

# REACT AND EXPRESS BOILERPLATE

BY QCC - TECH WORKS

---

## GOAL

- ▶ familiarize you with the installation packages
- ▶ How to define scripts in package.json
- ▶ Setting up a express server
- ▶ Integrating react with express
- ▶ Creating a boilerplate



# WE HAVE AN IDEA!!

- ▶ What do we do first when we have an idea for an application??? We create a directory that we're going to place this application in.
- ▶ Then we create a GitHub repo so that we can push this to the cloud, so we can reuse our boilerplate.

- ▶ Once we created this folder open it up in VS code or Atom. It should be an empty directory that we haven't installed anything in yet.
- ▶ We want to initialize this project by running the command `npm init` in our boilerplate directory. This will open the utility in the command prompt that will allow us to create a `package.json` file that we will list our dependencies in.
- ▶ When the prompt asks for the description you can add "React and Express Boilerplate" and when it asks for entry point - lets add in `server.js`

- ▶ When you're completed with the "npm init" command prompt. Your package.json file should look like:

```
1  {
2    "name": "reactandexpress",
3    "version": "1.0.0",
4    "description": "",
5    "main": "server.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Kirk Sullivan",
10   "license": "MIT"
11 }
```

Your name should be in the author.

- ▶ Now we want to install some dependencies - first we want to install express. Lets run the command `"npm install express"` to grab the express package. We also want the package called concurrently. So we run the command `"npm install concurrently"`. And last we also want nodemon package - So we run the command `"npm install nodemon --save-dev"` this will save it as a development dependency that will only be used while our app is in development, and not added to the bundle once we push it into production. Also lets add in the morgan package to grab run the command `"npm install morgan"`

- ▶ If all is done correctly. Your package.json file should look like:

```
1 {  
2   "name": "reactandexpress",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "server.js",  
6   "scripts": {  
7     "start": "node server.js",  
8     "server": "nodemon server.js"  
9   },  
10  "author": "Kirk Sullivan",  
11  "license": "MIT",  
12  "dependencies": {  
13    "concurrently": "^4.1.1",  
14    "express": "^4.17.1",  
15    "morgan": "^1.9.1"  
16  },  
17  "devDependencies": {  
18    "nodemon": "^1.19.1"  
19  }  
20 }
```

- ▶ Now we want to add some specific scripts into our package.json file. Inside the scripts object. We want to delete the test script, and replace it with a "start" key and have the value be "node server.js".
- ▶ Then we want to add another script called "server" and have the value be "nodemon server.js"

```
6  . "scripts": {  
7  . |   "start": "node server.js",  
8  . |   "server": "nodemon server.js"  
9  . },
```



- ▶ Now that we have our package.json file configured we need to create this server.js file and add some code.
- ▶ What we need to do first is start configuring this file to use the express package so we can create a server.

```
1  const express = require('express');
```

- ▶ We define our app variable that will listen on a specific port we define. Make sure this port is not 3000... because that's the port that the create-react-app package uses.

```
1  const express = require('express');  
2  const app = express();  
3  const morgan = require('morgan')  
4  const port = 4000;  
5  
6  app.listen(port, () => {  
7    console.log(`Server Started on Port ${port}`);  
8  });  
9
```

- ▶ Now let's put some dummy data in for a route to make sure our express server is running and serving data correctly.

```
10 app.get('/api/students', (req, res) => {  
11   ··const students = [  
12   ··  { id: 1, firstName: 'Captain', lastName: 'fancy' },  
13   ··  { id: 2, firstName: 'John', lastName: 'buttercup' },  
14   ··  { id: 3, firstName: 'Dusty', lastName: 'Trail' },  
15   ··];  
16   ··res.json(students);  
17   ··});
```

- ▶ Now lets start our server by running the command we defined in our package.json that will keep the server running.
- ▶ We do this by making sure we are in our boilerplate directory in the command prompt and run the command **npm run server**

```
$ npm run server
> reactandexpress@1.0.0 server /Users/kirksullivan/JavaScript/teachingMaterials/reactandexpress
> nodemon server.js

[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
body-parser deprecated undefined extended: provide extended option server.js:8:17
Server Started on Port 4000
```

- ▶ Now let's add our middleware

```
6  //Middleware
7  app.use(morgan("dev"));
8  app.use(express.json());
9  app.use(express.urlencoded({extended: true}));
```

- ▶ Okay... Its time for some react! YAYAYAY!!!
- ▶ What we need to do is go into our command prompt and make sure we are in our boilerplate directory. Then we need to run the "`npx create-react-app client`" command so we can install the create-react-package

- ▶ Once this is done... open your client/package.json file.
- ▶ We need to define our backend url as a proxy - then save and close the file.

```
25     "development": [  
26       "last 1 chrome version",  
27       "last 1 firefox version",  
28       "last 1 safari version"  
29     ],  
30   },  
31   "proxy": "http://localhost:4000"  
32 }
```

- ▶ Time to start our front end server!
- ▶ Be sure to navigate to your boilerplate directory then change directory into the client folder and then run the command "`npm start`"
- ▶ Now you should see the lovely create-react-app boilerplate running.



- ▶ Time to create a component that will handle our students.
- ▶ In our client/src folder we want to create a new folder that will contain our student component.
- ▶ Create a folder under the src called Students
- ▶ Then create a file called Students.js
- ▶ This is a stateful class component that we will use to create state that will represent the students in an array and will use a lifecycle method to fetch the data.

- ▶ Now lets bring in our component into our main App.js file
- ▶ You can simply replace the `<p>` tags and everything in between and add the Student component

```
6  function App() {
7    return (
8      <div className="App">
9        <header className="App-header">
10         <img src={logo} className="App-logo" alt="logo" />
11         <Students />
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Learn React
19         </a>
20       </header>
21     </div>
```

- ▶ Now we have to add state and make a api request in our students.js file so we can grab the data from our back end and populate a list, by using our friend fetch. Copy this

```
11  .. async componentDidMount() {  
12  ..    .. const res = await fetch('/api/students');  
13  ..    .. const data = await res.json();  
14  ..    .. this.setState({  
15  ..    ..   .. students: data  
16  ..    .. })  
17  .. }
```

- ▶ Now build out your student component with a list of the data we're pulling to get your boilerplate completed. Your app should look like this when finished.



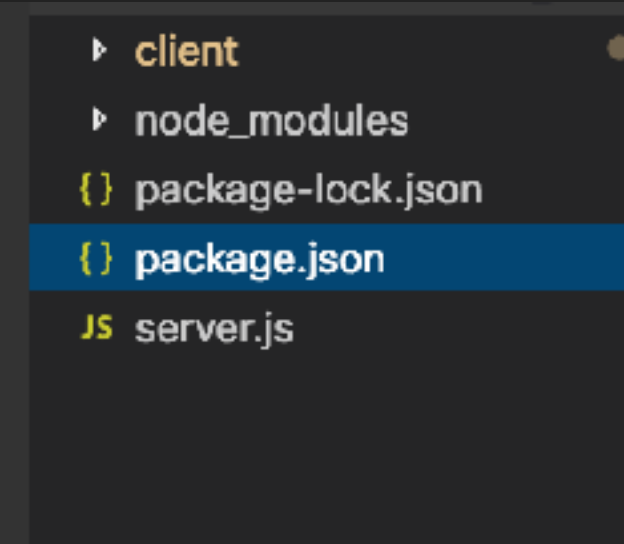
- Captain
- John
- Dusty

[Learn React](#)

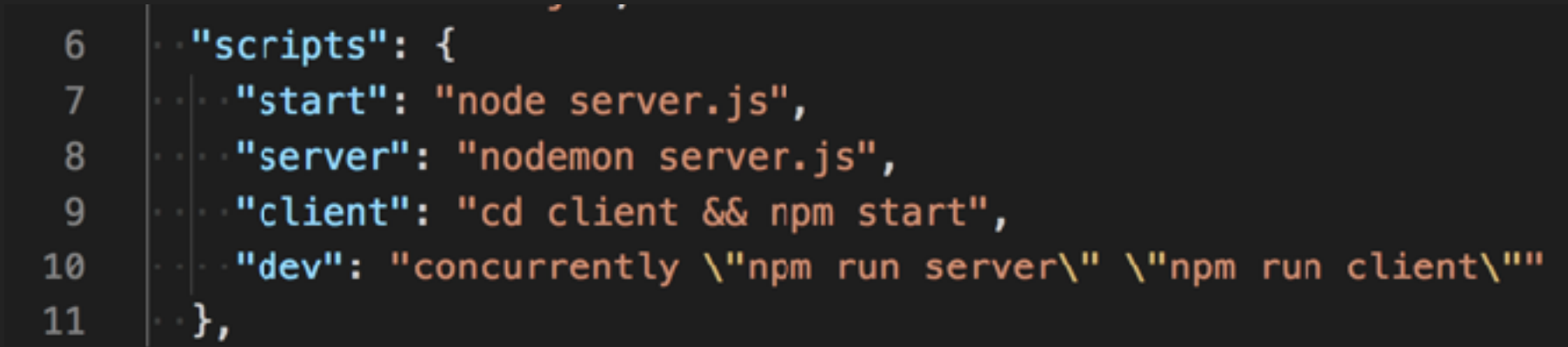
## REACT AND EXPRESS BOILERPLATE

---

- ▶ Now we want to make sure that both servers are running concurrently. Lets navigate to our package.json that is linked to our backend. What we want to add is a client script that will start our front end server and a dev script that will run both these servers concurrently.



```
▶ client
▶ node_modules
{} package-lock.json
{} package.json
JS server.js
```



```
6  .. "scripts": {
7    .. "start": "node server.js",
8    .. "server": "nodemon server.js",
9    .. "client": "cd client && npm start",
10   .. "dev": "concurrently \"npm run server\" \"npm run client\""
11   },
```

Now when you “npm run dev” from the main directory - both servers will run and you don’t need two console windows to handle both servers

- ▶ Now push this all to GitHub when you get it working correctly. Feel free to make customizations that make sense for you. Now when you have an idea for a React and Express application that you built from the ground up - you can just clone down this repo into a new project and have a great place to start!