SP3-GET OUT!

Final Report

DECEMBER 4, 2022

4850-01 SENIOR PROJECT
Fall 2022

Sharon Perry

Contents

1 Project Plan

1.1 Abstract

The goal of our project was to build a mobile app that helps users block unwanted phone calls. The app is built in React Native, JavaScript, and TypeScript. React Native is a JavaScript framework for writing real, natively rendering mobile applications for iOS and Android. JavaScript is a scripting and programming language for creating dynamically updating content on websites, mobile apps, and games. TypeScript is a strongly typed programming language that builds on JavaScript, which helps developers fix errors early and write better code. The app also uses RxDB as an offline database. RxDB is an offline-first, NoSQL database for JavaScript apps. We selected this technology platform because it allows us to deliver mobile apps for both Android and iOS.

1.2 Background

Smartphones play a large role in our everyday lives. We use them for communication, news, directions, and more. At least 85% of Americans own a smartphone, and that number is expected to continue climbing (Pew Research Center, 2021). Telemarketers and scammers found great success in taking advantage of smartphone users by harassing them with automatic phone calls and text messages, reaching millions of people each day without even pressing a button.

Mobile carriers offer some protection against these calls, but it is not enough. Instead, smartphone users depend on apps to protect them. We are creating GetOut, a mobile app for Android 7+ and iOS 10+, to provide smartphone users with another way to protect themselves from unwanted calls and texts.

1.3 Objectives

We decided to split our objectives into two separate phases. Phase 1 will consist of all the "must-have" requirements that we feel are the most important to users. This includes silent call blocking, configuration, a call log, and optionally blocking all non-contacts.

Phase 2 will consist of all the "nice-to-have" requirements that users will benefit from, but we may not have time to implement. This includes using public API's to heuristically identify spammers, SMS text blocking using the same algorithm as call blocking, an additional Machine Learning algorithm to intelligently block spam texts, and spoofed number detection.

1.4 Scope

The scope of the GetOut app is to intercept and block unwanted calls based on user configuration. This includes detecting incoming calls and checking the phone number against a whitelist and blacklist. The whitelist is a list of phone numbers to **allow**, while the blacklist is a list of phone numbers to **block**. Calls will be intercepted in the background, only showing a silent notification to the user if a call is blocked.

All calls will be displayed in a call log, which will have options for the user to block or unblock callers manually. A settings menu will allow users to view and edit the whitelist and blacklist directly, enable or disable features, and change the app's theme. Additionally, if time permits, we can extend this to text messages, add spoof detection, use 3rd-party APIs, and use machine learning to filter spam text messages.

1.5 Project Team

Roles	Name	Responsibilities	Contact
Project Owner	Sharon Perry	Ownership of project, provide requirements	sperry46@kennesaw.view.usg.edu 770-329-3895
Team Leader	John Hussey	Schedule meetings, assign tasks, submit group deliverables, code	john.luke.hussey@gmail.com 678-608-7254
Team Members	Ryan Kim	Frontend Development	ryankim0830@gmail.com 678-770-0121
	Nick Nguyen	AI Development	nicholasnguyen66@gmail.com 470-505-8993
	Colin Allen	Backend Development	colinallen.home@gmail.com 678-822-3861
	David Shipman	Documentation	dshipman@hotmail.com 404-731-5873
Advisor / Instructor	Sharon Perry	Facilitate project progress; advise on project planning and management.	sperry46@kennesaw.view.usg.edu 770-329-3895

1.6 Project Website

https://pretzeldev.com/getout/

1.7 Meeting Schedule

We meet every Sunday at 5:00pm EST.

1.8 Collaboration and Communication Plan

We have decided to use Notion to schedule tasks and Discord to communicate and host meetings. We are also utilizing Smartsheet to create and maintain the overall project schedule via the Gantt chart. For file sharing, we will use Discord and Google Drive. David (or John, if David is not present) will take and distribute meeting notes within an hour after each meeting. Furthermore, John will send a biweekly status report via email to the project owner, Sharon Perry.

General expectations for internal team communications are as follows:

- Members must reply to discussions and messages on Discord within 24 hours.
- Members must RSVP at least 24 hours before each meeting
    - If not present, members must go over meeting notes
    - Members who miss two consecutive meetings will be reported
- Members must be responsible for completing assigned tasks by given deadlines
    - Must notify John of any issues or delays of given tasks

1.9 Version Control Plan

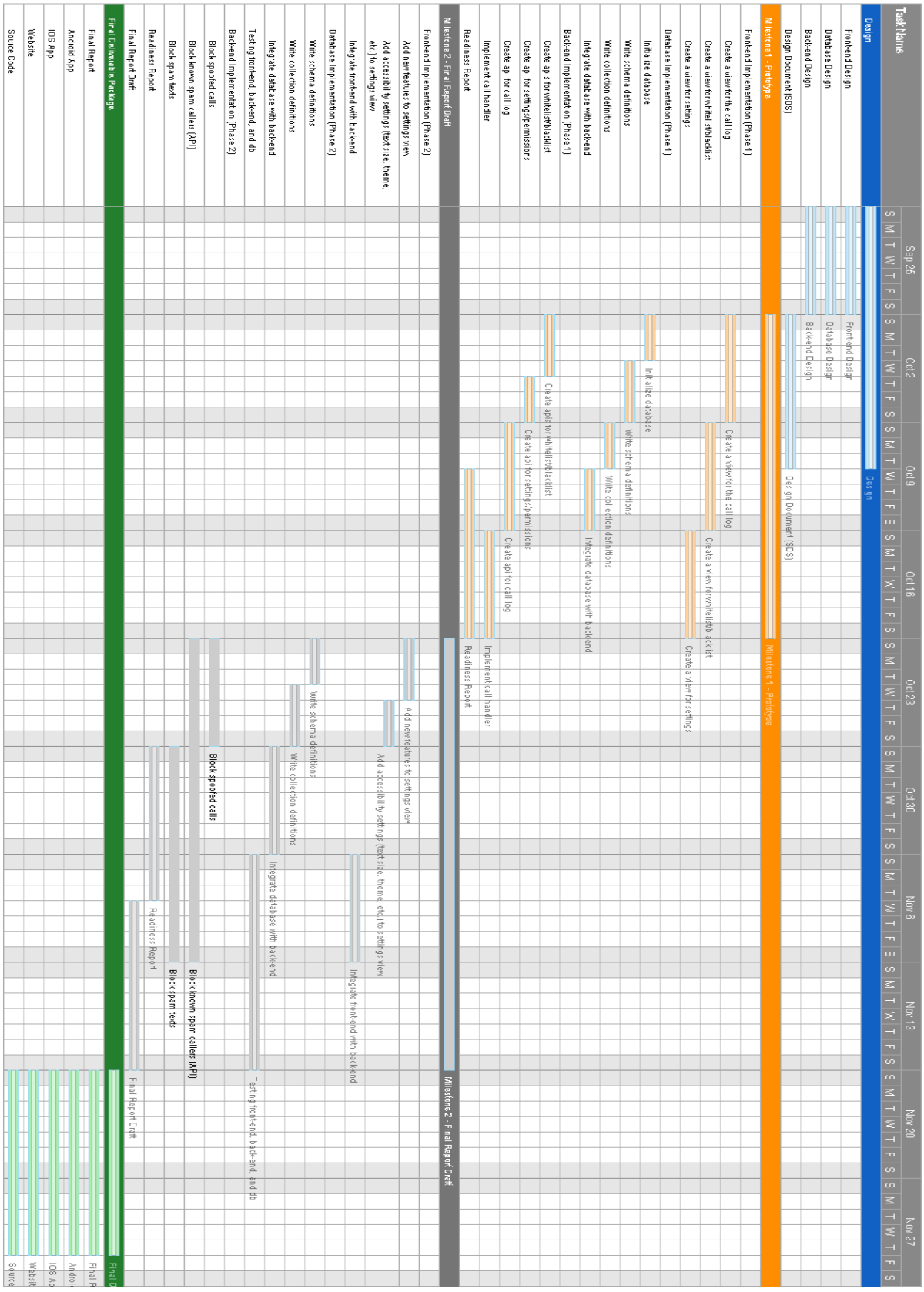
Using GitHub and GitHub Actions, we will be able to collaborate and partially automate the development process. We will have a master branch, development branch, and separate feature branches. Code changes must only be pushed to the feature branches. The code changes will then be reviewed and merged into the development branch. Then, the tester will clone, build, and test the development branch before merging it into the master branch.

1.10 Risk Assessment

Privacy is our primary concern, because this application will have access to users' information, including contacts, call logs, call states, and text messages (phase 2). While all the information will be stored locally on the user's phone, an attacker could have access to this information if the application or database is breached. To mitigate this risk, we will be transparent on what data we collect, and we will follow security practices to the best of our ability.

1.11 Project Schedule and Task Planning

See the link below for a full view version of the Gantt chart:
https://app.smartsheet.com/sheets/6Jg3J3qR54GcWGMH7wM8XJXJX6fmjJcfV5fxxfr1



2 OVERVIEW

2.1 General Overview

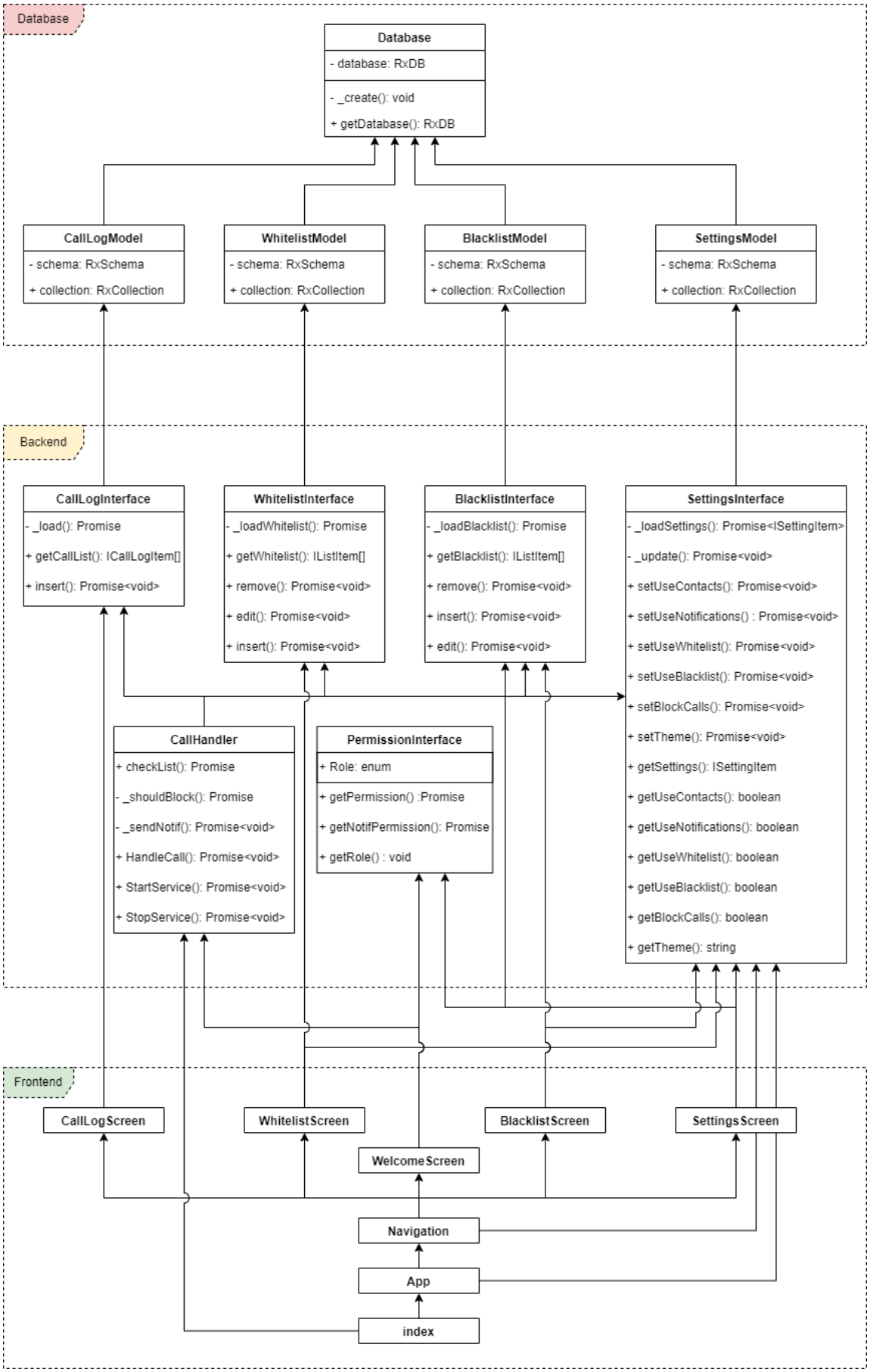
The GetOut app is designed to intercept and block unwanted calls based on a combination of user configuration and heuristic automation. This includes detecting incoming calls and checking the phone number against a whitelist and blacklist. The whitelist is a list of whole or partial phone numbers to allow through, and it contains the user's contacts plus anything manually added by the user. The blacklist is a list of whole or partial phone numbers to block, and it contains anything manually added by the user. A priority system will determine whether to use the blacklist or whitelist for a given number.

Calls will be intercepted in the background, and a silent notification will be sent to the user if a call is blocked. All calls will be displayed in a call log, which will have options for the user to block or unblock callers manually. A settings menu will allow users to view and edit the whitelist and blacklist directly, enable or disable features, and change the app's theme.

2.2 Design Strategy

The system was designed using a bottom-up approach, and it consists of 3 layers: database, backend, and frontend. The database layer connects to the database and performs operations on it. The backend layer contains convenient methods to interact with the database, adding another layer of abstraction on top of the database layer. It also handles background tasks, such as blocking calls and requesting permissions. The frontend layer provides the user with an interactive UI by using the backend layer's methods to display and edit data. The separation of the app into 3 layers gives us the freedom to change higher layers without significantly affecting the lower layers. Together, these 3 layers provide users with a functional and easy-to-use app.

2.3 System Design



3 PROJECT HISTORY

3.1 Project Selection and Issues

Most of the team that developed GetOut! was formed initially from John, David, Nick and Ryan. We had all worked together successfully on a project in Parallel and Distributed Computing and felt that, since we were able to work together successfully in the prior class, we would be as successful a group for this project as well.

We weren't looking for a specific type of project to work on, aside from something that would have real world applications and experience. During project selection, we decided against working on the robotic options, research projects regarding blockchain, and none of the individual app projects initially drew our interest. Quantum computing, while fascinating, did not translate to something immediately useful beyond initial capability exploration. The remaining projects that interested us were the grocery list, Phone Zapper, and Microtransactions in Gaming, as provided by Zebedee.

Our primary choice was to take on the project being offered by Zebedee, and so we put in our bid and began researching both the company and what they were offering. Thanks to our interest in this project, we were able to recruit another member for the team, Colin Allen, who has ended up being an invaluable addition to this project. We were set up, and ready to start working, and even had a very basic Unity application in the works.

Two weeks our initial discovery and research period, however, the group was informed that the Kennesaw State University legal department had discontinued the relationship with Zebedee, and that we would have to choose another project. Initially, John Hussey reached out to his company, and attempted to put together a project working on an add-on for Minecraft, but we were advised that said project would likely also be turned down by KSU's legal department. This left us with limited options, as the grocery list app now had several teams signed on to work on a version of it. So, we ended up working on the Phone Zapper.

3.2 Phone Zapper/GetOut! Research and Design Issues

Initially, the Zapper application was to be designed so that, once a spam call was detected and answered, a specific tone would be sent to the spamming software making the call. This tone would indicate that the number reached was a fax machine and said number would be removed from the number list that the spammer was utilizing. Ideally, this would prevent future calls from both that spammer, and any other spammer who was sold the updated list from that point forward.

We were able to find the tone in question but ran into a couple large issues when considering this approach. The two primary issues were:

- Phone spammer apps had long since added the ability to ignore the tone into their applications
- There was no way to send a command that would remove information from the back-end list without potentially violating the law through methods utilized to access the database of the call spammer application.
- A specific solution for a specific call spammer application could be developed, but that would require finding an active, current copy of the application in question, and it would also be too limited in scope to provide serious benefit.

As an alternative to this option, we intended to instead design the GetOut! application to send out various tones and messages indicating that the number in question was disconnected. The messages were going to be carrier specific. However, during research, we found out that to enact this functionality we would need to implement and maintain our own VOIP server to route phone calls through. We did not have the time or resources to build a VOIP server, so that stopped our attempts in this direction.

In the end, due to time and resource constraints, we decided to write GetOut! to be more of a standard multi-platform call blocking application, with the eventual potential to block and filter spam texts as well.

3.3 Development Choices and Issues

The initial decisions on tools were primarily driven by current utility in the job market. To that end, we decided to use React Native for the overall development of GetOut. RxDB had the built-in functionality to update the database automatically with every application update. Figma is an industry standard for front end development and is intuitive while being powerful. XCode is required to code anything for iOS, and Android Studio was a requirement to enable React Native the ability to interface with the Android APIs. The developers have added their own comments below.

Colin Allen:

"I'd say the main difficulty I struggled with (and John as well initially) was getting the hang of React's asynchronous execution in the backend when trying to write code that could be easily implemented and executed by our frontend devs. We ended up learning about functions like useEffect() and useState() to handle the results of our backend functions on the frontend, as well as knowing where to use keywords await and async appropriately in our backend database functions. I can definitely say that this has been the main hurdle to get over whenever I have wrestled with Javascript/web development, and I hope I can take some of what I learned through all the banging my head against the wall into my next projects."

Nick Nguyen:

"As of now, our implementation is complete for Android meaning our app, Get Out, can obtain the user's call log and history, thus allowing the "Recents" screen to manage missed and answered calls from the user. The "Recents" screen is crucial to our app as it grants the user to either block or unblock any given number. However, after completing the implementation process for Android, we ran into several problems with doing the same for iOS. We learned that acquiring the user's call log and history into our app is quite literally impossible to implement for iOS as it is not allowed for app developers. Unfortunately, this was not the only problem we had with the implementation process for iOS. There were abnormal instances with Metro, a JavaScript bundler for react native, where it was bundling incorrectly for iOS for no known reason. We also had trouble with changing the bundle identifier for iOS as well. After changing the bundle identifier for our app, it could no longer open as it could not find the new bundle identifier to open and build the app. Furthermore, implementing our app for iOS meant that you had to be on an apple device (macOS) as well. With this in mind, out of our group there happens to be only one person able to implement for iOS. Unfortunately, this limited the group's productivity as we had to schedule and implement with this one device. Overall, learning how to implement iOS has been a steep learning curve for everyone. We will certainly take these challenges and experiences with us to propel us further into our future."

Ryan Kim:

"Our first big step was choosing a database, and this proved to be quite difficult as there were a lot of choices to choose from. We were able to narrow our list to 2 to 3 different databases, but we chose to go with the one that had seemed to suit our needs the best which was RxDB. However, RxDB was a lot of work to get used to and to use as it had documentation, but the documentation was vague and did not explain enough about topics we had trouble with. Moving on to the front-end, initially working on the front end was quite easy as it only took getting use to react-native's syntax in implementing style choices like CSS. Even adding in buttons and different types of sliders was nothing serious as react-native's documentation was clear on these items. When it comes to the states of the buttons, however, it was an entirely different story. I spent days trying to research how states worked in react-native and every answer online was the same. The issue is that the only answer that kept popping up was not working with our code we had to come up with a solution on our own as there did not seem to be any answers online solving the problem we had at hand. Only after consulting with John were we able to get it fixed, and without him, I do not think I could have done it. I learned a lot through this, and hopefully it'll help me whenever similar issues appear before me."

John Hussey:

"Going into this project, we knew nothing about most of the tools, frameworks, and languages we ended up using. After finishing this project, I can say that RxDB had many challenges, but dealing with React Native's useState and useEffect hooks to get data to display and update correctly was the most difficult part of development. Working with Nick on iOS call blocking was also challenging, because of the limitations of the iOS platform. Outside of development, project management had its own challenges, particularly delegating tasks among members, as I had to learn what each teammate was skilled at. However, since I took an aggressive approach to scheduling and began successfully delegating tasks, we were able to complete all of phase 1 requirements ahead of schedule. We have all learned a lot and I feel confident that my team could build anything!"

4 DEVELOPMENT TOOLS AND STANDARDS

4.1 Development Tools

The following tools were utilized in the development of the application:

- Database: RXDB
- Framework: React Native
- UI Design: Figma
- iOS IDE: XCode
- Android IDE: Android Studio

4.2 Development Standards

Tick the appropriate box to indicate the standards being followed for this application:

Standard	√ indicates compliance
Database Design	✓
TypeScript	✓
JavaScript	✓
Accessibility	✓
Supported Android versions (7+)	✓
Supported iOS versions (10+)	✓

5 SYSTEM PROCESSES

5.1 Welcome Screen

Initial access to the application occurs through the WelcomeScreen on the front end. When the user clicks the button, all necessary roles and permissions are requested. Once the permissions are granted, the user is taken to the SettingsScreen to complete the setup process. The WelcomeScreen uses the SettingsInterface to request permissions and roles.

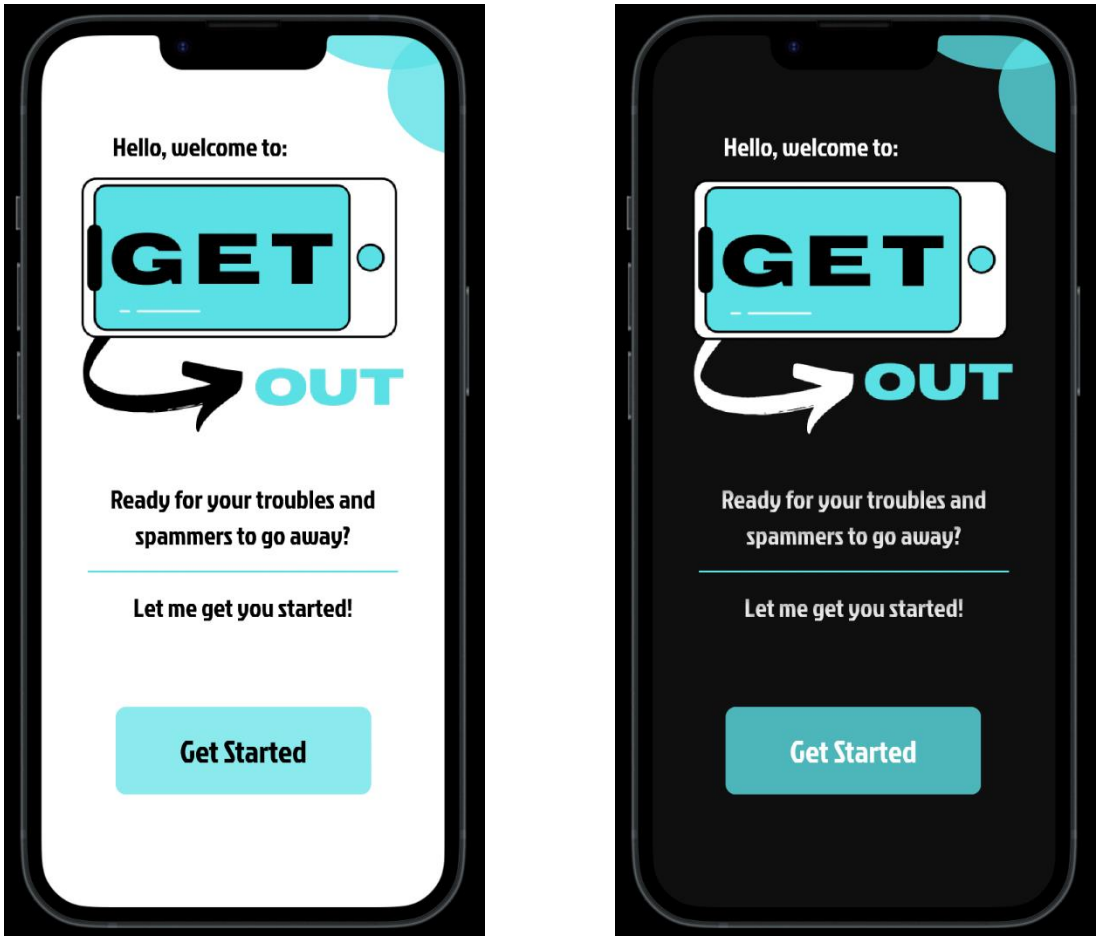


Image 1: initial welcome screen

5.2 Settings

The SettingsScreen allows the user to change how the app works and looks. The SettingsScreen uses the SettingsInterface to fetch the current settings via the getSettings() function and update the settings via individual setter functions. Some features will require permissions separate from the ones granted during the initial setup process. For these features, the SettingsScreen will use the SettingsInterface to request permissions when the user enables them.

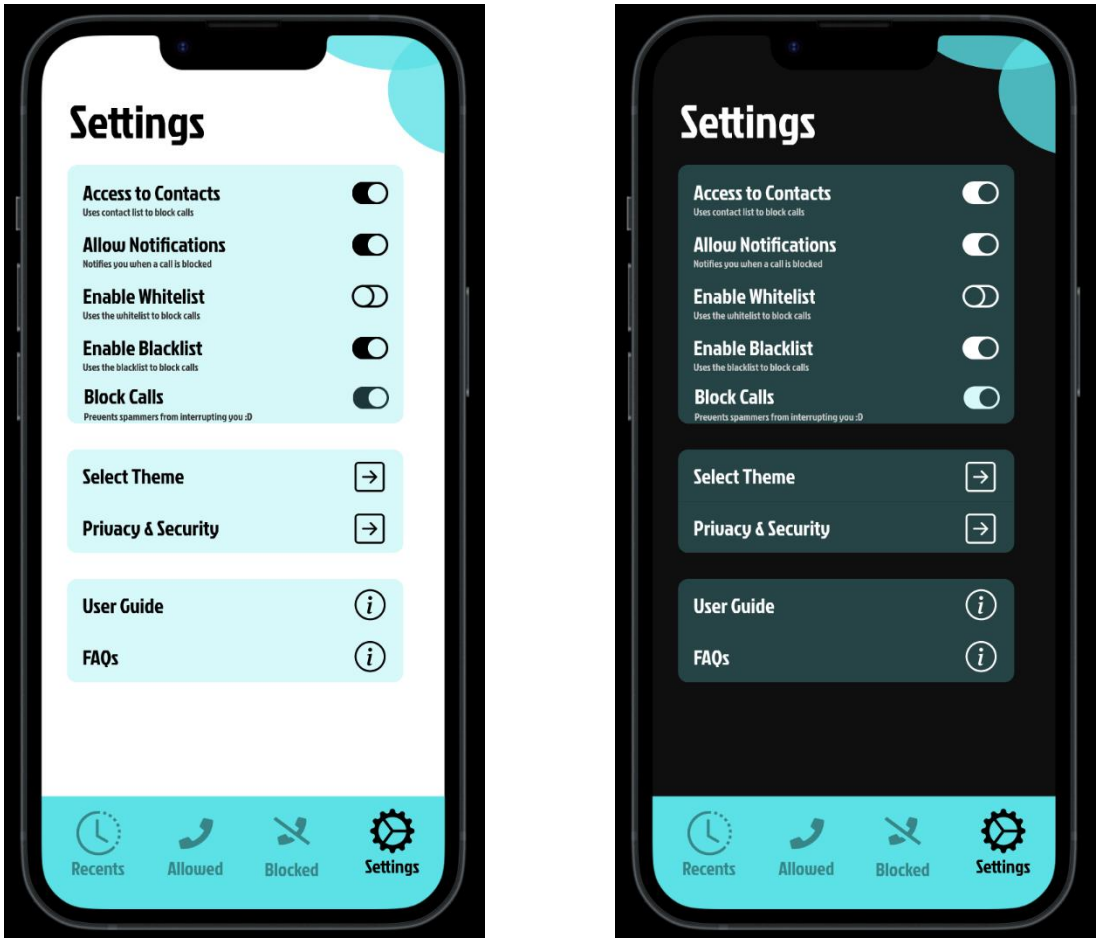


Image 2: settings screen

After initial setup (or settings adjustment), the user can access each screen from the bottom navigation bar, and the main screen will be the CallLogScreen. On the backend, the application will listen for incoming calls via the CallHandler. The onScreenCall() function handles incoming calls, and it will be implemented separately for iOS and Android. It checks the phone number of the incoming call against the contact list, whitelist, and blacklist, and determines whether to block the call based on the user's settings. For the call to be blocked, it must be added manually to the blacklist from the CallLogScreen or the BlacklistScreen.

5.3 Call Log

The CallLogScreen displays a list of the user's recent calls. It uses the CallLogInterface to conveniently communicate with the database. and from the backend by the CallHandler. The CallHandler gets the metadata of the incoming call from the CallLogInterface, checking against the whitelist and blacklist to determine whether to let the call through, and then the call's metadata is inserted into the database through the CallLogInterface's insert() function. The user can utilize the front-end interface to decide whether to add a number from a received call to either the whitelist or blacklist.

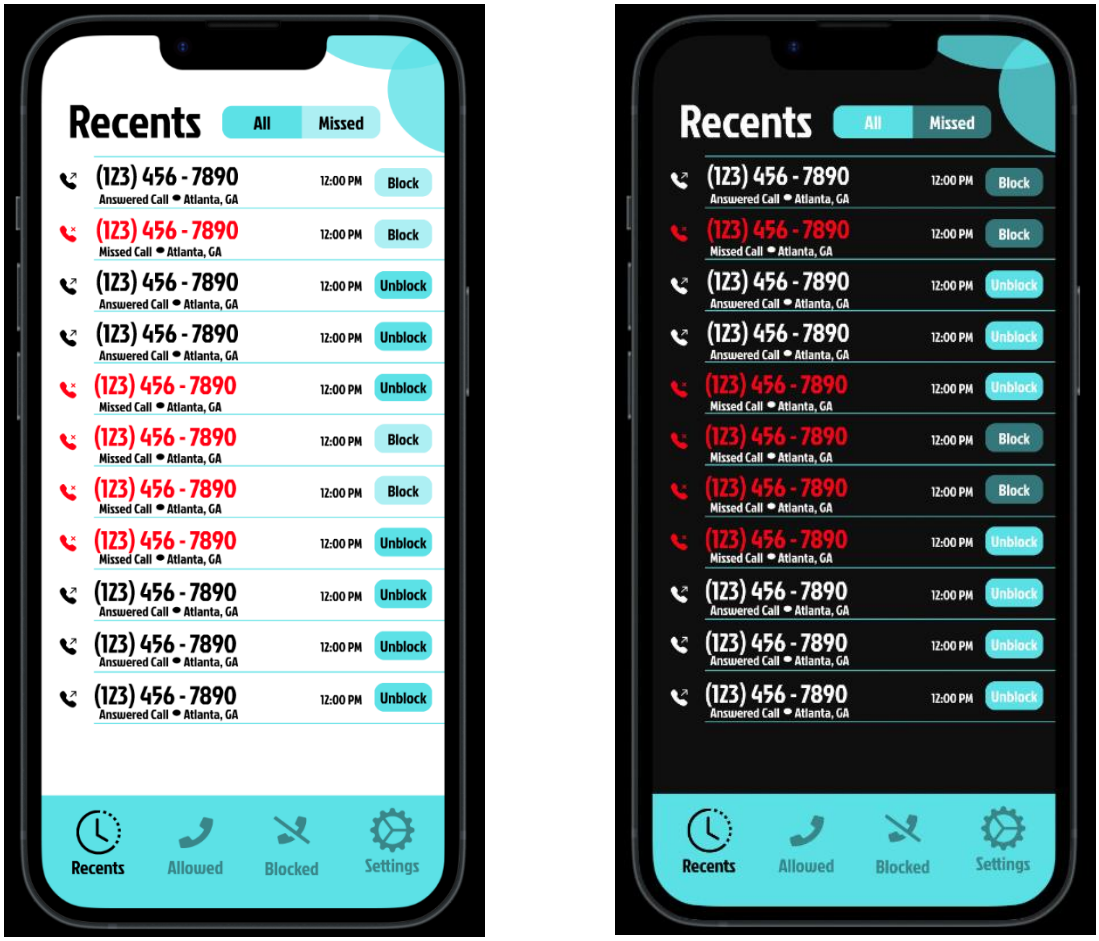


Image 3: call log/recent calls

5.4 Blacklist

The BlacklistInterface is accessed via the CallHandler on the back end, and by the user via the BlacklistScreen on the front end. The CallHandler checks incoming phone calls against the list and determines whether to block the call based on the inclusion of part (or all) of the number on the blacklist. The user can utilize the BlacklistScreen to both view and edit the blacklist. BlacklistInterface contains the getBlacklist() function that retrieves the current blacklist from the database, and also utilizes the insert() function to update the database in response to user input.

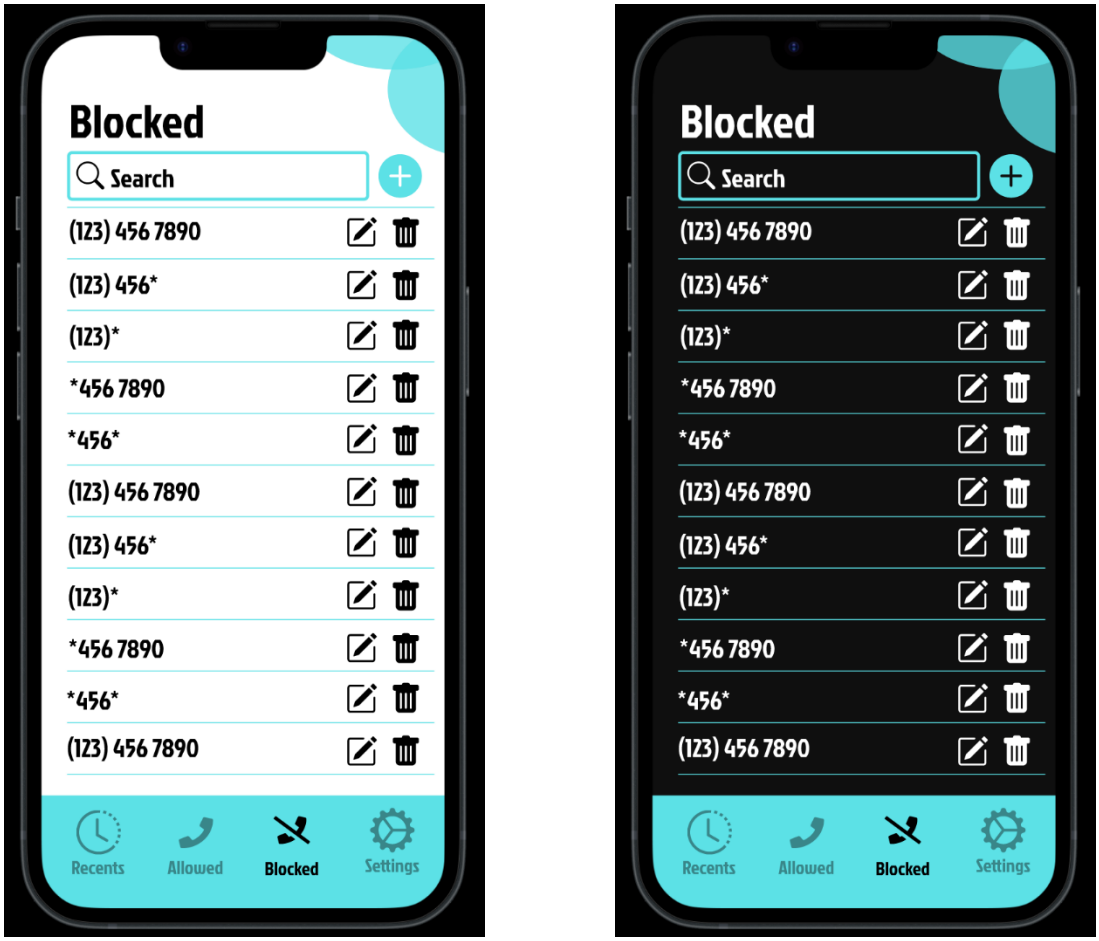


Image 4: blacklist screen

5.5 Whitelist

The WhitelistInterface is accessed via the CallHandler on the back end, and by the user via the WhitelistScreen on the front end. The CallHandler checks incoming phone calls against the list and determines whether to allow the call based on the inclusion of part (or all) of the number on the whitelist. The user can utilize the WhitelistScreen to both view and edit the whitelist. WhitelistInterface contains the getWhitelist() function that retrieves the current whitelist from the database, and also utilizes the insert() function to update the database in response to user input.

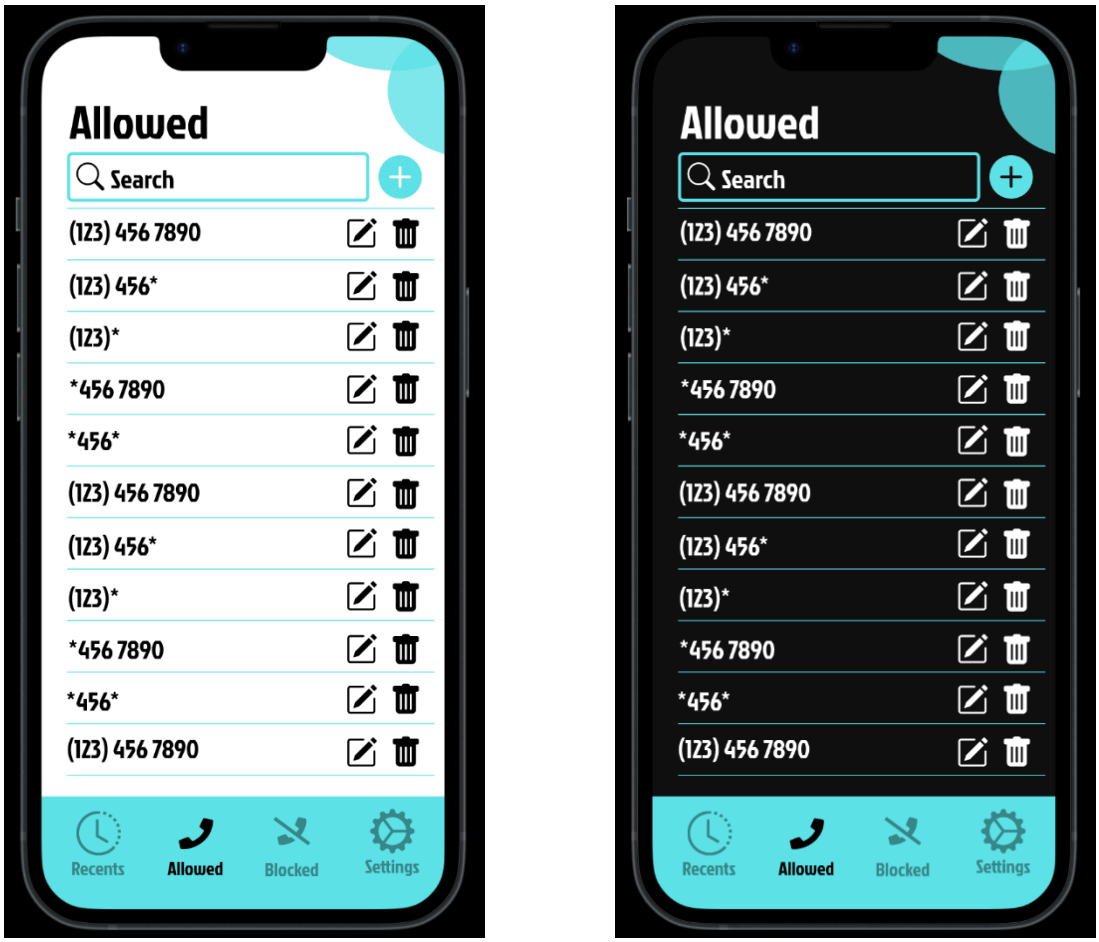


Image 5: whitelist screen

6 USER INTERFACE

6.1 Transactional Interface

On a fresh installation of the app, the user will be greeted with a welcome screen, followed by a quick setup process that will fetch permissions. The setup process will only be completed once per user. Whenever the user opens the app again, they will be taken to the call log screen.

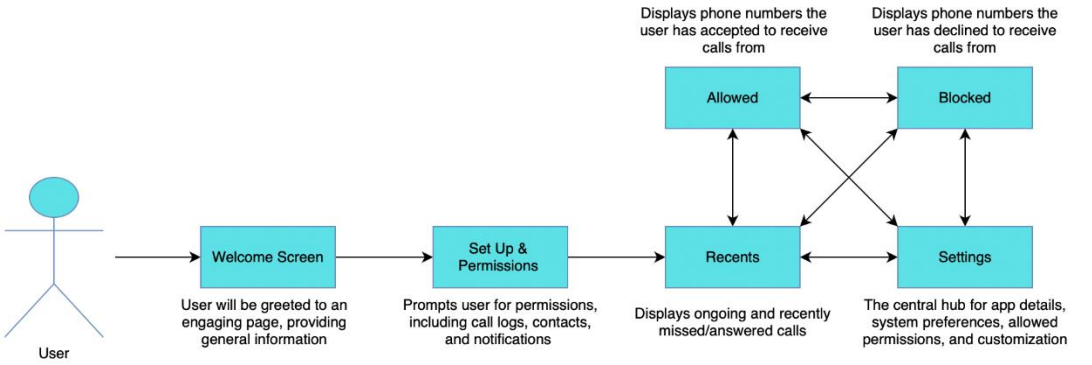


Image 6: Use case flow

6.2 Reporting Interface

Reporting will be done via the CallLogScreen on the front end. The screen displays the user's call history, and reports whether the call was blocked by GetOut or not. Additionally, a silent notification will be displayed for the user whenever a call is blocked. When clicked, the notification will take the user to the call log screen.

7 APPLICATION SECURITY

7.1 Authentication

Authentication is handled by the standard Android/iOS authentication procedures. There is no separate login currently for the app.

7.2 Authorisation

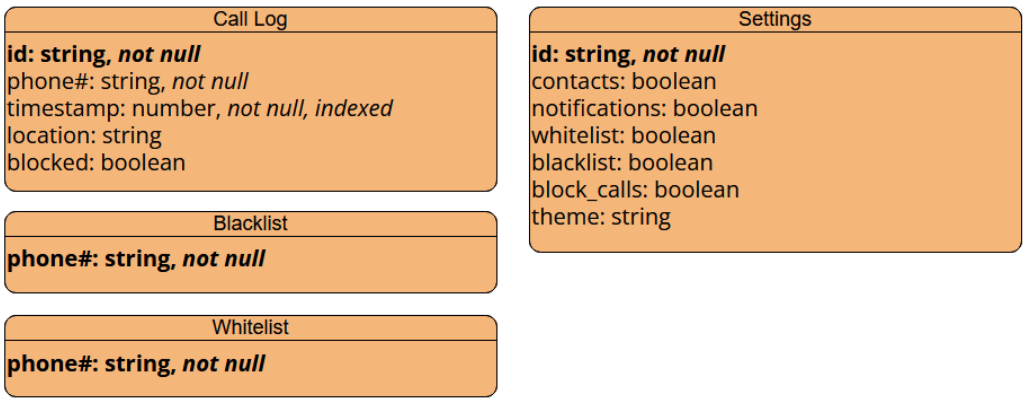
Authorisation to access various device functions is controlled by the back end; specifically, through the SettingsInterface, which will request permission from Android/iOS devices to access, contacts, notifications, and enable white and blacklists to control call handling.

7.3 Encryption

The user's contact list and call history will be stored in the database, which could expose sensitive information. For this reason, the sensitive information will be encrypted in the database by specifying it in the RxDB schema.

8 DATABASE DESIGN

8.1 Database ERD



8.2 Data Migration

Without a database migration system, the user would have to delete their entire database to update the app. GetOut will use the following database migration system, using the built-in functionality of RxDB. With every update after the initial distribution, we will update the version number of all altered schemas and carefully specified in the schema how the data should be migrated. This will prevent the user from losing their data with each app update.

9 TESTING PLAN

9.1 Testing Plan and Results

Requirements	Pass	Fail	Severity
Must get permissions from the user	✓		High
Must have a view for the call log	✓		Medium
Must have a view for the whitelist	✓		Low
Must have a view for the blacklist	✓		Low
Must have a view for the settings	✓		Low
Must intercept incoming calls	✓		High
Must verify that the call is incoming	✓		High
Must get the caller's phone number	✓		High
Must check if the number is in contacts	✓		Low
Must check if the number is blacklisted	✓		Low
Must check if the number is whitelisted	✓		Low
Must determine if call should be blocked	✓		Low
Must add details to call log	✓		Low
Must return response within 5 seconds	✓		High

10 RESULTS AND CONCLUSION

10.1 Results

Our team had little to no prior experience with building mobile apps or even using JavaScript, so we have spent much of our time learning JavaScript, TypeScript, React, React Native, Java, and Swift. For our project, we found that React Native has increased development time, because the call blocking functionality must be implemented separately for iOS and Android anyway. React Native only provides a cross-platform user interface and database. Using the React Native bridge to ensure the native call blocking functionality has the data it needs has been a real challenge, but in the end, we were able to successfully block calls in both the Android and iOS versions of the application.

10.2 Conclusion

This project has taught us many useful and marketable skills about mobile app development and React Native. Utilization of Smartsheet and other popular office collaboration tools have also been instrumental with regards to preparing the group for real world development and software design experience. Overall, this collaboration has been an extremely successful and beneficial one, and I am sure that we would all work together again on another project, given the opportunity.

11 APPENDIX

11.1 Software Requirements Specification

1. Introduction

Purpose

- The purpose of this document is to provide stakeholders and developers with an easy-to-follow roadmap of design requirements.

Intended Audience

- This document is for the developers, project manager, and other faculty.

Intended Use

- This SRS is intended to provide a development roadmap, as well as level set expectations for the project manager and faculty.

Scope

- The scope of the GetOut app is to intercept and block unwanted calls based on a combination of user configuration and heuristic automation. This includes detecting incoming calls and checking the phone number against a whitelist and blacklist. The whitelist is a list of whole or partial phone numbers to allow through, and it contains the user's contacts plus anything manually added by the user. The blacklist is a list of whole or partial phone numbers to block, and it contains anything manually added by the user. A priority system will determine whether to use the blacklist or whitelist for a given number.
- Calls will be intercepted in the background, and a silent notification will be sent to the user if a call is blocked. All calls will be displayed in a call log, which will have options for the user to block or unblock callers manually. A settings menu will allow users to view and edit the whitelist and blacklist directly, enable or disable features, and change the app's theme. Additionally, if time allows, we can add spoof detection and/or 3rd-party APIs to determine if a call should be blocked, and we can add machine learning to filter spam text messages.

User Needs

- Most users need a simple app that prevents unwanted callers from interrupting their day. For these users, the app needs to be easy to set up and use, which means the user interface needs to be simple and straightforward.
- Other users need more advanced configuration options to specify exactly which numbers will be blocked. For these users, advanced configuration options should be accessible. However, since most users only need a simple app, the advanced options must be separate from the core of the user interface, so that it does not disrupt the user experience.

2. System Features and Requirements

Functional Requirements

- Must be able to access incoming calls, compare them to white and blacklists, and block accordingly
- Must provide the user with the option to whitelist/blacklist incoming calls
- Must notify the user when calls are blocked
    - Must allow users the ability to access the settings menu to directly modify the black/whitelists as well as settings
    - Must allow first-time users the option to customize settings on initial load
    - Must have permission to access and intercept unknown incoming phone calls
    - Must have permission to access contacts and determine if a phone number or email address is in the user's contacts

3. External Interface Requirements

User Interface

- Buttons – The user must be able to interact with the application in some way via buttons
- Text boxes – User must be presented information via text
- Images – User must be provided visual stimuli through easy-to-understand images that are directly related to functionality

Software Interface

- On first time open:
    - The order the app must load in following first time access: Splash screen – first time setup – landing pad
    - First time setup: The app will need n screens regarding user instructions setting up access and functionality for the app, that contain links to the settings page referenced
    - Landing pad/Subsequent openings: The landing pad must display icons to access the blacklist, whitelist, call log, settings page (version, minor tweaks, assistance site), contact list
- Settings – Must contain n icons/menu items that allow users to adjust whitelist, blacklist, contact list, partial number blocking and screening, and any additional settings decided on
- Blacklist – Must contain a method to add/remove/modify numbers on the blacklist
- Whitelist – Must contain a method to add/remove/modify numbers on the whitelist, also add them to the contacts list
- Call log – Must display the calls received/blocked/made; must also provide the option to add a number to the contact list/whitelist/blacklist.

Back-End Operation

- The backend must connect to a database
- The backend must execute SQL statements on the database
- The backend must perform all logic necessary to handle incoming phone calls
- The backend must expose functions that fill in dynamic elements of the front end

Hardware Interfaces

- Must be configurable to show notifications on phone screen while phone is locked
- Must be operable on mobile devices running at least Android 7+, or IOS 10+.
- Must be able to answer calls utilizing phone hardware buttons

4. Non-Functional Requirements

Performance Requirements

- Must be able to run in the background constantly without draining system resources
- Must be able to quickly respond to incoming phone calls/texts
- The decision to block calls must be made within 5 seconds of receipt of call
- The decision to block texts must be made within 10 seconds of receipt of texts (phase 2)
- Must completely load app within 15 seconds of open
- Must download and install in a reasonably short amount of time
- Must be able to answer calls utilizing phone hardware buttons
- Must not slow down phone operation when app is minimized but still running

Security Requirements

- Must encrypt whitelist/blacklist
- Must encrypt/protect personal info input and saved in the app
- Must securely access incoming phone calls
- Must securely access the contact list on the phone
- Must securely access SMS/text messages (phase 2)