ***response*** data to a set state function that came from **useState**
11. Component re-renders due to state being updated

## Backend How It’s Connected

* Starting point: server.js This is the file that is actually executed, it connects everything else, the lines are run top to bottom.

1. **const express = require("express");**
   1. Gets a reference to express from **node\_modules**
2. **require("./config/mongoose.config");**
   1. this triggers the code in **mongoose.config.js** to be executed which configures and connects **mongoose**
3. **const app = express();**
4. **require** our **app**'s routes: **require("./routes/city.routes")(app);**
   1. This syntax invokes the function that is exported from this **require** and passes it our **app**
      1. **but, before the exported function is executed, the below steps happen**
5. modelName.routes.js runs **require("../controllers/modelName.controller");**
6. modelName.controller.js runs **require("../models/modelName.model");**
7. modelName.model.js runs **const mongoose = require("mongoose");** to get a reference to **mongoose** from node\_modules
   1. **mongoose** has already been connected and configured from step 2
   2. **const City = mongoose.model("City", CitySchema);** registers schema with **mongoose**
   3. The reference to the mongoose model is exported to the file that required our model: **module.exports = City;**
8. modelName.controller.js has now received the exported mongoose model from modelName.model.js
9. modelName.controller.js now exports an object full of methods
10. modelName.routes.js has now received the exported object from modelName.controller.js
    1. methods from the exported object can now be referenced in modelName.routes.js
11. modelName.routes.js now **exports** a function
12. **server.js** now receives the exported function from modelName.routes.js and executes it immediately
13. Everything is now set up, time to

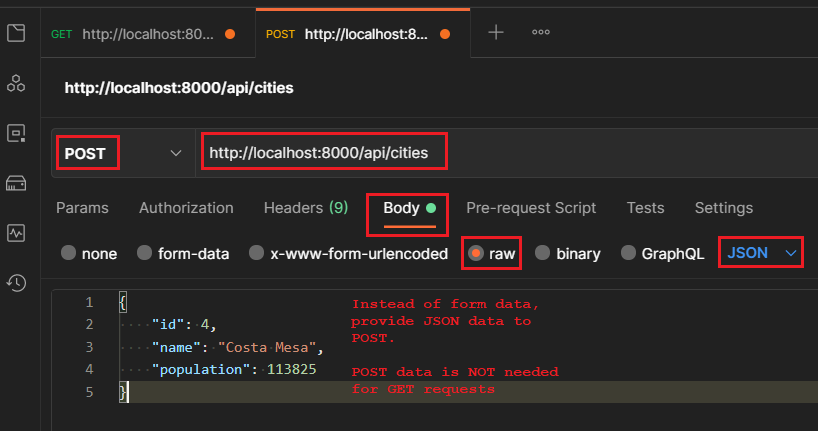
**app.listen(port, () => {**

**console.log(Listening on port ${port} for REQuests to RESpond to.);**

**});**

## 

## Postman

* When you have a back end, every single route should be tested in postman to verify it is 100% working BEFORE you start building the front end to make it easier to know that your future bugs are not from your back end.
* 

## [Troubleshooting](https://docs.google.com/document/d/1F0eILkNf8MeNcXusx0AMcptRdbf_DFXaWMxgzezjUZI/edit?usp=sharing)

## [How It’s Connected](https://docs.google.com/document/d/1Eo1wjO1HFtBROKd3luF5ynSpHNhYuyIgfyZtEUcMusQ/edit?usp=sharing)

## [Other MERN Resources](https://docs.google.com/document/d/1NLqRkvKGW2HJg5OLJz42cuChDS8hoCYmhGfbflOORAc/edit?usp=sharing)