

**CAREER***FOUNDRY*

# 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 each location (via a scatterplot), and another that visualizes the popularity of event genres (via a pie chart).

# The 5 Ws

1. **Who** — The users of your Meet app. 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.

## Your Project 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.
- Display Charts Visualizing Event Details.

### 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.

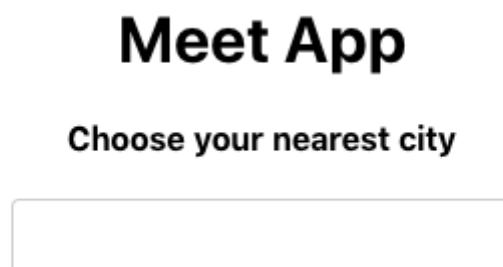
- The app *must* display well on all screen sizes (including mobile and tablet) widths of 1920px and 320px.
- The app *must* pass [Lighthouse's PWA checklist](#).
- The app *must* work offline or in slow network conditions with the help of a service worker.
- Users *may* be able to install the app on desktop and add the app to their home screen on mobile.
- The app *must* be deployed on GitHub Pages.
- The app *must* implement an alert system using an OOP approach to show information to the user.
- The app *must* make use of data visualization.
- The app *must* be covered by tests with a coverage rate  $\geq 90\%$ .
- The app *must* be monitored using an online performance 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.

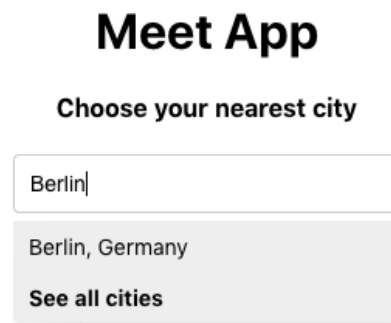
## User Input: City Name and Country

Default:



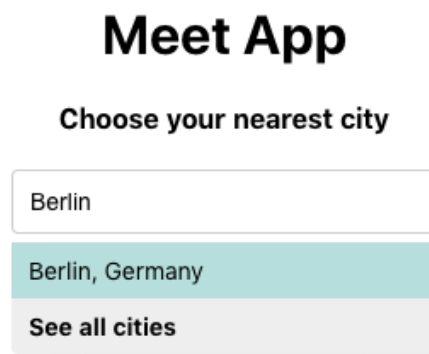
The mockup shows the title "Meet App" in a large, bold, black font. Below it is the subtitle "Choose your nearest city" in a smaller, bold, black font. At the bottom is a single-line text input field with a light gray border and rounded corners.

When typing a query:



The mockup shows the title "Meet App" and subtitle "Choose your nearest city". The text input field contains the text "Berlin". Below the input field is a dropdown menu with a light gray background and rounded corners. It contains two items: "Berlin, Germany" and "See all cities".

When hovering over a suggestion:



The mockup shows the title "Meet App" and subtitle "Choose your nearest city". The text input field contains the text "Berlin". Below the input field is a dropdown menu with a light gray background and rounded corners. It contains three items: "Berlin, Germany" (highlighted with a teal background), "See all cities", and a third item that is not visible.

After selecting a suggestion:

## Meet App

Choose your nearest city

Berlin, Germany

Specify number of events to show:

## Meet App

Choose your nearest city

Number of Events:

32

## Upcoming events in different screen sizes:

Small:

### Meet App

Choose your nearest city

Number of Events:

#### React is Fun

Mon Aug 24 2020 14:00:00 GMT+0200 (Central European Summer Time)  
@React is Fun | Berlin, Germany

[show details](#)

#### AngularJS Workshop

Mon Aug 24 2020 16:00:00 GMT+0200 (Central European Summer Time)  
@AngularJS Workshop | Cape Town, South Africa

[show details](#)

#### Fun with Node.js

Mon Aug 24 2020 18:00:00 GMT+0200 (Central European Summer Time)  
@Fun with Node.js | Nairobi, Kenya

[show details](#)

#### Intro to AngularJS-Remote

Mon Aug 24 2020 22:00:00 GMT+0200 (Central European Summer Time)  
@Intro to AngularJS-Remote | New York, NY, USA

[show details](#)

Large:

### Meet App

Choose your nearest city

Number of Events:  
32

**React is Fun**  
Mon Aug 24 2020 14:00:00 GMT+0200 (Central European Summer Time)  
@React is Fun | Berlin, Germany

show details

**AngularJS Workshop**  
Mon Aug 24 2020 16:00:00 GMT+0200 (Central European Summer Time)  
@AngularJS Workshop | Cape Town, South Africa

show details

**Fun with Node.js**  
Mon Aug 24 2020 18:00:00 GMT+0200 (Central European Summer Time)  
@Fun with Node.js | Nairobi, Kenya

show details

**Intro to AngularJS-Remote**  
Mon Aug 24 2020 22:00:00 GMT+0200 (Central European Summer Time)  
@Intro to AngularJS-Remote | New York, NY, USA

show details

**Node Gang**  
Tue Aug 25 2020 09:00:00 GMT+0200 (Central European Summer Time)  
@Node Gang | Sydney NSW, Australia

show details

## Event Details

“Details” button:

**React is Fun**  
Mon Aug 24 2020 14:00:00 GMT+0200 (Central European Summer Time)  
@React is Fun | Berlin, Germany

show details



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

## React is Fun

Mon Aug 24 2020 14:00:00 GMT+0200 (Central European Summer Time)  
@React is Fun | Berlin, Germany

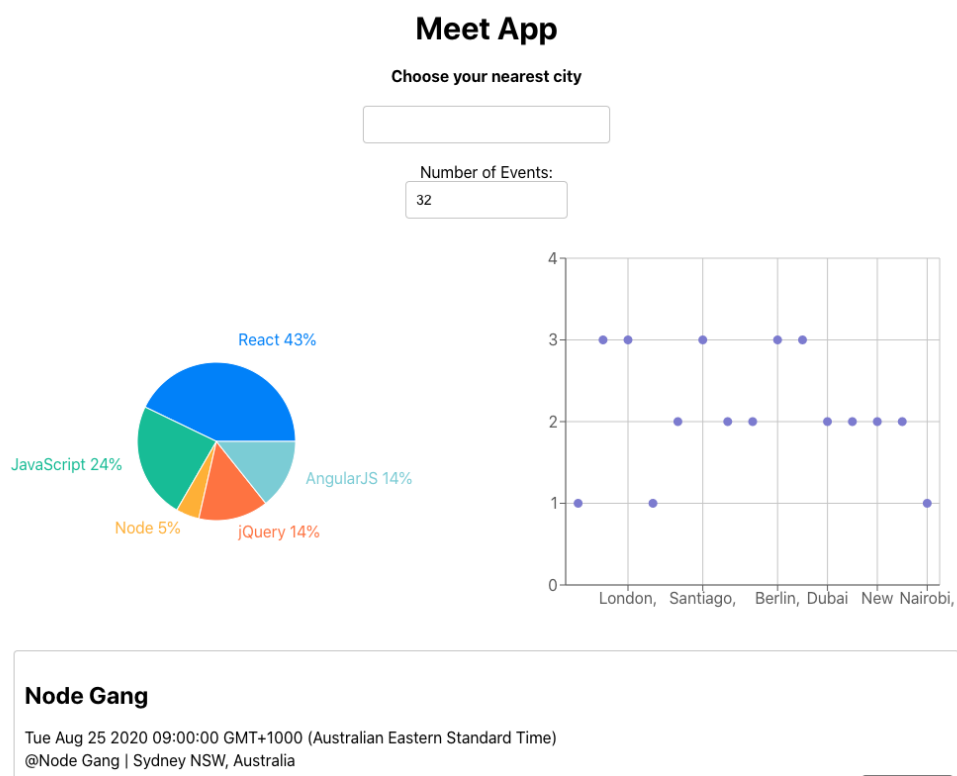
**About event:**

[See details on Google Calendar](#)

Love HTML, CSS, and JS? Want to become a cool front-end developer? React is one of the most popular front-end frameworks. There is a huge number of job openings for React developers in most cities. Join us in our free React training sessions and give your career a new direction.

hide details

With data visualization:



# Your Project Features & Scenarios

## **Feature 1: Filter Events By City**

Scenario 1: When user hasn't searched for a city, show upcoming events from all cities.

Scenario 2: User should see a list of suggestions when they search for a city.

Scenario 3: User can select a city from the suggested list.

## **Feature 2: Show/Hide Event Details**

Scenario 1: An event element is collapsed by default.

Scenario 2: User can expand an event to see details.

Scenario 3: User can collapse an event to hide details.

## **Feature 3: Specify Number of Events**

Scenario 1: When user hasn't specified a number, 32 events are shown by default.

Scenario 2: User can change the number of events displayed.

## **Feature 4: Use the App When Offline**

Scenario 1: Show cached data when there's no internet connection.

Scenario 2: Show error when user changes search settings (city, number of events).

## **Feature 5: Add an App Shortcut to the Home Screen**

Scenario 1: User can install the meet app as a shortcut on their device home screen.

## **Feature 6: Display Charts Visualizing Event Details**

Scenario 1: Show a chart with the number of upcoming events in each city.

# Your Project Deliverables

Throughout this course, you'll work from Exercise to Exercise to complete your project. For each task, you'll submit a deliverable that contributes to the final product—in this case, an app demonstrating your knowledge and skills of PWAs. Here's a breakdown of your project deliverables by Exercise:

## Exercise 4.1: 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 app and push it to GitHub.
- Deploy a React app to GitHub Pages.

## Exercise 4.2: Intro to Serverless Functions & Authentication

- Evaluate the merit and usefulness of serverless development
- Connect a React app with a protected API
- Prepare an OAuth client for authorization and authentication
- Obtain AWS credentials for future use

## Exercise 4.3: Writing & Testing AWS Lambda Functions

- Write Lambda functions to implement serverless technology in an app
- Test Lambda functions
- Create a serverless deployment package

## Exercise 4.4: Unit Testing

- Analyze use cases for unit tests
- Write unit tests for an app
- Test components using mock data
- Develop implementation code in response to unit tests

## Exercise 4.5: Integration Testing

- Analyze use cases for integration tests
- Write integration tests
- Develop implementation code in response to integration tests
- Integrate real data from an API into a web app

## Exercise 4.6: User Acceptance Testing & End-to-End Testing

- Describe the purpose of end-to-end testing during development
- Write acceptance tests for an app to help non-developer stakeholders understand implementation code
- Conduct automated end-to-end testing for an app
- Handle testing errors in the terminal

## Exercise 4.7: Continuous Delivery

- Discuss how CI and CD practices can help developers and organizations deliver high-quality products
- Integrate an APM tool into the development of a web app

## Exercise 4.8: Object-Oriented Programming (OOP)

- Define core concepts related to the OOP paradigm.
- Differentiate between when to use functional programming (FP) and when to use OOP to solve architectural problems (and when to use both).
- Implement a feature in your web app using OOP and React class components.

## Exercise 4.9: Progressive Web Apps

- Discuss what progressive web apps (PWAs) are and how they compare to regular web apps and native apps.
- Explain the core functionality of PWAs.
- Implement progressive functionality into an existing app, so that the app can be used offline and added to a user's home screen.

## Exercise 4.10: Data Visualization

- Transform API data into a visual format using a visualization library.
- Implement data visualization features into a web app's UI.
- Conduct research into a library's documentation to implement new features.

## 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.