The Competitive Subculture of Cryptocurrency Mining

A Technical Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Kevin Chen

March 20, 2023

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Kevin Chen

Technical advisor: Rosanne Vrugtman, Advisor, Department of Computer Science

Creating Dynamic Mock API Responses for UI Testing

CS4991 Capstone Report, 2022

Kevin Chen
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
kyc4mr@virginia.edu

Abstract
A large software company was planning to undergo large changes in their API contracts. However, this meant that they needed to create a way to smoothly integrate the current UI tests with the new API contracts without having to retype all their UI tests. UI tests use the API responses to execute different commands by mimicking human interactions. The idea was to create dynamic mock API responses to accomplish this. Instead of mocktail files storing API responses for testing, code done directly in Swift could easily be modified if changes occurred to API response structures. Our team interviewed teams for possible approaches, then implemented the code in Swift before creating robots using the newly created dynamic mock API responses. This code was eventually pushed to the main code repository. For next steps, dynamic API mocks will eventually be implemented in more teams across the company.

1. Introduction

Imagine you are working at a large company, conducting hundreds of thousands of tests to make sure your application works. However, one day you need to change every single one of these tests. This is what happened at the large company I worked at. API contracts, which receive application requests to perform third party functions, were going to change and the existing UI tests had to change in order to account for the testing of these third-party API calls. Our goal during the internship was to create dynamic mock API responses, as the solution.

2. Related Works

Creating API responses using Swift code is not a new thing (Mainguy, 2019) [1]. Creating these responses utilizes aspects of Swift, particularly "NetworkManager," to intercept API calls to the network and return the mocked API response. This is the approach we took in the internship project. However, the difference is that all previous mocks will also have to bypass a UI testing manager used in the company.

Another approach that has been used before was to create an entirely new architecture (Tirodkar and Khandpur, 2019) [2]. This approach builds upon Apple's own UI testing framework, XCUITest. It fixes several issues with the way user interactions were mimicked such as flakiness in the testing and adding more robust testing. In our project, we did not create a new architecture but instead used the existing framework's UI testing options and creating the dynamic mocks off the framework.

3. Process Design

Because the software company was large, creating a process for changing all of the UI tests would require several steps. We split the task into four parts: 1) asking other

teams what kind of approaches might work; 2) creating the response format; 3) designing robots to use the new mock API responses; and 4) creating robots using the new mock API responses to do UI testing automatically.

3.1 Interviewing Teams

The first step was the consult via Zoom other teams that were also working on iOS development. Our intern team had four interns. We split into two teams and interviewed other teams pursuing different methods. Suggestions on possible approaches included bypassing the current method of API responses. At the time, API response went through a company-developed tool to store API responses and return them whenever a method used an API for testing. This would be necessary since the changing of API responses would go to the tool and return the old API responses unless we re-recorded every single one through the tool, which would take a lot of time and would not be sustainable in the long-term. So several meetings were scheduled on ways to bypass this tool.

3.2 Creating the Response Format

Once we had an idea of the approach, we presented our findings to the team and began working on the response format. The four of us worked through peer-coding with our mentor. During the few coding days, we took a room in the office and worked through one person's laptop projected on the large screen in the room. We made several modifications to the current repository in CocoaPods, which was the dependency manager for the application. We also made a sample for this by removing the .tail file we used for holding a mock API response for email authentication and replacing it with the dynamic mock stub we created.

3.3 Creating Robots

Once we had sufficient evidence that our dynamic mock API response was working, we presented our approach to the team along with the intention of creating robots using to make UI testing easier. These robots functioned to execute certain UI tests on command, which allowed for testing of small snippets of the code such as individual functions without needing to test the whole code.

The structure of the robots were:

- a main, "orchestrating" robot which would be in charge of other robots and could direct them to do certain tasks such as make a change in the dynamic mock API response or execute a certain task;
- Robots directed by the orchestrating robot that would use the dynamic mock API response created and execute tests.

4. Results

Our solution is now being used in most of the iOS code and has been pushed to the main code repository. Before, one would need to manually change each .tail file if a certain aspect of the call is changed; however, if an API contract changes in the future, one only needs to make a change in one of the mock API stubs, and all the API calls that use that stub will change, saving hours for UI testing changes.

5. Conclusion

This project reduced the amount of work needed to maintain UI tests involving mock API responses. At the time, the large software company needed a way to continue mocking API responses in their UI tests for their iOS app. Creating dynamic mock API responses was the optimal solution that allowed for long-term use with relatively low effort. After this solution was implemented, there was time to create robots that utilized these new dynamic mock API

responses to automate the iOS app's UI tests.

6. Future Work

The next phase of this work would be to scale this for the whole iOS app at the company. Once this is completed, its integration would allow UI tests to be done more seamlessly and prevent errors in the application when API contracts are edited. This way, robots can also work together to do tests across the whole application; allowing for faster testing.

References

[1] Mainguy, J. 2019. Painless UI Testing in iOS (Part 1) - Mocking the Network | by Jean. *Medium*. Retrieved November 8, 2022 from https://code.egym.de/painless-ui-testing-in-ios-part-1-mocking-the-network-ffbd6ab4809a

[2] Tirodkar, A. and Khandpur, S. 2019. EarlGrey: iOS UI Automation Testing Framework. *IEEE*. Retrieved November 18, 2022 from https://ieeexplore-ieee-org.proxy1.library.virginia.edu/document/8816871