Upgrade Alerts: Preventing the Use of Bad App Versions

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

Grace Kisly

Spring, 2022

Technical Project Team Members

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

Aaron Bloomfield, Department of Computer Science

Upgrade Alerts: Preventing the Use of Bad App Versions

CS4991 Capstone Report, 2023

Grace Kisly
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
gck4mwf@virginia.edu

ABSTRACT
Mastercard, a financial services company, had no way of preventing users from using broken or outdated versions for their expense tracking application, which could potentially lead to app crashes and failures on the user end. In order to alleviate this, I developed two types of alerts for iOS to either suggest or force the user to update their version prior to entering the app. Through integrating Firebase Remote Config, the suggested and required app versions could be set and changed on the Firebase platform. The alert itself constructed in Swift, compares the user's app version to the Firebase app version parameters. The feature was tested successfully and deployed to over 500 users. However, given the limitation that the users would need to upgrade to the app version containing the alerts, during my time at Mastercard I was not able to see the feature in use. It would be beneficial in the future to evaluate how the feature functions on real users once they upgrade and gather user feedback.

1. INTRODUCTION

Software developers strive to develop the latest, most innovative, seamless apps for users, which would not be possible without software maintenance teams. But what could happen if these teams' work was not utilized? Although software maintenance may not appear as the most compelling or impressive job for a software engineer at the surface, it is all-encompassing and crucial to ensuring the longevity of an app. Maintenance includes fixing defects, improving performance, developing tests, and creating new features to improve the user experience. However, to have these improvements implemented, developers rely on users to update their apps to the latest versions. Without this, new features and fixes may become pointless.

In addition, bugs and defects will persist in the app as users do not upgrade, leading to an unsatisfactory user experience. As time progresses, outdated software can cause problems and even lead to app crashing, preventing customers from using the app at all. Left unaddressed, this could ultimately lead to current users being driven away from an application, and negative user feedback that would deter future users. A method for prompting users to upgrade their app is needed in order to prevent a poor user experience and the ineffectiveness of software maintenance teams' work.

2. RELATED WORKS

Ivaev (2019) detailed the use of Firebase Remote Config for iOS to control the state of an app without having to add code, to hardcode variables, and to republish the app to the App Store. The drawback of using Firebase Remote Config is the potential for throttling, which occurs when an app retrieves a value from the Remote Config platform too many times over a short duration and results in the fetch calls being halted. Ivaev shows how to incorporate Firebase Remote Config to dynamically update a label displayed in an app, which I utilized to manipulate the minimum app version as the text field in Firebase instead of a label. I addressed the throttling concern by setting the minimum time interval that the value could be fetched to a low interval only during testing, so in actual applications it would take much longer to fetch, preventing throttling.

For the user interface, Kargopolov (2020) walks through how to build an alert in Swift using the UIAlertController class. Although this is advantageous to create and display an alert to the user and to handle button selection actions, there are limited stylistic options, and he warns that if the button is not created correctly the user can never dismiss the alert. Kargopolov's demonstration of creating an alert and making an alert button call a function was helpful for me to implement the UI for the force and soft update alerts. The default style sufficed for my work, and I avoided the issue of the alert never disappearing by having the click of the button trigger a function that brings the user to the App Store link.

To connect the Firebase values to the alert, Memon (2021) describes how to incorporate a force update alert with Remote Config in Swift. The major advantage is that a user will have no way of using the app without updating their app to the new version**.** The drawback of Memon's work is that the code does not display how it intertwines within an existing app's View classes, just how to build the alert. This was integral in my project to understand how to connect the Firebase Remote Config values to displaying the alert. I addressed the drawback by using the existing code repository for the expense tracking app as an example to weave the update alerts code into the project. I also only used one Firebase Remote Config parameter, the minimum app version instead of three in Memon's work, retrieving the two other parameters in the Remote Config console from the source code.

3. PROJECT DESIGN

To design and create the Force and Soft Upgrade alerts, I worked with the other members on my team to outline design requirements and specifications prior to coding. During the development process, my team assisted me with conceptualizing solutions to the obstacles I encountered, utilizing the agile software life cycle to develop and test.

3.1 Requirements

As my team was transitioning to incorporate Firebase to test and monitor performance for the expense tracking app, it was required to set up Firebase Remote Config to set the minimum app versions. When the app is first opened, the Remote Config variable needed to be initialized prior to loading any of the initial display. The variable must to fetch the minimum app versions within 3 seconds; otherwise, the app will load without checking to see if an alert must be displayed.

For the alerts, they must be in the default "alert" style provided by Swift. Alerts must pop up when the Mastercard splash screen is being displayed. The user should not be able to interact with anything in the app other than the alert buttons when displayed. The text for the alerts must be identical to the approved strings for each alert. The Force Upgrade alert must only have one button, the update button. The Soft Upgrade alert will prompt the user to either update or dismiss. When the update button is pressed, the user will be brought to the expense app's page in the iOS App Store. When the dismiss button is pressed, the alert will disappear and the user will be brought to their initial display screen.

3.2 Specifications

A minimumAppVersion variable and targetAppVersion variable were created in the Remote Config console, set to 1.8.0 and 1.9.0 respectively as the highest available version of the app was 1.9.0. To connect these variables to the code repository, the Firebase Remote Config instance was added to the AppDelegate file, where the app launch is handled. In the InitialViewController file, responsible for creating the first view displayed upon opening the app, the Remote Config variable fetches both the minimumAppVersion and the targetAppVersion from the Firebase console. If the fetch is successful, these variables are inputted as parameters to the function that compares the minimum and target app versions with the current app version. The currentAppVersion variable is retrieved from the Info.plist file, a property list text file where all iOS apps must store an app version number.

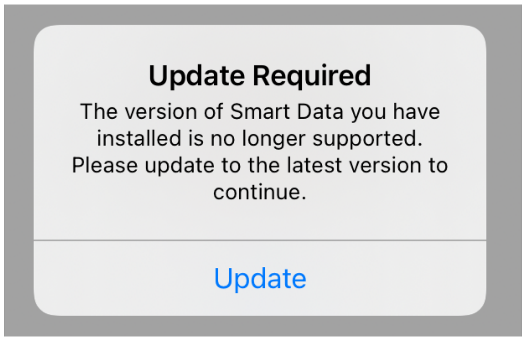
The compareAppVersions function checks for the conditions where the Force Upgrade alert or the Soft Upgrade alert should be displayed. The Force Upgrade alert is only shown if the currentAppVersion is less than the minimumAppVersion variable. The Soft Upgrade alert is only shown if the currentAppVersion is greater than or equal to the minimumAppVersion and less than the targetAppVersion. In all other cases no alert is displayed and the app launches to the initial view. Figure 1 below is the spreadsheet of edge cases I created to help me determine what the two conditions are for the alerts to be displayed.

minAppVersion	targAppVersion	currAppVersion	Result
1.8.0	1.9.0	1.7.9	Force Upgrade
1.8.0	1.9.0	1.8.0	Soft Upgrade
1.8.0	1.9.0	1.8.5	Soft Upgrade
1.8.0	1.9.0	1.9.0	Nothing
1.8.0	1.9.0	1.10.0	Nothing

Figure 1: Example variables for minimum, target, and current app versions to show the cases when Force or Soft Upgrade should appear.

If the compareAppVersion function returns a flag for the Force or Soft Upgrade alert to be displayed, then the alert is created in the InitialViewController file and initial app display is not shown. If the function returns no flag, then the app launches the inital app display.

When an alert flag is returned, a UIAlertController class is initialized in the InitialViewController file. The text strings of the alert's required titles and labels for the are retrieved from the Localized String file. This allows the alert text to match the language of the user's iOS device. The localized strings are set for the respective UIAlertController text values, and the buttons are created with UIAlertAction objects. Only one object is created for the Force Upgrade UIAlertController, the "update" button, while both an "update" and "dismiss" button is initialized for the Soft Upgrade. The "update" UIAlertAction handler will call a separate function to open the iOS App Store when the button is pressed. The function loads the URL to the expense app in the App Store and opens it. The "dismiss" UIAlertAction handler will remove the alert from the screen and start the loading the initial view display. A cropped screenshot of what the Force Upgrade alert looks like upon opening the app is shown in Figure 2 below.


Update Required

The version of Smart Data you have installed is no longer supported. Please update to the latest version to continue.

Figure 2: The finished Force Upgrade alert developed for the Mastercard Smart Data expense tracking app.

3.3 Challenges

One roadblock encountered was that when the alert was shown on the splash screen, it would disappear automatically when the splash screen had finished loading. This rendered the alert useless, as the app would continue loading in the background and dismiss without the user taking an action.

Another challenge I faced was in the case that the user exitted the app when the alert was shown. Before I addressed this, leaving the app would automatically dismiss the alert, so when the user returned the app would load the initial view without no alert.

3.4 Solutions

The first issue of the alert disappearing with the splash screen was originally resolved by an adjustment to requirements. At first, my team decided it would be acceptable for the app to finish loading the initial view and then displaying the alert, as the alert being shown would block the user from being able to access the screen. However, it was later determined that this undermines the function of the alert, as it should block outdated software versions from loading to prevent crashes. Thus, the issue was resolved by having the alert shown after the splash screen, instead of the alert on top of the splash screen, but prior to the initial view loading.

The next issue of the alert dismissing when exiting and returning to the app stemmed from the way the view was displayed upon reopening. This was addressed by checking for the alert flags in the function that handles when the app is reopened.

4. RESULTS

Continuous testing revealed the alert functioned as expected in the different version cases. This was done by personal unit testing and DevOps UI testing where mininum, target, and current app version values were adjusted. Tests for each case asserted that the expected lines of code were run and verified that the UI was displaying the expected screens. Although no real user data was collected prior to my departure from Mastercard, the feature was launched at the same time as the Android alert feature to the app's 1,000+ users.

5. CONCLUSION

Upgrade alerts for the Mastercard expense tracking app keeps users interacting with new features and improvements by prompting them to stay up to date with the latest version. The alerts also prevent broken and outdated software from being used, preserving user satisfaction. This project helped me develop skills in using XCode to build iOS applications in Swift. I also learned the value of setting dynamic parameters remotely with Firebase Remote Config. Most importantly, I learned how to adjust for unforeseen issues in app development environment and think critically to overcome obstacles.

6. FUTURE WORK

Moving forward it would be beneficial to track the success rates of users upgrading when the alert is displayed. Once a significant number of users are up-to-date on the version that contains the alerts, it would be worthwhile to gain insight on the number of people who receive soft or force upgrade alerts, and whether they update when they get the soft alert. It would be useful to get user feedback on the alert interface from clients, ensuring there are no difficulties. Beyond tracking real use, an improvement could be to only repeat displaying soft updates after a certain interval of time after dismissal. This would prevent a bombardment of suggested upgrade alerts every time the app is opened, but still ensures the user is frequently reminded to update.

REFERENCES

[1] Ivaev, Z. 2019. How to Implement Firebase Remote Config with Swift 5. (July 2019). Retrieved Feb 24, 2023 from https://medium.com/cleansoftware/firebase-ios-remote-config-minimalistic-step-by-step-tutorial-using-swift-5-f67cab5a3715.

[2] Kargopolov, S. 2020. How to Show an Alert in Swift. (April 2020). Retrieved [month, day, year] from

https://www.appsdeveloperblog.com/how-to-show-an-alert-in-swift/.

[3] Memon, S. 2021. Implementing Force Update Feature using Firebase Remote Config in iOS. (November 2021). Retrieved from https://medium.com/codechai/implementing-force-update-feature-using-firebase-remote-config-in-ios-52beaabebab5.