CAREERFOUNDRY

Achievement 4 Project: Meet App

Objective

To build a serverless, progressive web application (PWA) with React using a test-driven development (TDD) technique. The application uses the Google Calendar API to fetch upcoming events.

Context

Serverless and PWAs have grown in popularity over the last few years, and they're both considered to be the future of web development. By combining these two concepts, your app will not only work as a normal web application, but it will also reap the benefits of both serverless architecture and PWAs:

- **Serverless**: No backend maintenance, easy to scale, always available, no cost for idle time.
- **PWAs**: Instant loading, offline support, push notifications, "add to home screen" prompt, responsive design, and cross-platform compatibility.

For this app, you'll be using a TDD approach, where you write tests before writing the actual functionality for your app in code. Writing tests forces you to focus on the requirements of your application before jumping into the code. TDD relies on the repetition of a very short development cycle, allowing you to get immediate feedback and deliver high-quality code.

Last but not least, you'll add some graphs to your app, which will make it more visually appealing and allow you to more easily draw conclusions from the data. A picture is worth a thousand words, right? With a number of visualization techniques under your belt, you'll be able to display any type of data you want and produce a variety of output formats. Your app will allow users to search for a city and get a list of events hosted in that city. For the data visualization component, you'll add two charts—one that shows how many events will take place in that city on upcoming days, and another that visualizes the popularity of event genres in the form of a pie chart.

The 5 Ws

1. Who—The users of your Meet application. They could be you, your friends, your professional network, or your potential employers.
2. What—A progressive web app with the ability to work offline and a serverless backend developed using a TDD technique.
3. When—Users of this app will be able to use it whenever they want to view upcoming events for a specific city. Your recruiter will be able to see your code immediately on GitHub.

4. Where—The server, in this case, is a serverless function hosted by a cloud provider (e.g., AWS). The application itself is also hosted online to make it shareable and installable. It can be used even when the user is offline. As it's responsive, it displays well on any device.
5. Why—Serverless is the next generation of cloud infrastructure, PWA provides great user experience and performance, and the TDD technique ensures you have quality code and adequate test coverage. All of these skills, together with data visualization, will distinguish you from other web developers.

Features and Requirements

Key Features:

- Filter events by city.
- Show/hide event details.
- Specify number of events.
- Use the app when offline.
- Add an app shortcut to the home screen.
- View a chart showing the number of upcoming events by city.

User Stories:

- As a user, I would like to be able to filter events by city so that I can see the list of events that take place in that city.
- As a user, I would like to be able to show/hide event details so that I can see more/less information about an event.
- As a user, I would like to be able to specify the number of events I want to view in the app so that I can see more or fewer events in the events list at once.
- As a user, I would like to be able to use the app when offline so that I can see the events I viewed the last time I was online.
- As a user, I would like to be able to add the app shortcut to my home screen so that I can open the app faster.
- As a user, I would like to be able to see a chart showing the upcoming events in each city so that I know what events are organized in which city.

Technical Requirements:

- The app *must* be a React application.
- The app *must* be built using the TDD technique.
- The app *must* use the Google Calendar API and OAuth2 authentication flow.

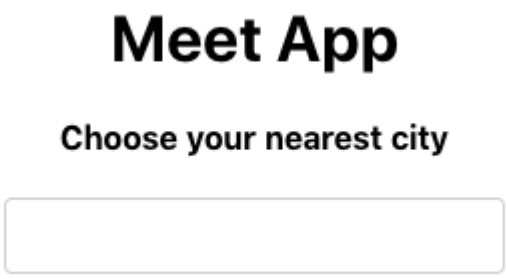
- The app *must* use serverless functions (AWS lambda is preferred) for the authorization server instead of using a traditional server.
- The app's code *must* be hosted in a Git repository on GitHub.
- The app *must* work on the latest versions of Chrome, Firefox, Safari, Edge, and Opera, as well as on IE11.
- The app *must* display well on all screen sizes (including mobile and tablet) widths of 1920px and 320px.
- The app *must* pass Lighthouse's PWA checklist.
- The app *must* work offline or in slow network conditions with the help of a service worker.
- Users *may* be able to install the app on desktop and add the app to their home screen on mobile.
- The app *must* be deployed on GitHub Pages.
- The API call *must* use React axios and async/await.
- The app *must* implement an alert system using an OOP approach to show information to the user.
- The app *must* make use of data visualization (recharts preferred).
- The app *must* be covered by tests with a coverage rate >= 90%.
- The app *must* be monitored using an online monitoring tool.

Mock-ups or Other Assets

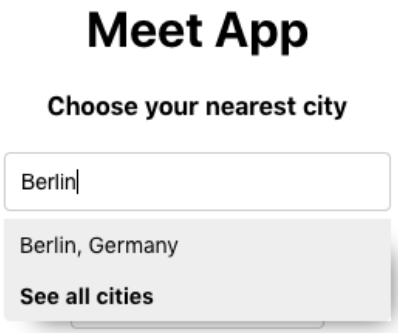
In this section, you'll find mockups for your app. We'll keep things simple for now so that you can focus on writing clean, readable code. Once you've mastered the foundational aspects of the code, we encourage you to add unique flair to your app.

User Input: City Name and Country

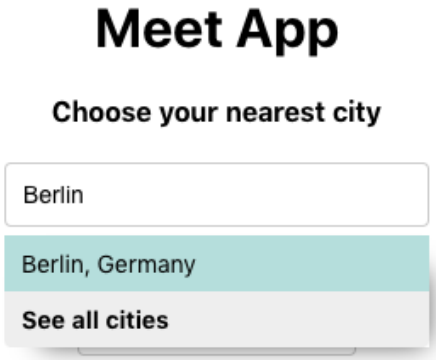
Default:



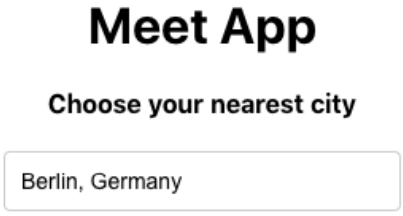
When typing a query:



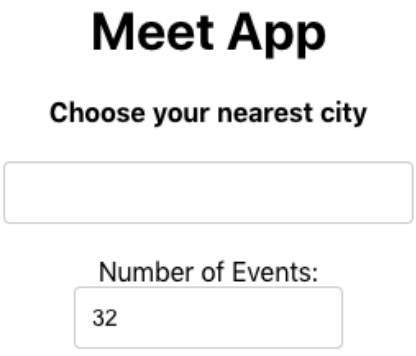
When hovering over a suggestion:



After selecting a suggestion:

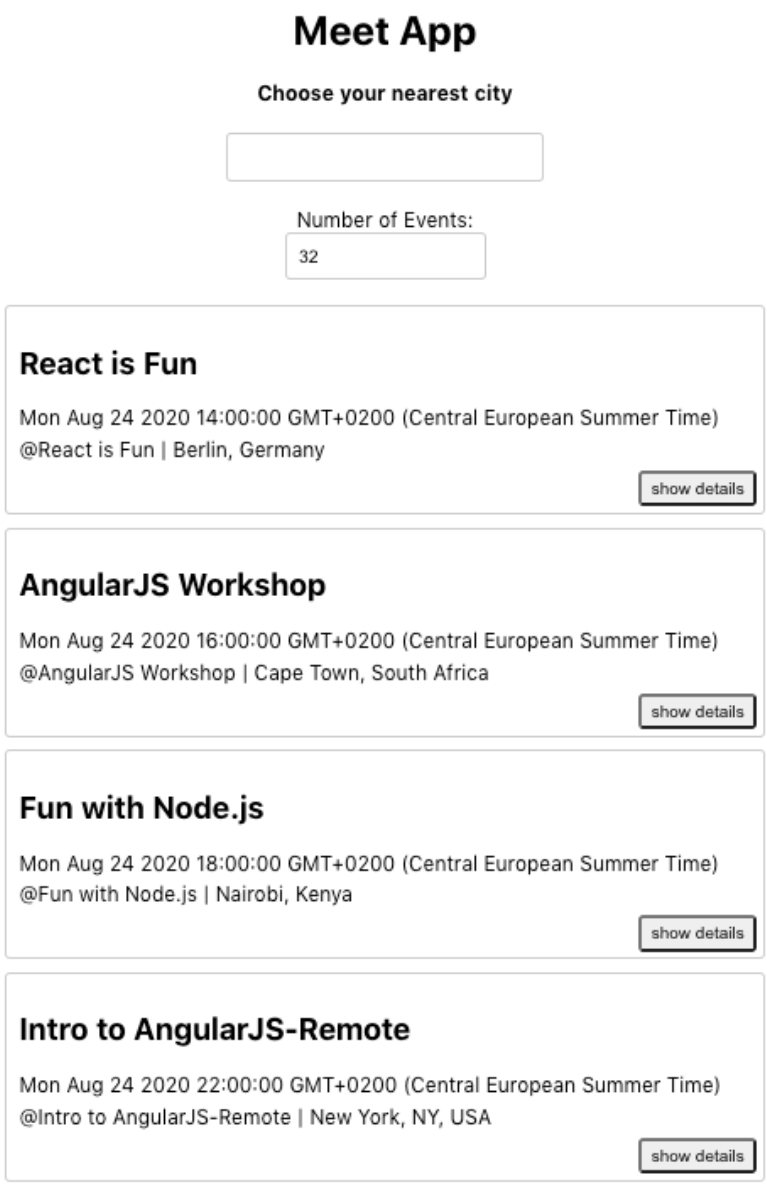


Specify number of events to show:

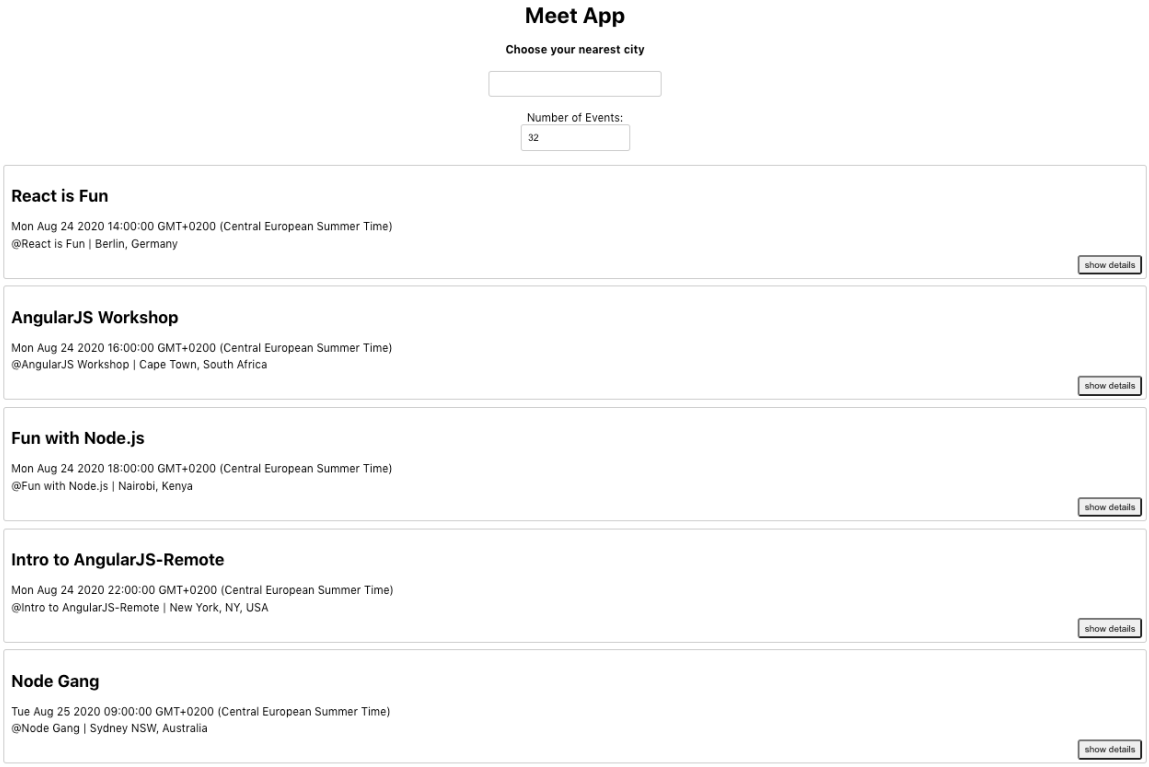


Upcoming events in different screen sizes:

Small:

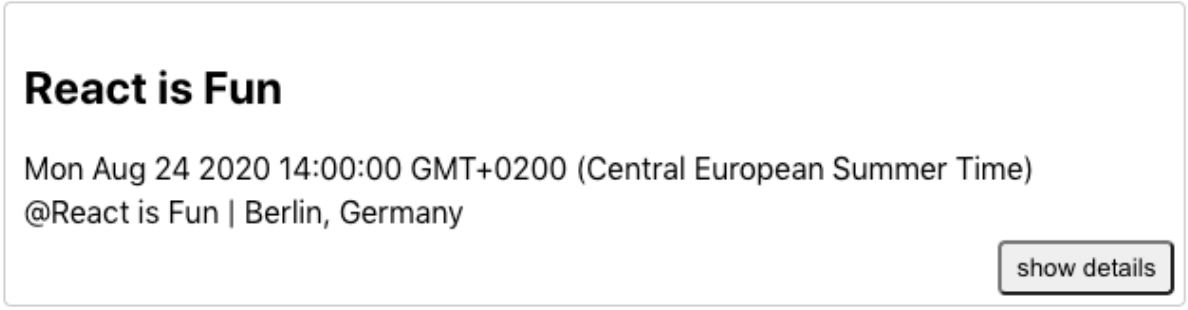


Large:

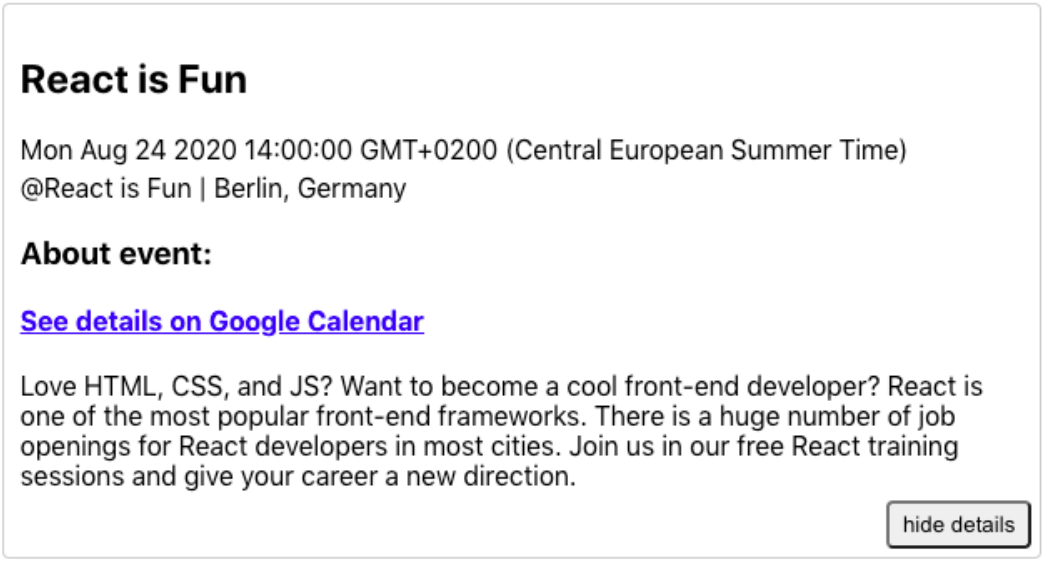


Event Details

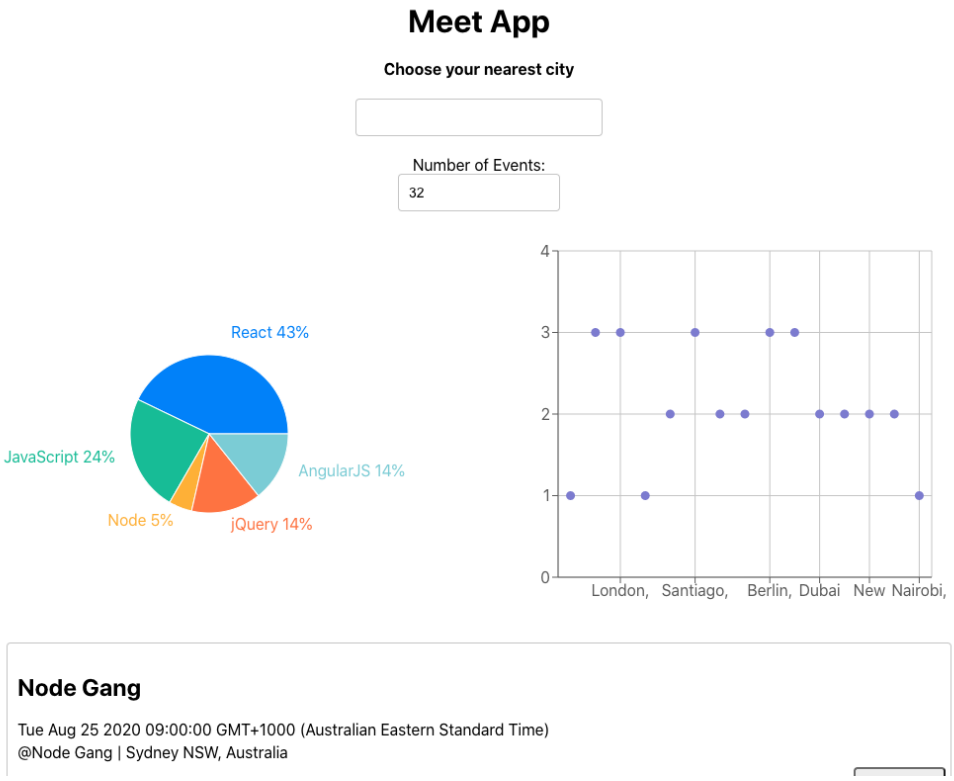
“Details” button:



After clicking on the “Details” button (before and after styling):



With data visualization:



Your Project Deliverables

Throughout this course, you'll be working from Exercise to Exercise to complete your project. For each Task, you'll submit a deliverable that directly contributes to the final product—in this case, an application to demonstrate your knowledge and skills of PWAs.

Below is a breakdown of your course project deliverables by Exercise:

Exercise 1: TDD & Test Scenarios

- Write user stories based on the app's key features.
- Translate user stories for each feature into multiple test scenarios.
- Use create-react-app to create a React application and push it to GitHub.

Exercise 2: Serverless Functions

- Obtain a consumer secret from the Google Calendar API.
- Write a simple serverless function to refresh the access token.
- Call this function from a static page.

Exercise 3: Unit Testing

- Using test scenarios, write frontend unit tests using mock data for the app's key features.

Exercise 4: Integration Testing

- Develop another feature for the app.
- Write integration tests to test the interaction between the app's React components.
- Write integration tests to test the data received from the mock API.

Exercise 5: User Acceptance Testing & End-to-End Testing

- Write user acceptance tests for two key features, covering all defined test scenarios.

Exercise 6: Testing in the Development Process (Continuous Delivery)

- Set up app monitoring for the application to monitor its performance.

Exercise 7: Object-Oriented Programming (OOP)

- Use an OOP approach to create alerts for the application.

Exercise 8: Progressive Web Applications

- Use a service worker to ensure your app works offline.
- Use a “manifest.json” file to make your app installable.
- Show a notification to the user to inform them the app is working offline (when the user is offline).

Exercise 9: Data Visualization

- Add charts, such as a ScatterChart, to your app’s UI to visualize data using the recharts library.
- Make visualizations responsive.

Optional: Advanced Deliverables

In addition to all of the above, you can add more advanced features to your app as you wish. Below are some topics you could explore in your app:

- Use the Lambda inline editor to create the authorization server instead of using the serverless toolkit.
- Style your app using React-Bootstrap.
- Write end-to-end tests for your app using Puppeteer.
- Conduct QA to test your app and fix any issues your testers may find.