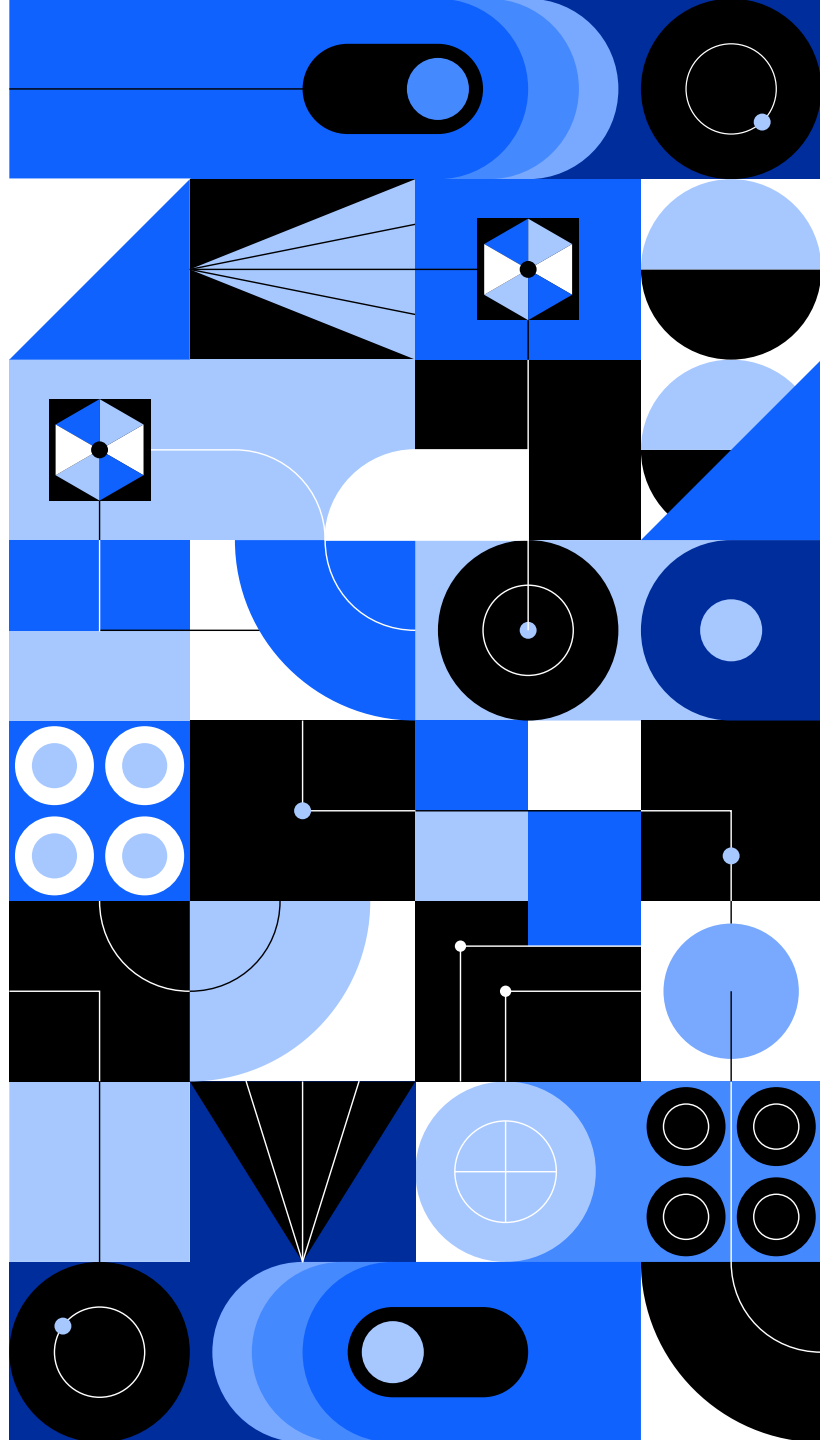
IBM Accelerate Software Developer Track

Wednesdays June 7th – July 26th
6:00pm – 8:00pm Eastern Time



• Optional practice test

Wednesday, June 14, 2023

JavaScript, React, and Styling

Today's speakers:

Garrett Bolter
UI Developer
MultiCloud SaaS Platform
IBM Software

Sujeily Fonseca
Software Engineering Manager, Technical Leader, MultiCloud Saas Platform
SW Track Lead

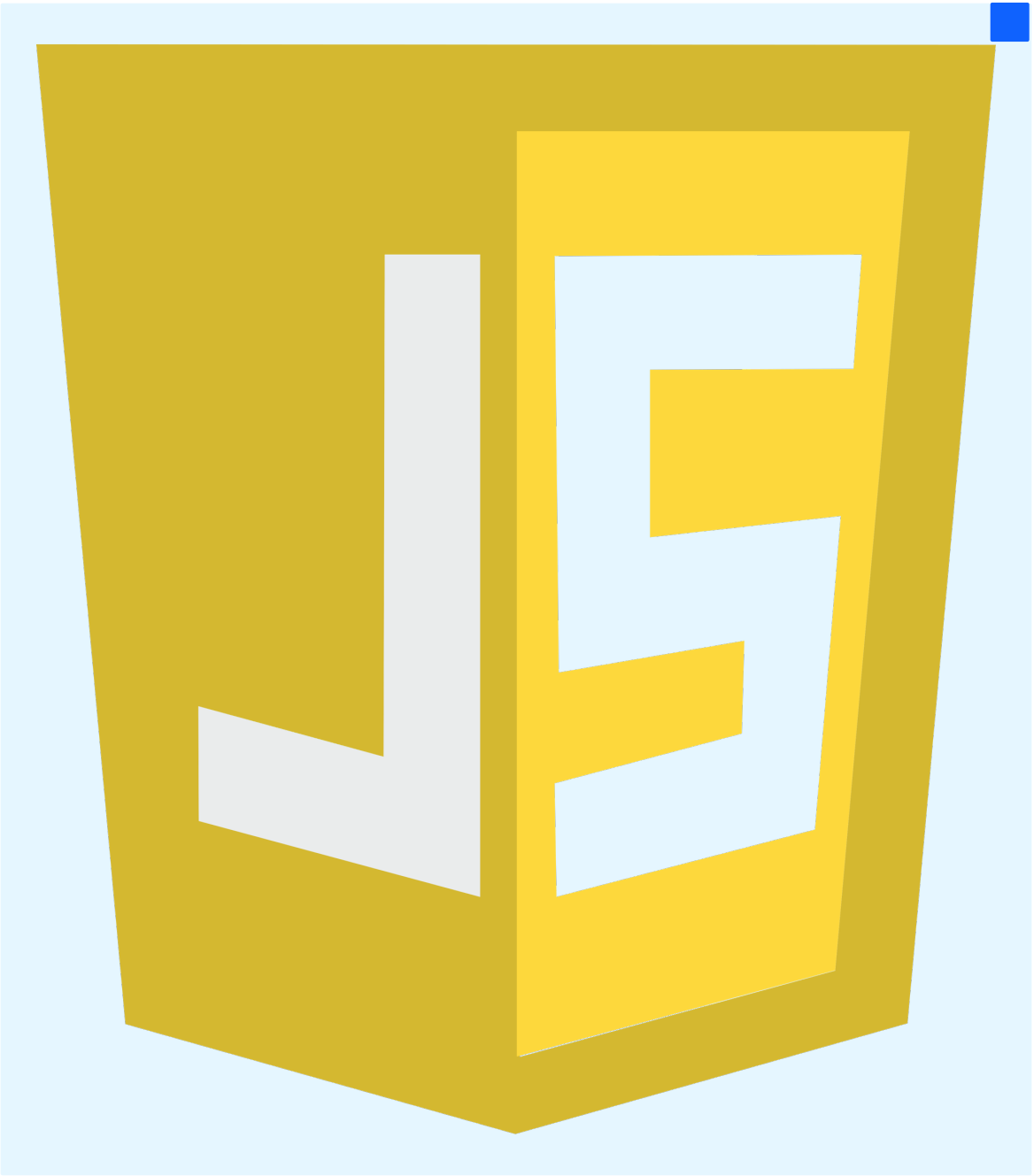
Eram Manasia
Software Engineer,
Sales Technology Engineering,
Finance and Operations

Session Agenda

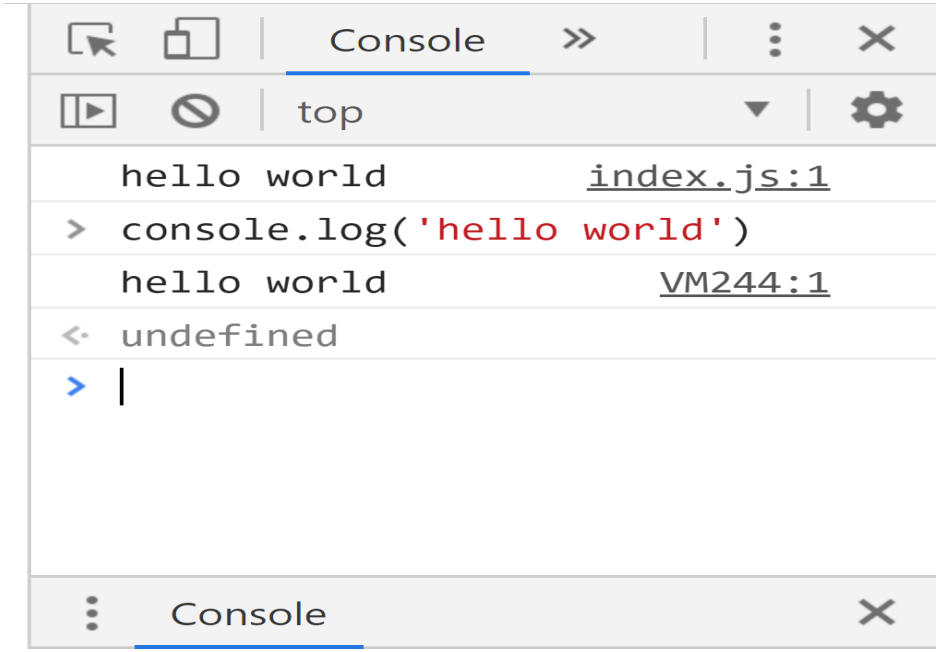
1. **JavaScript Fundamentals**
   History, usage, types, operators, identifiers, objects, functions, and asynchronous calls.
2. **React Deep-Dive**
   Rendering elements, components and props, state and lifecycle, event handling, and list and keys.
3. **Design and Styling**
   UX and UO design, styling in react, styling options, and Material UI (what/why/how).
4. **Q&A**
5. **HackerRank Practice Test**
6. **Lab/Project and GitHub Classroom**
7. **Next Steps**

JavaScript Fundamentals

Garrett Bolter
UI Developer
MultiCloud SaaS Platform
IBM Software



What do we know about JavaScript?



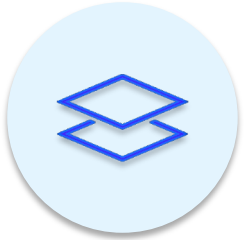
- JavaScript is a scripting language.
  - It gives the computer a series of instructions to perform.
- It interacts with the user, the page, and the internet.
- It is multi-paradigm, supporting event-driven, functional, and imperative programming styles.
- It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

JavaScript Fundamentals

VARIABLE DECLARATION

FUNCTIONS AND CLASSES

OBJECTS



SCOPING

ASYNCHRONOUS CALLS

Variables

Why are variables important?

- Variables are containers for storing data values.
- They allow programs to store and reuse values.
- There are different ways to declare variables in JavaScript.

Variable Declaration

`let`

What is it?
- Mutable
- Block-scoped

What is it useful for?
- Temporary values
- Most use cases

```
<script>
function example() {
  let x = 6;
}
</script>
```

`const`

What is it?
- Immutable
- Block-scoped

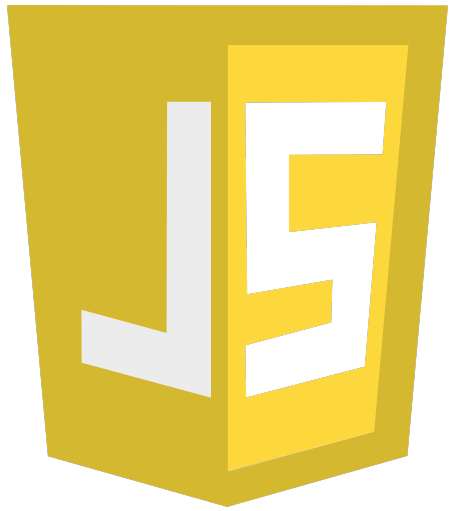
What is it useful for?
- Values that should not change

```
<script>
function example() {
  const y = 12;
}
</script>
```

`var`

What is it?
- ES6, same as let! <:-)
  - Recommendation to use let or const instead

Functions and Classes



Why use functions and classes?

- Functions and classes are both ways of logically organizing code.
- Although we will go over how they are used in JavaScript, functions, and classes are universal concepts in any object-oriented programming language.

Functions

- A function is a block of code designed to perform a particular task.
- Functions can be declared with the function keyword.
  - A **function declaration** consists of the function keyword, name, parameters, and curly brackets enclosing the function definition block

```
1.  // Function declaration
2.  function add(a, b) {
3.    return a+b;
4.  }
5.  add(2,4); // Returns 6


6.  function subtract(a, b) {
7.    return a-b;
8.  }
9.  subtract(4,2); // Returns 2
```

Functions (Cont.)

- Functions can be assigned to variables and defined as a **function expression.**
- The third way to define a function is called an **arrow function expression.**
  - Useful for passing a function as an argument.

```
1.  // Function expression
2.  const add = function(a, b) {
3.    return a+b;
4.  }
5.  add(2,4); // Returns 6


6.  // Arrow function
7.  a = [1,2,3]
8.  a.map((x) => x+1) // Returns [2,3,4]
```

Description

The `map()` method is an iterative method. It calls a provided `callbackFn` function once for each element in an array and constructs a new array from the results. Read the iterative methods section for more information about how these methods work in general.

```
const arrowFx = () => {
  console.log("boom");
}
```

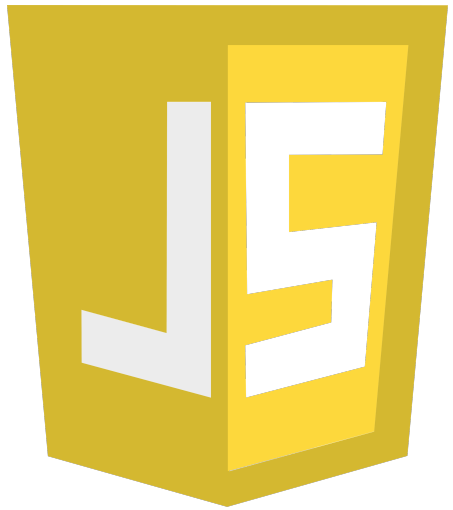
Classes

- Classes are logical objects that contain functions and properties.
- They are intended to encapsulate a functionality through a data structure, providing initial values and behaviors.
  - “Special functions” (see how it's "called")
  - Declaration must come before usage
- Classes can be declared similarly to functions

```
1.  // Class declaration
2.  class Person {
3.    constructor(name) {
4.      this.name = name;
5.    }
6.  }
7.  let user = new Person(“alex”)
8.  console.log(user.name) // Prints alex
```

```
1.  // Class expression
2.  let Person = class {
3.    // …
4.  }
```

Objects



Why are objects important?

- Objects in JavaScript are data types that share built-in properties inherited from the core *Object* class. These include Strings, Numbers, Arrays, Maps, and more.
- Objects are variables that can contain many values.

```
let person = {
    name: 'John',
    age: 20
};
```

Keys → `name`, `age` ; Values → `'John'`, `20`

Objects

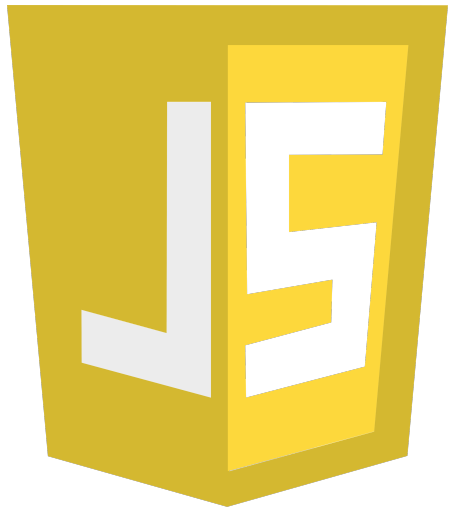
When an object in JS is passed to a function, it passes **a copy of the reference** to that object.

```
1.  // Modify the passed object
2.  var obj = { a: "wrong" };
3.  function modify(o) {
4.    o.a = "correct";
5.  }
6.  modify(obj);

7.  // Prints “correct”
8.  console.log(obj.a);
```

```
1.  // Do not do this
2.  var obj = { a: "wrong" };
3.  function modify(o) {
4.   o = { a: "correct" };
5.  }
6.  modify(obj);

7.  // Prints “wrong”
8.  console.log(obj.a);
```

```
1.  // Return a new object
2.  var obj = { a: "wrong" };
    function modify(o) {
3.   let copy = Object.assign({}, o);
4.   copy.a = "correct";
5.   return copy;
6.  }

7.  // Prints “correct”
8.  console.log( modify(obj).a );
```

Scoping



Why is scoping important?

- Determines the accessibility/visibility of variables.
- Provides a level of security to code by preventing access to objects where they are not meant to be used.
- Helps track down bugs, increase the reusability of code, and solve naming problems.

Scoping

Lexical

A.K.A. *Static scope*

- Inner functions retain the scope of the parent.

Global

Variables declared outside of a function are globally accessible to the webpage.

```
let n, m;
function test1() {
  console.log("n: ", n);
}
function test2(n) {
  console.log("n: ", n);
  test1();
}

n = 1;
m = 2;
test1();
test2(m);
```

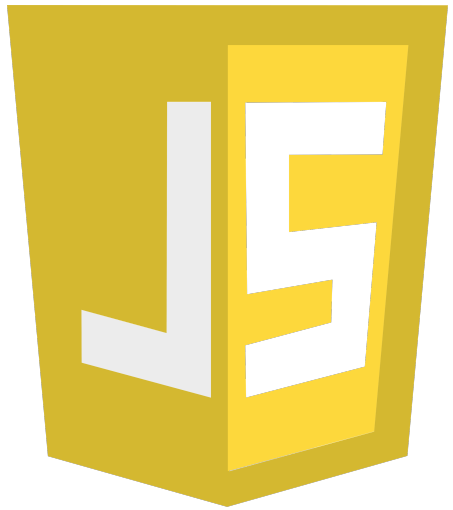
Functional

Functions create a new local scope:

- Variables defined inside functions are not accessible outside.

```
14:37:32.666 n:  1
14:37:32.666 n:  2
14:37:32.666 n:  1
```

Asynchronous Calls



Why are asynchronous calls important?

- Asynchronous calls are a key part of event-driven programming and user interfacing.
- User input is handled efficiently with events rather than looping checks.

Asynchronous Calls

JavaScript is **single-threaded,** meaning it will not spin up new threads to handle events.

So, how do we deal with events?

`Callbacks`

Callback functions are executed in an asynchronous way when an event is triggered:

- `onClick={handleClick()}`

`Promises`

Promises are an object that wraps an asynchronous operation:

- `then()`
- `catch()`

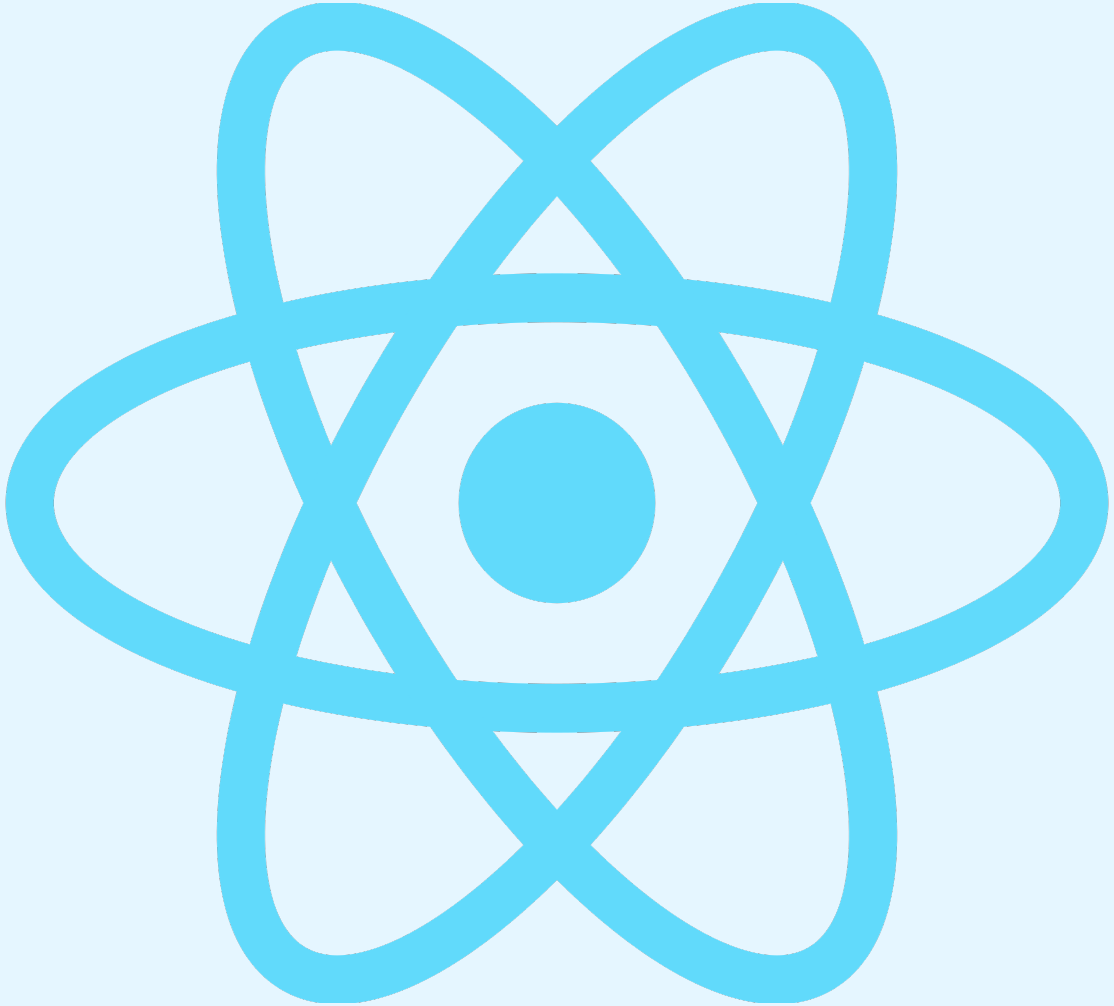
`Async/Await`

Keywords to handle asynchronous operations:

```
async function get(foo) {
  return await result(foo)
}
```

React Deep-Dive

Garrett Bolter
UI Developer
MultiCloud SaaS Platform
IBM Software



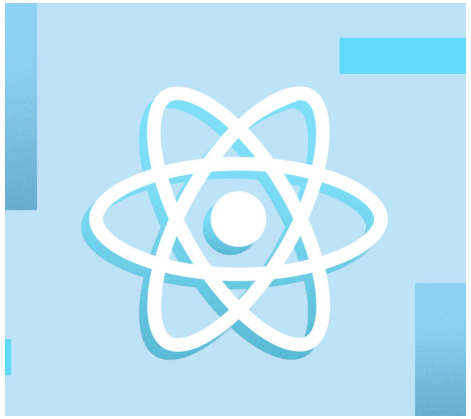
IBM

What do we know about React?

App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <div className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h2>Welcome
        </div>
        <p className="
          To get start
        </p>
      </div>
    );
  }
}

export default App;
```

```
localStorage
location
locationbar
logo                    import logo
Location
LongRange
LongRunningScriptDetectedEvent
MSLocalClientEvent
MSLocalClientEventBase
HTMLObjectElement
HTMLOListElement
HTMLOptGroupElement
```



```
to-do-list
  node_modules
  public
  src
    assets
    component
      navigation
      AddTodo.js
      todos.css
      todos.js
    pages
      About.css
      About.js
      Home.css
      Home.js
    AddTodo.test.js
    App.css
    App.js
    App.test.js
    index.js
    setupTests.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

- React is an open-source JavaScript library developed by Facebook.
- React is a User Interface (UI) library.
- React is a tool for building user interface (UI) components.
- React creates a virtual DOM in memory, where it does all the necessary manipulating before making the changes in the browser DOM.
- React only changes what needs to be changed.

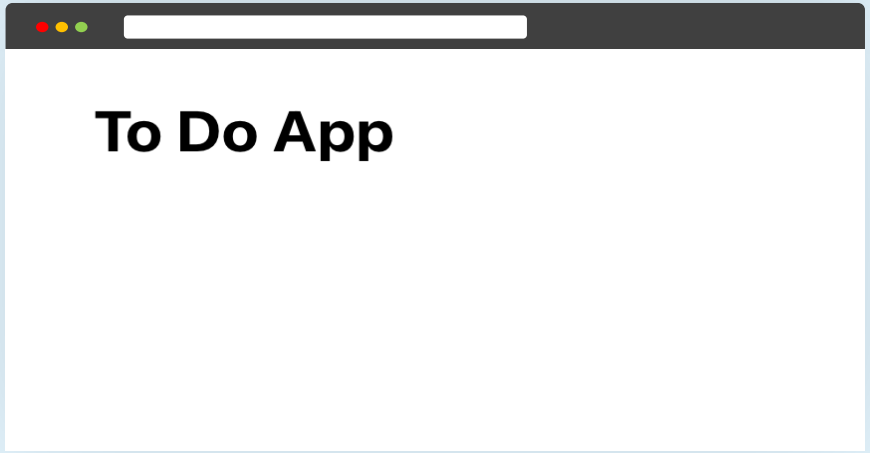
React:
Rendering Elements

- Elements are the smallest building block of a React application.
- React DOM controls the rendering of elements.
- When developing a React application, there should be one “root” element.
- To render a React element, first pass the DOM element to ReactDOM.createRoot(), then pass the React element to root.render():

```
const root = ReactDOM.createRoot(
  document.getElementById('root')
);
const element = <h1>Hello, world</h1>;
root.render(element);
```

```
// index.js
const element= <h1>To Do App</h1>;
ReactDOM.render(element,
document.getElementById('root'));
```

```
// index.html
<div id="root"></div>
```



React:
Components and Props

React components accept a single argument (called *props*) and return a React element.

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

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

React: Events

React can perform actions based on user events. It has the same events as HTML: click, change, mouseover, etc.

- Events are written in camelCase syntax.
- Event handlers are written inside curly braces.
- Arrow functions are used to pass an argument to an event handler.
- Event handlers have access to the React event that triggered the function.

```
1.  // Button event
2.  // Different syntax than JSX events
3.  <button onClick={doSomething}>
4.    Click me
5.  </button>
```

```
1.  // Passes event “e”
2.  // Arrow functions to pass parameters
3.  <button onClick={(e) => this.doSomething(e)}>
4.    Click me
5.  </button>
```

React:
List and Keys

Lists can be rendered in React by passing list items to a list element

```
1.  // List example
2.  const userItems = users.map((number) =>
3.    <li key={user.id}>
4.      {user.name}
5.    </li>
6.  );
7.  ReactDOM.render(
8.    <ul>{userItems}</ul>,
9.    document.getElementById('root’)
10. );
```

Keys should:

- Give a stable identity to an item
- Be unique among siblings

With keys, items can be added/removed from the original list (in our example, users) easily.

list.map is useful for performing transformations on lists.

React:

Hooks

Hooks allow function components to have access to state and other React features.

1

State Hooks

State lets a component “remember” information like user input.

- useState
- useReducer

2

Context Hooks

Context lets a component receive information from distant parents without passing it as props.

- useContext

3

Ref Hooks

Refs let a component hold some information that isn’t used for rendering. (helpful for counters)

- useRef
- useImperativeHandle

4

Effect Hooks

Effects let a component connect to and synchronize with external systems.

- useEffect

5

Performance Hooks

A common way to optimize re-rendering performance is to skip unnecessary work.

- useMemo
- useCallback

6

Others

React:
State and Lifecycle

React function components handle states with the useState react hook.

Declaring a state

const [count, setCount] = useState(0);

Here we declared a variable called `count` (with an initial value of `0`) and a setter function called `setCount`.

Updating state

Call setCount(100) to update the previously declared state (`count`) to `100`.

This will re-render the component.

Using state

In function components, you can simply reference the state value from the variable you declared. In our case `count`.

React:
State and Lifecycle (Cont.)

React class components should never modify their own props. Use **state** instead.

Setting state

The state of a component is declared in the constructor

```
import React, { Component } from 'react';

export default class App extends Component {
    constructor(props) {
        super(props);
        this.state = {
            text: 'Hello World'
        };
    }
    render() {
        return (
            <div>
                {this.state.text}
            </div>
        );
    }
}
```

Updating state

Call this.setState(*text*) to update the component's current state.

This will re-render the component.

Using state

The component's state can be accessed with:

this.state.<property>

this.state.text

React:
State and Lifecycle (Cont.)

In react function components, lifecycle functions are mostly handled with the useEffect react hook.

Mounting

```
useEffect(()=> {
  Console.log("component mount
ted")
})
```

Updating

```
useEffect(()=> {
  Console.log("component
updated")
})
```

Unmounting

```
useEffect(()=> {
  return () => {
    Console.log("unmounted"
  }
}
```

React:
State and Lifecycle (Cont.)

Lifecycle methods get called during three stages.

Mounting

1. `constructor()`
2. `getDerivedStateFromProps()`
3. `render()`
4. `componentDidMount()`

Updating

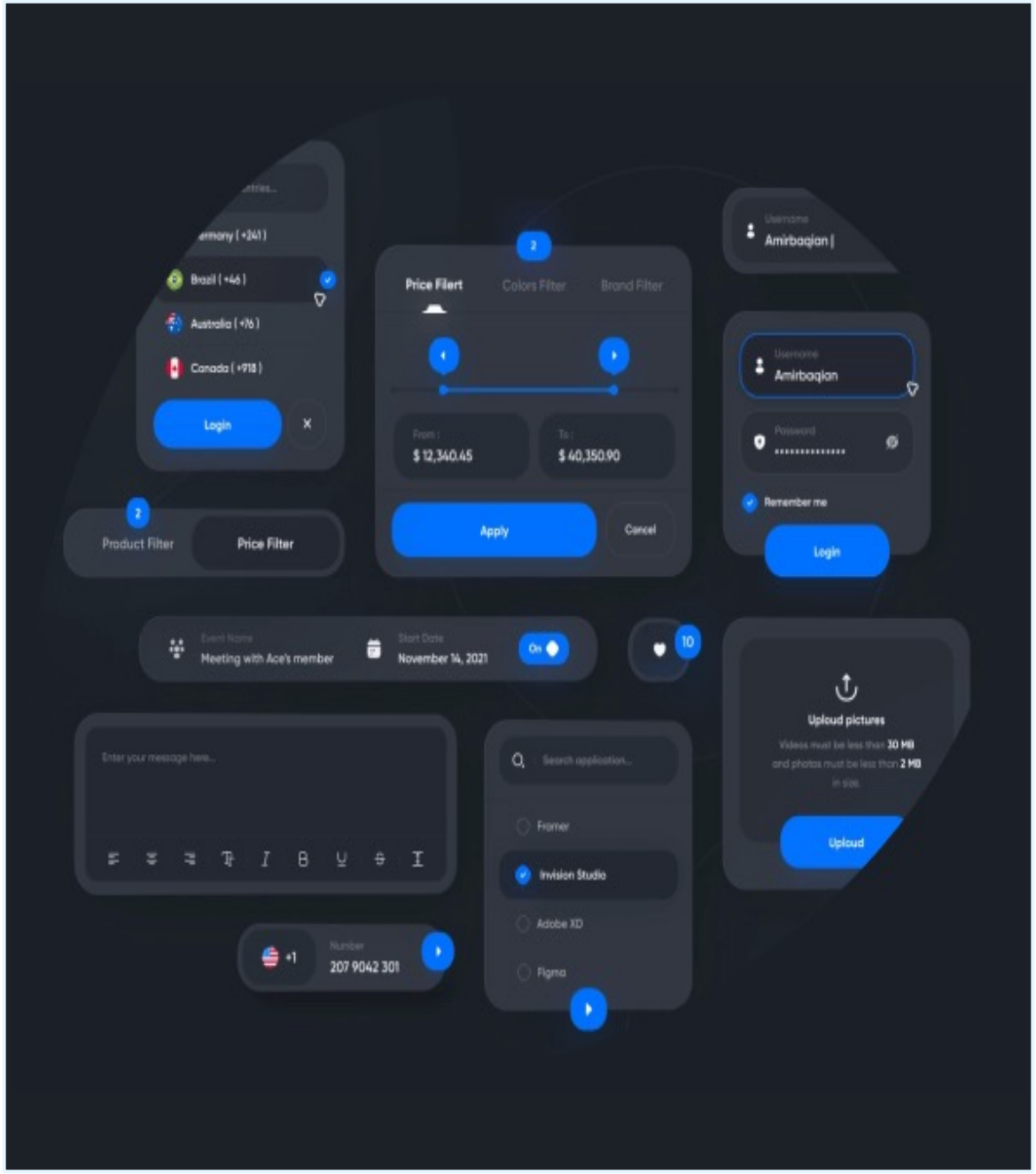
1. `getDerivedStateFromProps()`
2. `shouldComponentUpdate()`
3. `render()`
4. `getSnapshotBeforeUpdate()`
5. `componentDidUpdate()`

Unmounting

1. `componentWillUnmount()`

Design and Styling

Eram Manasia
Software Engineer,
Sales Technology Engineering, Finance and Operations



UX and UI Design

What is UX design?

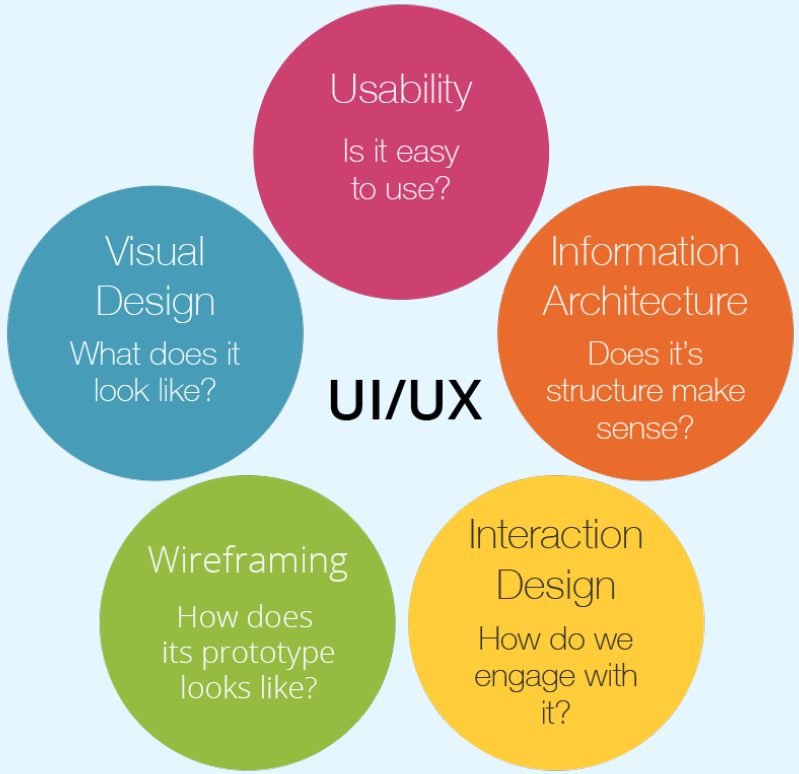
- User experience is the user's overall experience with a company’s products or services. Good and bad user experience design is determined by how easy or difficult it is to interact with each element or aspect of a product or service.

What is UI design?

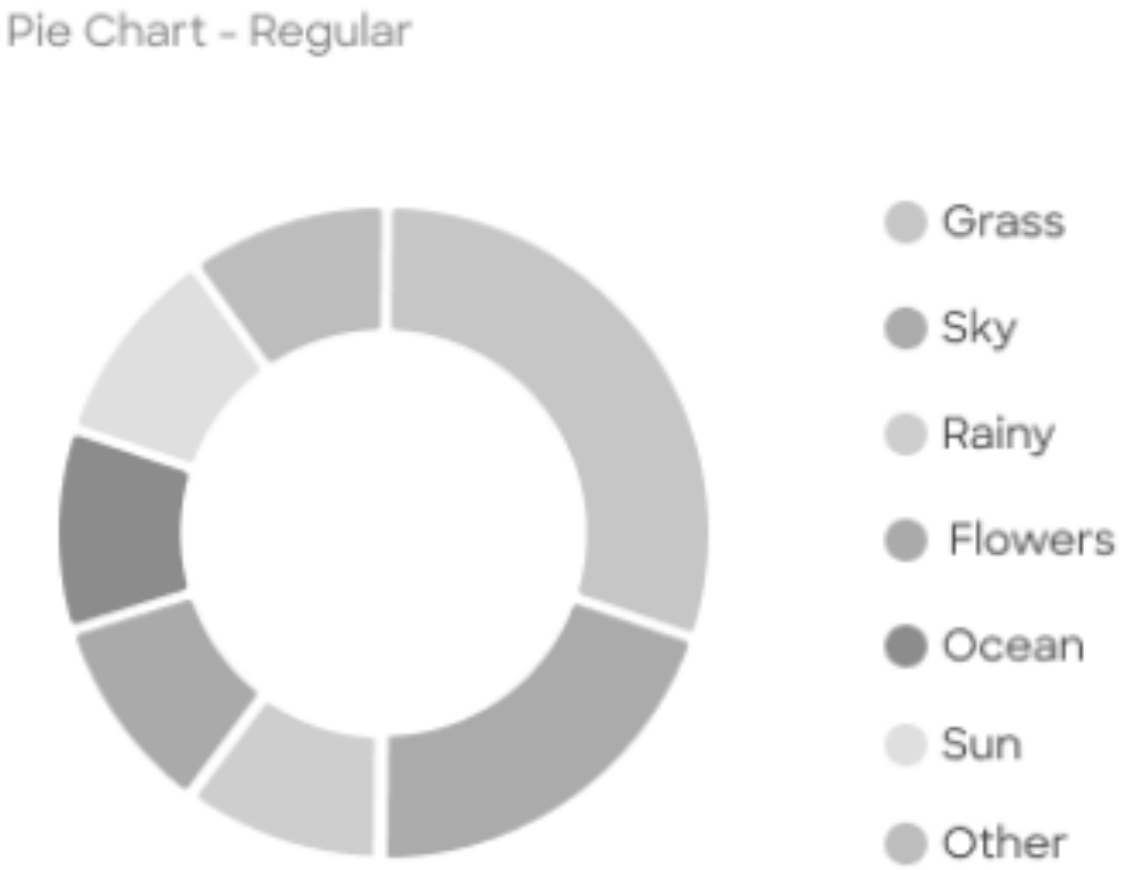
- The layout consists of how each element of the product will look, including buttons, placeholders, text, images, checkboxes, and any visual interface elements people interact with.

Why is it important?

- The UX/UI design of the application improves the user experience and customer satisfaction, which ultimately helps increase the number of users of the specific application.
- When working as a Frontend Developer, there could be a dedicated team helping to achieve an optimal user experience.



UX and UI Design (Cont.)



Accessibility

What is web accessibility?

- Ensuring that your application is able to be used by people with different abilities. Example: If you have a video on your site, having closed captions will allow people with hearing loss to be able to understand what is going.

Why is accessibility important?

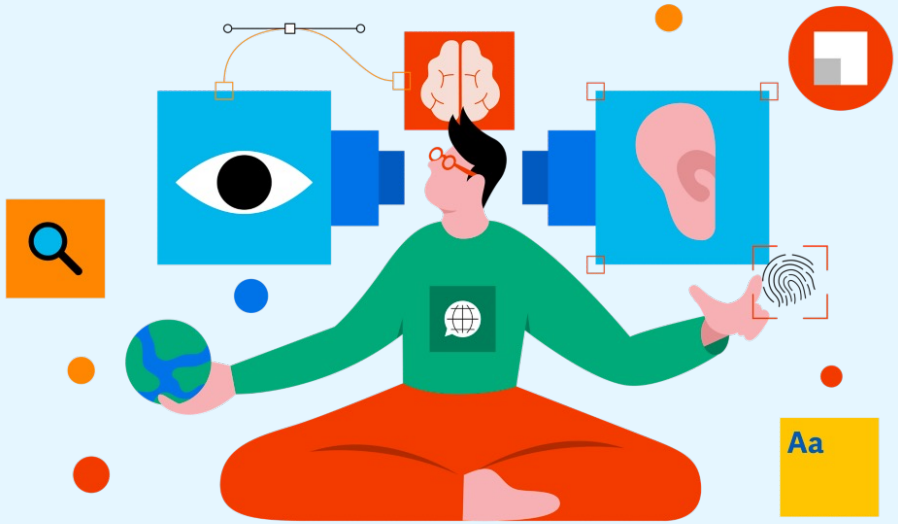
- It is important to be inclusive. Our audiences are from different backgrounds and ensuring a good user experience for them is vital.

When should accessibility be worked on for an application?

- The entire application lifecycle.

Are there any tools for accessibility?

- Yes: https://www.ibm.com/able/toolkit/tools

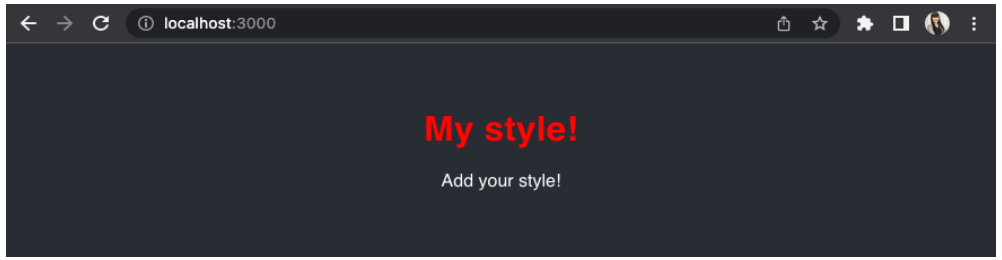


Styling React

There are different ways to style a React application.

Method #1:
Inline styling

- We can add inline styles to any React component we want to render.
- These styles are written as attributes and are passed to the element.



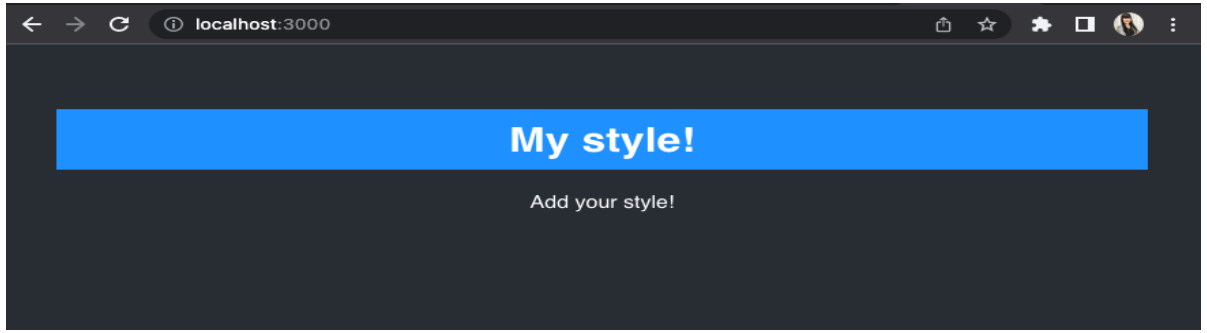

```
// Object with styling
const Header = () => {
  return (
    <>
      <h1 style={{color: "red"}}>My style!</h1>
      <p>Add your style!</p>
    </>
  );
}
```

Styling React (Cont.)

There are different ways to style a React application.

Method #1:
Inline styling (with style object variable)

- We create a style object variable the same way we create a JavaScript object.
- This object is then passed to the style attribute of the element we want to style.




```
1.  // Style object named “myStyle”
2.  const Header = () => {
3.    const myStyle = {
4.      color: "white",
5.      backgroundColor: "DodgerBlue",
6.      padding: "10px",
7.      fontFamily: "Sans-Serif"
8.    };
9.    return (
10.     <>
11.       <h1 style={myStyle}>My style!</h1>
12.       <p>Add your style!</p>
13.     </>
14.   );
15. }
```

Styling React (Cont.)

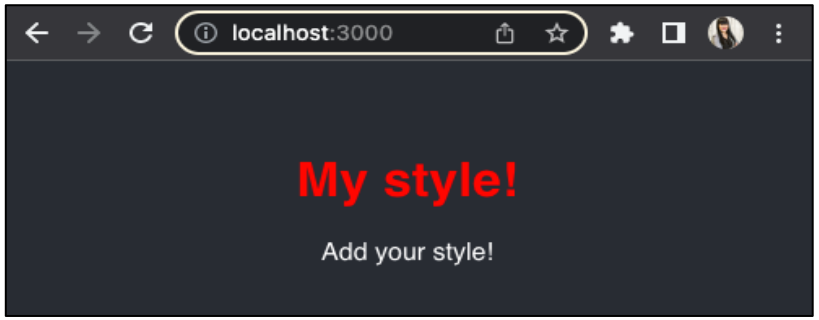
There are different ways to style a React application.

Method #2:
CSS Stylesheets

- We can write the CSS styling in a separate file (ending in .css).
- Then, import the file in the application.

```
body{
    Background-color: grey
}
```

App.css




```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './App.css';

const Header = () => {
  return (
    <>
      <h1>My style!</h1>
      <p>Add your style!</p>
    </>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Header />);
```

index.js

Styling React (Cont.)

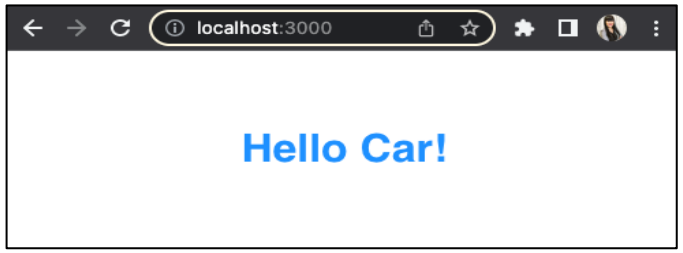
There are different ways to style a React application.

Method #3:
CSS Modules

- A CSS module is a CSS file in which all class names and animation names are scoped locally by default.
- When a CSS module is compiled, it produces two outputs:
  - CSS (modified version)
  - JavaScript object

```css
.bigblue {
  color: DodgerBlue;
  padding: 40px;
  font-family: Sans-Serif;
  text-align: center;
}
```

my-style.module.css




```jsx
import styles from './my-style.module.css';

const Car = () => {
  return <h1 className={styles.bigblue}>Hello Car!</h1>;
}

export default Car;
```

Car.js

```jsx
import ReactDOM from 'react-dom/client';
import Car from './Car.js';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car />);
```

index.js

What is Material UI? Why should you use it?

What is Material UI?

In Google’s words, the goal of Material is to: Create a visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.

Why Material UI?

1. Comprehensive
2. Familiar
3. Easy to Learn
4. Well-documented

Using Material UI

1

Install with node

```
npm install @mui/material
@emotion/react @emotion/styled
```

2

Import component

```
import Button from
'@mui/material/Button';
```

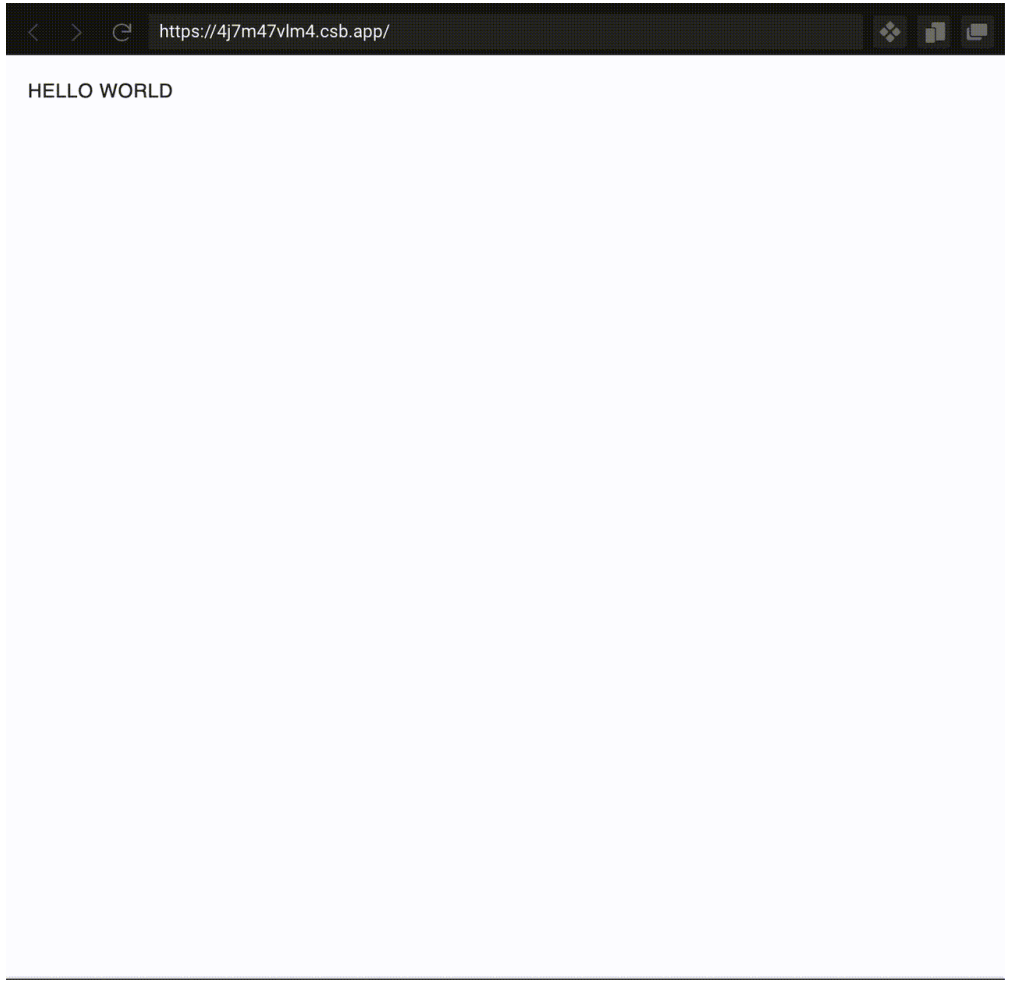
3

Use component in DOM

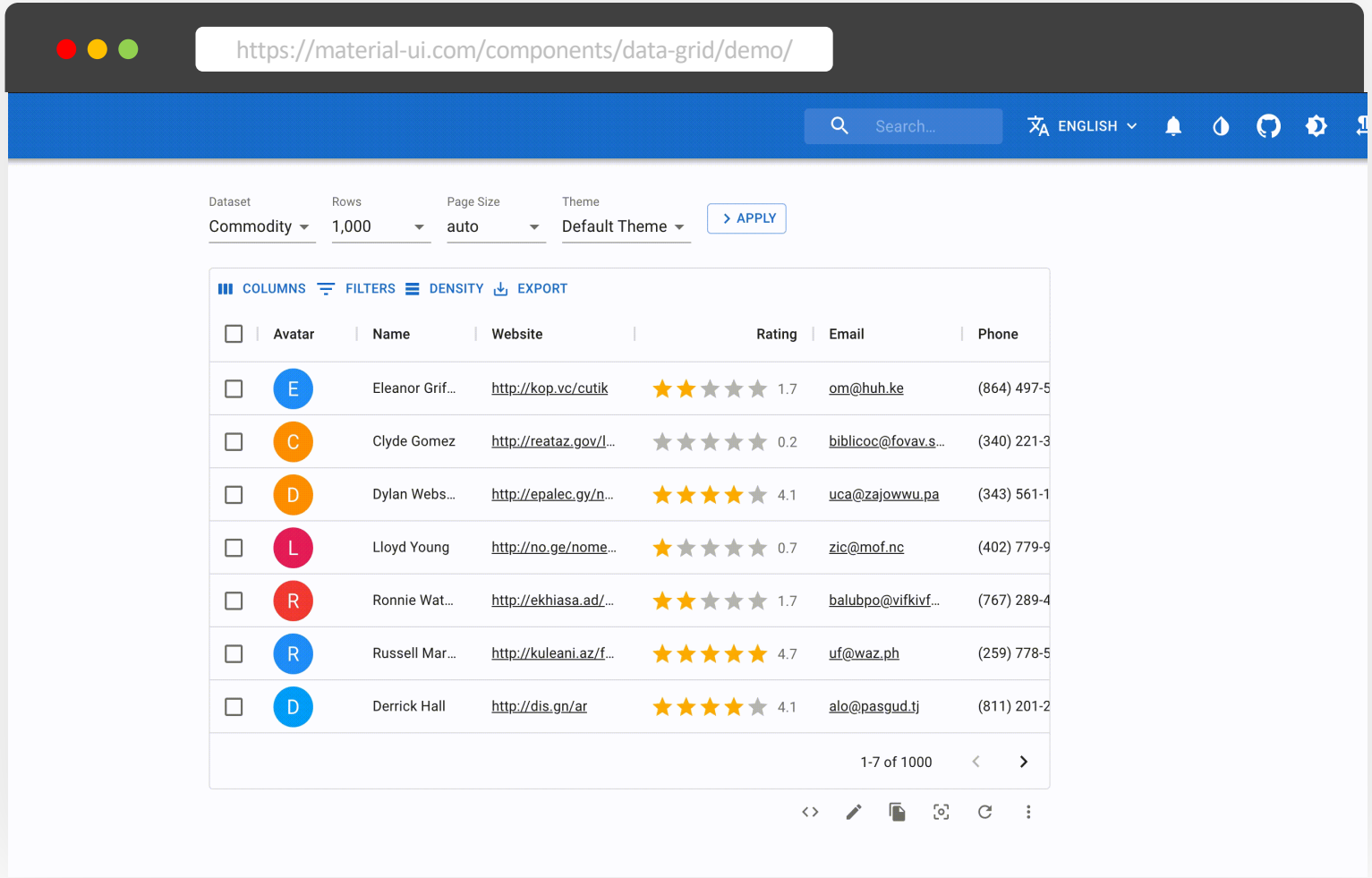
```
<Button> Hello World </Button>
```

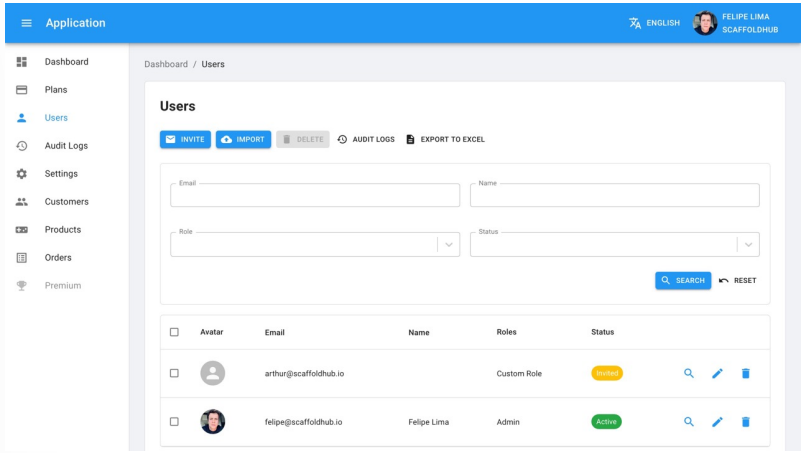
Material UI: Hello World

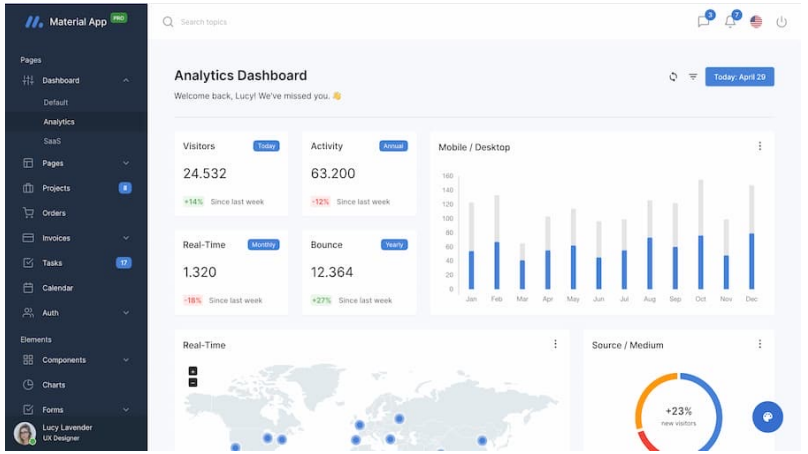
```
1.  import React from "react";
2.  import ReactDOM from "react-dom";
3.  import Button from
    '@mui/material/Button';

4.  function App() {
5.    return (
6.      <Button>
7.        Hello World
8.      </Button>
9.    );
10. }

11. ReactDOM.render(<App />,
    document.querySelector("#app"));
```



Material UI: Examples





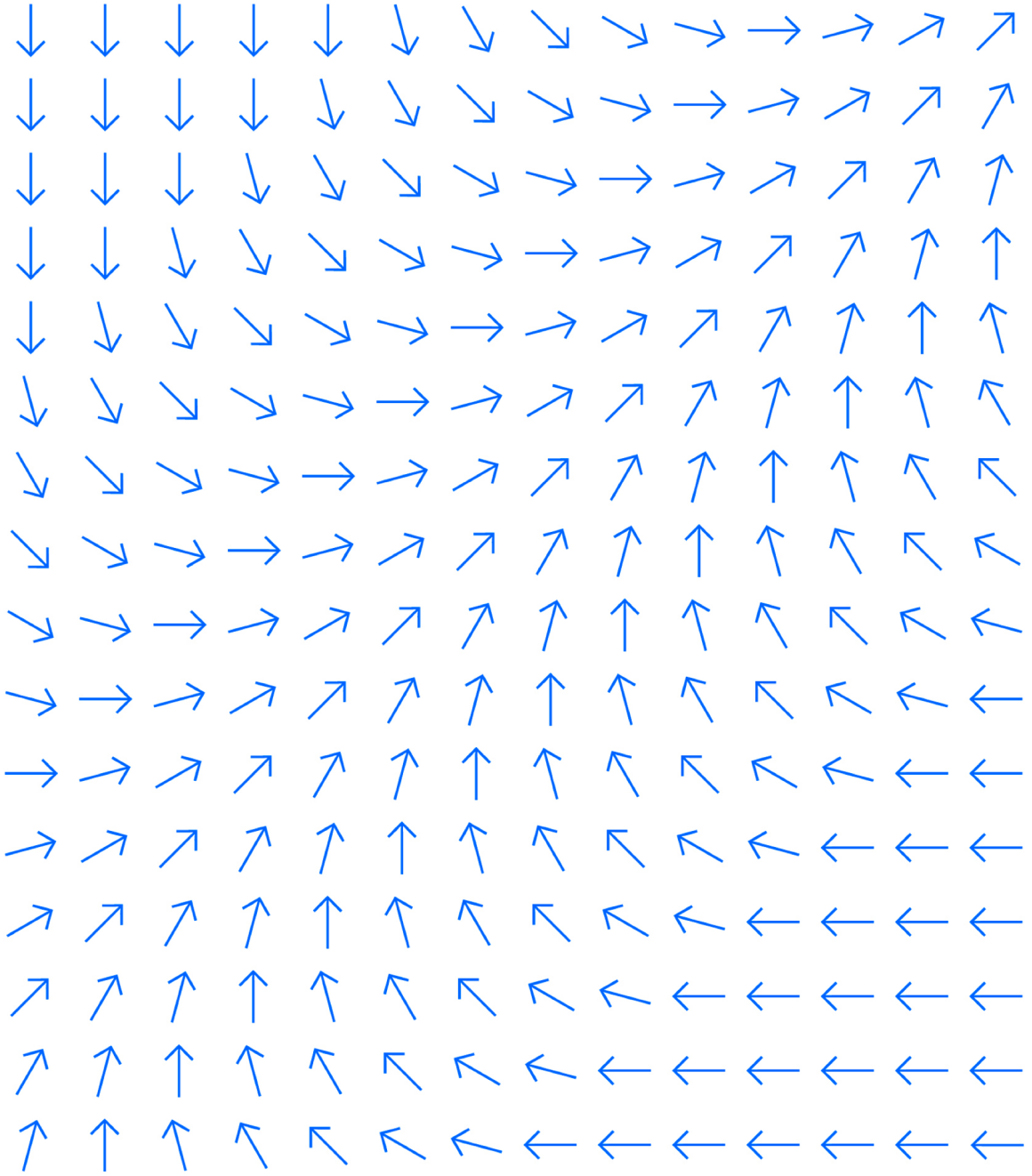


Q&A

Garrett Bolter
UI Developer,
MultiCloud SaaS Platform,
IBM Software

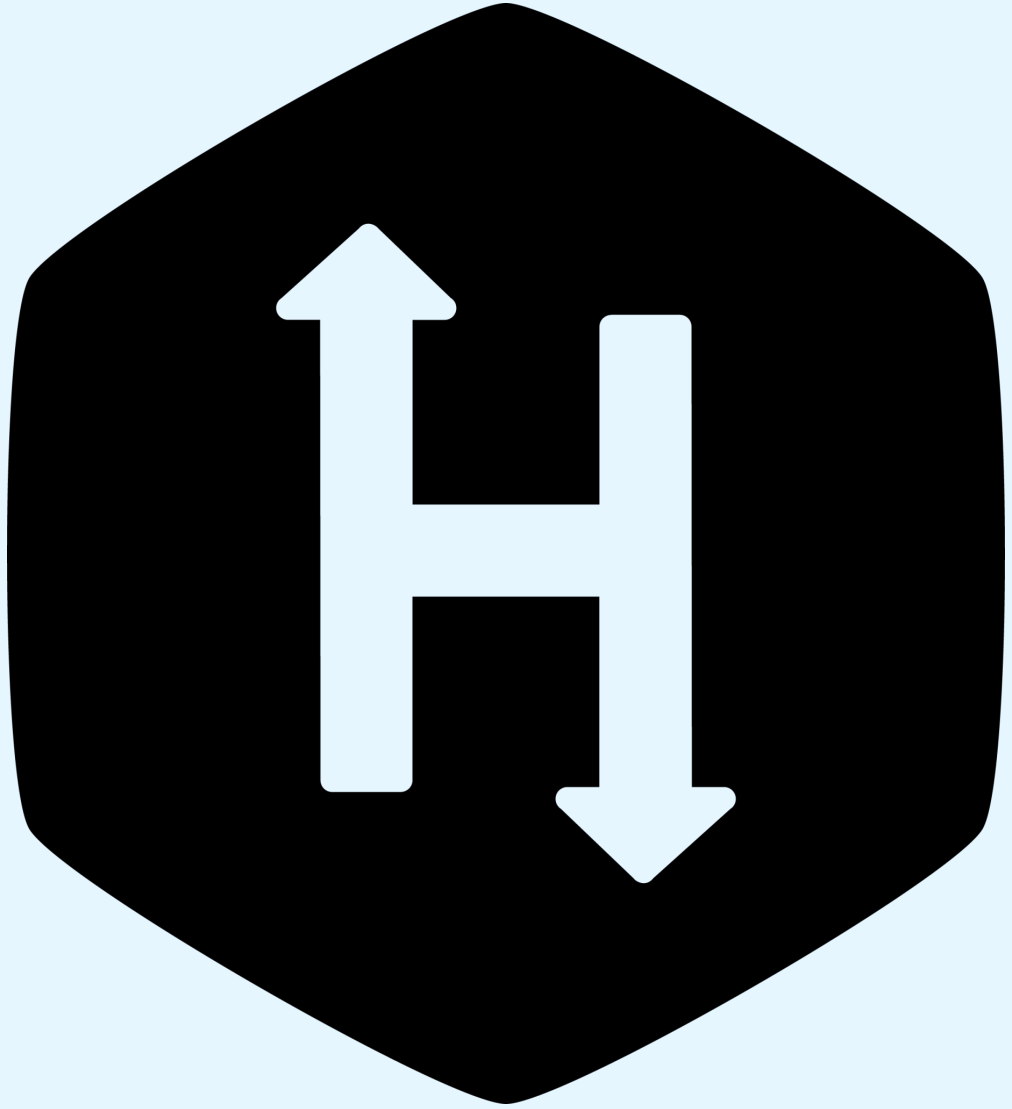
Sujeily Fonseca
Software Engineer Manager,
MultiCloud SaaS Platform,
IBM Software

Eram Manasia
Software Engineer,
Sales Technology Engineering,
Finance and Operations



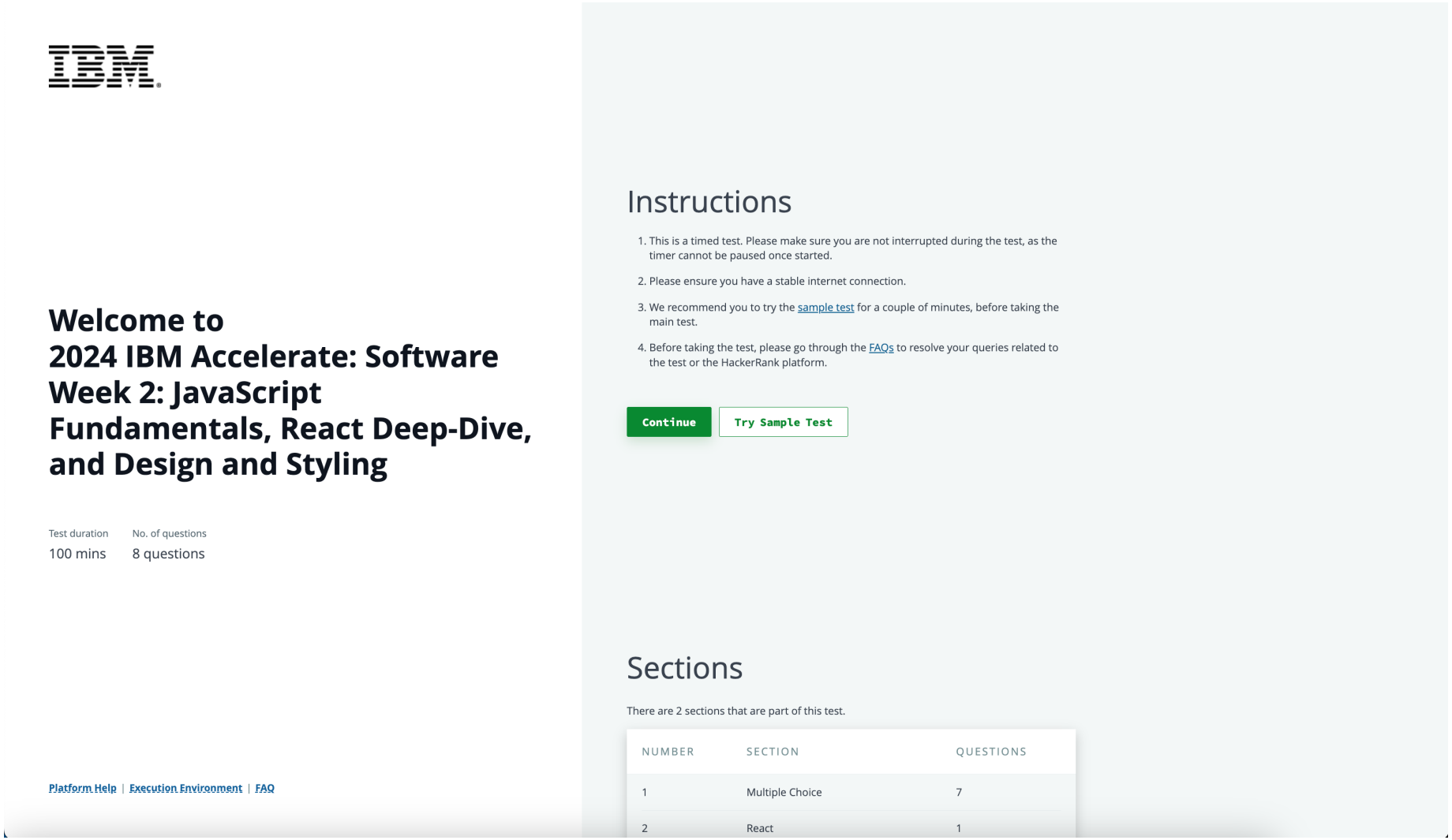
HackerRank Practice Test

Sujeily Fonseca
Software Engineering
Manager, Technical Leader,
MultiCloud Saas Platform
SW Track Lead



IBM

HackerRank Practice Test

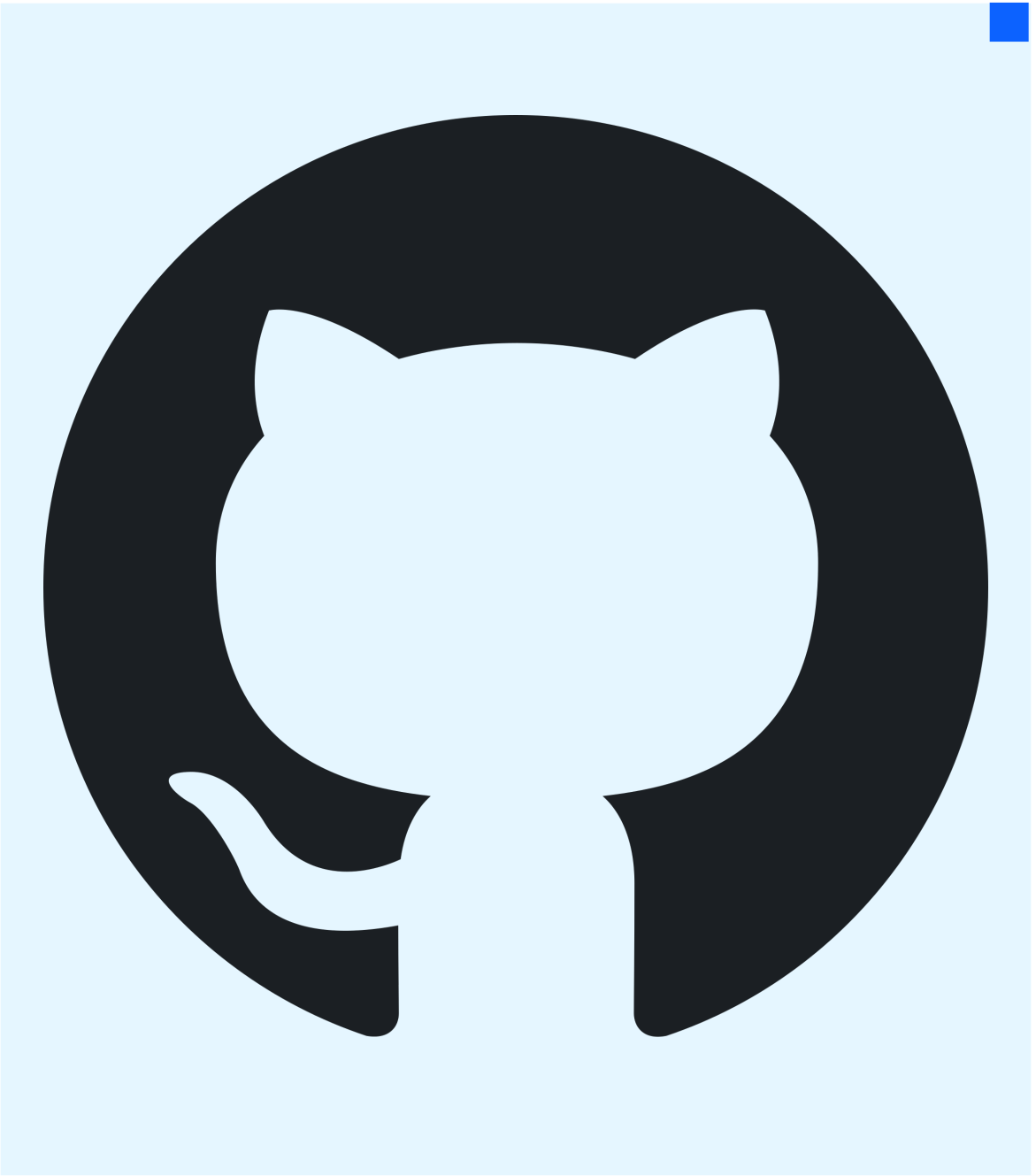


https://hr.gs/accelerate-sw-week2

(Due: June 18, 2024, 11:59 PM ET)

Project/Lab and GitHub Classroom

Sujeily Fonseca
Software Engineering
Manager, Technical Leader,
MultiCloud Saas Platform
SW Track Lead



To-do-list Lab (Week 2):
Review

- Submissions for Lab/Project 2 will close June 17, 2024, 11:59 PM ET
    - We will pull the code from your **master/main** branch at this time to provide feedback.
    - Make sure you merge any branches before then.
- Use the Slack channel and office hours for any questions.

To-do-list Lab:
Instructions

Feature Requirements:

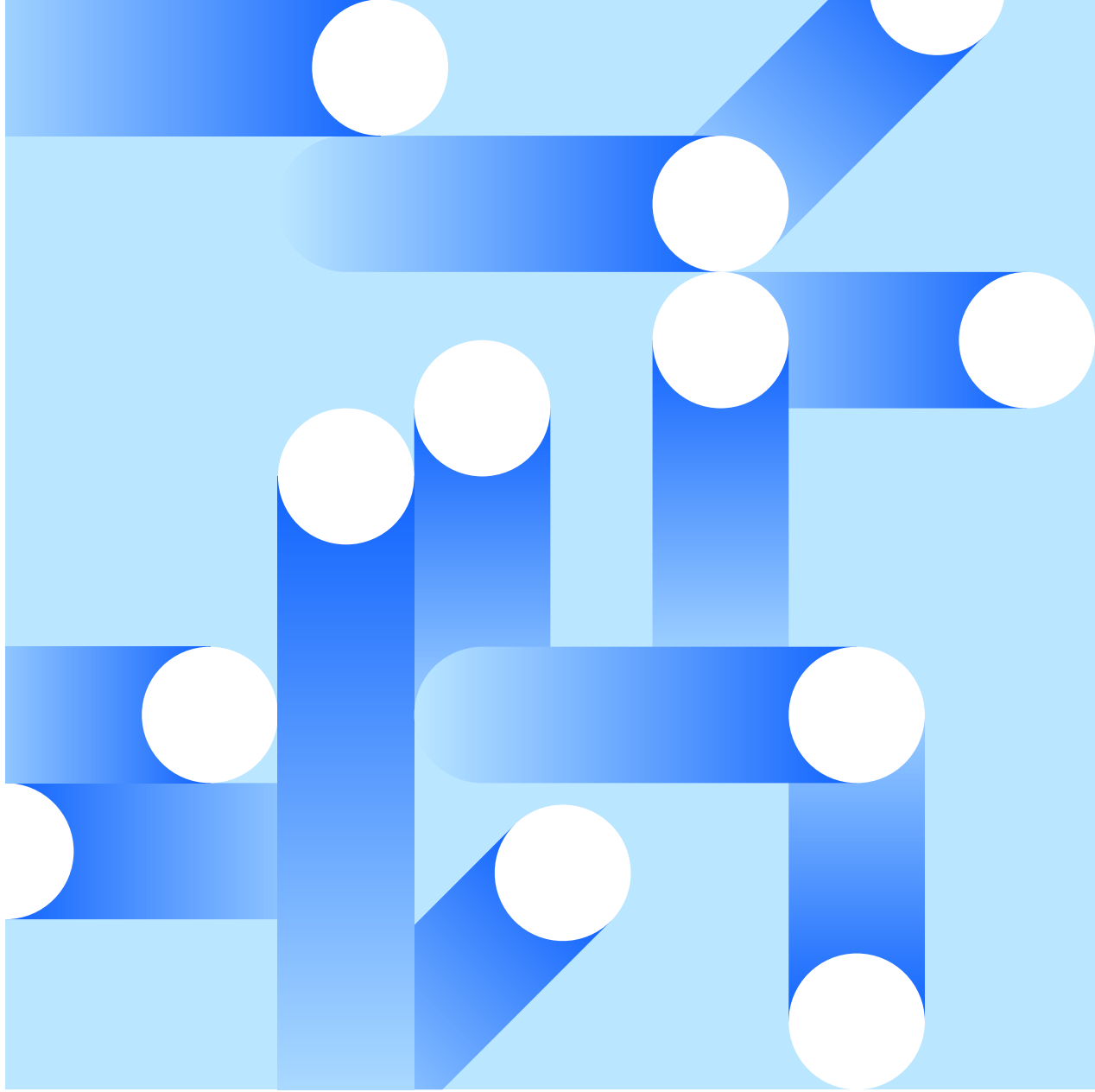
- Provide the date and time of item addition
- Allow users to mark items as complete
- Remove completed items from the list
- Validate there are no duplicated items

Implementation Requirements:

- Use Material UI components at least once throughout the app
- Use JavaScript's list.map function at least once to manipulate list items
- Implement at least one functional component

GitHub Classroom:

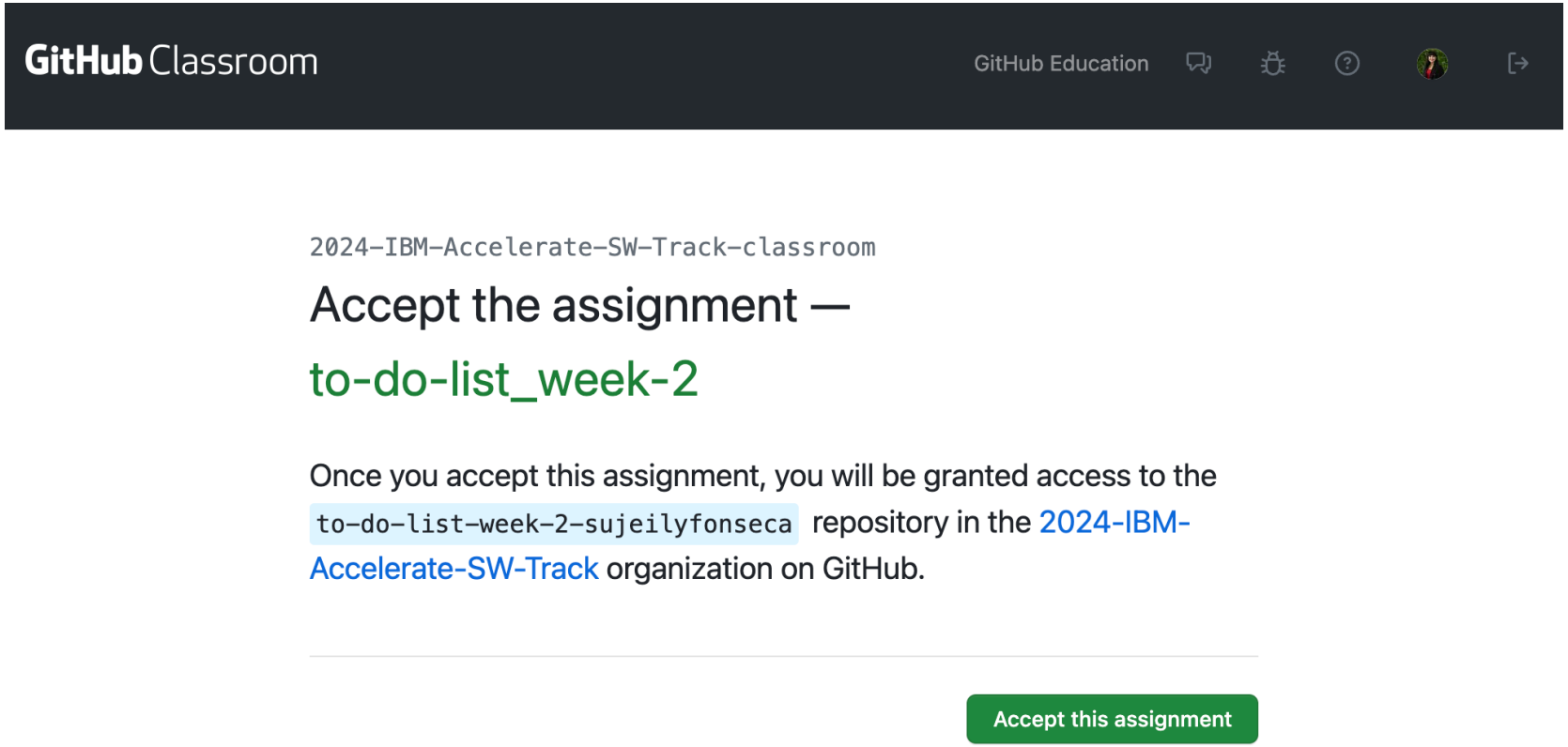
Let’s walk through the process together!



IBM

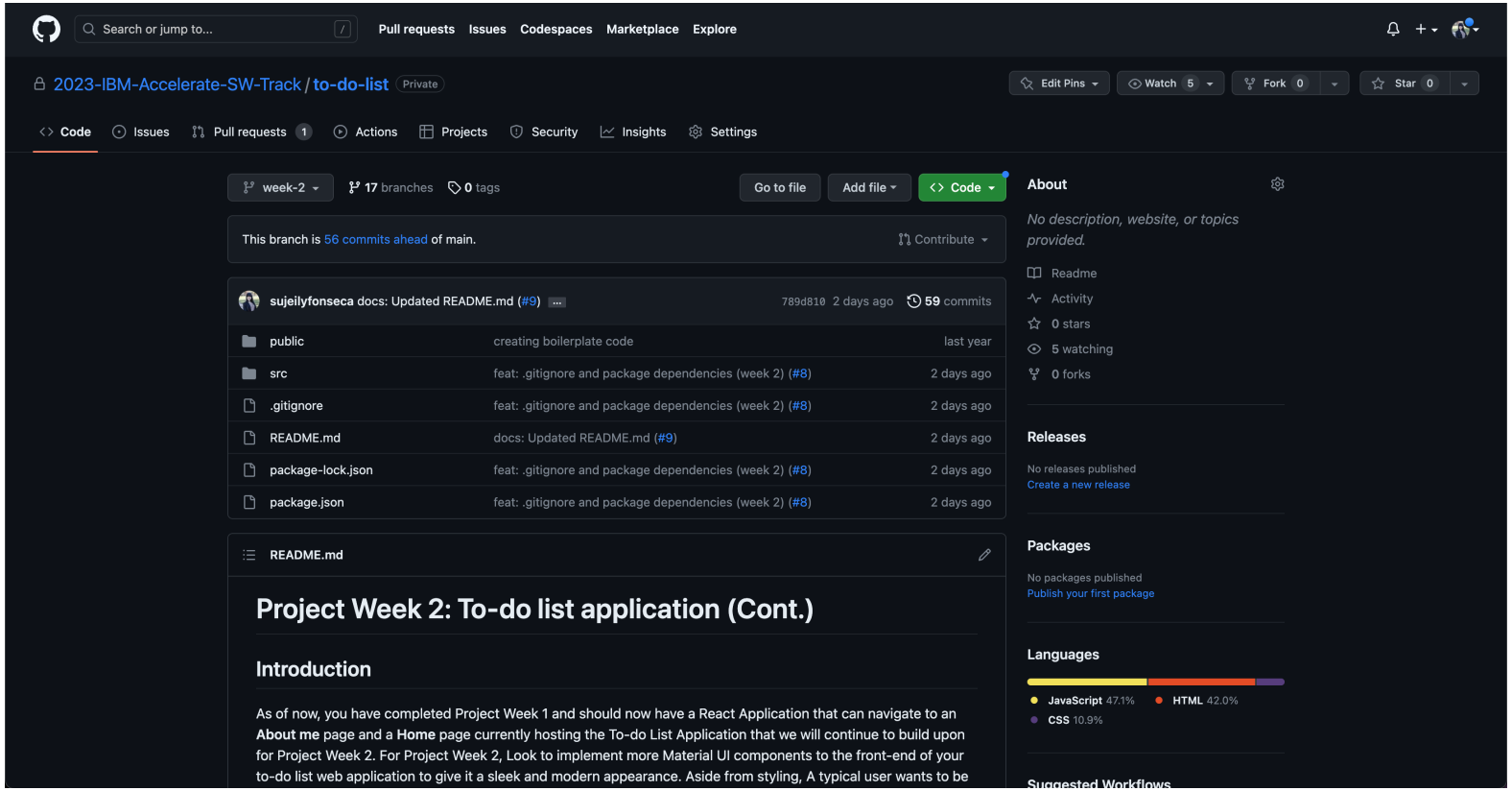
GitHub Classroom:

Let’s walk through the process together!



- Short link: https://ibm.biz/accelerate-sw-week2
- Direct link: https://classroom.github.com/a/r9S7wyqT

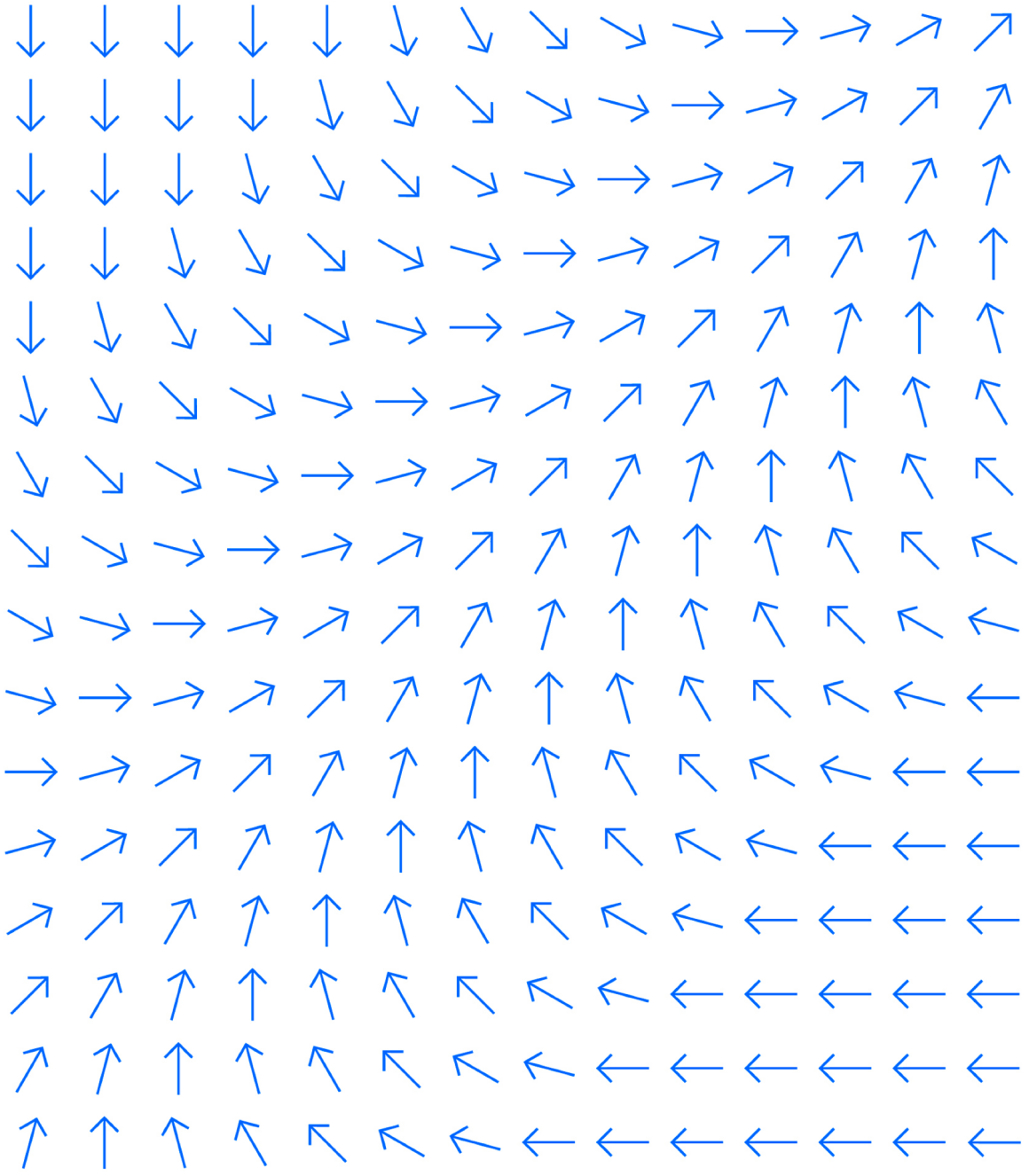
GitHub Classroom:
Automated feedback and code validation



- Automated grading/feedback was enabled.
- It will also give help you understand how you're doing.
- You can run the tests locally by executing "npm test".

Next Steps

Sujeily Fonseca
Software Engineering
Manager, Technical Leader,
MultiCloud Saas Platform
SW Track Lead



Next Steps

1. If you need help or have questions, ask in the slack channel (#ibm-accelerate-2024-software) or join the Office Hours (Thursday at 6:00 PM ET).
2. Complete your lab by Monday at 11:59 PM ET (https://ibm.biz/accelerate-sw-week2).
3. Complete the optional practice test by Tuesday at 11:59 PM ET
4. (https://hr.gs/accelerate-sw-week2).
5. Check out the solutions for the Project/Lab and HackerRank test from Week 1 (available) and Week 2 (to be made available prior to the next session).
6. Slides and video recordings of the session are available in our box folder.
7. Watch for week 3 materials to start to arrive by Monday.
8. Network and connect with your peers today and after today's session.