AMRITA VISHWA VIDYAPEETHAM

15CSE337
Cloud Computing and Services

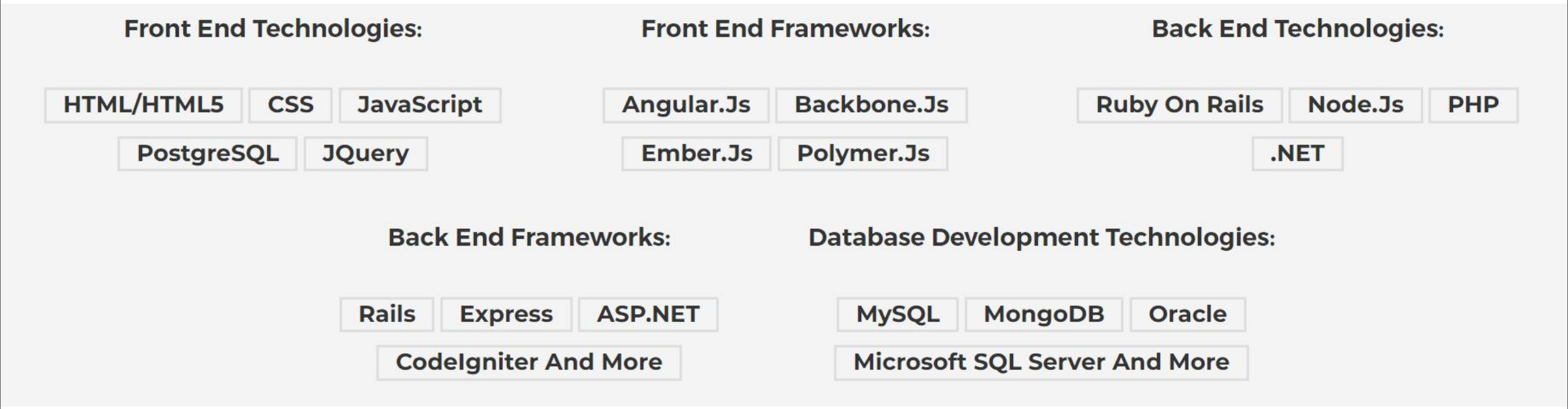
Introduction to stack – FULL vs MEAN vs MERN
Client side scripting - React

Dr Ganesh Neelakanta Iyer
Associate Professor, Dept of Computer Science and Engg
Amrita Vishwa Vidyapeetham, Coimbatore


What is a
stack?

Stack

- A stack is simply the different technologies being used to accomplish that goal

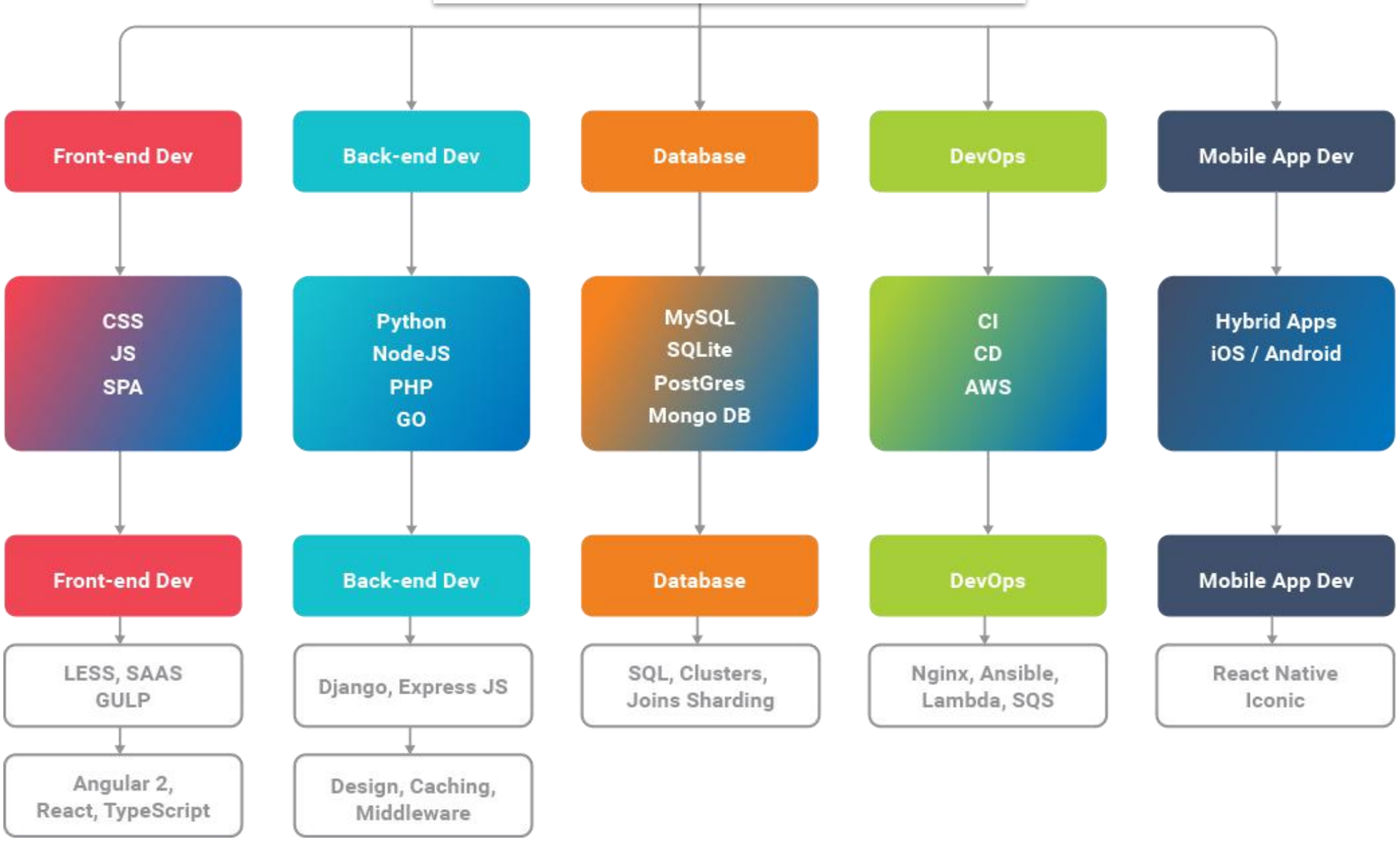


https://medium.com/datadriveninvestor/most-popular-technology-stack-to-choose-from-full-stack-vs-mean-stack-vs-mern-stack-in-2019-d12c0a17439a

Full Stack Development

- A full stack developer has the functional knowledge and the ability to work on all aspects involved in building an application. He is proficient in
  - Writing optimized front-end code in HTML, Java, JavaScript
  - Creating and using APIs and writing backend code in Ruby, Python/Java
  - Working with system infrastructure including hardware and OS
  - Networking, Security
  - Understanding, creating and querying databases
  - Project management and Client coordination

Full Stack Web Development



MEAN STACK

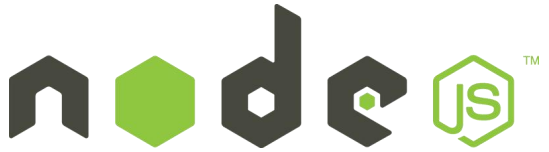
- The Friendly & Fun Fullstack JavaScript framework

MongoDB is the leading NoSQL database, empowering businesses to be more agile and scalable

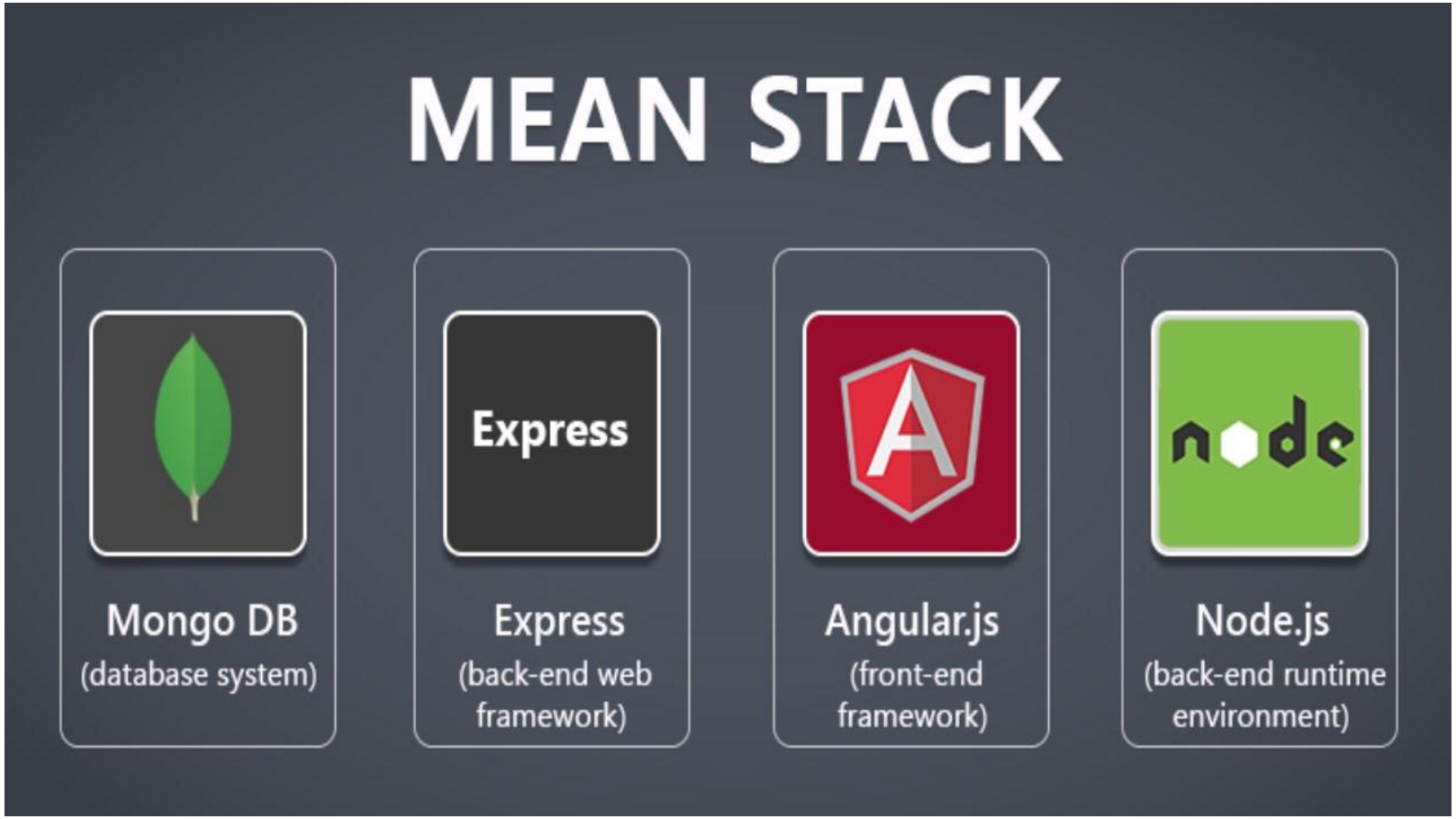
Express is a minimal and flexible node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications

AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.

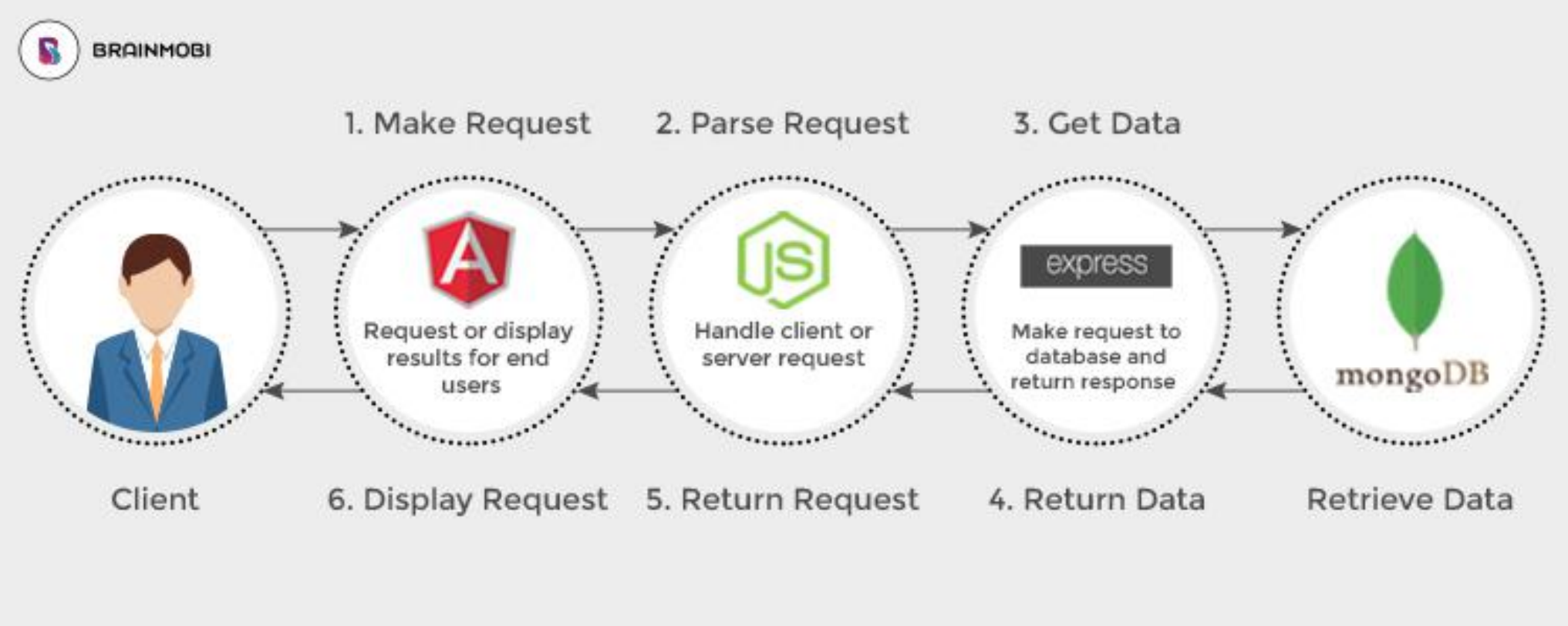
Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications



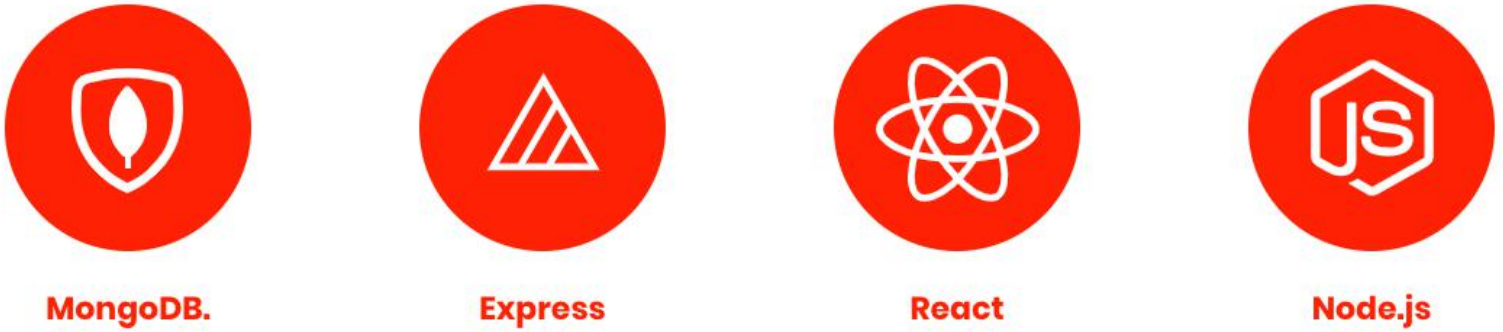
MEAN STACK


MEAN STACK
Mongo DB
(database system)
Express
Express
(back-end web framework)
Angular.js
(front-end framework)
node
Node.js
(back-end runtime environment)

MEAN STACK – HOW IT WORKS

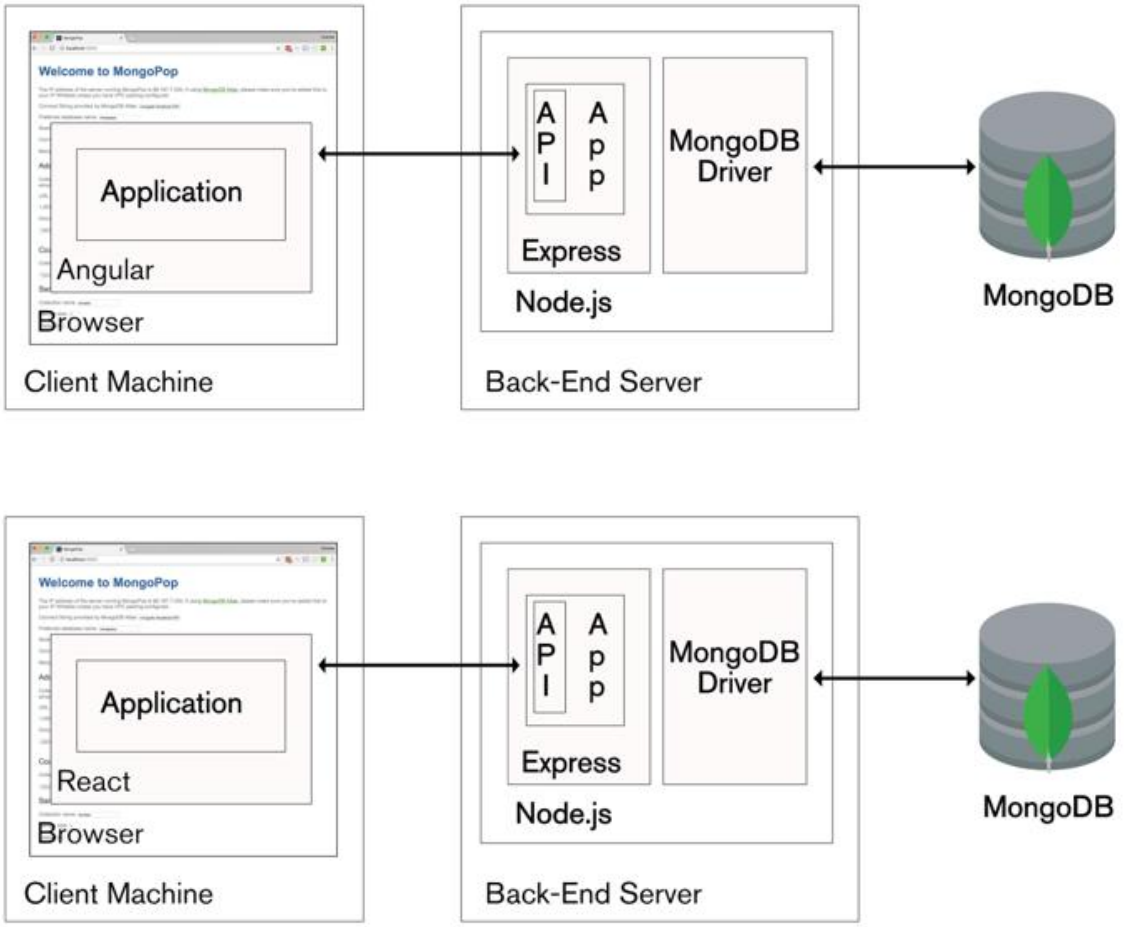


MERN Stack



- The MERN stack is very similar to MEAN stack
- The only difference here is that the MEAN stack is making use of Angular to build the front-end web application but in the MERN stack is using React instead

MEAN vs MERN



- An alternative to Angular is React (sometimes referred to as ReactJS), a JavaScript library developed by Facebook to build interactive/reactive user interfaces

https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack

Angular vs React

When to Choose React

- You know HTML, CSS and JavaScript.
- You need a highly customized specific app solution.
- You need an app with multiple events.
- You create shareable components in your app project.
- You wish to spend time on pre-development preparation

When to Choose Angular

- You know Java, C# and previous versions of Angular.
- You need a large-scale feature-rich application.
- You embrace ready-to-use solutions and need higher productivity.
- You wish to regulate app size.
- App complexity lies from Low to Medium level.

https://appinventiv.com/blog/react-vs-angular/

REACT	FACTOR	ANGULAR
2013	Release Date	2016
Facebook	Support from	Google
Javascript	Programming Language	Typescript
React 16.7.0	Latest Stable Version	Angular 7.2.0
Moderate	Learning Curve	Steep
Component Based	Architecture	Component Based
Relatively Small	App Size	Relatively Small
Virtual DOM	DOM	Real DOM
Client/Server Side	Rendering	Client/Server Side
JSX+JS (ES5 and Beyond)	Template	HTML+Typescript
Unidirectional (One-way)	Data Binding	Bidirectional (Two-way)
Medium	Abstraction	Strong

Our choice in this subject

- If you want to build an app with powerful UI, then React would be a better choice for you, and if you really want to build a complex web front-end and need a single modular framework to handle everything, then AngularJS will be an ideal option for you.
- We have decided to go with React (MERN stack) for rest of this course as the JavaScript Native Application Development


Introduction to
React

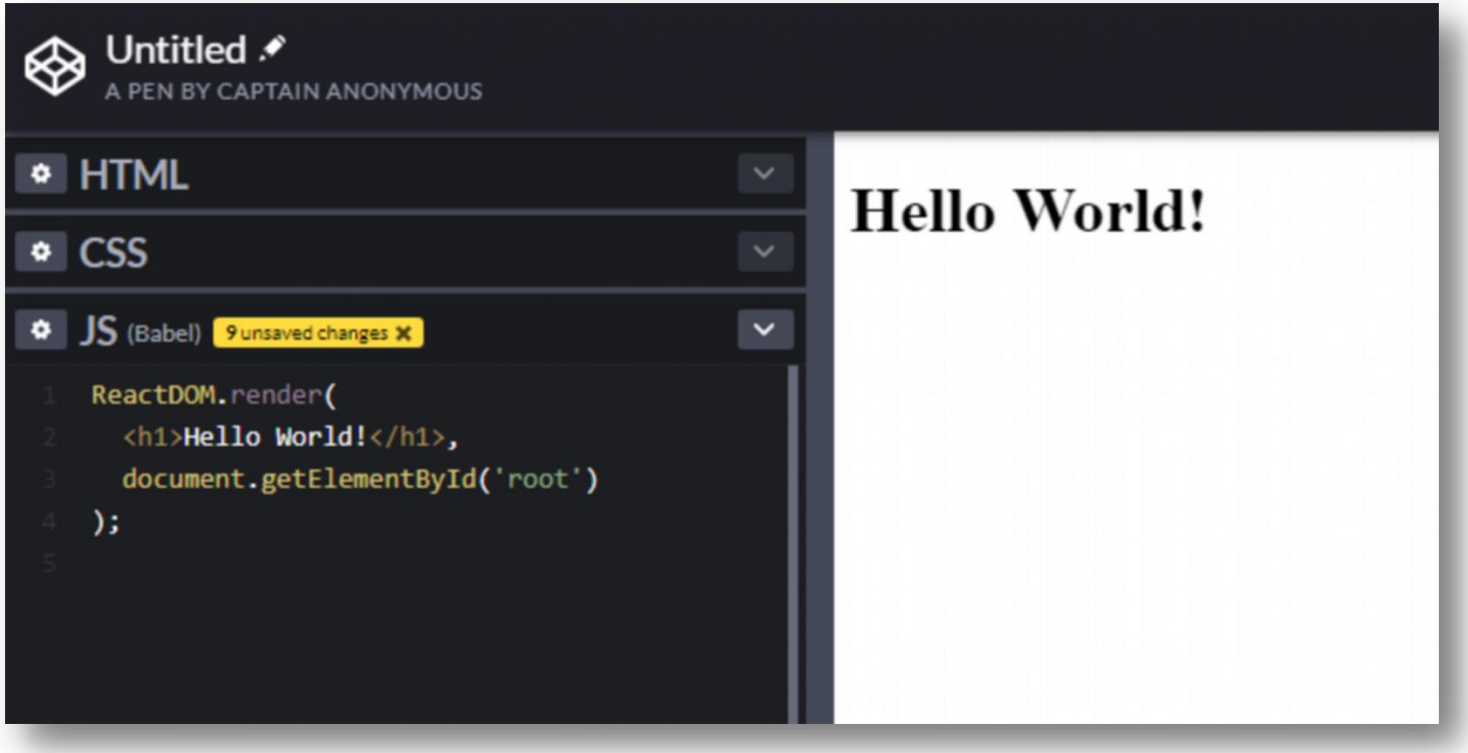
React

- A JavaScript library for building user interfaces
- React is an open-source project created by Facebook.
- React is the view layer of an MVC application (Model View Controller)
- React is a declarative, efficient, and flexible JavaScript library for building user interfaces
- It lets you compose complex UIs from small and isolated pieces of code called “components”.

Hello World

```
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

- Try your code online and see the output
- https://codepen.io/pen/



Getting Started

- The easiest way to create a React application is to use *create-react-app*
- This is a command-line interface (CLI) which allows you to quickly scaffold a React application
- It includes all of the development and workspace configuration so you can easily get coding

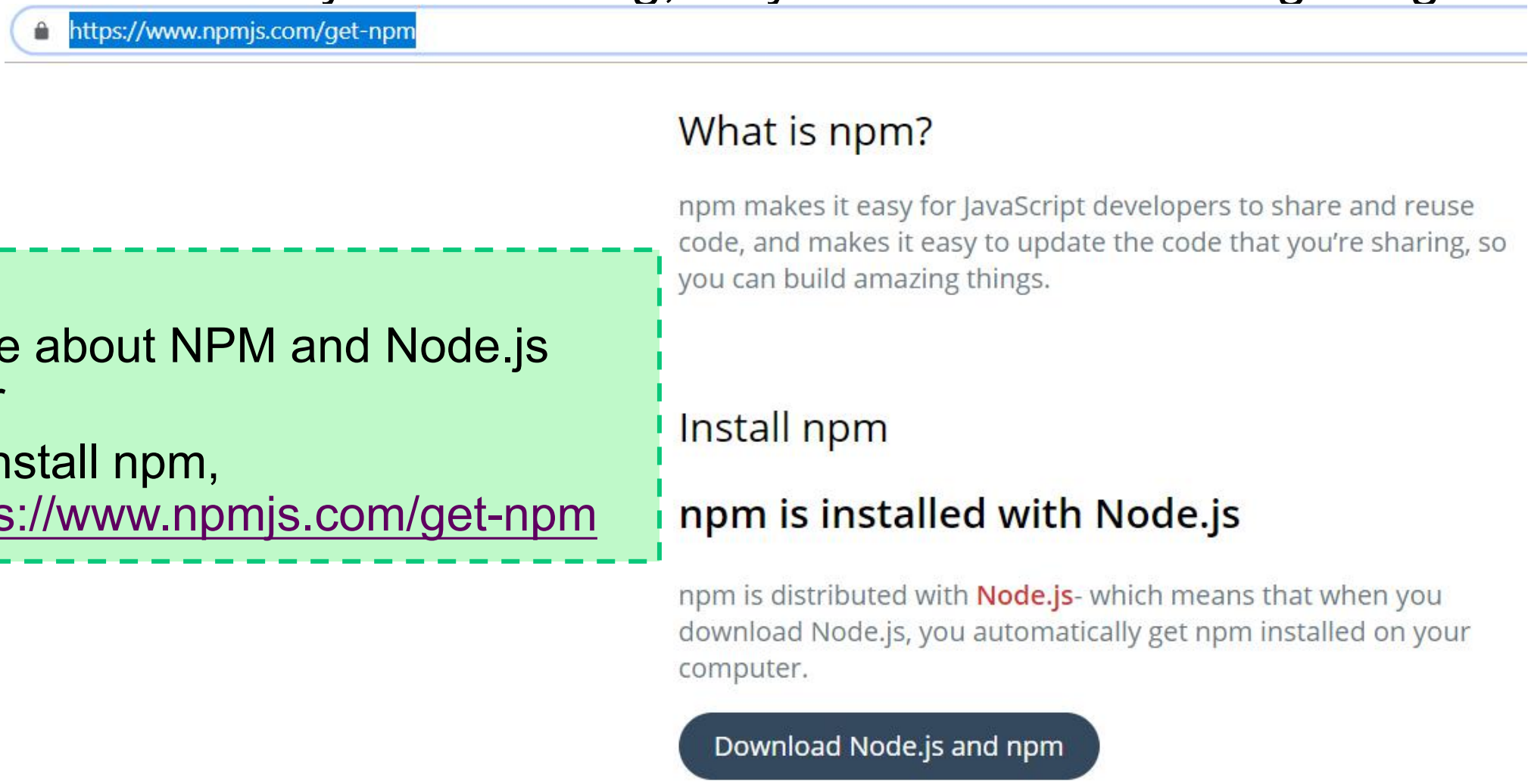
Installation and setup

- In order to use the React CLI, you have to first globally install it
- You can use npm or Node Package Manager to run the installation. If you have npm 5.2.+ you can use the following commands

- What is NPM? How do I set it up?

NPM

npm makes it easy for JavaScript developers to share and reuse code, and makes it easy to update the code that you're sharing, so you can build amazing things



More about NPM and Node.js later

To install npm, https://www.npmjs.com/get-npm

Create your first React App

- Create A React Application In A New Folder

```
npx create-react-app my-app
```

```
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...


> core-js@2.6.9 postinstall D:\Dropbox\Research\AMRITA\Course\Cloud computing\20
19\React\my-app\node_modules\babel-runtime\node_modules\core-js
> node scripts/postinstall || echo "ignore"


> core-js-pure@3.1.4 postinstall D:\Dropbox\Research\AMRITA\Course\Cloud computi
ng\2019\React\my-app\node_modules\core-js-pure
> node scripts/postinstall || echo "ignore"

+ react-dom@16.8.6
+ react@16.8.6
+ react-scripts@3.0.1
added 1402 packages from 727 contributors and audited 888986 packages in 197.913
s
found 0 vulnerabilities


Initialized a git repository.
```

Installation and Setup

```
Initialized a git repository.

Success! Created my-app at D:\Dropbox\Research\AMRITA\Course\Cloud computing\201
9\React\my-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd my-app
  npm start

Happy hacking!
```

Create your first React App

```
cd my-app
npm start
```

```
GiRi@GiRi-IyEr MINGW64 /d/Dropbox/Research/AMRITA/Course/Cloud computing/2019/React/my-app (master)
$ npm start

> my-app@0.1.0 start D:\Dropbox\Research\AMRITA\Course\Cloud computing\2019\React\my-app
> react-scripts start

Starting the development server...

Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://192.168.56.1:3000/

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Test your first app

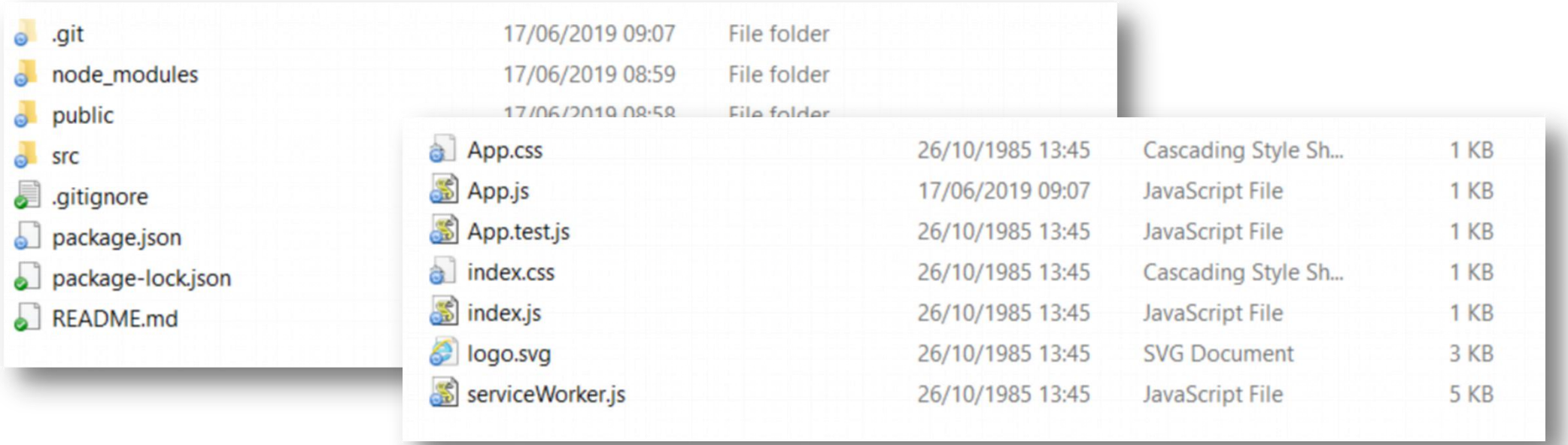
- http://localhost:3000/
- Can you get your name there?

Did you notice?

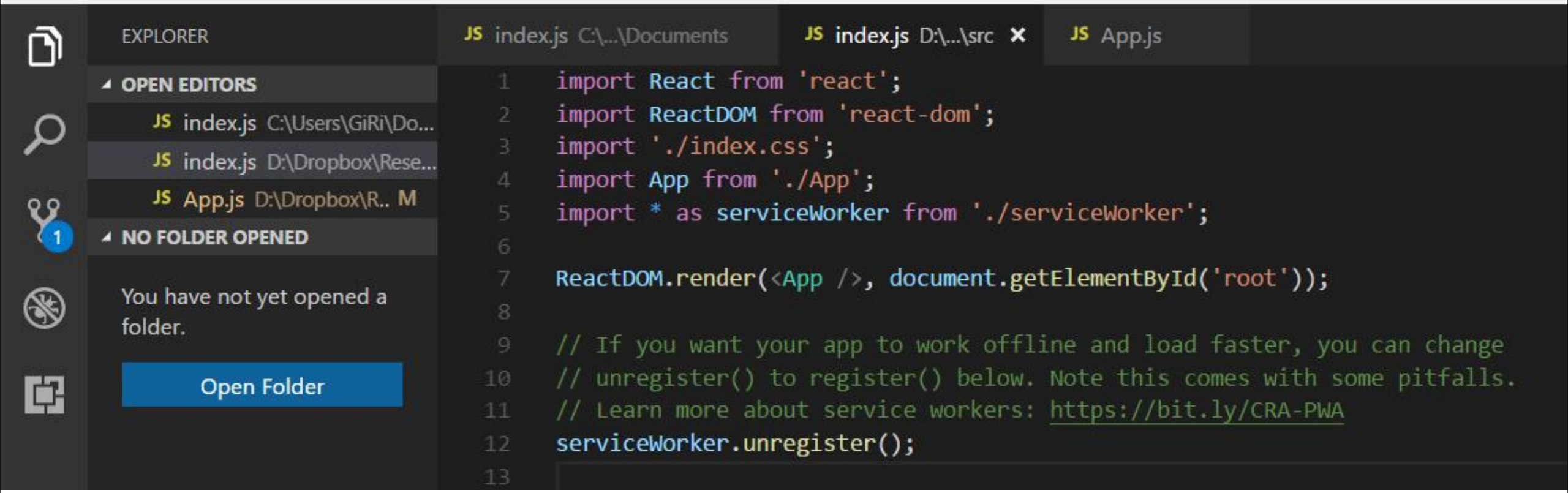
- When edits are made in the app.js file, it is automatically compiled and web page is refreshed

Lets understand first React app

- Go to the folder where you installed React app
- *my-app*
- Go to the folder *src*
- Open *index.js* in Visual Studio code or Atom or in your JS editor



index.js


EXPLORER
OPEN EDITORS
index.js C:\Users\GiRi\Do...
index.js D:\Dropbox\Rese...
App.js D:\Dropbox\R.. M
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
index.js C:\...\Documents
index.js D:\...\src
App.js
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
ReactDOM.render(<App />, document.getElementById('root'));
// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();

index.js

Line 1-2

- Import React and ReactDOM so we can use them in our application
- ReactDOM provides DOM-specific methods which you'll need to do things like mount your components

```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import App from './App';
5   import * as serviceWorker from './serviceWorker';
6
7   ReactDOM.render(<App />, document.getElementById('root'));
8
9   // If you want your app to work offline and load faster, y
10  // unregister() to register() below. Note this comes with
11  // Learn more about service workers: https://bit.ly/CRA-PW
12  serviceWorker.unregister();
```

index.js

Line 3

- Import the CSS file for the index.js file

Line 4

- Import the <App /> component so we can attach it to the DOM
- Rather that writing all the component codes in a single file, it is a good standard to write each components in different files
- The above line is used for importing the class you have written in the file *App.js* under the same directory

```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import App from './App';
5   import * as serviceWorker from './serviceWorker';
6
7   ReactDOM.render(<App />, document.getElementById('root'));
8
9   // If you want your app to work offline and load faster, y
10  // unregister() to register() below. Note this comes with
11  // Learn more about service workers: https://bit.ly/CRA-PW
12  serviceWorker.unregister();
```

index.js

Line 5

- Import the registerServiceWorker module
- Service Workers are wonderful tools which allow your app to function offline
- A service worker is a script that your browser runs in the background, separate from a web page, opening the door to features that don't need a web page or user interaction

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5  import * as serviceWorker from './serviceWorker';
6
7  ReactDOM.render(<App />, document.getElementById('root'));
8
9  // If you want your app to work offline and load faster, y
10 // unregister() to register() below. Note this comes with
11 // Learn more about service workers: https://bit.ly/CRA-PW
12 serviceWorker.unregister();
```

Read more here: https://developers.google.com/web/fundamentals/primers/service-workers/

index.js

Line 7

- Using ReactDOM, we can mount the App component to the DOM
- This method takes two mandatory parameters and one optional parameter
  - The component to render: <App />
  - The parent node to attach this component to: rootElement
  - A callback function which will execute after the component has rendered (optional)

```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import App from './App';
5   import * as serviceWorker from './serviceWorker';
6
7   ReactDOM.render(<App />, document.getElementById('root'));
8
9   // If you want your app to work offline and load faster, y
10  // unregister() to register() below. Note this comes with
11  // Learn more about service workers: https://bit.ly/CRA-PW
12  serviceWorker.unregister();
```

Read more here:
https://reactjs.org/docs/react-dom.html#render

index.js

Line 12

- Register the Service Worker so your app can be viewed without an internet connection

```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import App from './App';
5   import * as serviceWorker from './serviceWorker';
6
7   ReactDOM.render(<App />, document.getElementById('root'));
8
9   // If you want your app to work offline and load faster, y
10  // unregister() to register() below. Note this comes with
11  // Learn more about service workers: https://bit.ly/CRA-PW
12  serviceWorker.unregister();
```

Read more here:
https://reactjs.org/docs/react-dom.html#render

App.js

```jsx
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p> Hi Dr Ganesh,
          edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

App.js

Line 1-3

- Import React, logo.svg and css files

```
1  import React from 'react';
2  import logo from './logo.svg';
3  import './App.css';
4
5  function App() {
6    return (
7      <div className="App">
8        <header className="App-header">
9          <img src={logo} className="App-logo" alt="logo" />
10         <p> Hi Dr Ganesh,
11           edit <code>src/App.js</code> and save to reload.
12         </p>
13         <a
14           className="App-link"
15           href="https://reactjs.org"
16           target="_blank"
17           rel="noopener noreferrer"
18         >
19           Learn React
20         </a>
21       </header>
22     </div>
23   );
24 }
25
26 export default App;
```

Why import React from “react” in a functional component?

- Even though my functional component (app.js in this case) has no reference to React(since I could not see any mention of React keyword in the component), then why do I need to have import React from “react"; at the top of my file?

Why import React from “react” in a functional component?

- The code written will be compiled to another form using Babel
- Babel is a transpiler – a compiler which produces code in another language of same level
- Transpilers, or source-to-source compilers, are tools that read source code written in one programming language, and produce the equivalent code in another language
- In this case, it converts react code to backward compatible javascript

Why import React from “react” in a functional component?

```
import React from "react";
const App = () => (
  <div>Hello World!!!</div>
);
export default App;
```

Babel Transpilation

```
var App = function App() {
  return React.createElement(
    "div",
    null,
    "Hello World!!!"
  );
};
```

https://hackernoon.com/why-import-react-from-react-in-a-functional-component-657aed821f7a

The important thing to focus here is **React.createElement** and this is the reason why we need to import React at the start of any functional component

App.js

Line 5

- Create the *app* component
- **Functional Components** are declared with the function keyword and accept props as an argument. They simply return valid React element.
- **Class Components** are declared with the class keyword and extend React.Component or just Component if you have imported it using the destructured format
- Both functional and class components are logically equivalent (class just provides more functionality), so don't worry too much about the differences right now

```
1  import React from 'react';
2  import logo from './logo.svg';
3  import './App.css';
4
5  function App() {
6    return (
7      <div className="App">
8        <header className="App-header">
9          <img src={logo} className="App-logo" alt="logo" />
10         <p> Hi Dr Ganesh,
11           edit <code>src/App.js</code> and save to reload.
12         </p>
13         <a
14           className="App-link"
15           href="https://reactjs.org"
16           target="_blank"
17           rel="noopener noreferrer"
18         >
19           Learn React
20         </a>
21       </header>
22     </div>
23   );
24 }
25
26 export default App;
```

App.js

Line 7-22

- Describe what the UI should look like
- It is called **JSX** - A syntax extension to JavaScript
- React uses JSX for templating instead of regular JavaScript. It is not necessary to use it, however, following are some pros that come with it.
  - It is faster because it performs optimization while compiling code to JavaScript.
  - It is also type-safe and most of the errors can be caught during compilation.
  - It makes it easier and faster to write templates, if you are familiar with HTML

```
1  import React from 'react';
2  import logo from './logo.svg';
3  import './App.css';
4
5  function App() {
6    return (
7      <div className="App">
8        <header className="App-header">
9          <img src={logo} className="App-logo" alt="logo" />
10         <p> Hi Dr Ganesh,
11           edit <code>src/App.js</code> and save to reload.
12         </p>
13         <a
14           className="App-link"
15           href="https://reactjs.org"
16           target="_blank"
17           rel="noopener noreferrer"
18         >
19           Learn React
20         </a>
21       </header>
22     </div>
23   );
24 }
25
26 export default App;
```

App.js

Line 26

- Export the App component
- In React we use the keyword export to export a particular module or a named parameter or a combination so they can be used by other programs with the import statement

```
1  import React from 'react';
2  import logo from './logo.svg';
3  import './App.css';
4
5  function App() {
6    return (
7      <div className="App">
8        <header className="App-header">
9          <img src={logo} className="App-logo" alt="logo" />
10         <p> Hi Dr Ganesh,
11           edit <code>src/App.js</code> and save to reload.
12         </p>
13         <a
14           className="App-link"
15           href="https://reactjs.org"
16           target="_blank"
17           rel="noopener noreferrer"
18         >
19           Learn React
20         </a>
21       </header>
22     </div>
23   );
24 }
25
26 export default App;
```

That's it!

- This completes a basic introduction to React
- Reference:
  - https://medium.com/in-the-weeds/learning-react-with-create-react-app-part-1-a12e1833fdc
  - https://medium.com/in-the-weeds/learning-react-with-create-react-app-part-2-3ad99f38b48d
  - https://medium.com/in-the-weeds/learning-react-with-create-react-app-part-3-322447d14192
  - https://codeburst.io/react-for-beginners-part-1-59a983dedeec
  - https://programmingwithmosh.com/react/react-tutorial-beginners/

Self-study:
Learn how to create forms using React

- https://reactjs.org/docs/forms.html
- https://medium.com/@agoiabeladeyemi/the-complete-guide-to-forms-in-react-d2ba93f32825
- https://www.codementor.io/blizzerand/building-forms-using-react-everything-you-need-to-know-iz3eyoq4y
- https://dev.to/nsebhastian/react-form-real-time-validation-using-state-1eeg
- https://blog.logrocket.com/an-imperative-guide-to-forms-in-react-927d9670170a/
- https://itnext.io/building-a-dynamic-controlled-form-in-react-together-794a44ee552c
- https://medium.com/@geeky_writer_/using-react-hooks-to-create-awesome-forms-6f846a4ce57

Dive deep into React

Reference:
https://reactjs.org/docs/hello-world.html

Recap: JSX

- It is called JSX, and it is a syntax extension to JavaScript
- We recommend using it with React to describe what the UI should look like
- JSX may remind you of a template language, but it comes with the full power of JavaScript.

Embedding Expressions in JSX

- In the example below, we declare a variable called name and then use it inside JSX by wrapping it in curly braces:

```
function App() {
  const name='Dr Ganesh Iyer';
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p> Hi {name},
          edit <code>src/App.js</code> and save to reload.
        </p>
```

- You can put any valid JavaScript expression inside the curly braces in JSX
- For example, 2 + 2, user.firstName, or formatName(user) are all valid JavaScript expressions

In the example below, we embed the result of calling a JavaScript function, formatName(user), into <p> element

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Dr Ganesh',
  lastName: 'Iyer'
};

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Hi {formatName(user)} Edit <code>src/App.js</code> and sav
        </p>
```

Components and Props
Function and class components

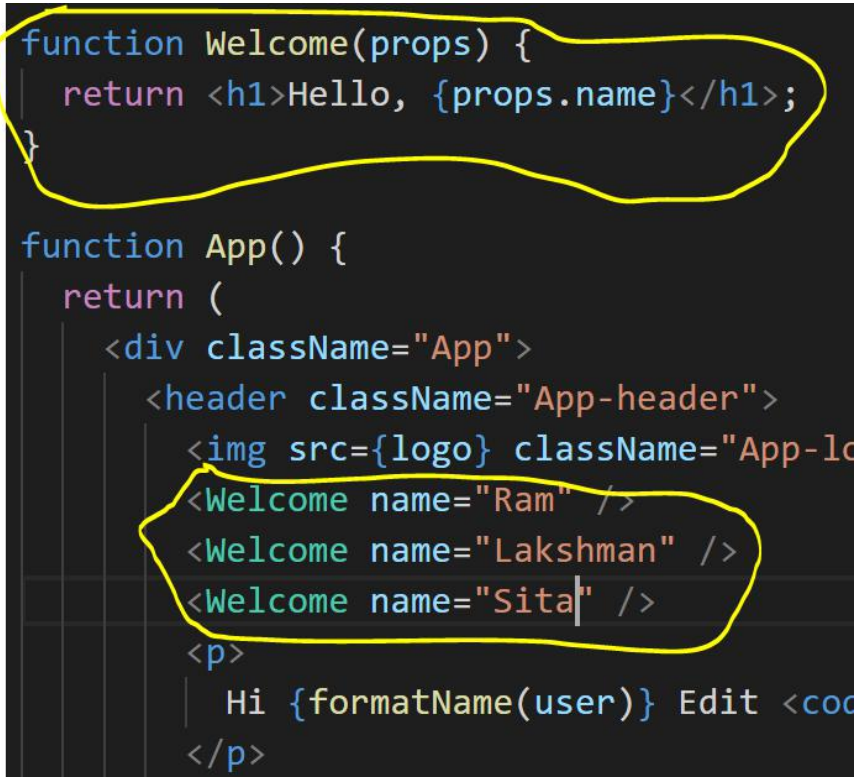
```jsx
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

```jsx
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

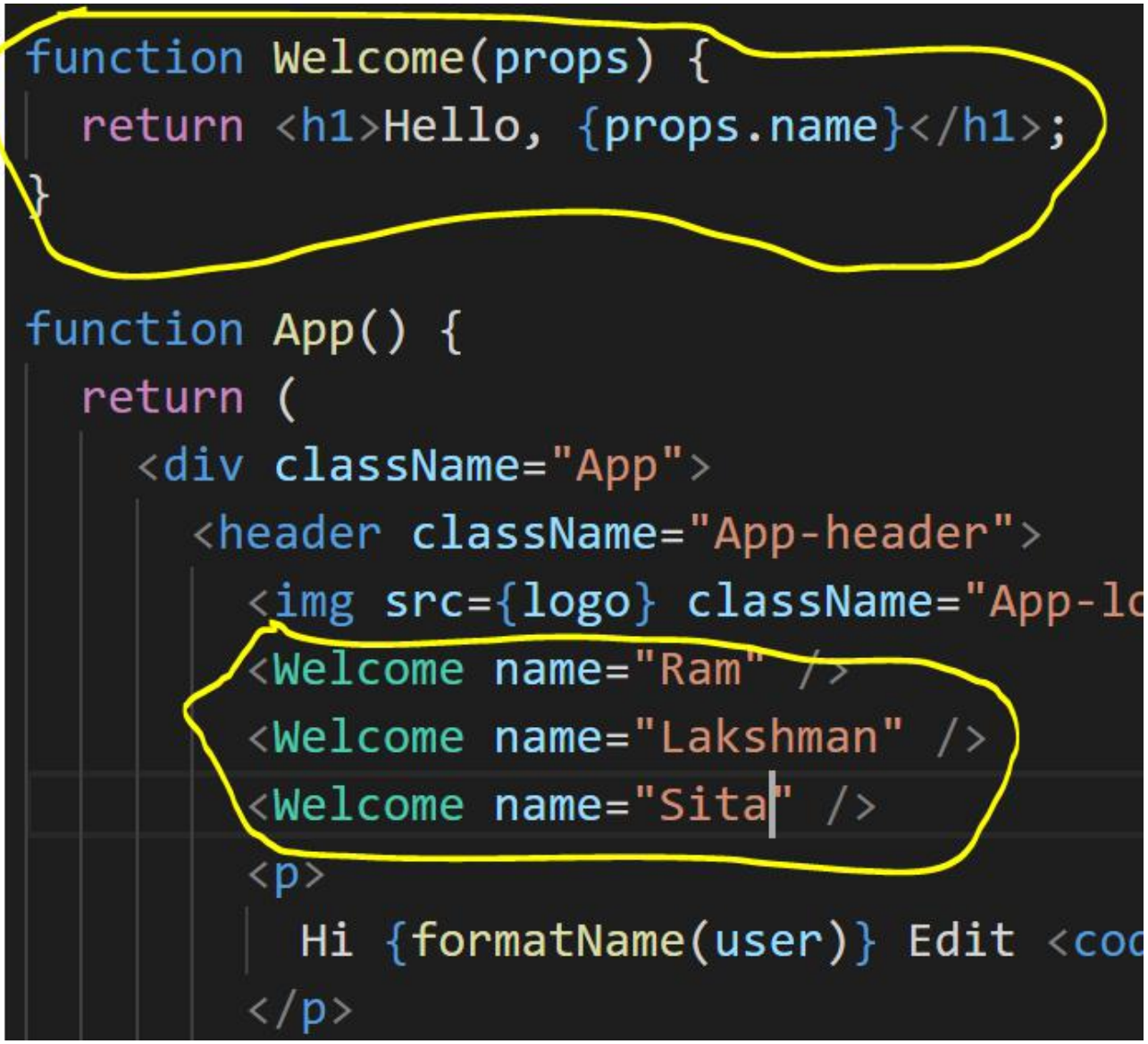
- The above two components are equivalent from React's point of view

Composing components

- 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



Composing components



Handling events

```
function App() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');}
  return (
    <div className="App">
      <header className="App-header">
      <a href="http://google.com" onClick={handleClick}>
      Click me
      </a>
        <img src={logo} className="App-logo" alt="logo" />
        <Welcome name="Ram" />
```

Conditional Rendering

- In React, you can create distinct components that encapsulate behavior you need
- Then, you can render only some of them, depending on the state of your application
- Conditional rendering in React works the same way conditions work in JavaScript
- Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them

Conditional Rendering

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />,
  document.getElementById('root')
);
```

- Greeting component is one that displays either of these components depending on whether a user is logged in
- This example renders a different greeting depending on the value of isLoggedIn prop

I have modified index.js

- Commented the App rendering
- Instead it renders Greeting now

```
import * as serviceWorker from './serviceWorker';

function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}

function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}

ReactDOM.render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={true} />,
  document.getElementById('root')
);
//ReactDOM.render(<App />, document.getElementById('root'));
```

Element Variables

- You can use variables to store elements
- This can help you conditionally render a part of the component while the rest of the output doesn't change.
- Consider these two new components representing Logout and Login buttons

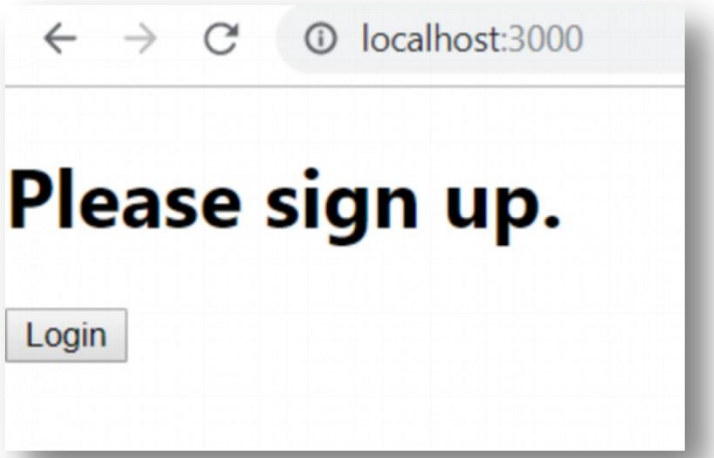
```
function LoginButton(props) {
  return (
    <button onClick={props.onClick}>
      Login
    </button>
  );
}

function LogoutButton(props) {
  return (
    <button onClick={props.onClick}>
      Logout
    </button>
  );
}
```

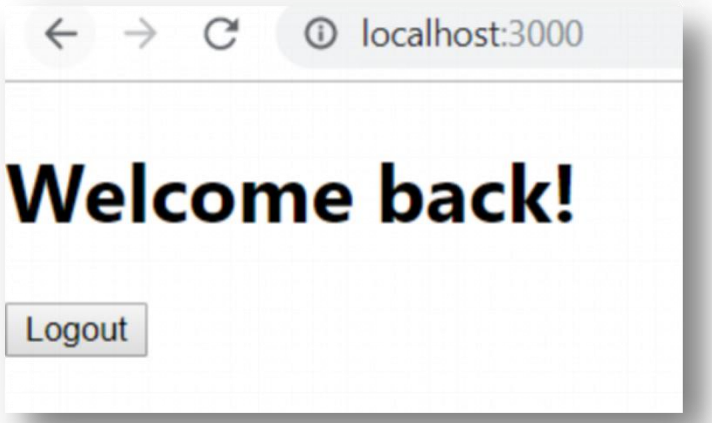
Stateful Components

- In the example below, we will create a stateful component called LoginControl.
- It will render either <LoginButton /> or <LogoutButton /> depending on its current state
- It will also render a <Greeting /> from the previous example

Stateful Components

http://bit.ly/31J3VGE






```jsx
class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.handleLoginClick = this.handleLoginClick.bind(this);
    this.handleLogoutClick = this.handleLogoutClick.bind(this);
    this.state = {isLoggedIn: false};
  }

  handleLoginClick() {
    this.setState({isLoggedIn: true});
  }

  handleLogoutClick() {
    this.setState({isLoggedIn: false});
  }

  render() {
    const isLoggedIn = this.state.isLoggedIn;
    let button;

    if (isLoggedIn) {
      button = <LogoutButton onClick={this.handleLogoutClick} />;
    } else {
      button = <LoginButton onClick={this.handleLoginClick} />;
    }

    return (
      <div>
        <Greeting isLoggedIn={isLoggedIn} />
        {button}
      </div>
    );
  }
```

Self exercise 1

[NAME]
[ADDRESS]
[PHONE #] – [EMAIL]

EDUCATION

University of Washington School of Medicine - Seattle, WA [DATE]-Present
Doctor of Medicine
Expected Graduation:

University of Washington School of Public Health – Seattle, WA [DATE]-[DATE]
[SECONDARY DEGREE]

[UNDERGRADUATE] – [LOCATION] [DATE]-[DATE]
[DEGREE], [FIELD] – [HONORS]
[MINOR DEGREES]

HONORS & AWARDS

[AWARD] [YEAR]

[AWARD] [YEAR]
- [DESCRIPTION]
- [DESCRIPTION]

[AWARD] [YEAR]
- [DESCRIPTION]

RESEARCH EXPERIENCE

[INSTITUTE/COMPANY] – [LOCATION] [YEAR]
- *[PROJECT TITLE]*
  - [PROJECT DESCRIPTION]
  - Advisor: [ADVISOR] – [ADVISOR'S INSTITUTE]

[INSTITUTE/COMPANY] – [LOCATION] [YEAR]
- *[PROJECT TITLE]*
  - [PROJECT DESCRIPTION]
  - Advisors: [ADVISOR1] – [ADVISOR'S INSTITUTE]
    [ADVISOR2] – [ADVISOR'S INSTITUTE]

[INSTITUTE/COMPANY] – [LOCATION] [YEAR]
- *[PROJECT TITLE]*
  - [PROJECT DESCRIPTION]
  - Advisor: [ADVISOR] – [ADVISOR'S INSTITUTE]

PRESENTATIONS

[CONFERENCE TITLE] – [LOCATION] [YEAR]
[CONFERENCE TITLE] – [LOCATION] [YEAR]
[SESSION TITLE] – [LOCATION] [YEAR]

Self exercise 2

- Design a login page with one field “User Name” and one button “Login”
- Once logged in with your First name, it should show a message “Welcome <First Name>, Login successful” and a logout button, Wrong input should show error message
- Once clicked on Logout button, show login page
- Bonus component: Let the login be with both user name and password

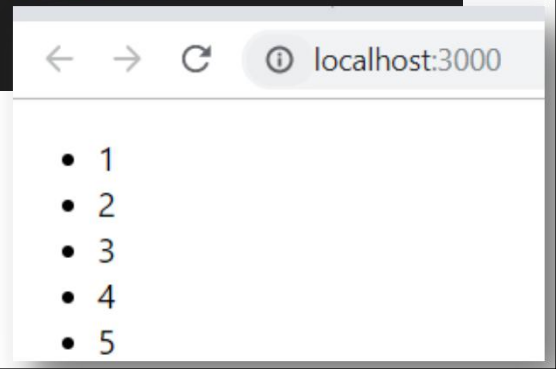
Complete the exercise by Monday!
Show me the output and book signed on Monday

Rendering multiple components

- You can build collections of elements and include them in JSX using curly braces {}.
- Below, we loop through the numbers array using the JavaScript map() function
- We return a <li> element for each item
- Finally, we assign the resulting array of elements to listItems:

index.js

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);

ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```



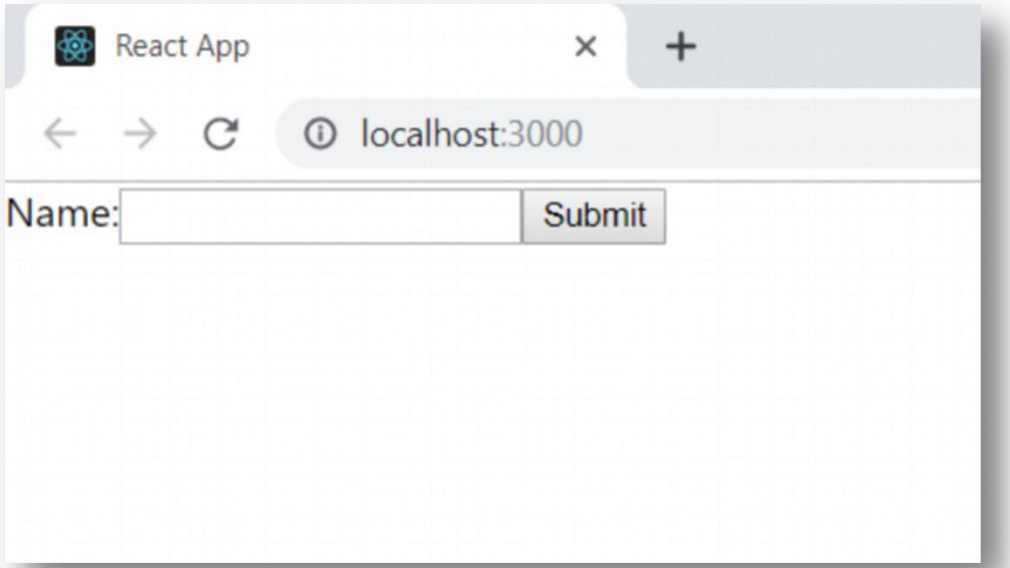
Try other options in keys here: https://reactjs.org/docs/lists-and-keys.html

Forms

- In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input
- In React, mutable state is typically kept in the state property of components, and only updated with setState()
- We can combine the two by making the React state be the “single source of truth”
- Then the React component that renders a form also controls what happens in that form on subsequent user input
- An input form element whose value is controlled by React in this way is called a “controlled component”.

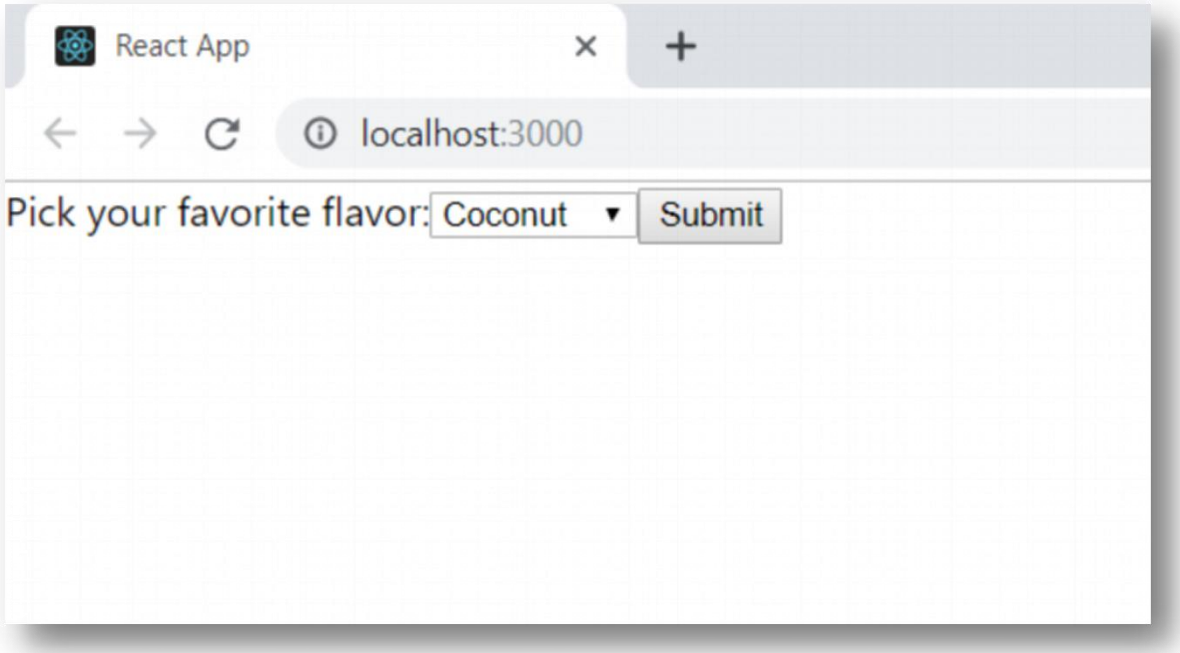
Forms → Example code available at dropbox folder → CC-Exercises

```jsx
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```



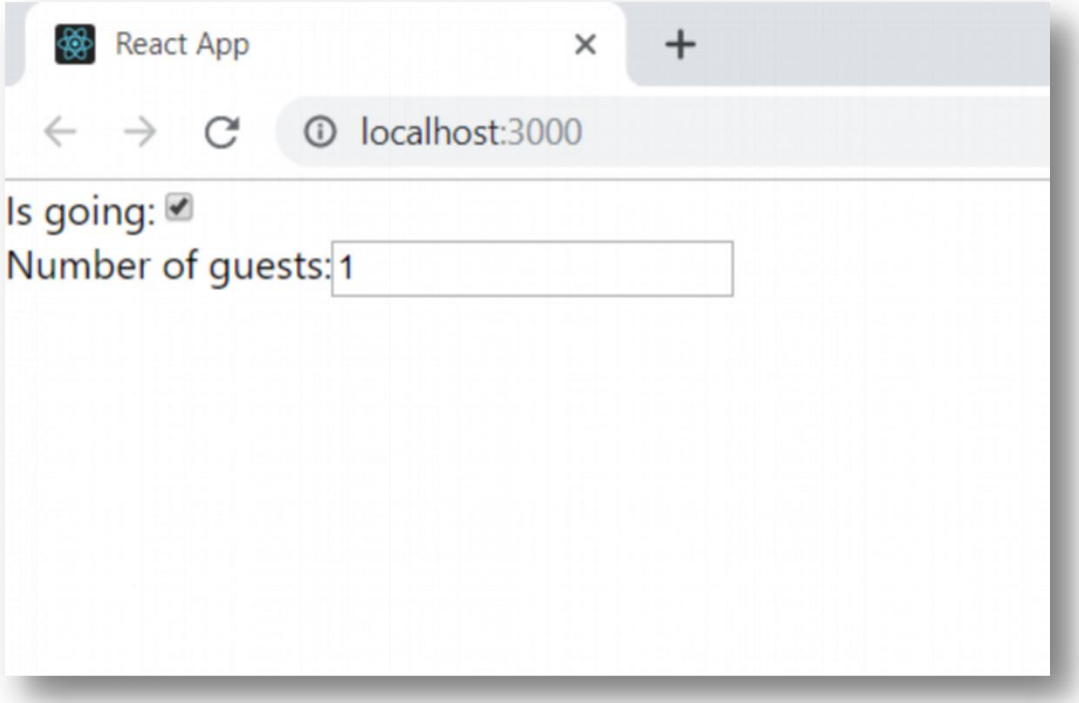
The select tag → Example code available at dropbox folder → CC-Exercises

```
class FlavorForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: 'coconut'};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('Your favorite flavor is: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Pick your favorite flavor:
          <select value={this.state.value} onChange={this.handleChange}>
            <option value="grapefruit">Grapefruit</option>
            <option value="lime">Lime</option>
            <option value="coconut">Coconut</option>
            <option value="mango">Mango</option>
          </select>
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
```



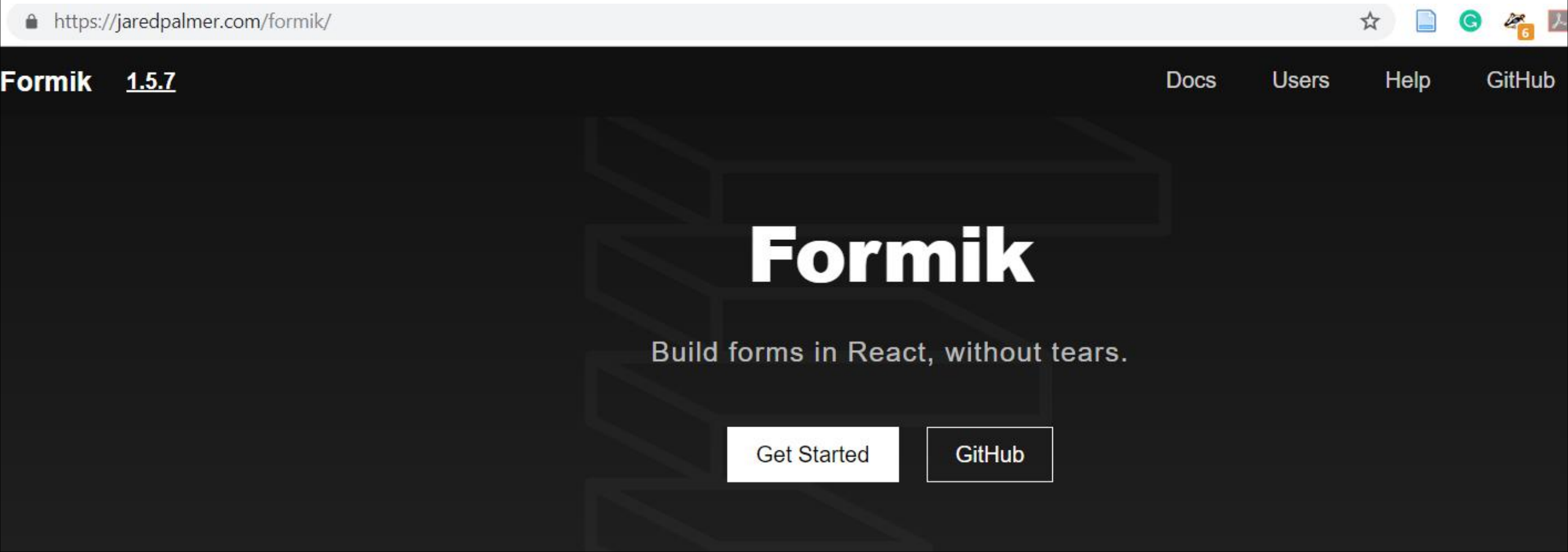
Multiple Inputs → Example code available at dropbox folder → CC-Exercises

```
class Reservation extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      isGoing: true,
      numberOfGuests: 2
    };

    this.handleInputChange = this.handleInputChange.bind(this);
  }

  handleInputChange(event) {
    const target = event.target;
    const value = target.type === 'checkbox' ? target.checked : target.value;
    const name = target.name;

    this.setState({
      [name]: value
    });
  }

  render() {
    return (
      <form>
        <label>
          Is going:
          <input
            name="isGoing"
            type="checkbox"
            checked={this.state.isGoing}
            onChange={this.handleInputChange} />
        </label>
        <br />
        <label>
          Number of guests:
          <input
            name="numberOfGuests"
            type="number"
            value={this.state.numberOfGuests}
            onChange={this.handleInputChange} />
        </label>
```

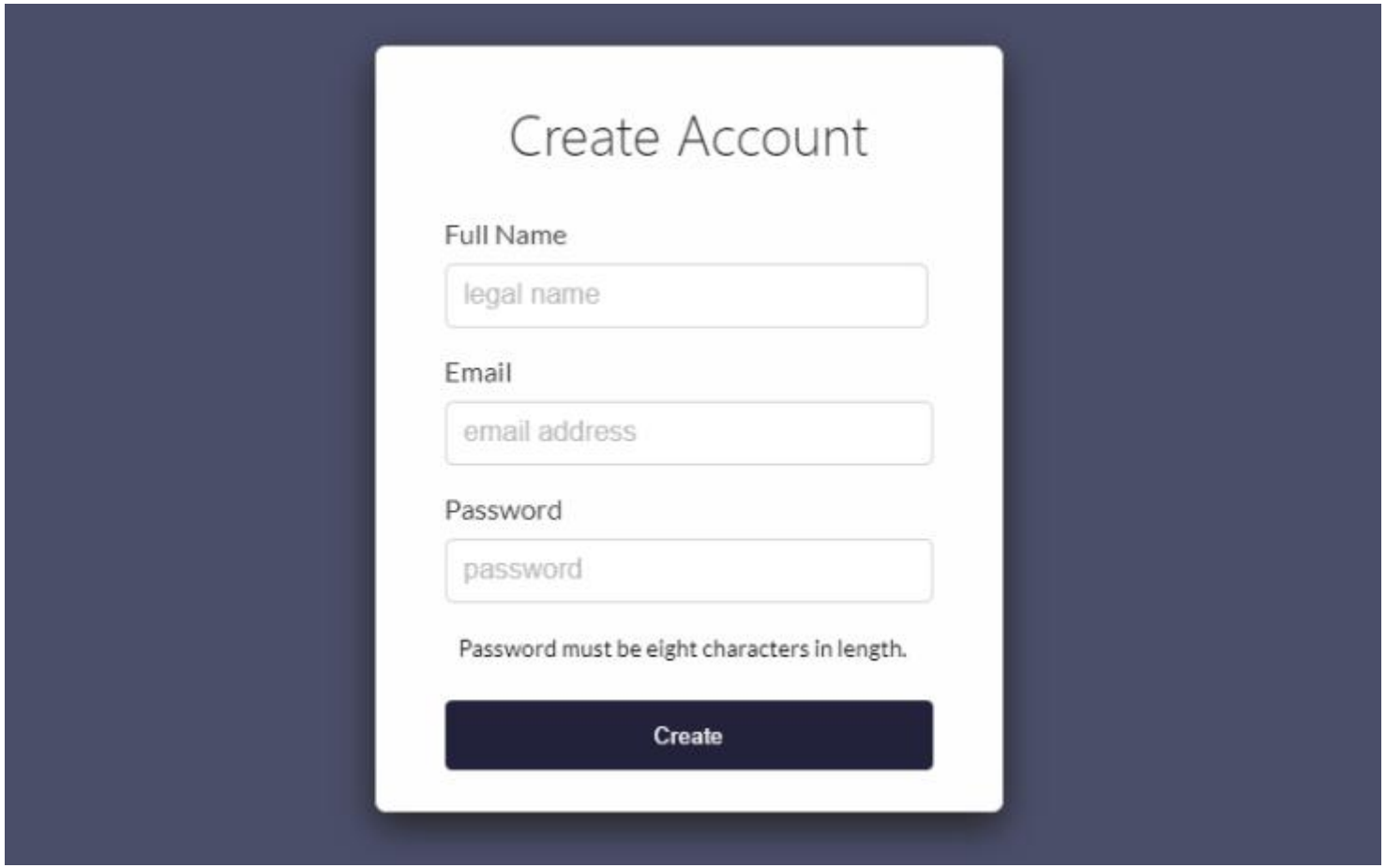


Fully-fledged form design (Optional, good to explore)

- Self Study: https://jaredpalmer.com/formik/

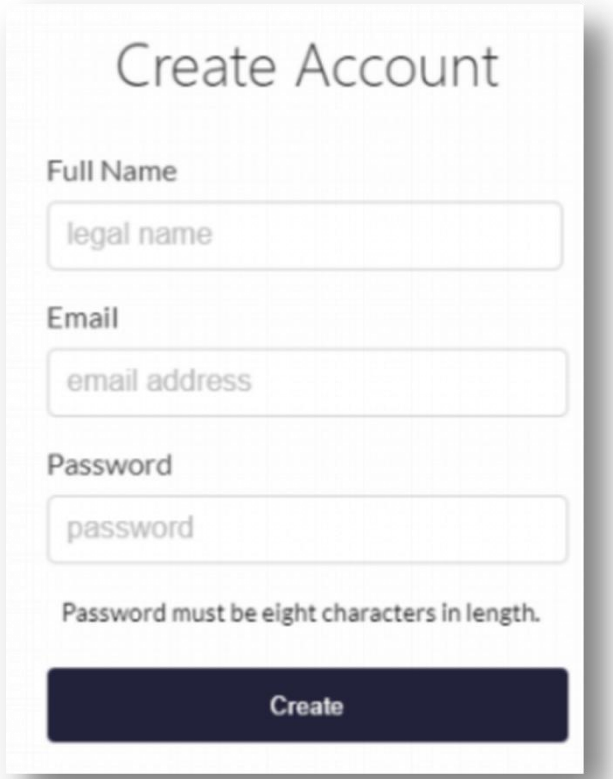


Custom Form validation in react



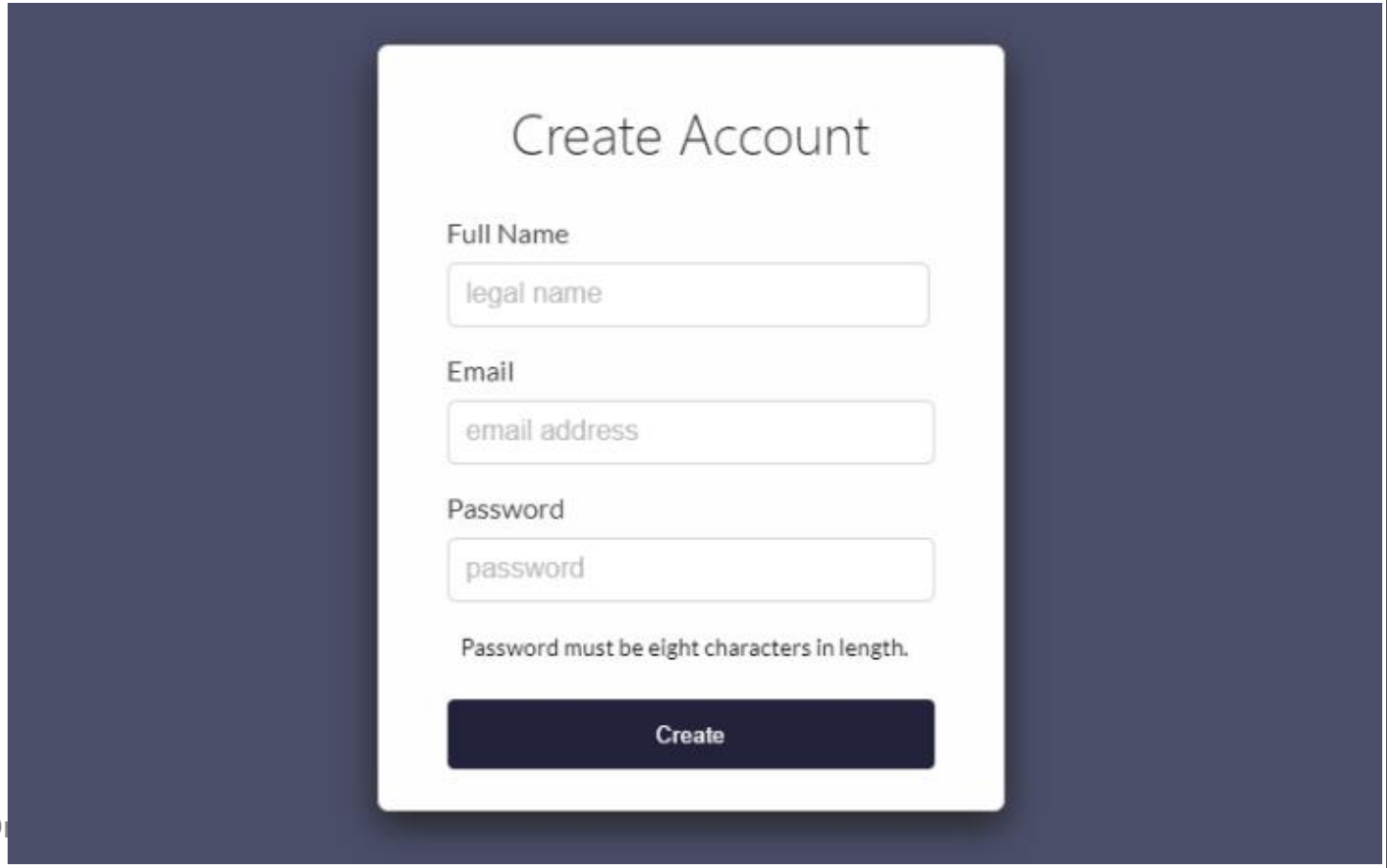
Design considerations

- One of the things to notice in the form I have setup for you is that we have specified three different types of inputs
- We have a fullName, email and password input
- It's very important to use the right type on each input as the behavior it provides is what users expect with a professional form
- It will assist their form fillers and allow for an obfuscated password which is also pretty standard



Lets start form vaildations

- Copy the folder react-forms-validation-start from dropbox into your computer
- npm clear cache –force
- npm install
- npm start



Validations

- We are going to build our form validation from this point and do all of the JavaScript logic ourselves
- Currently the form does not submit or work in anyway, it has only been styled
- The first thing we want to add is a constructor to our **Register component**:

- Our state will contain a property for each input as well as have an object (error) which will hold the text for our error messages
- Each form input is represented in this error object as well
- If we detect the input is invalid, this string will have a value, otherwise the value will be empty or zero
- If it's not zero, we will create logic to display the message to the user

```
constructor(props) {
  super(props);
  this.state = {
    fullName: null,
    email: null,
    password: null,
    errors: {
      fullName: '',
      email: '',
      password: '',
    }
  };
}
```

- Next we will add the handleChange() function
- This will fire every time we enter a character into one of the inputs on our form
- Inside that function, a switch statement will handle each input respectfully, constantly checking to see if we have for instance reached a minimum character limit or a found a RegEx match
- Each time a character is entered, an event will be passed to this function getting destructured. Destructuring assignment plucks our values out of the event.target and assigns them to local variables (name and value) inside of our function
- Let's add the handleChange() function. It should come right before the render method of our **Register class**:

```
handleChange = (event) => {
  event.preventDefault();
  const { name, value } = event.target;
  let errors = this.state.errors;

  switch (name) {
    case 'fullName':
      errors.fullName =
        value.length < 5
          ? 'Full Name must be 5 characters long!'
          : '';
      break;
    case 'email':
      errors.email =
        validEmailRegex.test(value)
          ? ''
          : 'Email is not valid!';
      break;
    case 'password':
      errors.password =
        value.length < 8
          ? 'Password must be 8 characters long!'
          : '';
      break;
    default:
      break;
  }

  this.setState({errors, [name]: value}, ()=> {
      console.log(errors)
  })
}
```

- The code above will enter into the correct switch case depending on which input you are typing in
- It will check that you have entered the correct length for that input or in the case of the email, it will check a RegEx that we still need to create and ensure that it matches the regular expression that checks for a proper email format
- Just above our Register class we can add a const that holds this RegEx and then we can call .test() on that RegEx string to see if our input matches and returns true, otherwise we will add an error message to our local copy of our error state.

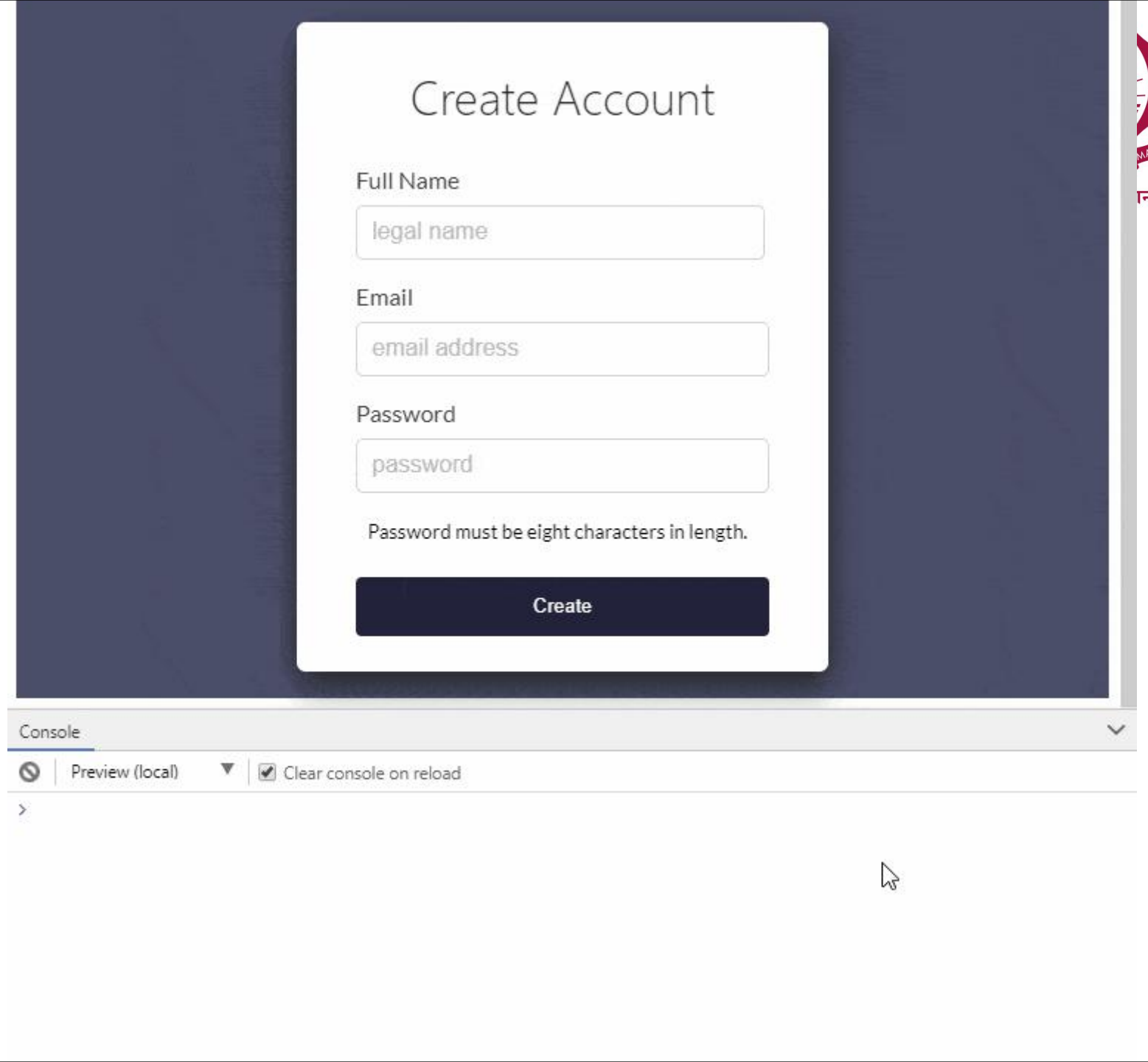
- const validEmailRegex =

RegExp(/^(([^<>()\[\]\.,;:\s@\"]+(\.[^<>()\[\]\.,;:\s@\"]+)*)|(\".+\"))
@(([^<>()[\]\.,;:\s@\"]+\.)+[^<>()[\]\.,;:\s@\"]{2,})$/i);

The RegEx is nearly impossible to read, but rest assured it covers most cases that we want to check including accepting unicode characters

Understand that this is just a test we perform on the frontend and in a real application you should test the email on the server-side with legit validation depending on your requirements.

This is a great spot to stop and check our work, in fact most of our validation is already working, if we go into our console for this page we can see what error messages are being created up until we satisfy each inputs validation:

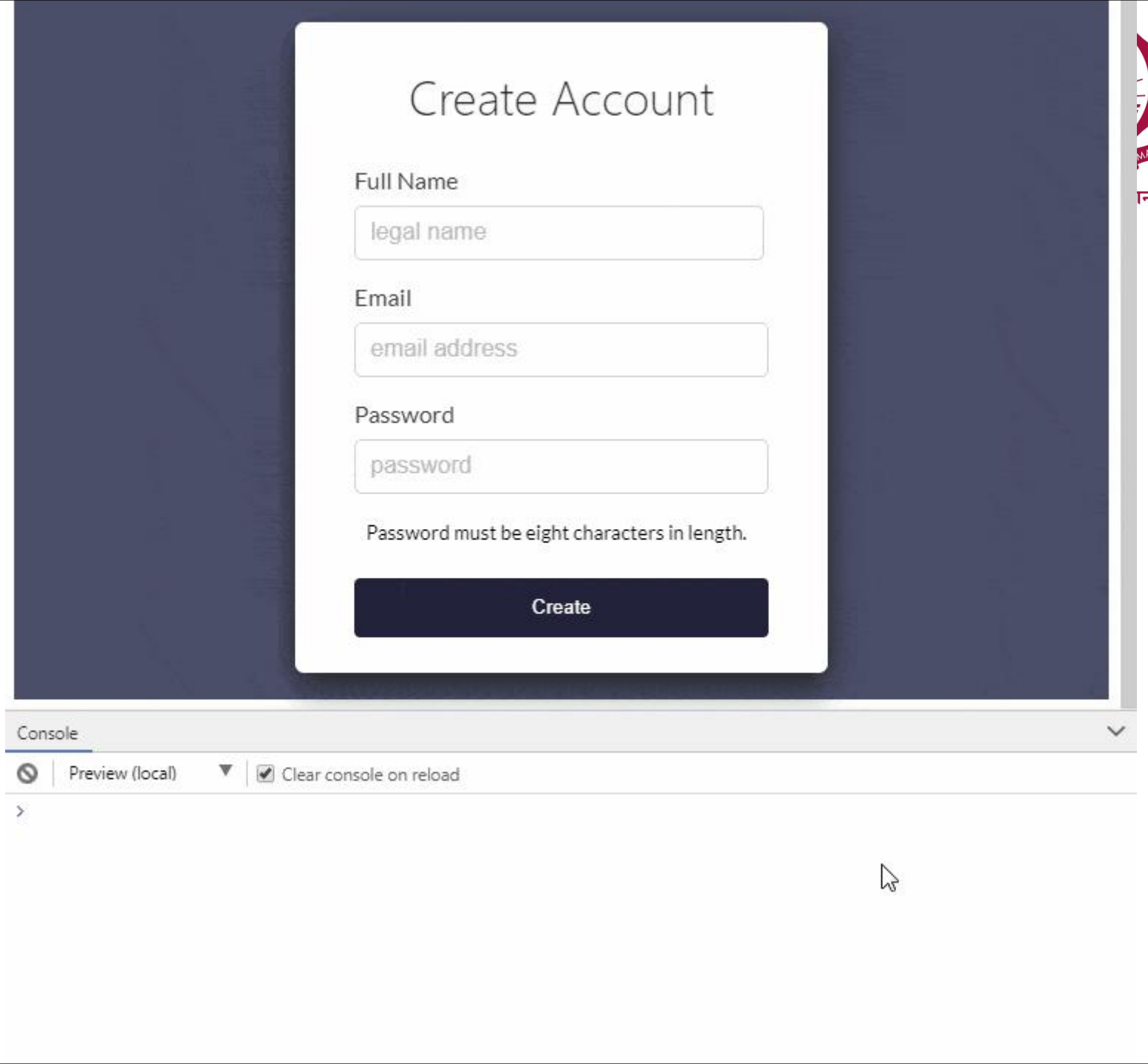


As you can see, as soon as we enter our first character in the fullName input, we get an error message

The fullName input requires that we enter at least 5 characters. We see that in our console up until we meet the criteria, then the error message disappears.

Although we will not continue logging these errors in the console, we will pay attention in future code to the fact that we either have an error message or not

If so, we will display that error message to the user directly underneath the input



References and next steps

- Look at react-forms-validation-start.zip in the dropbox folder
- In src folder, there is in index.js which is the form without validation
- Then copy the contents of index-withvalidation.js to get full picture
- Based on the concepts learnt in these slides, understand and run the code

- https://www.telerik.com/blogs/up-and-running-with-react-form-validation

Next class

- Node.js

Thank you!

Dr Ganesh Neelakanta Iyer

ni_amrita@cb.amrita.edu
ganesh.vigneswara@gmail.com

Office Hours

– Thursday 1350-1440
@ My office

Useful debugging commands

- npm install -g create-react-app
- npx create-react-app my-app

- npm cache clean –force
- Alternatively copy my-app form someone else and run npm install followed by npm start