International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Frontend Testing Improvement: A New Approach to Ensuring Web Application Quality

Chakradhar Avinash Devarapalli

Software Engineer
Email: *avinashd7[at]gmail.com*

Abstract: ***This paper looks at the numerous challenges that are present in frontend testing for developers. It then proposes a novel solution inspired by Netflix's proactive failure testing framework and the Molly algorithm. We look into the integration, deployment, and adoption strategies necessary for the successful implementation of proactive failure testing. With the help of lightweight components and automation frameworks, developers can streamline testing workflows and improve overall software quality. Through effective training initiatives and feedback mechanisms, organizations can foster a culture of continuous improvement in frontend testing practices.***

Keywords: Frontend Testing, Proactive Failure Testing, Molly Algorithm, Web Application Quality, Integration

1. Introduction

Frontend testing is a very critical element of web or application development. We must make sure that every element works right, the code does not break up, or there are no interdependencies that lead to performance issues. There are several ways to test the application or the website's front end, like checking small parts (unit testing) bit by bit or testing everything together (integration testing) in one go. But each approach has its own problems [1].

Sometimes, when we test small parts, we miss how they work together. Other times, when we test everything together, it's hard to see what's going wrong. Much like fixing a household item or a car. We can check each part separately, but we need to see how they all work together as well to know if it runs smoothly.

Testing websites may be problematic because there are a number of things to consider at all times. For instance, some common issues with frontend testing include:

- Poor interaction or difficulties because of the complex UI elements, such as the dropdown components, breadcrumbs, etc.
- Inability to test aspects like CORS setup or GraphQL calls comprehensively.
- Complex and unintuitive test authoring and debugging processes [2].
- Challenges in making assertions on spies/mocks or executing code within the application.
- Limited capability to handle enterprise - level applications with sophisticated authentication mechanisms and build processes.

Over the years, a large number of work structures and theories have come forth to solve this issue. However, in one form or another, these problems still exist.

For instance, some tools help with small parts, but they're not good at checking the bigger picture. Alternatively, other tools help with the big picture but don't work well with complex stuff.

Element to element (E2E) Component Testing has become a widespread tool for showcasing the progress in enhancing testing methodologies, yet gaps persist in addressing the nuanced demands of modern web applications [3].

However, that doesn't cover the entire picture, either. E2E takes a lot of time and efforts to text, especially in larger applications or websites with more complicated functionalities.

SafeTest by Netflix [4]is a concept under development, focusing on utilizing a number of unique features such as deep linking, two - way communication between browsers and test contexts, and associated reporting capabilities. It is a novel approach to front end testing focusing on proactive failure testing, automating the process across the production phase as well. While there isn't much information about SafeTest yet, Netflix has introduced automated and proactive means as a prelude to this testing methodology.

This paper takes a closer look at the challenges that developers currently face for front end testing and how automated failure testing is set to help tackle these challenges – especially with newer technologies currently under development.

2. Literature Review

Frontend testing in web development is crucial for ensuring reliability and user satisfaction. Integration testing faces challenges like comprehensive coverage and managing dependencies [2]. Debugging is complex due to intricate UI elements and testing framework limitations [1] [3].

Netflix's proactive failure testing, exemplified by the Molly algorithm, advances testing methodologies [7]. E2E testing is vital but struggles with scalability and resource consumption [4]. Frontend development evolution influences testing practices, emphasizing agility and performance optimization.

3. Current Landscape

When it comes to testing, one of the most prevalent factors to consider is time and developer exhaustion. Duplication of

testing efforts not only leads to wasted time, but also leads to developers getting frustrated due to the repetitive nature of work therein. The manual labor involved in testing may end up delaying projects – especially in the case of larger, more extensive and complicated projects.

Netflix has been actively looking for improved development and testing of the frontend – especially when it comes the application and its website. One key issue that the company faces is that its application needs to span across a very wide range of appliances. From telephone screens all the way to computers and the wide array of monitors and TV screens; its application needs to cater to a rather diverse environment.

At Netflix, proactive failure testing has proven effective in ensuring product reliability by preparing systems for production environment issues. Manual efforts, though beneficial, are limited in scope and efficiency. The pursuit of a more comprehensive approach led Netflix to develop a failure testing approach called Molly, inspired by Peter Alvaro's work [5].

The concept of Molly came to in 2018 to solve the significant challenges that frontend testing developers face due to the nature of modern web applications. Despite advancements in testing methodologies, several persistent problems hinder efficient and comprehensive testing of frontend components.

3.1 Difficulty in Interaction with UI Elements

Testing UI elements like dropdown components or interactive widgets often prove challenging.

Traditional testing approaches end up struggling to simulate user interactions accurately, leading to incomplete test coverage and potential functionality gaps.

3.2 Limited Testing of Backend Integration

Frontend testing often overlooks comprehensive testing of backend integration points such as CORS setups or GraphQL calls.
Ensuring seamless communication between frontend and backend systems requires robust testing strategies, which are often lacking in conventional testing frameworks.

3.3 Test Authoring and Debugging

The process of authoring and debugging frontend tests can be cumbersome and unintuitive.

Traditional testing frameworks may require verbose test scripts and lack user - friendly debugging tools, leading to inefficiencies in the testing workflow.

3.4 Inability to Make Assertions on Application Behavior

Testing tools today often lack the capability to make assertions on spies, mocks, or execute code within the application context effectively.

This limitation restricts the ability to validate critical application behaviors and edge cases, leading to potential functionality issues in production environments.

3.5 Limited Scalability for Enterprise Applications

Conventional frontend testing approaches face scalability challenges when applied to large - scale enterprise applications with sophisticated authentication mechanisms and build processes.

Testing frameworks may struggle to handle the details of enterprise - level applications, resulting in inadequate test coverage and reliability.

The talent shortage seen since COVID - 19 also plays a major role here, as 88% of companies have reported that they have been struggling to find, hire, and retain QA engineers [6].

3.6 Inefficiencies in End - to - End Testing

End - to - end (E2E) testing, while essential for validating complete user workflows, often consumes significant time and resources, especially in larger applications with functionalities [7].

The time - intensive nature of E2E testing limits its feasibility for frequent testing cycles and agile development workflows.

However, since 2018, the frontend development world has evolved considerably – to the point that there are now a number of challenges that even Molly cannot deal with. For example, one of the most pressing issues that need to be dealt with includes the limitations in testing due to limited numbers of QA engineers available. The general inefficiencies in end - to - end testing is also a major concern that companies need to deal with. Automation stands as a critical element here, providing solutions to almost all the challenges developers currently face.

4. Proposed Solution

The world of frontend development and testing has seen significant changes over the years. COVID - 19 shifted consumer demands considerably, and as a result, the issues within applications and websites also changed quite a bit. Consequently, so did the need for front end development models [7].

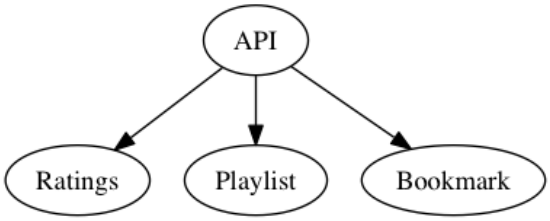
In response to the challenges that the current developers face for frontend testing, Netflix has pioneered a proactive failure testing approach aimed at enhancing product reliability and streamlining testing workflows. The proposed solution, inspired by Peter Alvaro's Molly framework, leverages a lineage - driven fault injection technique to identify potential failure points and preemptively address system vulnerabilities [1].

The core of the proactive failure testing approach lies in the Molly algorithm, which analyzes successful requests and retroactively identifies failure points that could have prevented the desired outcome. Systematically injecting failures at various points within the request execution path,

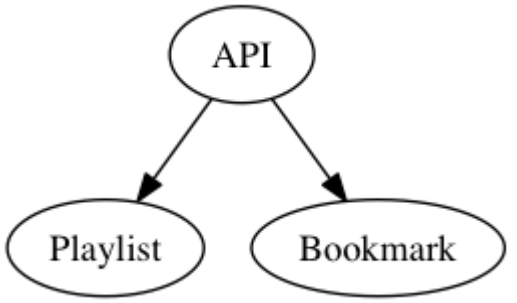
the algorithm iteratively explores potential failure scenarios and evaluates their impact on system performance.

The workflow of the proposed solution is as follows:

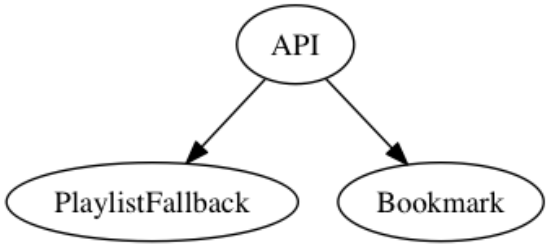
1) Identification of Necessary Components:
Molly begins by identifying the components necessary for a successful request execution, including API calls, resource loading, and service interactions.



2) Failure Injection and Experimentation:
a) Failure points are randomly selected from the identified components and injected to simulate system failures.
b) The request is rerun with the injected failure to evaluate its impact on request success or failure.



3) Outcome Analysis:
a) Molly categorizes request outcomes into three possibilities: request failure, successful request without critical failure points, and successful request with alternative execution paths.
b) Each outcome is analyzed to update the failure equation and explore new failure points iteratively.



4) Exploration Strategy:
Molly adopts an exploration strategy that computes all solutions satisfying the failure equation and randomly selects from the smallest solution sets to prioritize critical failure points.

Netflix's implementation of proactive failure testing integrates seamlessly with the company's existing infrastructure and testing frameworks. With the help of Netflix's tracing system and injection points provided by the FIT service, Molly builds a comprehensive request tree to analyze request execution paths and potential failure points. This also makes for a very compelling use case, where normally, this would take more than 2^{100} attempts and hours lost via developer frustration. However, with the help of this proposed methodology, Netflix was able to find five distinct failure points in just 200 tries.

Netflix prioritizes member experience as the primary metric for evaluating request success. Tapping into device - reported metrics streams, Netflix assessed whether a request resulted in member - facing errors, providing valuable insights into system performance and reliability.

Furthermore, to address challenges related to request idempotence and behavior mapping, Netflix employed equivalence classes to group requests with similar behaviors. Analyzing request features such as paths, parameters, and device information, Netflix creates request classes that cover similar request behaviors and failure scenarios, enabling efficient testing and analysis.

4.1 Idempotence and Request Class Mapping

In addressing the challenges of frontend development, particularly in the world of proactive failure testing, the concepts of idempotence and request class mapping play crucial roles.

Developers often come face to face with the task of determining whether certain requests are idempotent and safe to replay, especially when analyzing the impact of failure injection on request outcomes.

To overcome this challenge, a sophisticated approach to request class mapping has been devised, focusing on requests generated within the framework's context. Getting back to our use case, these requests typically adhere to structured JSON paths, such as 'videos', 'profiles', and 'images', providing insights into the internal services required to fulfill them. [5] Using the inherent structure of Falcor requests, Netflix created request classes that cover similar request behaviors and failure scenarios. This approach, in turn, enhanced the efficiency of failure testing by grouping requests with comparable characteristics, enabling streamlined analysis and targeted failure injection.

5. Academic Review of Perceived Challenges

Table 1: Table of Studied Literature Regarding Challenges

Name	*Title*	*Challenge Discussed*
Leung and White	A study of integration testing and software regression at the integration level	Challenges in integration testing and regression
Wolfgang Mayer	Model - Based Debugging – State of the Art And Future Challenges	Advancements and challenges in model - based debugging
Y. Li	Front - end testing: an important part of quality assurance in Front - end development	Importance and challenges of front - end testing
Netflix Technology Blog	Automated Failure Testing	Implementation and benefits of automated failure testing
Alvaro	Molly	Overview of the Molly project and its significance

6. Implementation & Deployment

The implementation and deployment of Netflix's proactive failure testing framework, inspired by the Molly algorithm, involves a comprehensive approach to seamlessly integrate this novel testing methodology into existing frontend development workflows. This process is characterized by several key steps aimed at optimizing the testing framework's efficacy and ensuring its wide adoption among development teams.

5.1 Infrastructure Integration

The first step involves assessing and integrating the testing framework with current infrastructure components, including version control systems like Git, and CI/CD platforms. This ensures the framework can be easily adopted without disrupting existing development processes.

For example, the following is a sample code for the integration of the code frontend testing repository to accommodate testing scripts, configurations, and dependencies, thereby facilitating versioning and collaboration among development teams. This example focuses on integrating a proactive failure testing framework with version control systems (e. g., Git) and CI/CD platforms.

```
frontend - testing - repo/
|
├── tests/
|   ├── unit/
|   |   ├── test - utils. js
|   |   ├── componentA. test. js
|   |   └── componentB. test. js
|   |
|   ├── integration/
|   |   ├── integration - test - utils. js
|   |   ├── featureA. test. js
|   |   └── featureB. test. js
|   |
|   └── e2e/
|   ├── e2e - test - utils. js
|   ├── scenarioA. test. js
|   └── scenarioB. test. js
|
├── configurations/
|   ├── jest. config. js
|   ├── babel. config. js
|   └── eslint. config. js
|
└── package. json
```

Here,

- **tests/**: This directory contains subdirectories for different types of tests, including unit tests, integration tests, and end - to - end (E2E) tests. Each test file focuses on a specific aspect or feature of the application.
- **tests/unit/**: This directory contains unit tests that verify individual components or functions in isolation. Each test file typically corresponds to a single component or function being tested.
- **tests/integration/**: Integration tests in this directory verify the interactions and behavior of multiple components or modules within the application. These tests may involve testing API endpoints, data fetching, or UI components working together.
- **tests/e2e/**: End - to - end tests simulate real user scenarios and interactions with the application. These tests typically cover complete user workflows, including navigation, form submissions, and UI interactions.
- **configurations/**: This directory contains configuration files for testing tools and frameworks used in the project. Examples include Jest configuration (jest. config. js) for unit and integration tests, Babel configuration (babel. config. js) for transpiling JavaScript code, and ESLint configuration (eslint. config. js) for code linting and quality checks.
- **package. json**: The package. json file specifies project dependencies, scripts, and metadata. It includes dependencies for testing frameworks (e. g., Jest), testing utilities, and other development dependencies.

For reference, these are the sample code snippets for the respective syntax:

5.1.1 jest. config. js (Jest configuration):

```
module. exports = {
testEnvironment: 'jsdom',
testMatch: ['<rootDir>/tests/**/*. test. js'],
setupFilesAfterEnv:    ['<rootDir>/tests/setupTests.
js'],
};
```

5.1.2 babel. config. js (Babel configuration):

```
module. exports = {
 presets: ['[at]babel/preset - env', '[at]babel/preset -
react'],
plugins: ['[at]babel/plugin - transform - runtime'],
};
```

5.1.3 eslint. config. js (ESLint configuration):

```
module. exports = {
 root: true,
 env: {
 browser: true,
 es6: true,
 jest: true,
 },
 extends: ['eslint: recommended', 'plugin: react/recommended'],
parserOptions: {
ecmaVersion: 2018,
sourceType: 'module',
ecmaFeatures: {
jsx: true,
 },
 },
 plugins: ['react'],
 rules: {
 'react/prop - types': 'off',
 },
};
```

Organizing the frontend testing repository with clear directory structures and configuration files allows for a much more streamlined integration with version control systems, CI/CD platforms, and development workflows. Developers can collaborate effectively, maintain code quality, and automate testing processes using the defined structure and configurations.

5.2 Tooling and Automation

Selecting and customizing the right set of tools and automation frameworks is crucial for facilitating comprehensive test coverage. The framework should support integration with established testing libraries like Jest, enabling automated test suites that can run within CI/CD pipelines for continuous testing.

5.3 Developer Training and Adoption

To maximize the framework's benefits, development teams require training on its methodologies, tooling, and best practices. This could involve workshops, documentation, and ongoing support to encourage widespread adoption and proficiency in proactive failure testing practices.

5.4 Iterative Improvement and Feedback Mechanisms

Implementing feedback mechanisms to gather insights from users of the framework is essential for its continual refinement. Regular reviews and surveys can help identify areas for improvement, ensuring the testing approach remains effective and relevant to developers' needs.

7. Significant Impact on the Field

The introduction of Netflix's proactive failure testing framework, particularly its reliance on the Molly algorithm for lineage - driven fault injection, marks a significant advancement in the field of frontend testing.

By automating the identification and injection of potential failure points, the framework significantly reduces the time and manual effort required for thorough testing. This enables developers to focus on other critical aspects of development, fostering faster iteration cycles and product enhancements.

Furthermore, the ability to preemptively identify and rectify potential failures before they impact the user experience directly contributes to higher quality web applications. This proactive approach to testing ensures that products are more reliable and performant, enhancing user satisfaction and trust.

The framework's ability to handle complex, enterprise - level applications with sophisticated authentication mechanisms also addresses a critical gap in existing testing methodologies. Its scalability supports the growing complexity of web applications, making it a valuable tool for organizations of all sizes.

8. Conclusion

Frontend development and testing highlights the critical importance of ensuring the reliability and quality of web applications. The challenges outlined in this paper, ranging from UI interaction issues to the scalability issues of enterprise - level applications, highlight the need for innovative approaches to frontend testing.

Netflix's proactive failure testing framework, inspired by the Molly algorithm, presents a promising solution to address these challenges.

The integration and deployment of the proactive failure testing framework require careful consideration of infrastructure components, tooling, and developer adoption strategies. Leveraging lightweight, modular components and automation frameworks compatible with popular development frameworks ensures broad applicability and seamless integration into existing workflows.

Furthermore, effective training initiatives and feedback mechanisms are essential for improvement and accountability within development teams.

References

[1] H. Leung and L. White, "A study of integration testing and software regression at the integration level, " in *IEEEXplore*, San Diego, USA, 2002. Original Date: 26 - 29 November 1990. [Accessed 01 11 2019]
[2] M. S. Wolfgang Mayer, "Model - Based Debugging – State of the Art And Future Challenges, " 30 05 2007. [Online]. Available: https: //www.sciencedirect. com/science/article/pii/S157106610700196X. [Accessed 01 11 2019].
[3] Y. Li, "Front - end testing: an important part of quality assurance in Front - end development, " 30 01 2019. [Online]. Available: https: //www.theseus. fi/handle/10024/265272. [Accessed 01 11 2019].
[4] Netflix Technology Blog, "Automated Failure Testing, " 20 01 2016. [Online]. Available: https: //netflixtechblog. com/automated - failure - testing - 86c1b8bc841f. [Accessed 01 11 2019].

[5] P. Alvaro, "Molly, " Github, 2018. [Online]. Available: https: //github. com/palvaro/molly. [Accessed 01 11 2019].
[6] Pulasthi Perera; Roshali Silva; Indika Perera, " Improve software quality through practicing DevOps, " *IEEEXplore*, 15 01 2018. [Online]. Available: https: //ieeexplore. ieee. org/abstract/document/8257807. [Accessed 01 11 2019].
[7] Thanh - Toan Do; Anh Nguyen; Ian Reid, " AffordanceNet: An End - to - End Deep Learning Approach for Object Affordance Detection, " *IEEEXplore*, 13 09 2018. Original Conference: 21 - 25 May 2018 [Online]. Available: https: //ieeexplore. ieee. org/abstract/document/8460902. [Accessed 01 11 2019].
[8] Kevin Hoffman, "*Building Microservices with ASP. NET Core*" Book. Publisher: O'Reilly®. Beijing, Boston, Farnham, Sebastopol, Tokyo.31 08 2017 [Accessed 01 11 2019].