Case Study – Meet App

Isabel Matula

Overview

Objective

The aim of the project was to develop a serverless, progressive web application (PWA) with React using a test-driven development (TDD) technique.

Purpose & Context

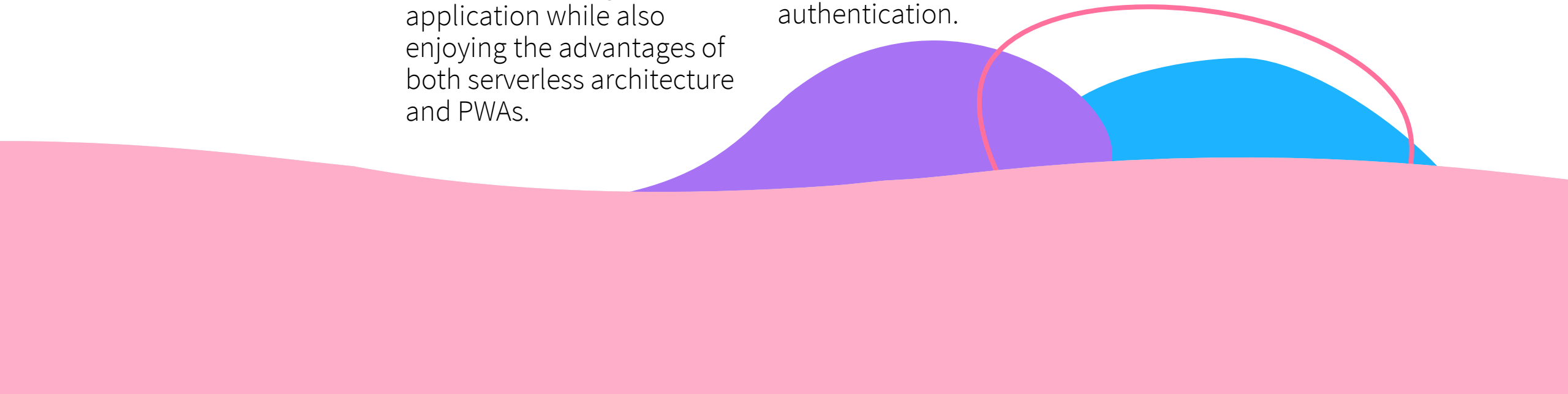
Over the past few years, Serverless and PWAs have become increasingly popular in web development. By combining these two concepts, the app can function like a regular web application while also enjoying the advantages of both serverless architecture and PWAs.

Tools

This project utilized React and JavaScript for app development, integrating Recharts for data visualization, and Google Calendar API for event retrieval and OAuth2 for authentication.

Duration

The project was completed within a timeframe of one and a half months.



User Stories

- As a user, I should be able to filter events by city, so that I can see a list of events taking place in that city.
- As a user, I should be able to show and hide event details, so that that I can get more details on an event only when needed.
- As a user, I should be able to specify the number of events displayed, so that I can decide how many I want to see at once.
- As a user, I should be able to use the application when I am offline, so that so I can use the app even when I don’t have an internet connection.
- As a user, I should be able to add shortcut for the app to my home screen, so that so I can easily navigate to the app whenever I open my browser.
- As a user, I should be able to see charts visualizing event details, so that I can quickly see what kind of events there are and where.

Unit & Integration Testing

Unit Testing

For testing my Meet App, I used **Jest**, which is included with create-react-app, for unit testing. I wrote test cases to verify individual components and grouped similar tests into test suites. **React Testing Library**, also bundled with create-react-app, provided necessary functions for testing React components. Running **npm test** allowed me to execute all test suites and get immediate feedback. This approach ensured each component functioned correctly and independently.

Integration Testing

I also used **Jest** for integration testing, focusing on component and data dependencies. I tested parent-child interactions to ensure data and function props were correctly handled. Additionally, I verified that components could fetch and render data from the Google Calendar API. These integration tests ensured seamless component interactions and reliable data handling. This approach improved the stability and robustness of the application.

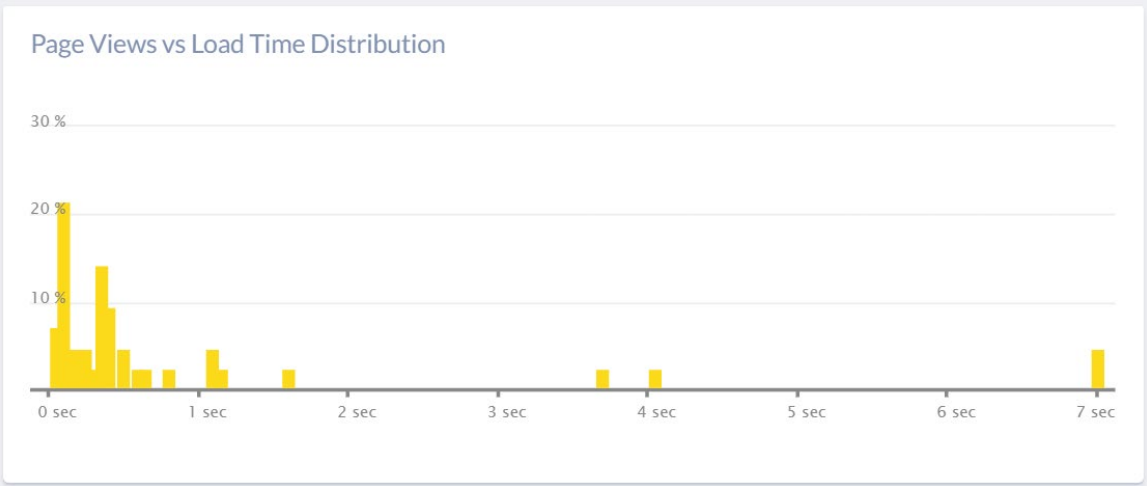
User Acceptance & End-to-End Testing

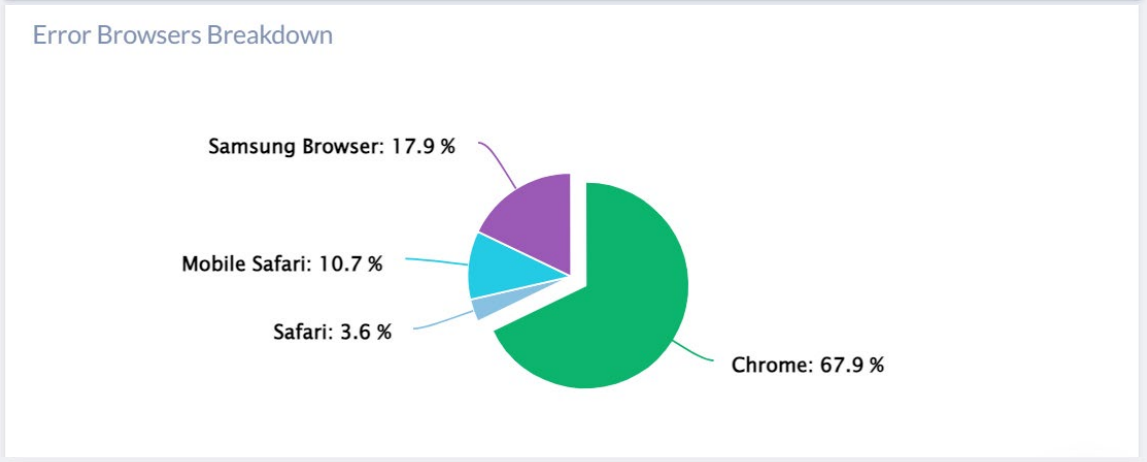
User Acceptance Testing

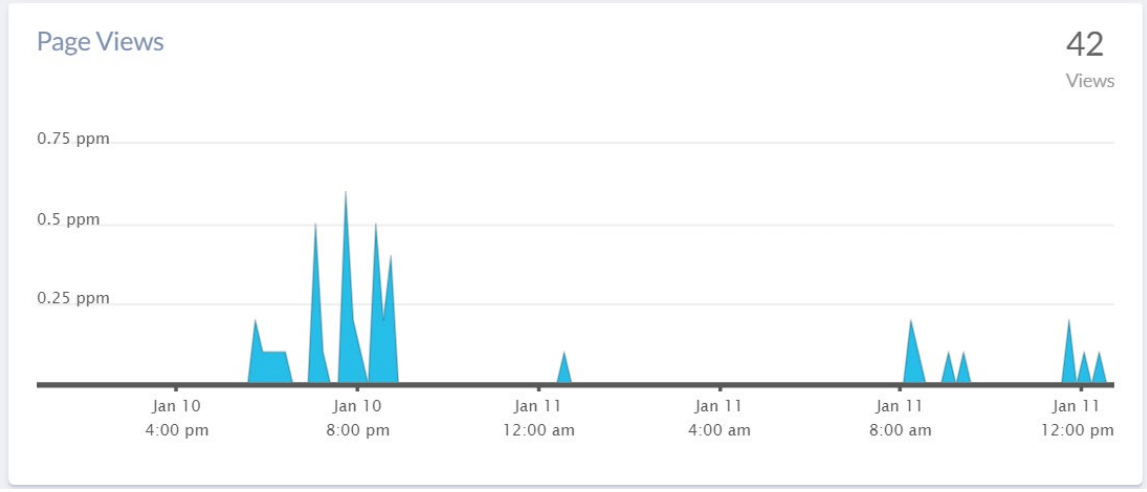
I performed User Acceptance Testing using a behavior-driven development (BDD) approach with jest-cucumber. I wrote feature tests in Gherkin's "Given, When, Then" syntax, stored in .feature files within a dedicated "features" folder. These tests ensured the application's features met stakeholder expectations and were easily understandable by non-developers. Each feature had corresponding step definition files in JavaScript that connected the Gherkin scenarios to the actual test code. This method allowed me to verify the functionality of the application from an end-user perspective.

End-to-End Testing

I performed User End-to-End Testing (E2E) using Puppeteer with Jest to ensure the application functioned correctly from the user's perspective. Puppeteer simulated user interactions, such as typing in search queries, setting limits on results, and clicking buttons. I wrote tests that opened the Chromium browser, navigated to the locally hosted app, and interacted with UI elements. By running these tests, I was able to verify that the application performed as expected, ensuring a seamless user experience. This testing approach confirmed the app's reliability and usability.





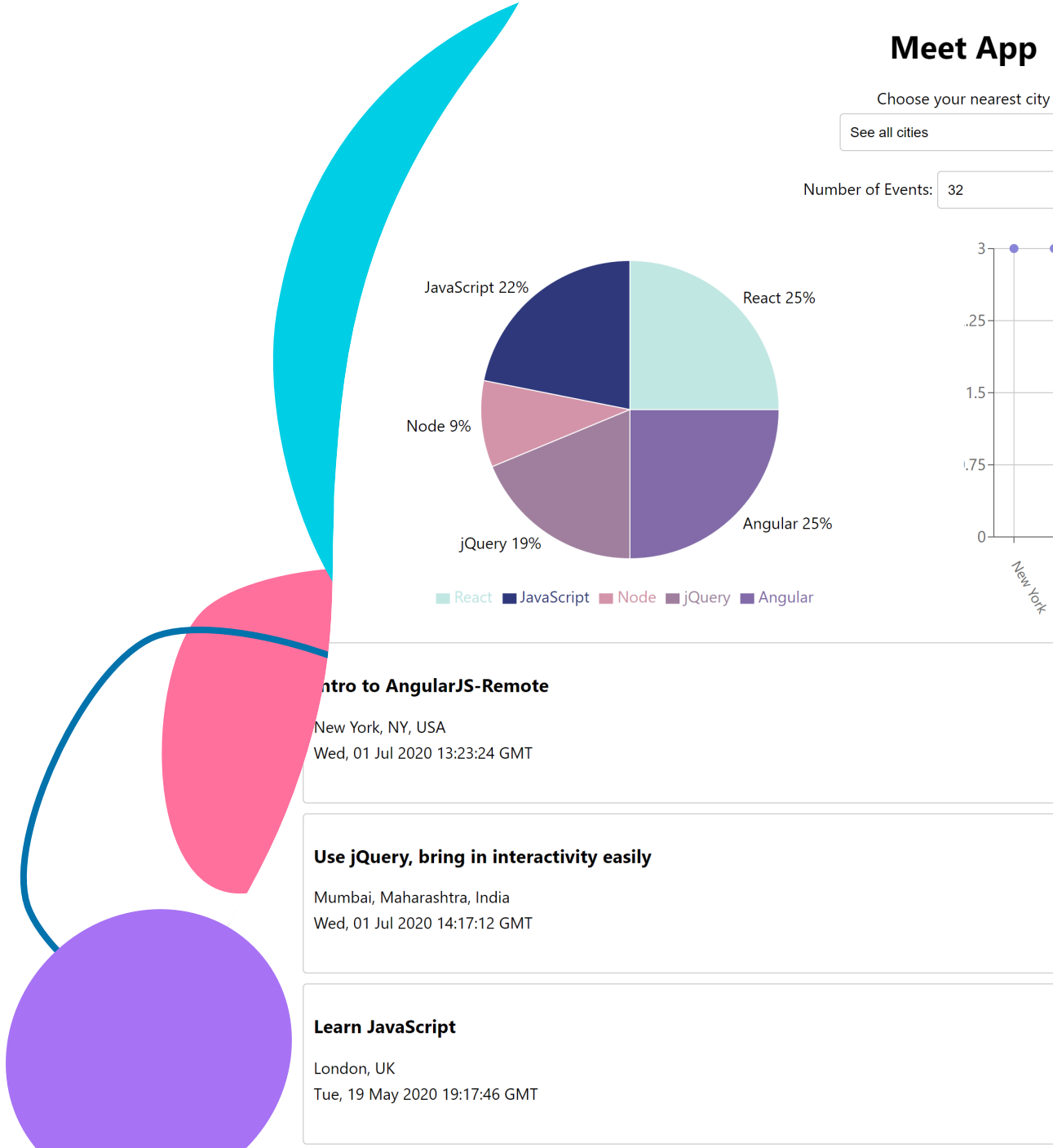


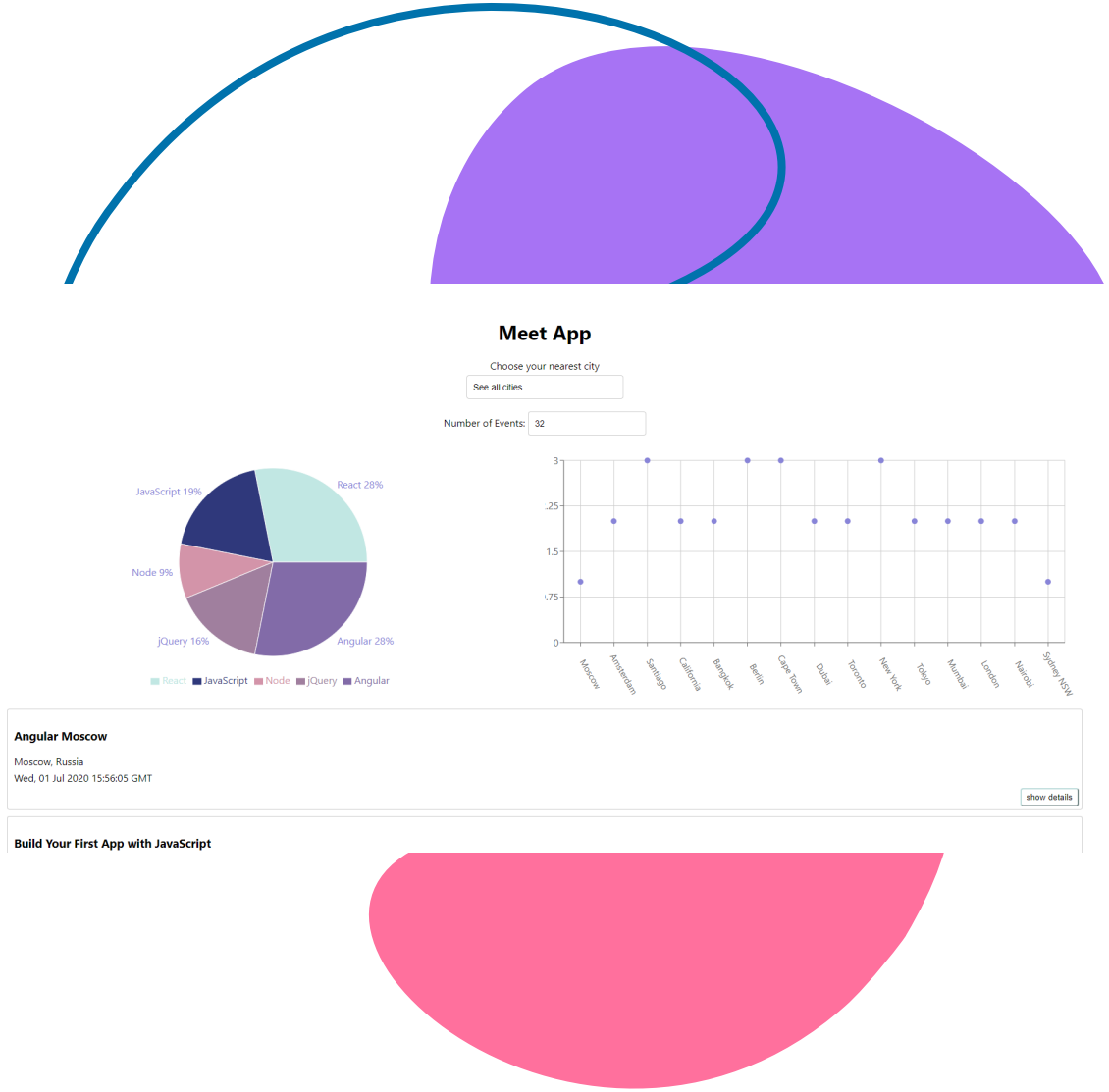
Performance Monitoring

I performed Performance Monitoring using Atatus to ensure the application functioned well under real-world conditions. I integrated Atatus by installing the atatus-spa package and configuring it in the application's index.js file. Atatus captured data on page load times, errors, and user interactions. I tested the setup by generating a test error and confirmed the integration through the Atatus dashboard. This allowed me to track the app's performance, identifying slow pages and frequent errors, ensuring a positive user experience by proactively addressing any issues.

Data Visualization

I used the **Recharts** library to create a **scatter chart** that visualized the number of events in various cities worldwide. I explored Recharts' capabilities, set up the ScatterChart, and formatted the data to count events per city. The chart was customized for readability and responsiveness, handling label overlap. I also implemented a **pie chart** to display event genres, applying customized labels for better clarity. Both charts were integrated into a grid layout for a responsive design. Finally, I deployed the updated app and verified its functionality in the browser.





Conclusion

The goal of developing a serverless, progressive web application with React using a test-driven development technique was successfully achieved. The final product allows users to filter events by city, show and hide event details, specify the number of events displayed, use the app offline, add the app to their home screen, and view charts visualizing event details. Integrating Recharts for data visualization and the Google Calendar API for event retrieval and OAuth2 authentication posed significant challenges, particularly in ensuring seamless functionality and data security. Despite these hurdles, the app's development, which included extensive unit testing, integration testing, user acceptance testing, and end-to-end testing, was rewarding. Moving forward, I would focus even more on enhancing user experience. Overall, this project highlighted the importance of comprehensive testing in web development.