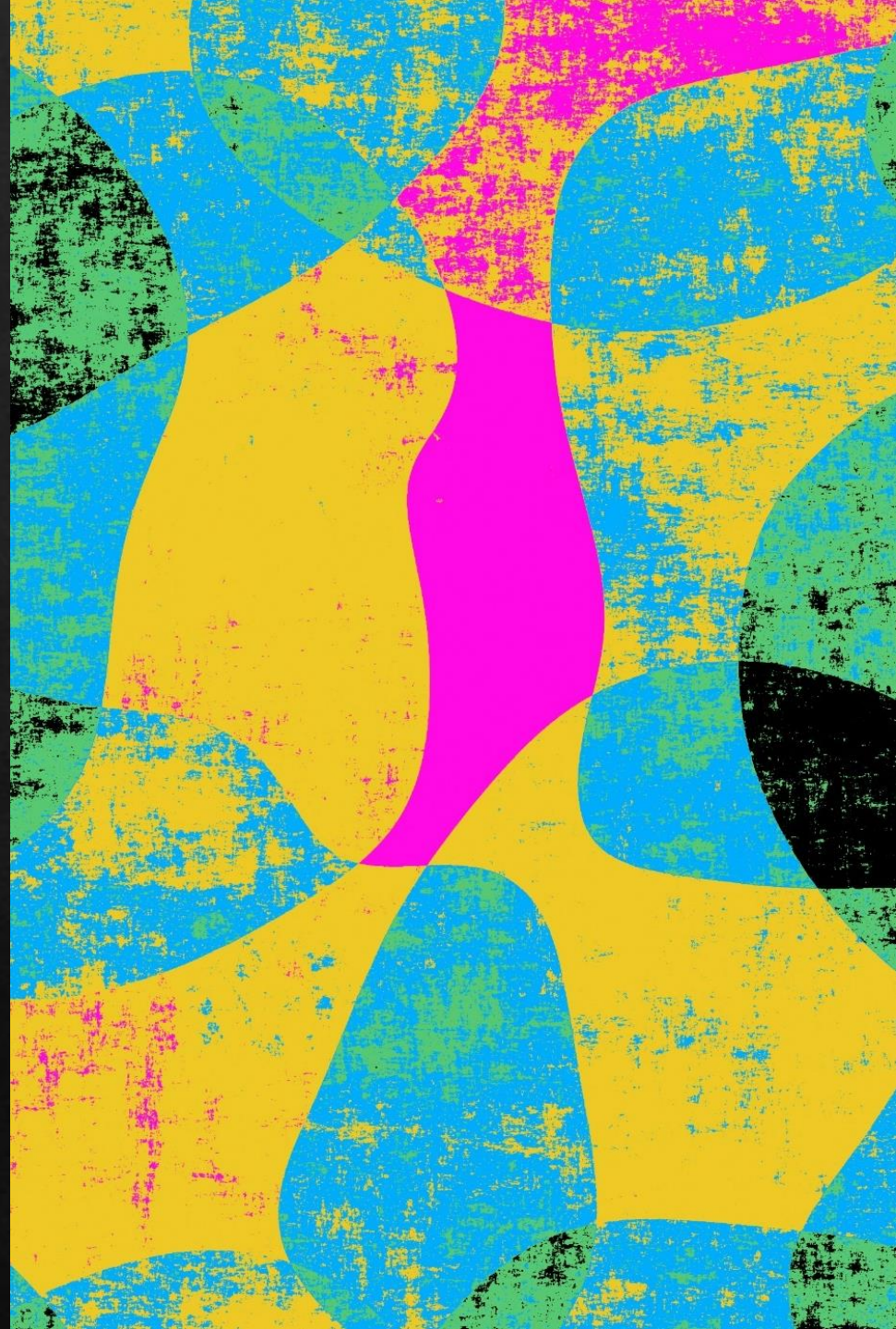
ReactJS

Component based development



Course Introduction

- Welcome to Introduction to React, the popular JavaScript library used for building user interfaces.
- Objective: By the end of this course, you should understand what React is, its benefits, main concepts, and be able to build a simple React application.
- Prerequisites: To make the most of this course, you should already have a basic understanding of HTML, CSS, JavaScript, Node.js, and npm.

Introduction to React

- React is a JavaScript library (that behaves like a framework) for building user interfaces, maintained by Facebook and the developer community.
- Why use React?
  - Speed: React allows developers to use a virtual DOM to boost performance.
  - Modularity: React promotes the use of reusable components.
  - Scalability: React is used in large-scale applications by many big companies, proving its scalability.
  - Large Community: React has a massive community of developers contributing to its growth and improvement.
- Structure of a React application: A React application is typically organized into components, which can be functional or class-based. However, as of 2023, functional components using hooks are the preferred method.

Environment Setup
&
Creating a New React App

Installing Node.js and npm

- Node.js is an environment that allows JavaScript to run on the server-side.
- npm (Node Package Manager) is a tool that facilitates package installation, version management, and dependency management.
- Both Node.js and npm are critical for setting up a React application.
- Action Points:
  1. Download and install Node.js and npm: Visit https://nodejs.org/ to download and install Node.js and npm. The website will automatically suggest the best version based on your operating system.
  2. Verify the installation: Open your terminal or command prompt and run the following commands:
     - node -v
     - npm –v
- Each command should display a version number, confirming the successful installation of Node.js and npm.

Creating a new React app using Create React App

- Create React App is a utility provided by Facebook to create a new React application skeleton.
- It sets up a functional, modern JavaScript development environment with zero configuration, thereby enabling use of the latest JavaScript features.
- It provides a local development server, powerful developer experience, and optimizes your app for production.

Creating a new React app using Create React App

- Action Points:
- Navigate to the directory where you want to create your new React project using the cd command.
- Run the following command to create a new project:
  - npx create-react-app my-app
  - Replace "my-app" with the name of your new project.
- Once the process is complete, navigate into your new project directory:
  - cd my-app
- Start the development server:
  - npm start
- npm start
- Your new React app will now open in a new tab of your default web browser. Keep the terminal running as it's hosting your development server.

Exploring the React Project Structure

- After running npx create-react-app my-app, you'll see a new directory called my-app with a predefined folder structure.
- The main folders and files you'll work with are:
- node_modules/: This directory contains all the JavaScript libraries and modules your project depends on. These dependencies are defined in the package.json file and are installed via npm.
- public/: This directory contains the static files you might need in your project that will not be processed by Webpack. The key file in this directory is index.html.
- src/: This is the heart of your React application. All of your React components, images, CSS, and JavaScript files live in here. The key file is App.js, which is the root component of your React app.
- package.json: This file contains metadata about your app, like its name and version, and the list of dependencies to install from npm when setting up the project.
- All other files and folders are configuration and script files necessary for development and can be largely ignored for now.

Overview of React Application

- React is a library for building user interfaces, primarily through building components.
- A component in React serves as a blueprint. You define a component once (like a HTML button with a specific style) and then use it as many times as needed.
- Components can be composed together to form complex user interfaces. Each component defines its own UI and logic that can be reused wherever you need them.
- React follows a unidirectional data flow, which means the data in a React application flows downward from parent components to child components.
- The interactive UIs are built in React by designing views for each state in your application, and React efficiently updates and renders the right components when your data changes.

Overview of React Application

- Action for You:
- Open your new React project in your favorite text editor. Start from src/App.js and public/index.html and follow the import statements to explore how the different components and files are connected.
- Make a small change in App.js (like changing the text in <p> tag) and save it. You should see your change instantly in your web browser. This is thanks to the hot reloading feature of Create React App.

Creating a new React app using Create React App

- Action Points:
- Navigate to the directory where you want to create your new React project using the cd command.
- Run the following command to create a new project:
  - npx create-react-app my-app
  - Replace "my-app" with the name of your new project.
- Once the process is complete, navigate into your new project directory:
  - cd my-app
- Start the development server:
  - npm start
- npm start
- Your new React app will now open in a new tab of your default web browser. Keep the terminal running as it's hosting your development server.

Overview Of React

JSX

- JSX stands for JavaScript XML. It is a syntax extension for JavaScript, which allows you to write HTML in your React JavaScript code.
- How to write JSX:

```
const element = <h1>Hello, world!</h1>;
```

- Embedding JavaScript in HTML with JSX Expressions:

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

Components

- In React, UIs are split into independent, reusable pieces called components.
- Functional components are simpler and easier to test and debug. They use “hooks” for managing state and lifecycle methods, replacing the need for class components.

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

Creating a Welcome Page

Creating a Welcome Page

- React apps are made up of components, and each page in our app can be thought of as a separate component.
- Let's start by creating a Welcome component.

Creating a new React app using Create React App

- Action for You:
  1. Create a new folder components in the src directory.
  2. Within the components folder, create a new file Welcome.js.
  3. Add the following code to Welcome.js:

```
import React from "react";

function Welcome() {
  return (
    <div>
      <h1>Welcome to Our Website!</h1>
      <p>We're glad to have you here.</p>
    </div>
  );
}

export default Welcome;
```

Creating a new React app using Create React App

- Action for You:

4. Now, import and use this Welcome component in App.js:

```
import './App.css';
import Welcome from "./components/Welcome";
function App() {
  return (
    <div className="App">
      <Welcome />
    </div>
  );
}

export default App;
```

Creating a Form Page

Creating a Form Page

- Within the components folder, create a new file FormPage.js.
- Add the following code to FormPage.js:

```jsx
import React, { useState } from "react";

function FormPage() {
  const [name, setName] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Hello, ${name}!`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input
          type="text"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}

export default FormPage;
```

Creating a Form Page

3. Now, import and use this FormPage component in App.js:

```
import Welcome from "./components/Welcome";
import FormPage from "./components/FormPage";
import './App.css';
function App() {
  return (
    <div className="App">
      <Welcome />
      <FormPage />
    </div>
  );
}

export default App;
```

Creating a Form Page

3. Now, import and use this FormPage component in App.js:

```
import Welcome from "./components/Welcome";
import FormPage from "./components/FormPage";
import './App.css';
function App() {
  return (
    <div className="App">
      <Welcome />
      <FormPage />
    </div>
  );
}

export default App;
```

Creating a Form Page

- Go to your web browser, and you should see your Welcome message followed by the form. Try entering a name in the form and submitting it. You should see an alert with your name.

Styling and Separating Form Page

Styling the FormPage

- Action for You:
- Create two new folders inside the components directory: Welcome and FormPage.
- Move the Welcome.js to the Welcome folder, and FormPage.js to the FormPage folder.
- In the FormPage directory, create a new file FormPage.css.
- Add the following CSS rules in FormPage.css:

Styling the FormPage

```css
.form-container {
  margin: auto;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  width: 300px;
  padding: 20px;
  background-color: #f2f2f2;
  border-radius: 5px;
  box-shadow: 0px 0px 15px 0px rgba(0, 0, 0, 0.2);
}

.form-container form {
  display: flex;
  justify-content: center;
  flex-direction: column;
}
.form-container input {
  height: 30px;
  border-radius: 5px;
  border: 1px solid #ccc;
  padding: 5px;
  margin-bottom: 10px;
}
```

Styling the FormPage

- Action for You:
- Import this CSS file in FormPage.js and use the form-container class. Don't forget to correct the import path since we've moved files around:

Styling the FormPage

```jsx
import React, { useState } from "react";
import "./FormPage.css";

function FormPage() {
  const [name, setName] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Hello, ${name}!`);
  };

  return (
    <div className="form-container">
      <form onSubmit={handleSubmit}>
        <label>
          <input
            placeholder="Enter your name"
            type="text"
            value={name}
            onChange={(e) => setName(e.target.value)}
          />
        </label>
        <input type="submit" value="Submit" />
      </form>
    </div>
  );
}

export default FormPage;
```

Implementing Routing

Implementing Routing

- We will use the react-router-dom library to enable routing in our React app.
- Action for You:
  - Install react-router-dom by running the following command in your terminal:
  - npm install react-router-dom
- Run your app again:
  - npm start

Implementing Routing

- Setup the routing in App.js. We will use the BrowserRouter, Routes, and Route components from react-router-dom. Make sure to adjust the import paths for Welcome and FormPage:

```jsx
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Welcome from "./components/Welcome/Welcome";
import FormPage from "./components/FormPage/FormPage";
import './App.css';
import React from "react";
function App() {
  return (
    <Router>
      <div className="App">
        <Routes>
          <Route index path="/" element={<Welcome />} />
          <Route path="/form" element={<FormPage />} />
        </Routes>
      </div>
    </Router>
);
}
```

Implementing Routing

- Update Welcome.js:

```
import React from "react";
import { Link } from "react-router-dom";
function Welcome() {
  return (
    <div>
      <h1>Welcome to Our Website!</h1>
      <p>We're glad to have you here.</p>
      <Link to="/form">Go to Form Page</Link>
    </div>
  );
}

export default Welcome;
```

Implementing Routing

- Update FormPage.js:

```jsx
import React, { useState } from "react";
import "./FormPage.css";
import { Link } from "react-router-dom";
function FormPage() {
  const [name, setName] = useState("");

  const handleSubmit = (event) => {
    event.preventDefault();
    alert(`Hello, ${name}!`);
  };

  return (
    <div className="form-container">
      <h1>Welcome to Our Form!</h1>
      <form onSubmit={handleSubmit}>
        <label>
          <input
            placeholder="Enter your name"
            type="text"
            value={name}
            onChange={(e) => setName(e.target.value)}
          />
        </label>
        <input type="submit" value="Submit" />
      </form>
      <Link to="/">Go back to Welcome Page</Link>
    </div>
  );
}

export default FormPage;
```

Fetching Data

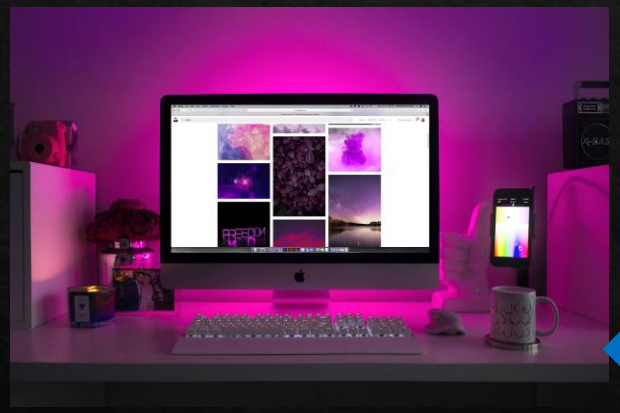
JSONPlaceholder

- Today we're going to play around with a handy tool called JSONPlaceholder. It's a free online service that gives us fake data for testing. The best part? It's easy to use and no sign-up required.
- We're going to write a function that uses fetch to get some data from JSONPlaceholder. Specifically, we want to grab a user's name.
- This is a great way to learn how we can ask for data from a website (that's what fetch does) and then use that data in our app. Once we've nailed this, we can do all sorts of cool things like getting real data from all sorts of places!
- https://jsonplaceholder.typicode.com

Behind the Scenes

React

Server



POST
GET
PUT
DELETE

CREATE
READ
UPDATE
DELETE

Initial FormPage Component to Current Version

- Initially, our FormPage component was a simple form that accepted a user's name and displayed an alert on submission. This component served as a basic input form and didn't interact with any external data.
- However, to make this application more dynamic and to expose the students to handling API requests in React, we decided to refactor the component to fetch data from an API instead of just accepting user input.
- We introduced an asynchronous fetchData function to make a GET request to the 'jsonplaceholder' API. We then stored this fetched data in the component's state.

Initial FormPage Component to Current Version

- Instead of an input field for user's name, we now have an input field that displays the name field from the fetched data. This field is read-only, meaning it only displays data and cannot be edited by the user.
- A FetchButton component was added. This button triggers the fetch operation when clicked.
- In summary, we transitioned from a static form to a dynamic component that fetches and displays data from an API. This serves as a practical introduction for students to learn about API requests and asynchronous operations in React.

FetchButton Component

- Finally, we define the FetchButton component. It accepts the handleFetch function, background color bgColor, and font color fontColor as props.
- When clicked, the button calls the handleFetch function, triggering the fetch operation.

```jsx
// FetchButton.js
import React from "react";

function FetchButton({ handleFetch, bgColor, fontColor }) {
  return (
    <button
      onClick={handleFetch}
      style={{ backgroundColor: bgColor, color: fontColor }}
    >
      Fetch Data
    </button>
  );
}

export default FetchButton;
```

Creating the Fetch Function

```
// FormPage.js
import React from "react";
import { useState } from "react";
import "./FormPage.css";
import { Link } from "react-router-dom";
import FetchButton from "../FetchButton/FetchButton";

async function fetchData(url) {
  const response = await fetch(url);

  if (!response.ok) {
    throw new Error(`HTTP error! status: ${response.status}`);
  }

  return response.json();
}

function FormPage() {
```

- First, we create an async function fetchData(url) that uses the Fetch API to make a request to the provided URL.
- The function checks if the response is OK (HTTP status 200-299). If not, it throws an error.
- If the response is OK, it proceeds to parse the response as JSON and returns the resulting data.

Setting up State and Fetch Handler in the FormPage Component

- In our FormPage component, we set up a piece of state data to hold the fetched data.
- We also define a handleFetch function that calls our fetchData function with the URL of the user we want to retrieve, then sets the data state with the returned data.
- If an error occurs during fetching, it is logged to the console.

```
function FormPage() {
  const [data, setData] = useState("");
  const url = "https://jsonplaceholder.typicode.com/users/2";

  const handleFetch = () => {
    fetchData(url)
      .then((fetchedData) => setData(fetchedData))
      .catch((error) => console.error("Error:", error));
  };

  return (
    <div className="form-container">
```

FormPage Component Render

- We return a JSX structure that includes an input box displaying the fetched user's name and a FetchButton that will trigger the fetch operation when clicked.
- The input box is read-only, meaning the user cannot modify the fetched data.

```
  };

  return (
    <div className="form-container">
      <h1>Welcome to Our Form!</h1>
      <input type="text" value={data.name} readOnly />
      <FetchButton handleFetch={handleFetch} bgColor="blue" fontColor="white" />
      <Link to="/">Go back to Welcome Page</Link>
    </div>
  );
}

export default FormPage;
```

Next Steps

Full CRUD Operations with Node.js, MongoDB, or Firebase

- Now that we've set up the front-end and learned how to fetch (read) and display data from an API, the next big step is to learn how to implement the other CRUD (Create, Read, Update, Delete) and HTTP (Get Post, Put/Patch, Delete) operations. This will allow our application to interact fully with a database, not only fetching data but also creating, updating, and deleting it. There are two main paths you can take for this:

1. Node.js and MongoDB:
2. Firebase (Or another Backend as a Service):

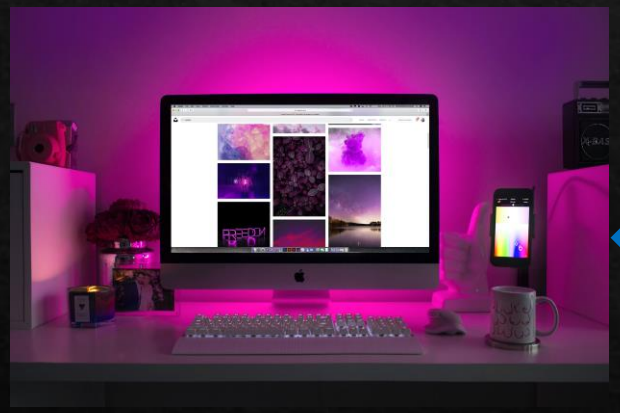
Behind the Scenes

React

Node

Firebase
(both)

MongoDB



Node.js and MongoDB

- If you prefer to build and control your own back-end, this is the route for you. With this approach, you'll gain a deep understanding of server and database management. You'll learn to:
  - Set up a server with Node.js (Slides In the classroom)
  - Define endpoints for various operations
  - Handle different types of requests and responses
  - Manage a MongoDB database (Slides In The Classroom)
- This option will require more time and effort for setup and understanding, but offers flexibility and full control.

Firebase

- If you prefer a quicker setup with less back-end management, Firebase is a great choice. As a Backend-as-a-Service (BaaS) provided by Google, Firebase provides a variety of services including a real-time database, authentication, cloud storage, and more. With Firebase, you'll learn to:
- Integrate Firebase into your application
- Use Firebase services for CRUD operations
- Handle authentication, cloud storage, and more
- This option is quicker to set up and simpler to manage, but offers less control over the back-end operations.

Full CRUD Operations with Node.js, MongoDB, or Firebase

- Set up Node.js and Mongo DB or Firebase
- Test with the Fetch you currently have
- Add in the other HTTP protocols:
  - Post
  - Put
  - Delete