Extend Maximizes Efficiency Through Cypress Component Testing

Extend is a leading product and shipping protection platform that aims to amplify brand revenue and streamline issue resolution. The company boasts a robust engineering team of 80 professionals, which includes 7 dedicated internal developer experience (DevX) engineers. Extend engineers consistently implement end-to-end (E2E) tests in its React-based front-end applications and backend AWS services, all of which are developed in Node.

Extend uses Cypress E2E tests extensively, running hundreds of tests that span dozens of services and four primary front-end applications. As part of their continuous integration (CI) process, they conduct over four million tests, achieving impressive release times, moving from a pull request to production in under an hour. They have been using Cypress for two years, prior to which they had no successful automation in CI, relying on WebDriverJS locally and Jest with Supertest for API E2E automation.

Recently, the company embarked on a pilot initiative to integrate Cypress Component Testing (CT) into its most extensive front-end application. This initiative aimed to enhance the development and testing experience for their front-end engineers.

03 million
Cypress end-to-end tests

01 million
Cypress component tests

< 60 minutes
From PR to production

< 10 minutes
for CI feedback with Cypress Cloud Parallelization

Challenge

Extend's front-end engineering team had previously used component testing tools like Enzyme and React Testing Library (RTL). However, they encountered substantial difficulties with this method. Firstly, the testing procedure was notably time-consuming, partly because of the requirement to perform intricate mocking of frequently used libraries within the React ecosystem, including Redux, Redux Toolkit, React Router, and Formik. Secondly, the

existing testing tools only allowed them to inspect a component within the command-line interface (CLI), primarily in its HTML format, which imposed limitations on their diagnostic capabilities.

Solution

Recognizing the urgency for a change in their low-level UI testing approach, Extend acknowledged that their existing solution, Enzyme, was becoming obsolete due to the lack of ongoing support from Airbnb. The limited testing coverage in their component library, extensively used across their four front-end applications, further underscored the necessity for a new solution. Cypress Component Testing (CT) emerged as the clear answer to address Extend's challenges.

In Cypress CT, custom mounts serve as specialized render functions, efficiently mounting a component directly into the DOM, bypassing the entirety of the app. This approach mirrors the component's integration within the actual application, but within the confines of the test context. By adopting custom mounts in Cypress CT, the team abstracted intricate application details, ensuring an authentic replication of component behavior during tests.

Upon embracing Cypress, the team introduced a novel approach that simplified the process of mocking compared to their prior methods. They transitioned to network-level mocking, which proved more straightforward and bolstered their confidence in the overall functionality of the component code. While they had the capability for low-level mocking with Sinon (a built-in Cypress feature), the shift to network-level mocking emerged as a preferred and more efficient choice.

The improved developer experience provided by Cypress CT rekindled the team's enthusiasm, leading them to revisit and revitalize component testing examples in the Cypress repository. Today, the repository is a trove with over 400 Cypress component test examples for React.

Staff engineer at Extend, Murat Ozcan, reflects:

"Component testing stands as the linchpin in front-end software development. It bridges unit tests and end-to-end tests. Cypress CT, with its transformative potential, not only refines the developer experience but also optimizes the development workflow. With visual feedback in the actual browser environment, we have full observability into our component in isolation. At Extend, our mission with Cypress CT was clear: master our UI framework and make it a joy to engineer front-end components. Immersed in React, we discovered patterns worth highlighting and curated these as code examples in the Cypress component test repository. Our evaluative efforts,

> "Component testing stands as the linchpin in front-end software development. It bridges unit tests and end-to-end tests. Cypress CT, with its transformative potential, not only refines the developer experience but also optimizes the development workflow."

— Murat Ozcan, Staff Engineer, Extend

such as CyCT vs. RTL, highlighted Cypress's edge over console-based tools."

The significance of Cypress CT became particularly clear when Extend used combinatorial testing alongside it for their component library. Being able to configure their common components in dozens, sometimes hundreds of ways, they utilized the most effective test models to derive minimal, fault-finding component configurations. This approach, which involved mounting and rendering fault-finding variations of component configurations, received team endorsement and offered a universal method to avoid problems without the need to create tests for each individual variant. Additionally, the ability to inspect and test common components, combined with visual testing integration using Cypress CT and Percy, brought about a further transformative impact.

Before the team-level rollout of Cypress CT, Extend's DevX team embarked on initiatives to address existing E2E testing challenges. Daily office hours ensured all E2E testing experiences were optimized. Following that, they focused their efforts towards component testing, utilizing custom mounts to eliminate redundant component complexities and harness network-level mocking with Cypress' intercept API. With the integration of Cypress's Vite support, which enhances the startup time for Cypress CT, the developer experience was notably elevated.

"Testimonials from our engineers underscore Cypress CT's revolutionary stature. Its intuitive design, vivid feedback systems, and usability have transformed our developmental landscape," states Murat. To sum it up, Extend enthusiastically supports the capabilities of Cypress CT and encourages other organizations to explore its vast potential.

Results

The adoption of Cypress CT led to remarkable outcomes for Extend. The DevX team efficiently trained engineers to compose component tests with Cypress CT in a fraction of the time compared to Enzyme. According to Murat, this resulted in "usually half or third of the code compared to before, with the added benefit of visual feedback for the engineer in the real browser with devtools". He further noted, "Because of the built-in retry mechanism of the Cypress API, any low level tests doing meaningful work—beyond just rendering—are exponentially more stable with Cypress Component Testing."

Cypress CT also simplified and shortened the test-writing process by enabling effective network mocking. Tests run live during development, allowing developers to observe the component as opposed to relying on Jest watchers. This marks a significant departure from the previous need to use low-level mocks with Jest to mock 80% of their components, along with the intricate process of mocking hooks and other providers.

> "Testimonials from our engineers underscore Cypress CT's revolutionary stature. Its intuitive design, vivid feedback systems, and usability have transformed our developmental landscape."

— Murat Ozcan, Staff Engineer, Extend

This shift was so influential that, within a quarter, Cypress CT surpassed E2E testing in the pilot application. The success of the approach served as a guiding example for all developers engaged in testing within the application.

The positive results from Cypress CT adoption were so compelling that developers who initially adopted it began contributing to other applications and advocated for a shift towards Cypress in those as well. This success has set the stage for Extend's ultimate goal of fully migrating from React 16 and transitioning all Jest & Enzyme component testing to Cypress CT.

Extend currently executes close to 3 million Cypress E2E tests a year in CI, and 1 million Cypress component tests. The expected number post-migration is under 2 million component tests per year. Their applications generally can be released from a PR to production in under an hour, with migration of monorepos to individual repositories aimed at further reducing the mean time to production in the near future.

With over a thousand Cypress component tests in the pilot application, the CI feedback time is under 10 minutes, thanks to parallelization with Cypress Cloud. Comparatively, unit testing, type checking, and linting take upwards to 15 minutes in their monorepo's PRs.

Cypress Component tests are, on average, half to a third the lines of code compared to their Enzyme pre-migration counterparts. Coupled with visual feedback in the real browser and recorded feedback in CI, Extend's engineers achieve higher value for less cost.

Extend's journey with Cypress Component Testing showcases how a strategic shift in testing methodologies can significantly impact test efficiency and developer productivity. By embracing Cypress CT, the organization streamlined testing processes, reduced test creation times, and promoted seamless front-end engineering among its developers. The success of this endeavor underscores the pivotal role of component testing and Cypress CT in optimizing front-end software engineering processes and ensuring application quality.

> "Because of the built-in retry mechanism of the Cypress API, any low level tests doing meaningful work—beyond just rendering—are exponentially more stable with Cypress Component Testing."
>
> — Murat Ozcan, Staff Engineer, Extend