



MEET APP

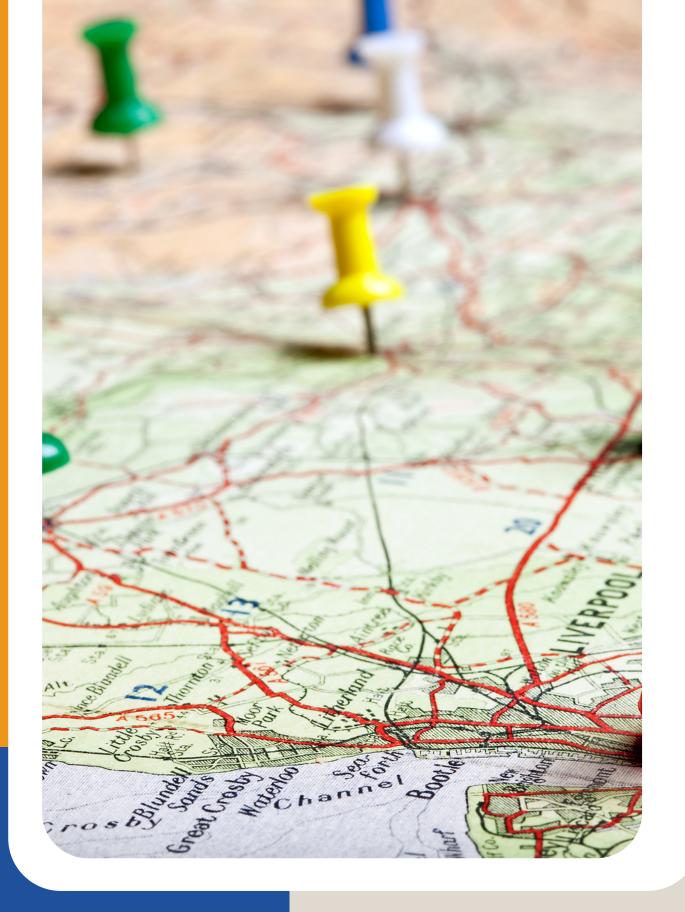
A FULL STACK CASE STUDY











OVERVIEW

This serverless, progressive web application (PWA) was built with React using a test-driven development (TDD) technique. The application uses the Google Calendar API to fetch upcoming events. Users can login using their Google information and search for relevant events in their area.

<u>CHECK IT OUT</u>

```
viceWorkerRegistration.js
                       JS Alert.js
                                      meet-app-144.png
                                                             favicon
mponents > JS Event.js > [@] Event
import { useState } from "react";
const Event = ({ event }) => {
 const [showDetails, setShowDetails] = useState(false);
 const toggleDetails = () => {
   setShowDetails(!showDetails);
  return (
   <div className="event">
       kh2 style={{ color: "#1e847f" }}{event.summary}</h2>
       <div className="location">{event.location} </div>
       <div className="dateTime">{event.start.dateTime}</div>
       {showDetails && <div className="description">{event.descripti
       <button className="details-btn" onClick={toggleDetails}>
         {showDetails ? "Hide Details" : "Show Details"}
       </button>
      </div>
   };
export default Event;
⊗0∆0 🗭0 🕏 Live Share
```

OBJECTIVE

This project was part of the the full-stack immersion course from the web development program at CareerFoundry to demonstrate my knowledge and skills of PWAs. The goal was to create a serverless progressive web application with authentication and authorization of users to showcase in my portfolio.

DETAILS



Roles

Role: Lead Developer

Tutor: Ebere Iweala

Mentor: Mahesh Rodrigo



Duration

This project took me nearly 8 weeks to complete from start to finish.



Methodologies

- React
- React Bootstrap
- React Testing Library
- Jest
- Puppeteer
- AWS



THE PROCESS





Getting Started

I started a Create React App and wrote user scenarios using the Gherkin "given, when, then" syntax.

Setting things up

I connected my app with a Google Calendar API for events and set up OAuth authorization/authentication for users.

Going serverless

I set up the serverless toolkit through AWS and and created and deployed AWS Lambda serverless functions in my code.

Testing

I conducted unit tests and integration tests for React components and app features using the React testing library and Jest.

Continued Testing

I then conducted end-to-end testing using Puppeteer.

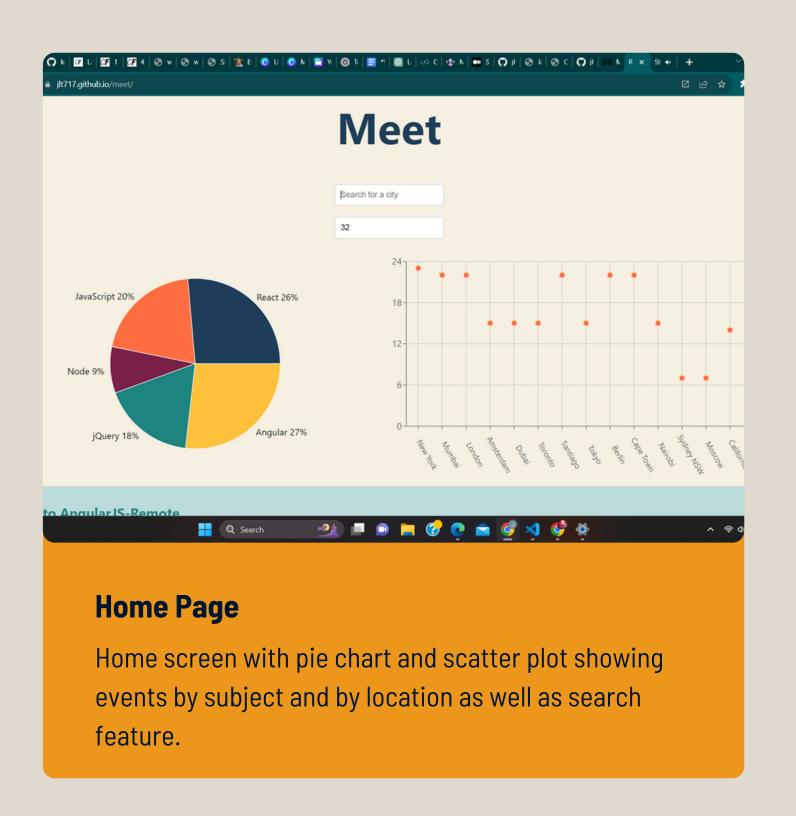
App Performance

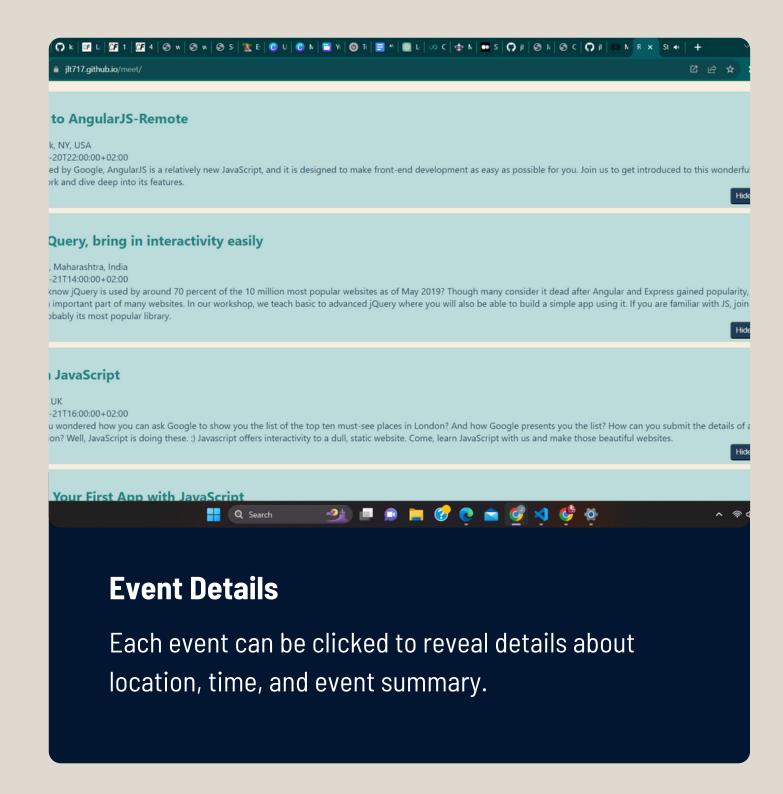
I followed up by using Atatus to track overall app performance and Lighthouse to convert my app to a PWA.





SCREENSHOTS









RETROSPECTIVE

This project challenged me and pushed me to persevere and try to break down and understand new concepts.

What Went Well

I enjoyed focusing on user scenarios and functionality in this project. It was very informative and user-focused.

Challenges

I struggled with reinterpreting code into different scenarios for various testing. This project was among the most frustrating I've worked on.

Final Thoughts

JS CitySearch.test.is >

: allLocations = extractLocations(eventsList);
ribe("<CitySearch /> component", () => {
st("renders text input", () => {

expect(cityTextBox).toBeInTheDocument();
expect(cityTextBox).toHaveClass("city");

render(<CitySearch allLocations={allLocations} />);
const cityTextBox = screen.queryByRole("textbox");

st("suggestions list is hidden by default", () => {
render(<CitySearch allLocations={allLocations} />);
const suggestionList = screen.queryByRole("list");

Looking back, I can see that I should've spent more time researching and practicing each type of testing. I did, however, after the fact.

```
expect(suggestionList).not.toBeInTheDocument();
st("renders a list of suggestions when user types into city textbox", async () => {
render(<CitySearch allLocations={allLocations} setInfoAlert={() => { }}/>);
 onst cityTextBox = screen.getByPlaceholderText("Search for a city");
 wait userEvent.type(cityTextBox, "a");
 onst suggestionItem = screen.queryAllByRole("listitem");
expect(suggestionItem.length).toBe(2);
 onst suggestionList = screen.queryByTestId("suggestions");
expect(suggestionList).toHaveClass("suggestions");
"updates list of suggestions correctly when user types in city textbox", async () => {
nder(<CitySearch allLocations={allLocations} setInfoAlert={() => { }}/>);
st cityTextBox = screen.queryByRole("textbox");
ait userEvent.type(cityTextBox, "Berlin");
                      ocations
                      (location) => {
                     rCase().indexOf(citvTextBox.value.toUnnerCase()) > -1
                                                                                                                    Ln 174, Col 4 Spaces: 2 UTF-8 CRLF {} JavaScript
                                      Q Search
```

meet-app-144.png