Best Practices for Authenticating Cypress Tests in CI

Introduction

Continuous Integration (CI) is a crucial part of the software development process, allowing teams to automate the building, testing, and deployment of their applications. When it comes to end-to-end testing with Cypress, one important aspect is authenticating the tests to simulate real user interactions. In this report, we will explore the best practices for authenticating Cypress tests running in CI environments.

Custom Commands for Authentication

One of the recommended approaches for authenticating Cypress tests is to use custom commands. Custom commands allow you to encapsulate and reuse Cypress test logic, making it easier to add your own functionality to your test suite. By creating a custom command for authentication, you can simplify the process of logging in and reuse it across multiple test cases.

The Cypress documentation suggests creating a custom `cy.login()` command to handle authentication. This command can be added to the `cypress/support/commands.js` file, making it available for use in any spec file. The custom command can encapsulate the login code and provide a more maintainable and elegant solution compared to copying and pasting the login code into every test or using a `beforeEach` block.

Programmatic Authentication

To improve performance and avoid the overhead of re-authentication in each test case, Cypress best practices recommend programmatic authentication. Instead of going through form-based authentication for every test case, you can create a custom Cypress command, such as `authenticate`, to retrieve test credentials and sign in via a direct call to the authentication mechanism.

By using programmatic authentication, you can avoid the need to interact with the UI for authentication and directly authenticate the user using the provided credentials. This approach can speed up test runs and ensure consistent authentication across test cases.

Caching Authenticated Credentials

In Cypress, by default, test isolation does not propagate state such as cookies and local storage between test cases. However, in some cases, it may be desirable to cache authenticated credentials to speed up test runs. For example, if Amplify stores user credentials in browser local storage, authentication will not persist between test cases without additional code.

To address this, you can use hooks like `before` and `beforeEach` to cache the localStorage in memory and restore its contents before each test case. The `onBeforeLoad` callback provided by the Cypress `visit()` command can be used to restore the contents of the window's localStorage. This ensures that the authenticated state is maintained between test cases and avoids the need for re-authentication.

AWS Amplify Integration

When using AWS Amplify for authentication in Cypress tests, there are specific considerations for CI/CD use cases. The AWS Amplify SDK provides a credential provider chain that supports CI/CD scenarios. The Cypress configuration, which runs in a NodeJS context, can access the credentials required for testing.

To retrieve and decrypt the credentials from Parameter Store, the AWS SDK for JavaScript v3 provides the `fromNodeProviderChain()` function, which returns a CredentialProvider object. This function can be used in the Cypress configuration to access the credentials needed for testing.

By leveraging the AWS SDK and Cypress configuration, you can securely retrieve the necessary credentials for authentication in CI environments.

Avoiding CORS Errors

In some cases, when running Cypress tests, you may encounter CORS (Cross-Origin Resource Sharing) errors. These errors can occur if the cross-origin request is not authenticated. To avoid CORS errors, it is important to ensure that the authentication steps are in place and the requests are properly authenticated.

By following the best practices for authentication in Cypress tests, such as using custom commands and programmatic authentication, you can ensure that the requests are authenticated and avoid CORS errors.

Conclusion

Authenticating Cypress tests in CI environments is an important aspect of ensuring the accuracy and reliability of end-to-end tests. By following best practices such as using custom commands, programmatic authentication, and caching authenticated credentials, you can streamline the authentication process and improve the performance of your tests.

Integrating with authentication providers like AWS Amplify requires additional considerations, such as retrieving credentials securely and leveraging the AWS SDK for JavaScript. By following the recommended approaches and leveraging the capabilities of Cypress and authentication providers, you can create robust and efficient CI pipelines for your Cypress tests.

Overall, the best practices for authenticating Cypress tests in CI environments provide a solid foundation for ensuring the accuracy and reliability of your end-to-end tests, enabling faster delivery of high-quality software.

References

- Cypress Documentation: Testing Your App
- AWS Amplify Blog: Continuous Integration and Testing AWS Amplify Apps with Cypress
- Cypress Documentation: Best Practices
- Auth0 Blog: End-to-End Testing with Cypress and Auth0
- Cypress Blog: Authenticate Faster in Tests with the cy.session() Command
- GitHub Issue: Define strategy for testing authentication providers in CI