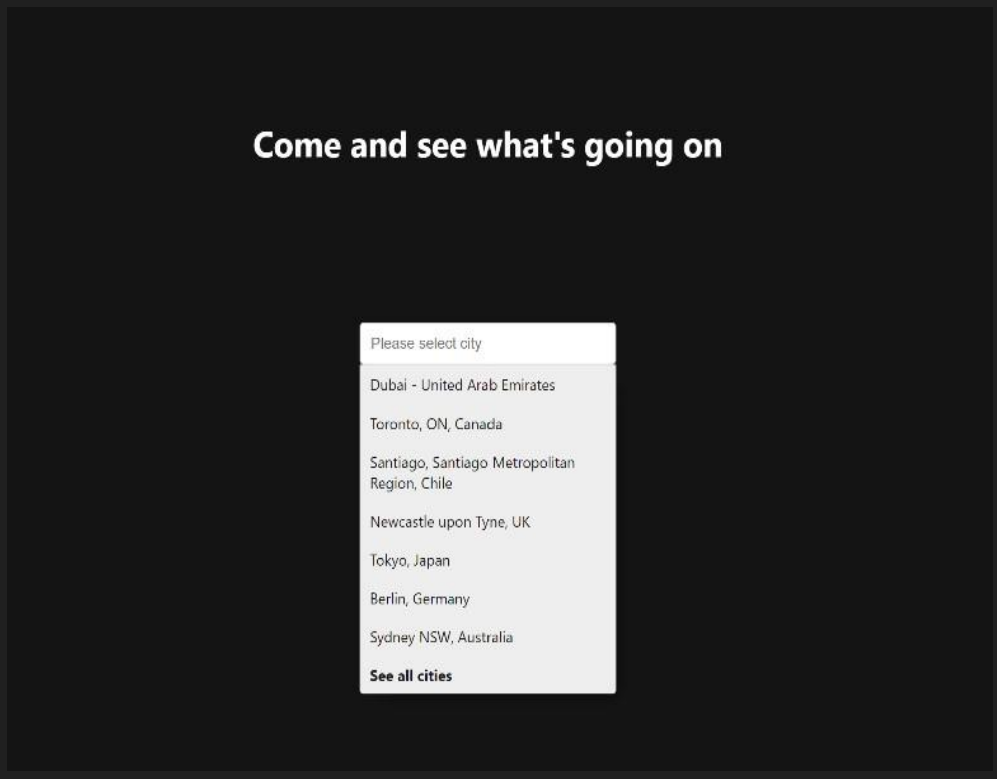
MEET APP

Find web dev events near you

Overview

This app was developed during the Full Stack Web Development course in Career Foundry. The progressive web application (PWA), built with React, allows users to search for a city and get a list of events hosted in that city. The events are fetched from the Google Calendar API using AWS Lambda as an authorized server for issuing OAuth2 authentication. It was developed following test-driven development (TDD) supported by Jest.



Objective

The goal was to implement a complete project. The problem I wanted to solve was to develop an application that you can discover web development events near your area. Also, I especially wanted to gain hands-on experience with Node.js and React.

Built with

React Native, JavaScript and firebase

Project Scale

1-month project

```
const { google } = require("googleapis");
const OAuth2 = google.auth.OAuth2;
const calendar = google.calendar("v3");

const SCOPES = ["https://www.googleapis.com/auth/calendar.readonly"];

const credentials = {
  client_id: process.env.CLIENT_ID,
  project_id: process.env.PROJECT_ID,
  client_secret: process.env.CLIENT_SECRET,
  calendar_id: process.env.CALENDAR_ID,
  auth_uri: "https://accounts.google.com/o/oauth2/auth",
  token_uri: "https://oauth2.googleapis.com/token",
  auth_provider_x509_cert_url: "https://www.googleapis.com/oauth2/v1/certs",
  redirect_uris: ["https://bilal-32.github.io/meet_/"],
  javascript_origins: ["https://bilal-32.github.io", "http://localhost:3000"],
};

const { client_id, client_secret, redirect_uris, calendar_id } = credentials;

const oAuth2Client = new google.auth.OAuth2(
  client_id,
  client_secret,
  redirect_uris[0]
);

// Generate an URL
module.exports.getAuthURL = async () => {
  const authUrl = oAuth2Client.generateAuthUrl({
    access_type: "offline",
    scope: SCOPES,
  });

  return {
    statusCode: 200,
    headers: {
```

Development Process

1. Creating an Authorisation Server and setting up OAuth2

The first thing to do was ensure that the user has access to the Google Calendar API. This was a complex process because involved on one hand the creation of a server to handle the authorisation requests and on another hand, the creation of an OAuth Consumer that contains the access required by Google to the API.

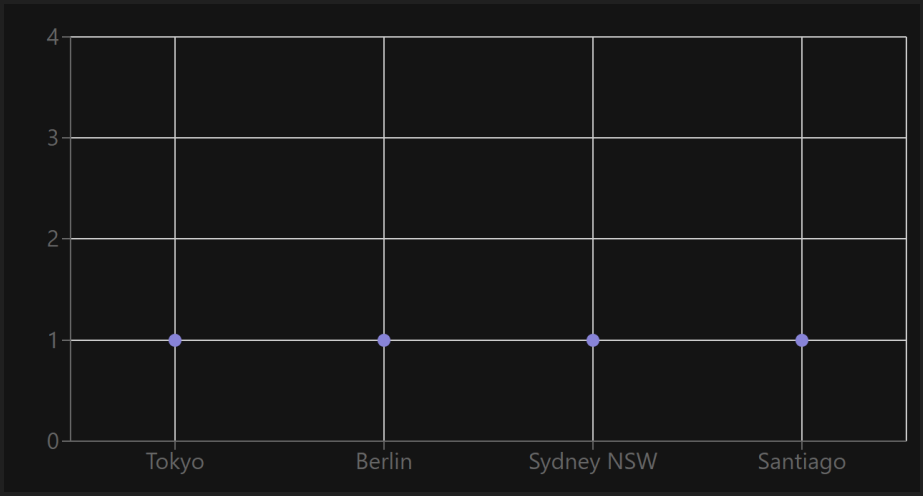
The authorisation Server used the services of AWS Lambda to host serverless functions created with the authorisation requests. Through the OAuth2 protocol, the application gains access to the API with its own OAuth Consumer.

The biggest challenge I encountered during this task was the commands I was asked to use in the terminal as I was not familiar with them. The way I resolved this was by researching the commands online and seeing answers from other web developers on stack overflow

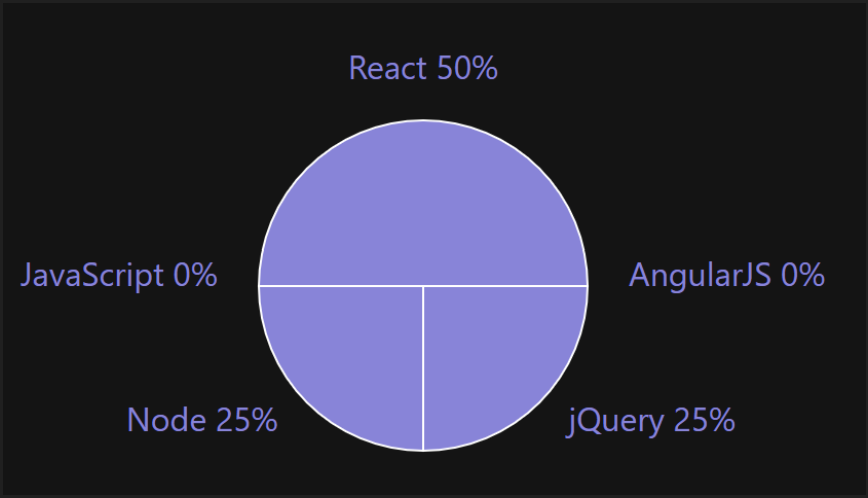
2. Testing

To guarantee that every part would work properly, unit tests were set up with the help of the Jest, first with mock data, then with the actual data loaded from Calendar API, followed by integration tests and user acceptance tests made with Puppeteer. This was the essential topic in this project and for me, as a developer student, was a big step to be able to know how to prevent errors and write more structured code. The way I learned to write more structured code was by using indentation and comments as this makes the code neat and easier to read. The way I prevented errors and bugs was by having the live website on the screen while coding so I could see any errors popping up also I used the console log as it is very simple to use.

The challenge I faced in this task was incorporating an interactive chart and a drop-down button so you can select the city and how many events you want to see. The way I got around this challenge was by researching the code to use in HTML and CSS to incorporate the interactive element



The grid above shows how many events you have for each image also it contains what city has events coming up.



The pie chart above shows how many events are in each language or framework.

3. Converting to a PWA (Progressive Web Application)

A web application with the advantages of a native application: the ability to be installed (on both mobile devices and computers) and the ability to work offline. Service Workers handle the requests and responses made between the application and the data stored in the browser's cache, making it possible for the application to load data even when there is no internet connection. And thanks to the Web App Manifest (which defines how the app will be displayed to the user) meet, the app can look and feel like a native application. The struggle I found when using PWA was that there was less functionality, an example of this is accessing the calendar but I solved this by ensuring I didn’t use the functionality that would affect the application’s performance.

Summary

The most challenging task was setting up authorisation/authentication for Google API: the reason it was so complicated was I had not done it before. The way I solved it is researching how to authenticate google API and the best easiest way I found was through OAuth 2. For the second React project, the focus was testing: an essential skill for every developer and one that I, personally, enjoyed learning. It was also interesting to know how the testing phase integrates the most modern development processes like CI (Continuous Integration) and CD (Continuous Delivery). The value of CI and CD is that it makes creating the project easier and faster. With CI I can continuously integrate code into a single shared and easy-to-access repository and with CD it allowed me to take the code stored in the repository and continuously deliver it to the production. If I had a chance to do this project all over again, I would first make a solid plan on paper for each step and also outline the places where I have a skill gap for example the commands that I needed to use in the terminal.