iOS Development: Creating A Command Line Tool to Auto-Generate Boilerplate Code

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Alex Chan

Spring, 2023

Briana Morrison, Department of Computer Science

iOS Development: Creating A Command Line Tool to Auto-Generate Boilerplate Code

CS4991 Capstone Report, 2022
Alex Chan
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ac5ug@virginia.edu

Abstract
Capital One Financial Corporation, an American bank holding company, wanted to increase production of its iOS feature developers by replacing its inefficient, manual process of writing repetitive code for plugin request files with a tool that automatically generated such code. I worked with a team and created a command line tool that generates boilerplate code for the API request file of a given plugin. I learned to use Apple's integrated development environment Xcode and programming language Swift, the version control software GitHub, and code generation framework SwiftGen. Additionally, I studied the iOS environment, worked through an agile process, and constantly communicated with my team to ensure the success of the project. As a result, I developed a minimum viable product that speeds up the request file creation process by roughly 85%. There are plans to further improve upon the base product, to allow more unique plugins to utilize the command line tool, as well as expand a similar concept for Android development.

1. Introduction and Background

With over 5.6 million reviews on the Apple app store, Capital One Mobile is widely used by the company's 44 million users. In order to continue its 4.9-star rating (out of 5), iOS developers must continue to update the app, ensuring its features are up-to-date with modern software tools and always functioning properly. However, apart from making more hires for software developer positions, there is no simple way for Capital One to further increase its speed of production. This applies to the corporation's iOS application, where feature developers work to create new features as well as improve pre-existing ones offered by Capital One Mobile.

For any of these features to appear on a user's screen, an API request must be made to obtain the necessary designs and functionalities from Capital One's API. Therefore, code must be written for an API request for every single plugin created. Additionally, the general code structure for each API request is very similar, so the work to be done by feature developers tended to be repetitive. Furthermore, developers would place the API request files in different locations within their plugins, making things more difficult to locate when looking at another's code. By making a tool that created starter code for the API request files, the relative location of such files could be standardized and feature developers would be able to save time that can be put towards implementing new functionalities.

2. Related Works

According to Apple Developer (2022), Xcode is an integrated development environment (IDE) used to write code in the programming language Swift, which is used to create iOS applications [1]. Xcode inspired my internship project because it is, in a sense, what I was trying to improve. The Apple IDE is used by all iOS developers to create powerful projects, offering many helpful features such as auto-completion and syntax highlighting to increase productivity. Despite these great tools, there is no way for Xcode to create large chunks of code, and this is where my team's command line tool came into consideration.

With the evolution of mobile banking, the need for financial corporations to allocate more resources into mobile development is only increasing. Mobile banking has grown very prevalent in people's everyday lives, from only being able to show account balances to now offering the ability to make transactions online. As Yuen (2002) points out, customers can even personalize alerts to ensure they pay their paperless bills on time [2]. Such a technological innovation is related to my internship project because it automated basic financial procedures, making them much more efficient and accessible. This led to an increase in demand for highly functional, modern, mobile banking applications, especially from the big hitters in the banking industry.

3. Project Design

We gathered the project specifications before beginning to work on the project. The development process consists of research on the iOS environment and plugin files, deciding on tool logistics, and finally creating the tool. Challenges that arose throughout the development process included understanding the company's iOS environment and acquiring necessary input for our tool to function.

3.1 Project Overview

The project manager gave my team the task of using a code generation framework to develop an auto code generation tool for API request files. Apart from restrictions of the code being in Swift, Apple's programming language for iOS products, there was plenty of creative freedom given to use in terms of how our end product would function.

3.2 Project Specifications

As long as certain project specifications were addressed, there was some flexibility in the design of the code generation tool. Most of the requirements revolved around making things easy for the user, including:

1. No need to download any additional resources to utilize the tool
2. No need to create any additional files or directories to use the tool
3. Ability to reuse the tool to update and replace preexisting request files

Essentially, the tool was to be designed so that the user had to do minimal work.

3.3 Development Process

The development process consists of the research done to make design decisions on the tool, and the subsequent creation of said tool.

3.3.1 Research and Decisions

The first thing to learn was the architecture of the iOS application. I am unable to disclose exact details, but for the purpose of this project, API information for a given plugin was obtained from a site in the form of either a JSON or YAML file. Each request file varied from plugin to plugin. Therefore, one task was to determine the commonalities among the majority of plugin files, so we knew what portions of code we could generate that would be helpful for most users. This mainly included creating data structures that held information on a given plugin API.

Additionally, any open-source software we wanted to utilize had to first be WhiteSource scanned to ensure its safety before being implemented into the iOS environment. This was done for all Swift code generation frameworks in consideration. In the end, we decided to use a code generator tool called SwiftGen because of its compatibility with JSON and YAML files and structured use of a configuration file. We also decided to create a command line tool because an environment for such implementation already existed in the working iOS system.

3.4 Designing a Command Line Tool

We broke up our tool's working process into its individual tasks in chronological order. We developed or revised new and preexisting Swift files with the necessary code to complete each task.

3.4.1 Implementing SwiftGen

First, we integrated the code generation tool SwiftGen into the iOS architecture through a pod install.

3.4.2 Creating SwiftGen Inputs

Second, we performed several tasks in parallel. One such task was the creation of templates using a language called Stencil. This was a necessary input for SwiftGen so that it knew what code to output given the desired plugin API's JSON or YAML file. Additionally, a struct was created to hold the required information for SwiftGen's configuration file, which was necessary to state what files were being used as inputs, their file type, and what the outputted file should be named.

3.4.3 Parsing the APIs

Third, we created functions to parse the JSON and YAML files containing the plugin APIs, obtaining the necessary data for the request files to be outputted.

3.4.4 Combining the Components

Finally, all of the aforementioned components which were created in separate files were integrated into a central file containing the process of using our command line tool. This was done so that, when our command was used given a plugin name and its respective API as a JSON or YAML file, a folder called API is created in the plugin's directory and temporary files necessary for SwiftGen are created. SwiftGen then generates the request files in the API folder and the temporary files are subsequently deleted.

3.4 Challenges

The main challenge was learning the iOS environment and general structure of the request files. When looking through examples, there were numerous variations of variables for the API description structs that needed to be accounted for. Additionally, having had no experience to in working with mobile development, quickly understanding the architecture proved to be difficult.

One of the bigger challenges was figuring out how to obtain the JSON/YAML file of the API description for a plugin. These could be obtained by selecting a download option from the webpage containing the APIs, but the security of the site made it difficult to automate the download process. Consequently, for the final product we achieved, it was still necessary for users to provide the JSON/YAML file.

4. Results

The command line tool was demonstrated for potential users, which consisted of mobile developers on both feature and platform teams. Mainly, the tool is to be used by those on feature teams who develop new plugins and features for Capital One's iOS application. Through prototyping, our command line tool is expected to speed up the creation of request files by ~85%. At the time of writing this, my manager's team is working to bring the tool into mass consumption for such users' benefit.

5. Conclusion

The tool created for our project is a great example of how technology can be used to help the engineers who are innovating as opposed to just the end users of a company. Our demo indicated that our tool will save iOS feature developers time which they may allocate towards more pressing matters. Consequently, this project revealed the possibilities provided by code generation frameworks such as SwiftGen. Such software could greatly improve work efficiency especially for larger organizations with a structured software environment.

6. Future Work

Considering our tool creates code for the basic structure that nearly all request files require, we left the project with plenty of ideas for further customization of subgroups of request files. This would mean adding flags as parameters for our command to auto generate more case-to-case code for different plugins. Another idea was to create a user interface for the tool instead of having to run it through the command line. This would make the tool more intuitive since users would not have to learn the different parameters and flags required to run the command.

There was also hope that our project for the iOS environment could be used to help develop a similar resource for the Android side of things. Nonetheless, the main goal for the future left by my intern team was to have our tool be approved and integrated into the main working branch of the iOS environment so that any and all developers may utilize it.

References

[1] Apple Developer. 2022. Xcode14. Retrieved September, 23, 2022 from https://developer.apple.com/xcode/

[2] Yuen, M. 2022. State of mobile banking in 2022: top apps, features, statistics and market trends. Retrieved September, 23, 2022 from https://www.insiderintelligence.com/insights/mobile-banking-market-trends/