Bookstand

A Major Qualifying Project submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

By:
Stephen Andrews
John Bieber
Ted Bieber

Advisors:
Professor Michael Ciaraldi
Professor Joshua Cuneo
Date: 03/11/2019

Report submitted to
Professor Michael Ciaraldi
Worcester Polytechnic Institute

Abstract

The goal of this project was to create a platform that made it easier for students to get books for their classes, and earn back some of the value from books that they no longer use. After releasing Bookstand onto the iOS App Store and Google Play Store, a number of WPI students created accounts and listed books on the app. Additionally, some of the app's users submitted survey feedback, where the overall distribution was positive. The feedback also contained comments on UI functionality and bugs, both platform specific and platform independent. The team took this feedback into account by fixing bugs and improving the app's UI. Due to the number of users and positive feedback, the team considers the project to be successful.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Motivation

Currently, there are a limited number of ways for students to get textbooks for their classes. Some classes rely on digital textbooks which are typically cheaper and more convenient to purchase than print textbooks, but others may require materials only available through print or sold by the campus bookstore. Over a four year undergraduate degree, the cost of textbooks can be quite expensive. According to studies, students spend an average of $1,298 on textbooks per year.[1] With rising tuition costs, these textbook prices make it increasingly difficult for students to secure their education. Figure 1.1 shows the rising cost of textbooks compared to recreational books over the past 15 years.

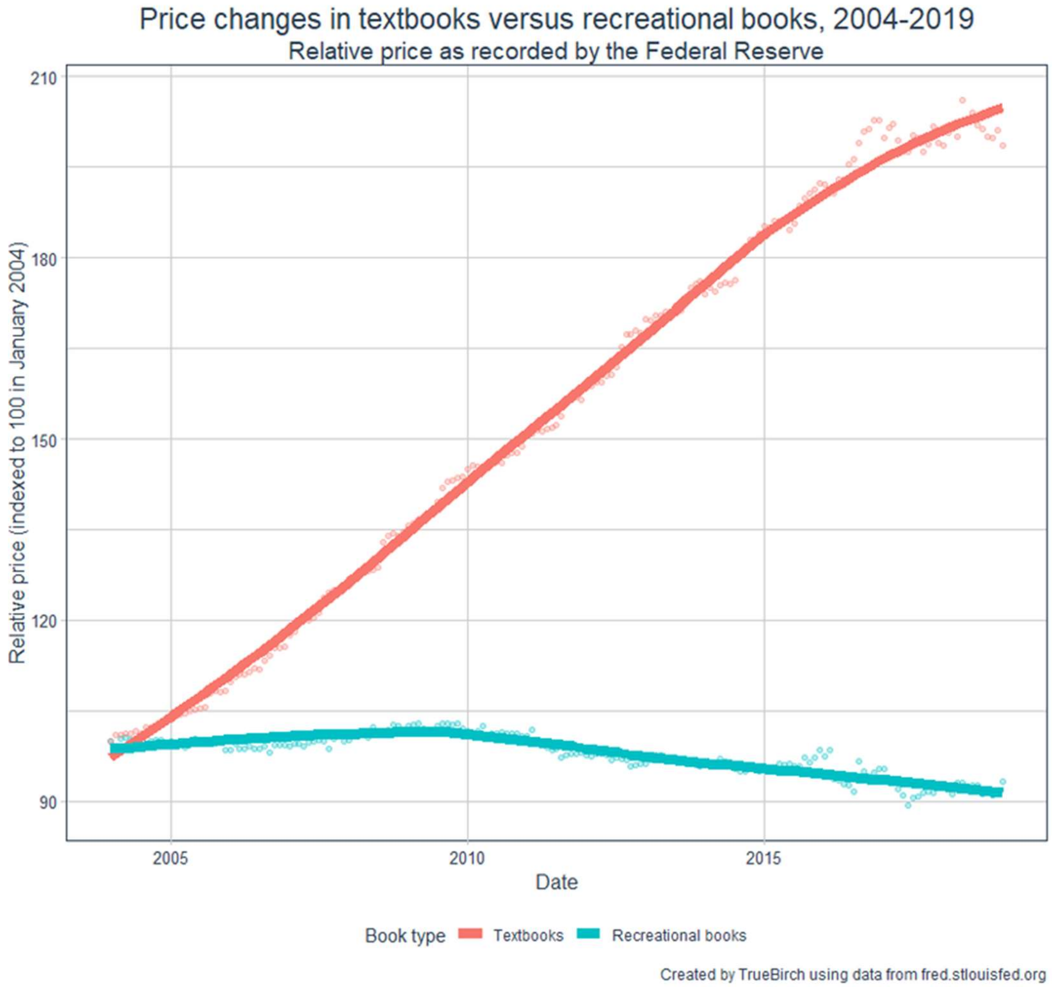


Figure 1.1: The rising cost of textbooks compared to recreational books over the past 15 years

1.2 Project Description

Many students enroll in college immediately after high school and often seek out loans to cover their tuition, which can range from an average of $9,970 for in-state residents at public universities to an average of $34,740 at private non-profit colleges.[2] Furthermore, students are spending an exorbitant amount of money on textbooks they may only use for a single term or semester. As a result, students are left with a collection of expensive textbooks that will provide them with no additional value.

Bookstand aims to address this multifaceted problem by providing a platform, specifically for WPI students, to buy, sell, or trade textbooks with one another. These transactions are facilitated through student-created listings. These listings are visible to all other students using the app. The users of the app can track their transaction history to see past trades and the earnings they have made from selling their books.

The intended result is that students can help themselves and others by selling their old textbooks to other students taking these classes. This should both cut down on the cost of the textbook for the new student, and offset the cost to the original buyer of the textbook.

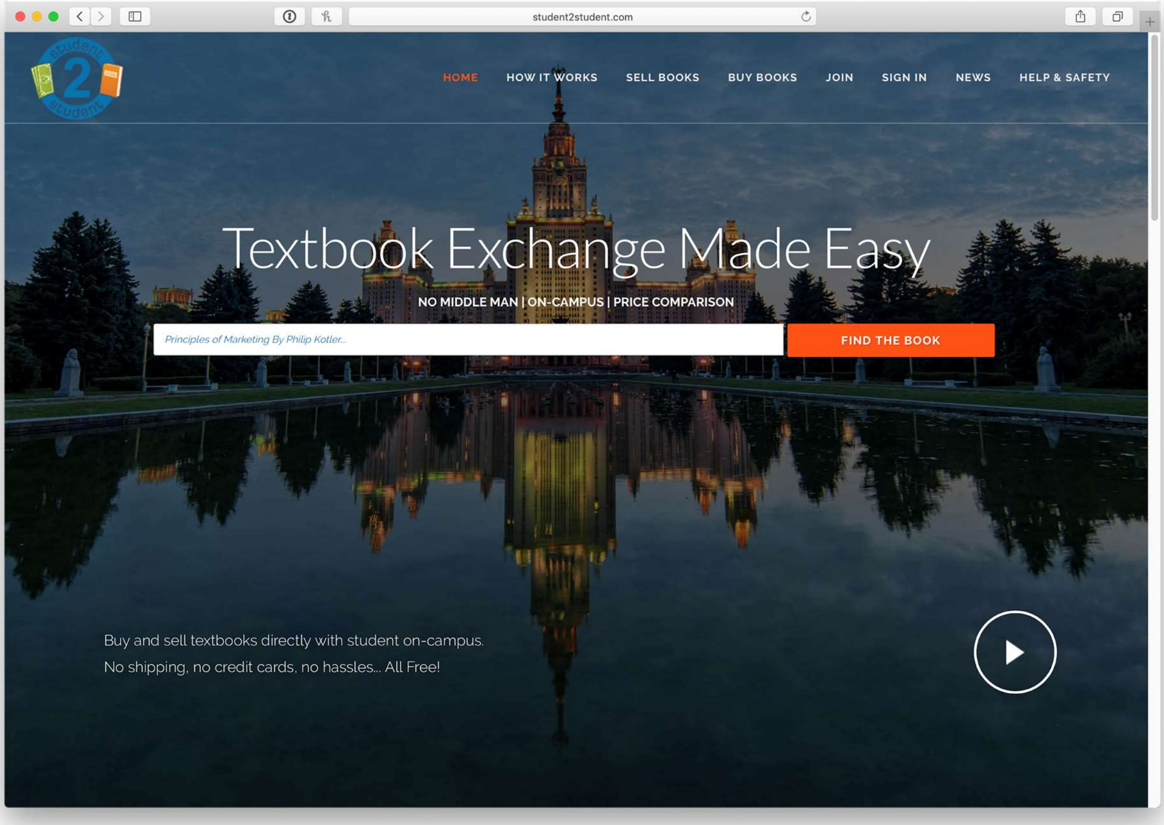
1.3 Prior Work

Prior research shows that a handful of applications have been attempted over the course of several years, but it appears that none have been successful. The reasons for failure appear to be varied, including web-only design, poor user interface, and lack of a critical mass of users and available titles.

1.3.1 Student2Student

Student2Student positions itself as a textbook exchange program. According to its site, the company was founded in 2007.[3] However, despite the supposed founding date, the site shows no sign of traffic. Searching for common textbooks returned zero results on multiple university campuses. In addition to the textbook exchange, Student2Student aggregates postings from various online retailers to provide students with lower prices.

Another indication of popularity can be determined by viewing the social media accounts belonging to Student2Student. Their Twitter account was created in 2016 and contains only two tweets with a total of four followers. The Facebook page shows more promise with posts dating back to 2009 and a total of 48 likes. Due to the limited postings on their site and lack of social media presence, we can only determine the project has been a failure thus far.



Textbook Exchange Made Easy

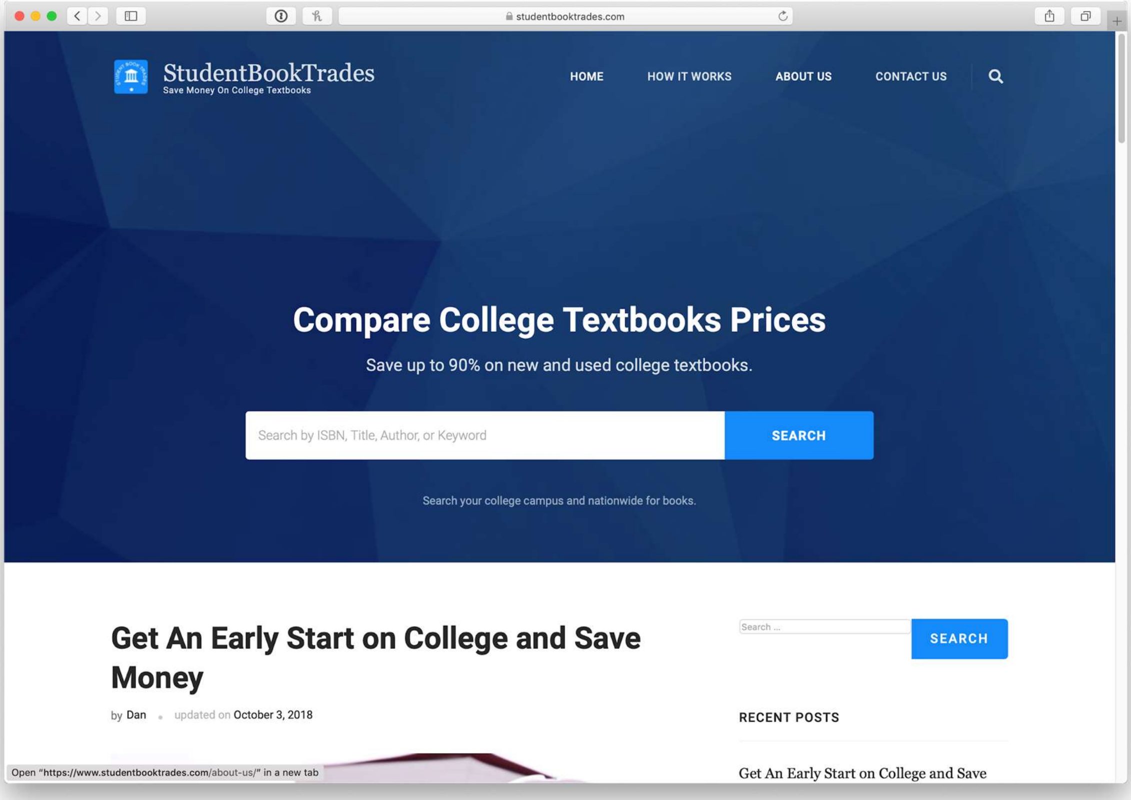
NO MIDDLE MAN | ON-CAMPUS | PRICE COMPARISON

Buy and sell textbooks directly with student on-campus.
No shipping, no credit cards, no hassles... All Free!

Figure 1.2: Student2Student homepage

1.3.2 StudentBookTrades

StudentBookTrades is yet another textbook exchange program lacking any indication of traffic. The site lists a 2018 copyright date on its website.[4] When searching for a book, it seems to only provide Amazon links of both new and used copies for purchase. Additionally, the search results are missing the ability to filter by university. StudentBookTrades advertises that users can buy via PayPal, but nothing on their site indicates that it is currently being used by anyone.


studentbooktrades.com
StudentBookTrades
Save Money On College Textbooks
HOME
HOW IT WORKS
ABOUT US
CONTACT US
Search by ISBN, Title, Author, or Keyword
SEARCH
Search ...
SEARCH
RECENT POSTS
Open "https://www.studentbooktrades.com/about-us/" in a new tab

Compare College Textbooks Prices

Save up to 90% on new and used college textbooks.

Search your college campus and nationwide for books.

Get An Early Start on College and Save Money

by Dan • updated on October 3, 2018

Get An Early Start on College and Save

Figure 1.3: StudentBookTrades homepage

1.3.3 BookU

BookU markets itself as a mobile application for students to exchange study materials with each other. It is a for-profit company founded in 2016 at the University of Tampa by Tom Shields.[5] Visiting their domain now shows an empty page with the option to purchase the domain. BookU is the only prior art that offered a mobile version of their product. However, it appears the app has since been removed from the app store. As of early 2019, the site appears to now represent a booking agency.

Chapter 2: Methodology

This chapter introduces the methodology behind the Bookstand concept. One of the main goals of the project was to create something the WPI community could benefit from with the hope of ultimately expanding to other universities. We hope that someone may be able to benefit from the functionality of our app, or choose to contribute to the remaining milestones documented at the end of this paper.

2.1 Research and Requirements

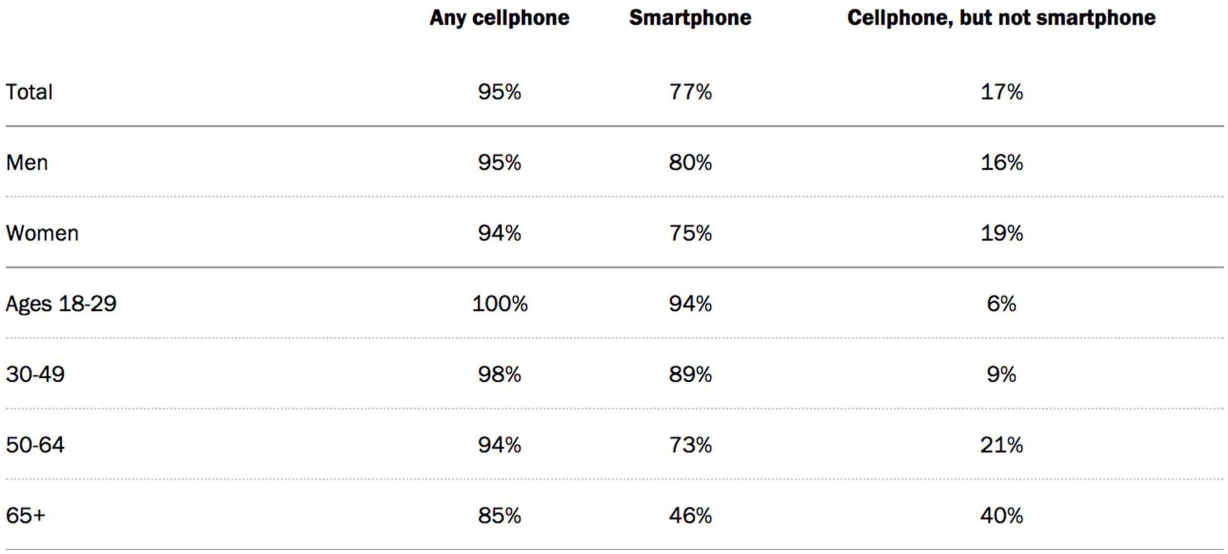
The team began by identifying some of the positives and negatives of college living. As members of the same community, all students share many of the same advantages and disadvantages of attending a university. Some of these advantages include the opportunities presented by the university such as free transportation, discounts on food, or a state of the art athletic facility. For these reasons, it becomes difficult to offer an improvement in these areas. For many students there is no cost effective method of purchasing a textbook that they may only use one time. We sought to improve this process.

There are many options for purchasing textbooks, none of which are financially compelling. With Bookstand, we wanted to offer low textbook prices with an even greater convenience than renting from Amazon. In order to accomplish this, we would need our own platform to foster a free market.

2.2. Why Mobile?

When planning this project, it was important to determine which platform would best suit this type of market place. Traditionally, these sorts of applications are web based. However, due to convenience and features such as image uploads, it was determined a mobile app would provide users with the best experience.

There are a few major reasons why Bookstand was written as an app. The first reason is that as of 2017, 94% of people in the United States between the ages of 18 to 29 reported owning a smartphone as shown in Figure 2.1.[6] Smartphones were also chosen because of their technical abilities; they provide a built-in camera that can be used to scan a book's barcode or take a picture as proof of condition or ownership. Lastly, smartphones are a good medium to collect data on, since most people do not share smartphones with others, whereas a computer might be shared within a household.



	Any cellphone	Smartphone	Cellphone, but not smartphone
Total	95%	77%	17%
Men	95%	80%	16%
Women	94%	75%	19%
Ages 18-29	100%	94%	6%
30-49	98%	89%	9%
50-64	94%	73%	21%
65+	85%	46%	40%

Figure 2.1: Percentage of U.S. adults who own either a cellphone or smartphone

2.3 Development Options

An important consideration when developing mobile applications is the inherent difficulty of supporting a wide variety of devices. There are many different types and versions of smartphones; and as it currently stands, the mobile device ecosystem is comprised mostly of iOS and Android devices. iOS holds the majority share in the United States of America at roughly 60% (see figure #2.2), but to develop solely for iOS would be to discard 40% of an app's potential user base. [7] As of today, there are two options for developers seeking to create an app for both Android and iOS; these options are discussed below.

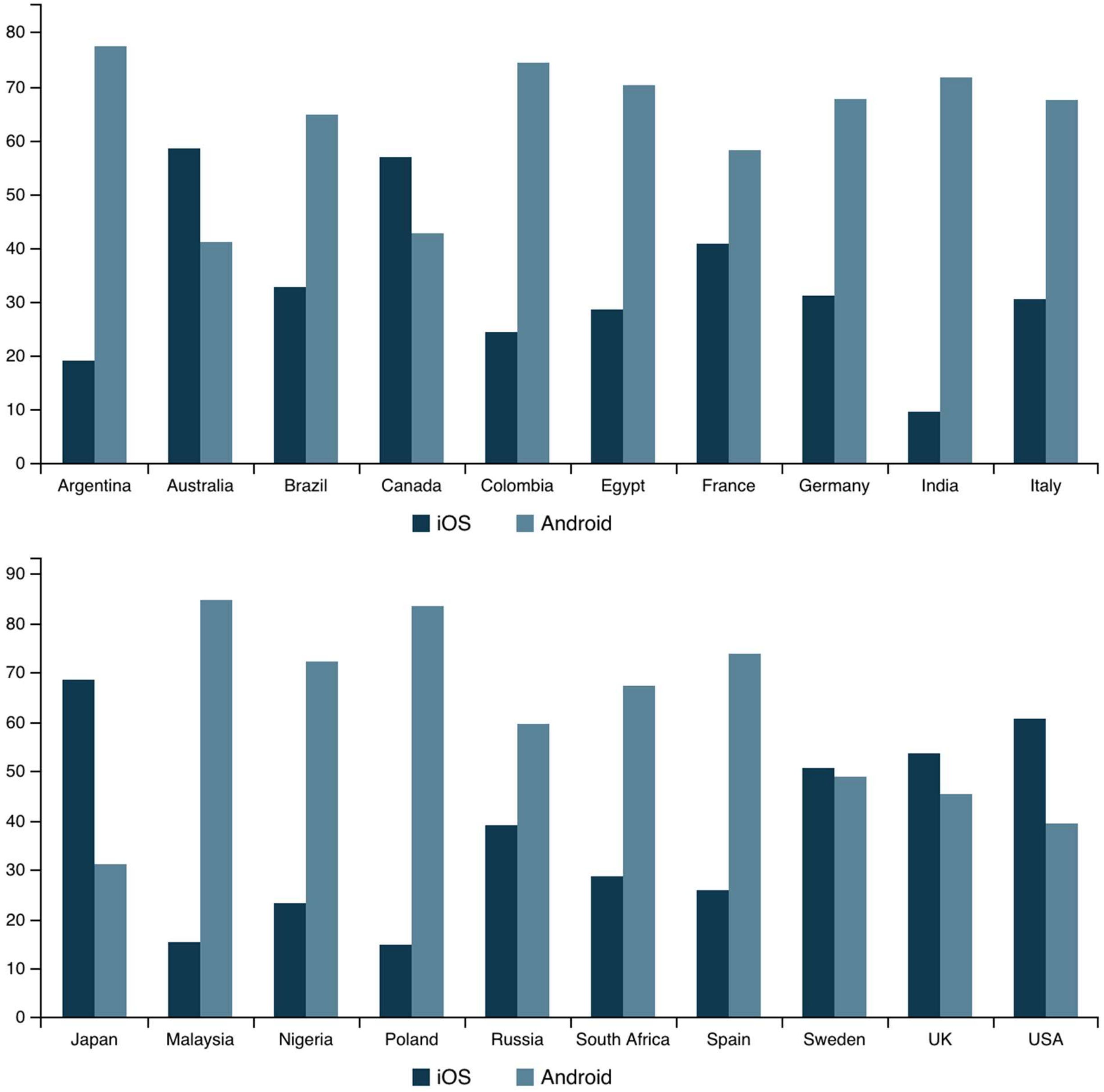


Figure 2.2: iOS vs Android market share by region

2.3.1 Native Applications

Native applications are apps that are written exclusively for iOS or Android. In order to support both operating systems, a developer must maintain a code base for each one. This carries with it the difficulty of having to be familiar with the development process of each operating system. iOS applications are written in Objective-C or Swift using Xcode, whereas Android applications are written in either Kotlin or Java using Android Studio. To develop an app for both platforms, developers would potentially have to learn two programming languages and familiarize themselves with two IDEs. Additionally, there could be platform-specific constraints or bugs that may require special handling on one app, but not the other. For developers who want to create an app that runs on both iOS and Android without doing double the work, there exist hybrid application frameworks.

2.3.2 Hybrid Applications

Hybrid application development has not been as popular in the past because it required a JavaScript wrapper in order to natively interact with devices. The introduction of frameworks such as React Native made it possible to develop an app with a single code base that compiles directly into native code, which performs comparably to a natively developed app.[8] React Native leverages techniques used in web-based layouts and allows developers to program in JavaScript. This enables a developer with prior web experience to leverage his or her skills into designing a mobile application, which saves a lot of time that would have otherwise been spent learning how to develop a native app. After looking into native and hybrid development options, we chose to use React Native to develop Bookstand. This has allowed us to support both platforms in a fraction of the time, without sacrificing any potential users or performance.

Chapter 3: Technology Stack

Complex applications are typically broken down into a client and server approach where the client represents a user, and the server consists of the data layer and any kind of processing required to serve data to the client. The following section discusses the implementation of the client, the server, and the libraries used in each.

3.1 Backend

In order to store information such as user accounts and listings, we created our own backend service consisting of a database and a RESTful API. The backend is a web server running on physical infrastructure. Since Bookstand is a mobile application, it was important to store and process as much information as possible on the backend to respect the limited resources available on the users' mobile devices.

3.1.1 Database Structure

When creating a database, the two main approaches that can be used are the relational schema and non-relational schema. Relational databases serve as the more traditional approach, where all data is represented in terms of tuples and grouped into relations.[9] On the other hand, non-relational databases, commonly referred to as NoSQL, are best suited for large scale projects that require a high level of performance.[10] In addition to scalability, NoSQL is also better equipped to handle unstructured data.

For the purposes of this project, PostgreSQL, an open source object-relational database management system (ORDBMS), was chosen since our data model is best suited for a relational approach. Additionally, PostgreSQL is one of the most mature ORDBMSs and in certain scenarios, has been shown to outperform NoSQL for large datasets when configured properly.[11]

Figure 3.1 below represents an entity-relationship diagram of the Bookstand database. Each table consists of an auto-incrementing primary key as well as "created" and "updated" columns for auditing purposes. Most of the table relationships are user-centric since nearly all interactions revolve around the user.

Users and Universities

The user table has a single foreign key reference to the university table. This relationship is essential for maintaining separate listings for each university participating in the app. While WPI is the only university currently supported, the data model has been designed in such a way that allows for additional universities to have independent listings from the others.

Books and Authors

Books and authors are captured in three separate tables: book, author, and author_identifier. The author_identifier is a junction table that creates a many-to-many relationship between the book and its authors. Since a book may have multiple authors, this is a necessary table to include for the purposes of normalization.

Listings

The listings table is the mechanism in which users can publicly display their books for sale to other users in the app. The table contains foreign key relationships to the user creating the listing and the book being sold, as well as additional user specified information such as the book condition, listing description, and the URL to uploaded photos. Initially, we had considered supporting two types of listings: sales and trades. Trades were intended for users who wanted to swap books with another user on the app. Due to time constraints, this feature was never fully implemented.

Messages

Since Bookstand acts as a marketplace for the WPI campus, it was important to include some means of communication. Initially, we had considered relying on SMS, but after further consideration, determined that an in-app solution would allow users to conceal their phone numbers, providing them with better privacy. The messages are comprised of two tables: thread and message. The thread table provides a communication channel between two users. The message table contains the actual message contents and a foreign key to the thread the message belongs to.

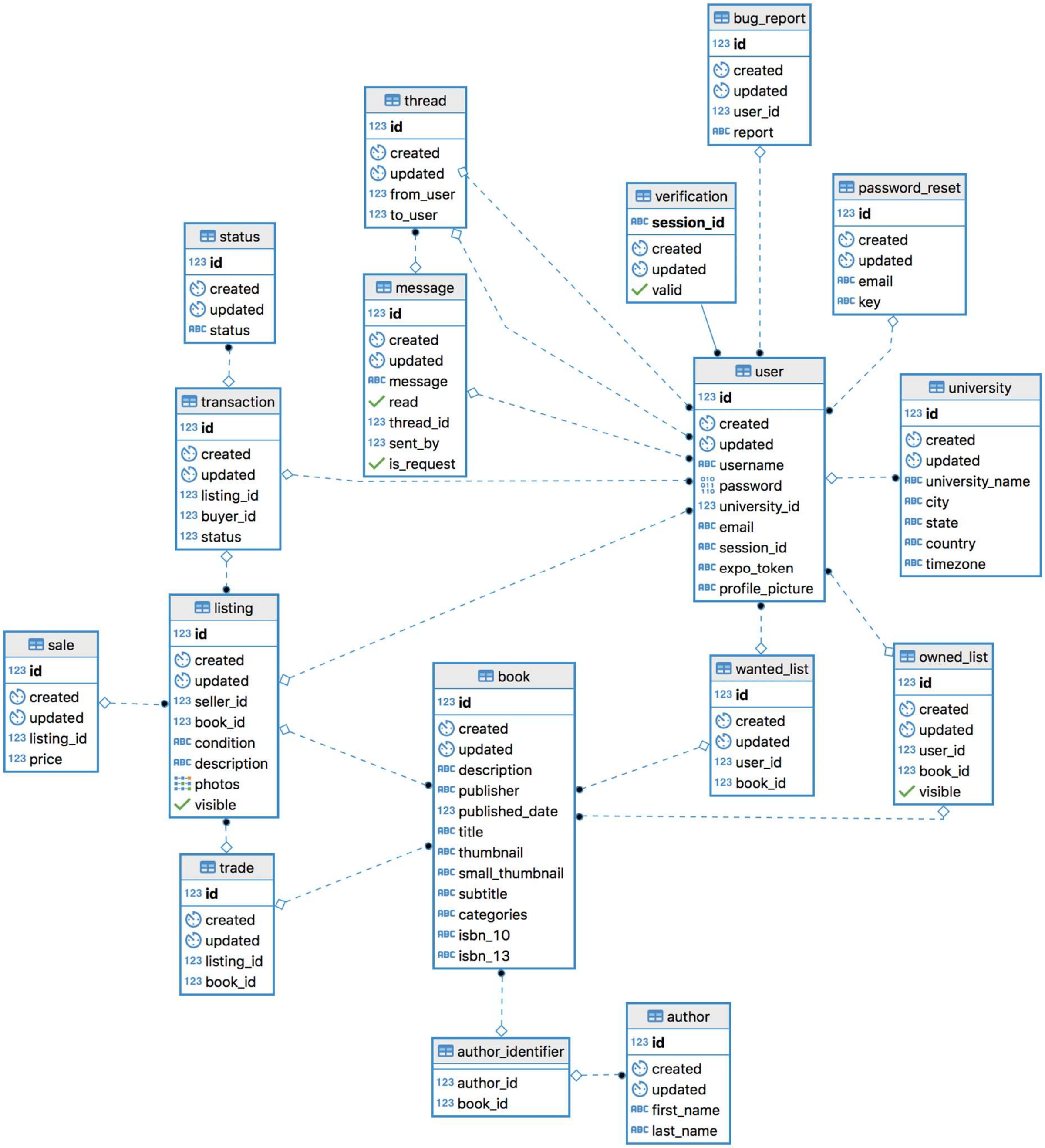


Figure 3.1: An entity-relationship diagram of the Bookstand database

3.1.2 API Structure

In order to access the backend, a RESTful API was designed and implemented on bookstand.io. One of the main considerations when developing the API was the security of user data. Having an exposed API means that malicious users could potentially steal or tamper with the data. These threats were accounted for in a couple of ways.

Since Bookstand uses SQL Alchemy, an object-relational mapping (ORM) for Python, SQL Injections are not a threat. This is because any special characters in the data are automatically quoted by the SQLEngine object when subclassing the model class.[12] In order for an SQL Injection to be successful, a developer would have to intentionally circumvent this functionality and rely on raw execute statements.[13]

Another point of risk is the potential of someone discovering the endpoints used by the application. To protect against this, the API only accepts requests from verified origins. Bookstand accomplishes this by assigning each user a unique session ID, and only users with an active session ID are allowed to interface with the API. On top of this, the session IDs are stored alongside a boolean indicating whether or not they are active. This allows for a particular session ID to be marked as inactive if it is linked to malicious activity. There is still the possibility that users will make a new account and use that session ID, but even then, identifying the session ID associated with their account is quite difficult.

The core purpose of the API is to separate the app logic from the data processing and reduce the need for local storage on a user's phone. Any network related functionality in the app is provided by the endpoints of the API such as managing user accounts, handling books, image uploads, user listings, messages, and transactions.

The endpoints for user accounts provide functionality to register, set personal information, reset passwords, upload a profile picture, submit bug reports, and see all of the listings that the user has.

The endpoints for books provide functionality to identify all of the books currently stored in Bookstand's database, fetch a book by its ISBN, and search for books by title. With a few exceptions, these actions use the Google Books API to find books. Books that are found via the Google Books API are then stored on Bookstand's backend. This is done to improve performance. If a user searches for a book, then it is likely that the book will be searched for again, so it is cached on Bookstand's backend, meaning that it does not need to interface with the Google Books API in the future.

The listing endpoints provide functionality to fetch all listings, fetch all listings for a specific ISBN, filter listings, post listings, and remove listings. Since these endpoints may return a lot of results, the returned data is paginated. By default, only 25 listings will be shown on the first page. From the app, subsequent pages can be accessed by scrolling towards the bottom of the browse screen. Another important design aspect of these endpoints is that removing a listing does not actually remove the record from the database. Instead, a "hidden" boolean is set to be true, thus hiding the listing from future search results. This can allow for additional features and the potential for data mining in the future.

The messaging endpoints provide users with the ability to send messages and initiate transactions with one another. Messages are organized into threads or conversations. These take place between two users: the buyer and the seller. Typically, the buyer is the user that initiates the message thread, when they press the buy button on the individual listing screen. The first message they send includes a trade request, which is automatically sent after they have confirmed that they want to buy the book that is listed. This trade request is limited in functionality, meaning that the buyer can only send or cancel a trade request, and the seller can only accept or deny it. Other useful features that did not make it into the minimum viable product would be the ability to negotiate a price.

The transaction endpoints provide functionality for users to buy books from each other. This solves a problem where users could potentially list a book, sell it, and then not do anything else. The issue with this is that the listing would still be visible. The solution is the transaction because it provides a convenient way for users to communicate that they want to buy a book and that they want to sell a book. It also provides the ability for a user to track their transaction history. Their transaction history can be viewed in the profile screen by pressing the transaction history button. This aggregates the total value of the books they have sold and summarizes the transactions in a simplistic UI.

3.1.3 Google Books

In order to support the searching of books within the app, a third party service was required. After exploring several APIs, we ultimately decided to use the Google Books API as it offers free access to non-commercial solutions and includes a sizable amount of metadata to display in the app such as cover art.

3.2 Frontend

The frontend of our app refers to the software that runs on the user's phone. In other words, it is the actual app. This section covers the technologies that we used to build our frontend, as well as why they were used.

3.2.1 React Native

React Native is a user interface library for JavaScript created by Facebook. It allows developers to build mobile apps using only JavaScript. There are a variety of features that make it much easier to develop, debug, and deploy apps.[14] Many large companies have seen success with a React Native code base. Some notable examples include Instagram, Bloomberg, Discord, and Tesla.[15]

React Native was first introduced in 2015 at a React conference. Initially, it had begun development in the summer of 2013 as an internal hackathon project at Facebook. However, since its launch, it has quickly gained the support of thousands of people. At the time of this writing, the React Native repository has a total of 1,894 contributors with over 70,000 stars making it one of the most popular repositories on GitHub.[16]

A large part of the success behind React Native is the ability to write one code base that supports multiple platforms. React Native users were able to develop at unparalleled speeds all while still supporting both iOS and Android platforms with native performance. This has allowed smaller teams to reach massive success. For example, Discord's iOS team is comprised of only two engineers and the app has reached millions of active users and a 4.8 star rating with over 240,000 total ratings.[17] Additionally, since React Native is based on React, many web engineers will find creating mobile apps to be much like that of a web application providing less of a barrier for new developers.

While React Native has seen a lot of success, there are still a handful of naysayers. Despite React Native avoiding the criticism that hybrid platforms like Electron have faced for poor performance, many critics argue that React Native users are at the mercy of Facebook to supply critical updates to the platform, thus preventing React Native applications from being a viable option for large scale applications. Recently, one of React Native's major supporters, Airbnb, published a blog post detailing why they were moving away from React Native. This sparked a lot of debate around the viability of React Native as a competitor in the mobile app development space. One of the main reasons they chose to abandon the platform stemmed from the large amount of native code they had and still have when they first began developing with React Native. Despite discontinuing React Native at Airbnb, the company ran an internal survey to gather information from their developers about their experience working with the platform. Of the engineers that worked with React native, 60% described the experience as amazing, 20% as slightly positive, 15% as slightly negative, and only 5% as strongly negative.[18]

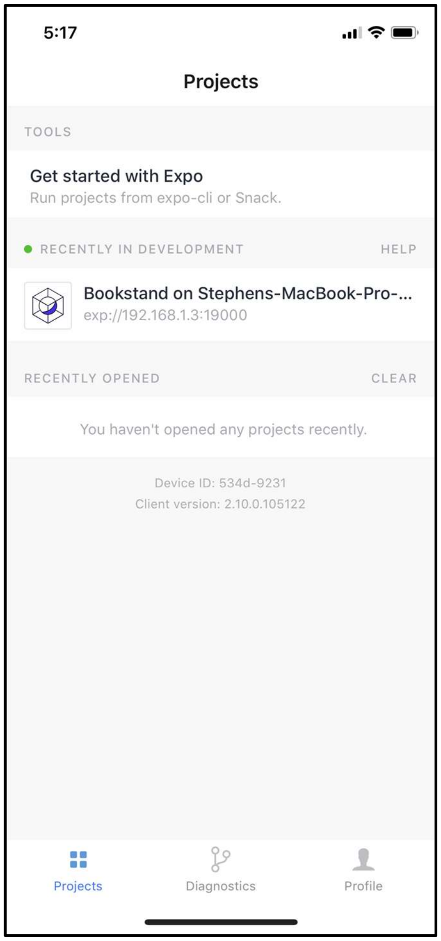
Overall, individual experience with React Native may vary but it was paramount to the success of this project and we would highly recommend it to anyone looking to develop a mobile application.

3.2.2 Expo

Expo is a free and open source toolchain built around React Native to help developers build iOS and Android apps using JavaScript and React. Developers can create Expo apps by using the Expo SDK. It provides a multitude of native modules absent from the React Native API such as access to the camera, contacts, and local storage. This allows the app to run in any native environment containing the Expo SDK.[19]

Expo also provides a command line interface (CLI) and accompanying app to improve the development workflow. The Expo CLI can be installed via NPM and provides commands for running a local Expo server which can then be connected to the app as shown in figure 3.2. While running an Expo server, any changes made to the codebase will be applied and hot reloaded in the Expo app. This allows for rapid prototyping all over a wireless connection. The CLI also provides commands for building iOS and Android variants of the application to submit to the App Store and Play Store.

Many of the modules used throughout the Bookstand app have come directly from Expo. The camera, barcode scanner, and push notifications are all modules provided by Expo that interface with the native APIs. The functionality provided by Expo offers developers a solid foundation to build on. However, throughout the development of the project, there were several issues related to the Expo SDK used in the app. Upgrading to newer versions of the SDK introduced bugs that could only be fixed by contributing to the repositories for the modules being used. This would require fixing the bug in the SDK, submitting a pull request, and having it approved. Several GitHub issues were opened on the project and received no attention from any of the Expo developers for several months over multiple SDK releases.



5:17

Projects

TOOLS

Get started with Expo
Run projects from expo-cli or Snack.

RECENTLY IN DEVELOPMENT HELP

Bookstand on Stephens-MacBook-Pro-...
exp://192.168.1.3:19000

RECENTLY OPENED CLEAR

You haven't opened any projects recently.

Device ID: 534d-9231
Client version: 2.10.0.105122

Projects Diagnostics Profile

Figure 3.2: Shows a local Expo server running in the Expo app

3.2.3 Third Party Packages

This section provides an overview of the libraries used by the frontend. Specifically, it covers the libraries that provide core functionality necessary in every app, as well as some additional libraries used to improve the appearance of the user interface.

React Navigation

React Navigation is a library that provides flexible navigation options for developing apps. It has built-in, easy to use functions and behavior that can be modified in the event that a developer needs more specific behavior for their app. It provides everything a developer needs for basic and advanced navigation within their app, and it even supports basic animations and gestures.

The core principle behind React Navigation is that each screen is part of a navigation object, and can only navigate to something within that same navigation object. This allows for logical design of screen navigation management. The basic navigation object is called a Stack Navigator, in reference to its underlying data structure, but there exist more concrete objects that render nicely. An example of this is the Bottom Tab Navigator, which will let users navigate to any screen that is part of this object by providing a footer with different customizable icons for each screen.

Redux

React Navigation is especially powerful when combined with effective state management. Unfortunately, a popular library for state management, Redux, was no longer guaranteed to be compatible with React Navigation (and vice versa) at the time of the team's development of the app. For state management, we ultimately decided to use React Native's built in functions, which are effective but can become convoluted when used at scale.

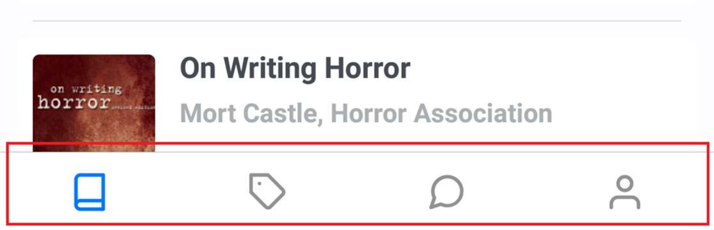
Redux is a library that provides mechanisms to manage a centralized immutable state within an app. In essence, if one screen on an app receives data, it is available anywhere within the app. The advantage of this over the built in features of React Native are immense. Firstly, the state in vanilla React Native is dependent on the screen. This state is also only partially reset when navigating to and from screens. This makes it difficult to predict the state when navigating to and from an array of screens. For a novice React Native developer, it will require a series of trials and errors to fully understand the fundamentals of maintaining state.

Redux approaches this problem as a finite state machine. The only way to get from one state to another is through a developer defined function that takes in a state and some action and returns a new state. This means that the only way to update the state is to store an updated version of it, making it easy to predict, stay organized, and compare present and past states.

Since Redux and React Navigation are no longer guaranteed to be compatible, it is highly recommended that developers either create their own centralized state management, or seek other libraries either for navigation or state management.[20]

React Native Vector Icons

React Native Vector Icons provides a large array of icons that can be used within React Native to indicate what a button or tab is for. Within Bookstand, these are used on the Bottom Tab Navigator to indicate the Browse, Create Listing, Messages, and Profile Screens. [21]



On Writing Horror

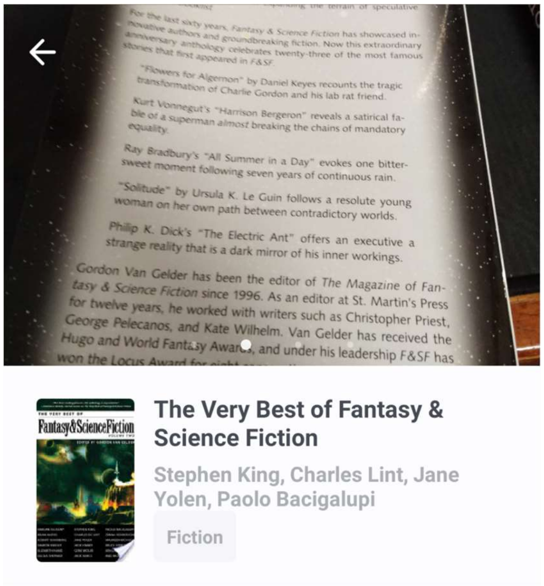
Mort Castle, Horror Association

Figure 3.3: React Native Vector Icons as used in Bookstand

Figure 3.3 above shows how Bookstand uses vector icons in the app. Users will see this screen after logging in. Vector icons are used in the bottom tab navigator, signifying four distinct sections of the app. On the far left, an icon of a book indicates the browse screen. The second icon from the left is a price tag, meant to indicate the create listings screen. The third is a message bubble, which is meant for the messages screen. Lastly the icon of a person is meant to indicate the profile screen.

React Native Snap Carousel

React Native Snap Carousel provides an image carousel that is capable of handling a variety of features. Within Bookstand, it is used in the detailed listing screen that is displayed when looking at a specific listing that a user has posted. The component is used to display images at the top of the screen, showing pictures that the user has taken of their book. [22]



For the last sixty years, *Fantasy & Science Fiction* has showcased innovative authors and groundbreaking fiction. Now this extraordinary anniversary anthology celebrates twenty-three of the most famous stories that first appeared in *F&SF*.

"Flowers for Algernon" by Daniel Keyes recounts the tragic transformation of Charlie Gordon and his lab rat friend.

Kurt Vonnegut's "Harrison Bergeron" reveals a satirical fable of a superman *almost* breaking the chains of mandatory equality.

Ray Bradbury's "All Summer in a Day" evokes one bittersweet moment following seven years of continuous rain.

"Solitude" by Ursula K. Le Guin follows a resolute young woman on her own path between contradictory worlds.

Philip K. Dick's "The Electric Ant" offers an executive a strange reality that is a dark mirror of his inner workings.

Gordon Van Gelder has been the editor of *The Magazine of Fantasy & Science Fiction* since 1996. As an editor at St. Martin's Press for twelve years, he worked with writers such as Christopher Priest, George Pelecanos, and Kate Wilhelm. Van Gelder has received the Hugo and World Fantasy Awards, and under his leadership F&SF has

The Very Best of Fantasy & Science Fiction

Stephen King, Charles Lint, Jane Yolen, Paolo Bacigalupi

Fiction

Figure 3.4: React Native Snap Carousel as used in Bookstand

Figure 3.4 above is an example of how Bookstand uses the snap carousel component. In the detailed listings screen, at the top are images that users have taken of their listed book. In the image above, the snap carousel shows the second image of the four available. The small dots at the bottom of the carousel display the number of images, corresponding the number of dots, as well as the currently selected image, which corresponds to the larger, brighter dot in the grouping.

React Native Keyboard Aware Scroll View

React Native Keyboard Aware Scroll View is a component that takes into account the additional space that the keyboard takes up when a user is entering information into a field. Since the keyboard comes up from the bottom of the screen, it pushes the rest of the screen upwards. Without this component, that portion of the screen would not be accessible until the keyboard has been dismissed. With this component, a user can swipe up or down to access the rest of the screen. This component is used within Bookstand wherever there is a text field that would be obscured by the keyboard.[23]

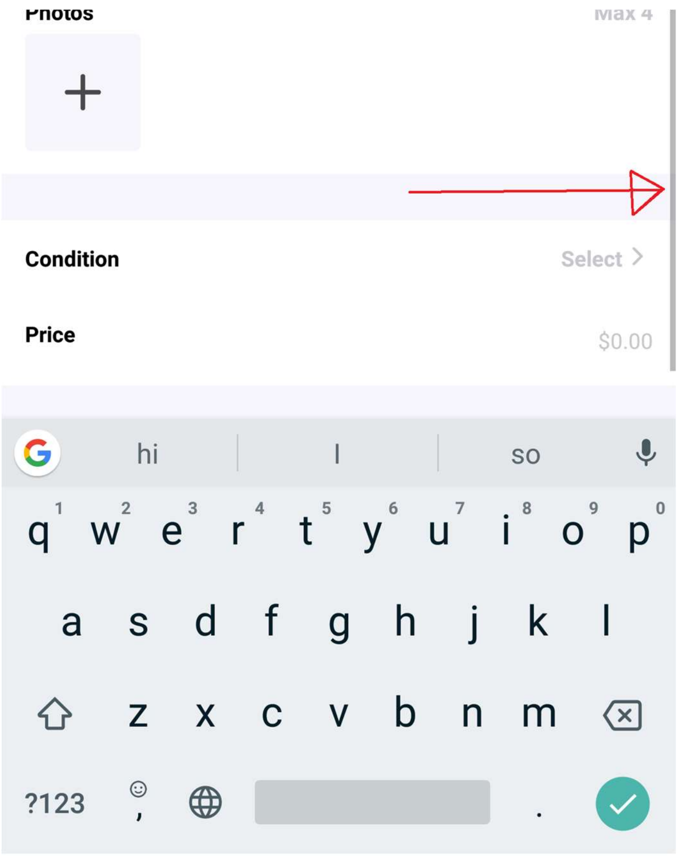


Figure 3.5: React Native Keyboard Aware Scroll View as used in Bookstand

Figure 3.5 above is an example of how the keyboard aware scroll view component is used in Bookstand. This screenshot was taken from the create listings screen after an input was selected and the keyboard was brought up. On the right there is a bar indicating how far into the page this view is from. This bar will only appear if the screen takes up more space than is provided. When a keyboard opens, that will almost always force this situation, so the bar will appear.

React Native Multi Slider

React Native Multi Slider is a component package that provides the ability to have two draggable nodes on a slider (vanilla React Native does not provide this). In Bookstand, this is used in the filtering of listings after selecting a textbook from the browse screen.

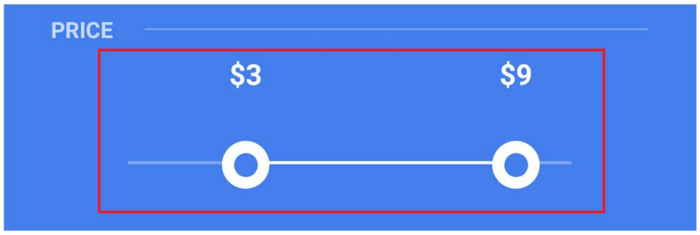


Figure 3.6: React Native Multi Slider as used in Bookstand

Figure 3.6 above is an example of where the multi slider component is used in Bookstand. It is used to set the minimum and maximum prices for textbooks when searching for a specific ISBN or textbook entry. This component has been modified to support labels above each of the sliders.

3.3 Deployment

Deployment is an important aspect to consider when developing mobile applications. Once an app has been published and has users, seamless deployments are paramount in providing a good user experience. Once an app has been deployed, the developers no longer have the luxury of taking systems offline to push new changes. This section discusses the steps required in order to fully deploy the app, including the services used, continuous integration, and deployment to both Apple's App Store and Google's Play Store.

3.3.1 Environments

In software development, an environment describes the system in which an application is deployed and executed. As an application becomes more complex with the introduction of a database or some other outside service, it is important to introduce multiple environment tiers to ensure its stability. The most commonly used environments are local, staging, and production. Local environments are contained within the developer's own computer, while staging and production environments are generally located on a remote server. An important distinction between the environments is that they do not share any outside services with each other. For example, if an application requires a database connection, each environment will typically have its own database instance while maintaining an identical structure.

3.3.2 Continuous Integration/Delivery

Continuous integration and delivery are development practices that help ensure changes made to an application will behave as expected when deployed to a production environment. Over the past couple of years CI/CD has become a standard developer practice that greatly reduces the amount of bugs that reach the end user. There are a variety of solutions for setting up a CI/CD workflow, some of which are open source such as Jenkins.

For the purposes of this project, Travis CI was chosen. Travis CI is a continuous integration service specifically integrated with GitHub. Since GitHub was used as the source control management utility for the Bookstand application, integrating the project with Travis CI was incredibly simple. Additionally, Travis CI offers a free student tier for private repositories.

3.3.3 Deploying to App Store

Apple requires every developer to maintain a paid developer account to test and publish apps in the App Store. The developer account requires a yearly subscription of $99 USD. While providing a means to publish apps, the developer program also offers access to integrations such as Apple Pay and iCloud backup.

With a developer account, users can log into the App Store Connect dashboard. The App Store Connect is a web-based tool for managing all aspects of an application. Through this dashboard, developers can publish new apps, track app analytics, monitor sales and trends, and much more.

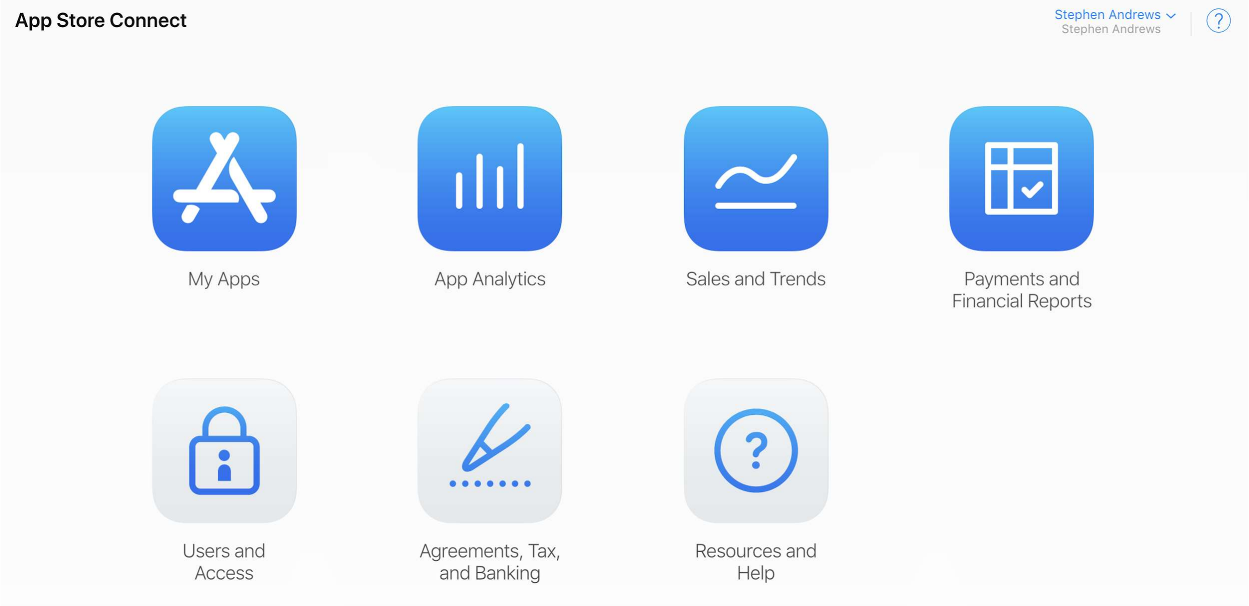


Figure 3.7: The App Store Connect dashboard

In order to submit an application to the App Store, the application must be built as an iOS App Store Package (.ipa). In the React Native ecosystem, explicit action must be taken to produce this file. To produce an IPA, developers must have the appropriate certificates generated to sign the app.[24] Application certificates are valid for one year and are used to verify an app is being published by an active developer account. Failure to comply with Apple's guidelines may result in revoked certificates.

Once the appropriate certificates are generated, running `expo build:ios -c` will prompt the user for the certificate paths. For novice users, Expo has the option to handle certificate generation. At the end of the build process, a signed IPA file will be created that can be uploaded to the App Store Connect dashboard. To upload the IPA, drag and drop the IPA file in the application loader located in Xcode under Xcode > Open Developer Tool > Application Loader as shown in figures 3.8 and 3.9.

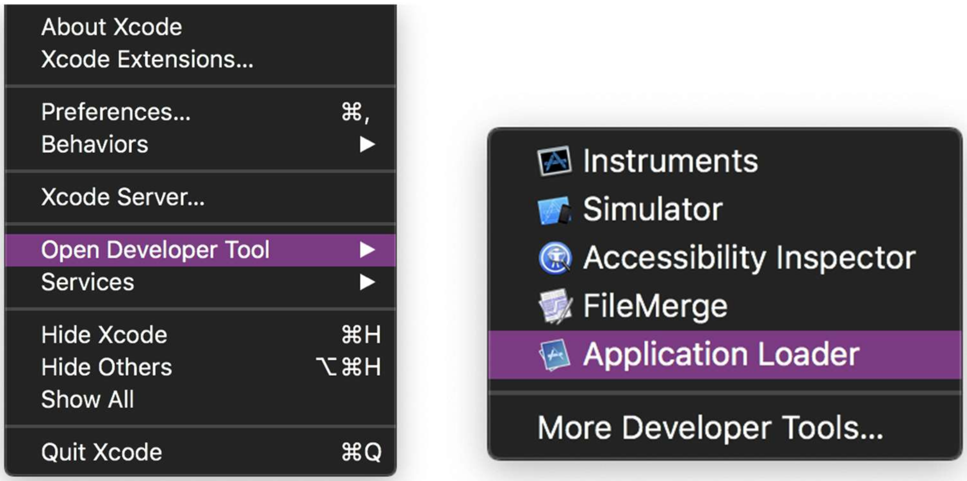


Figure 3.8: Accessing the Application Loader from Xcode menu

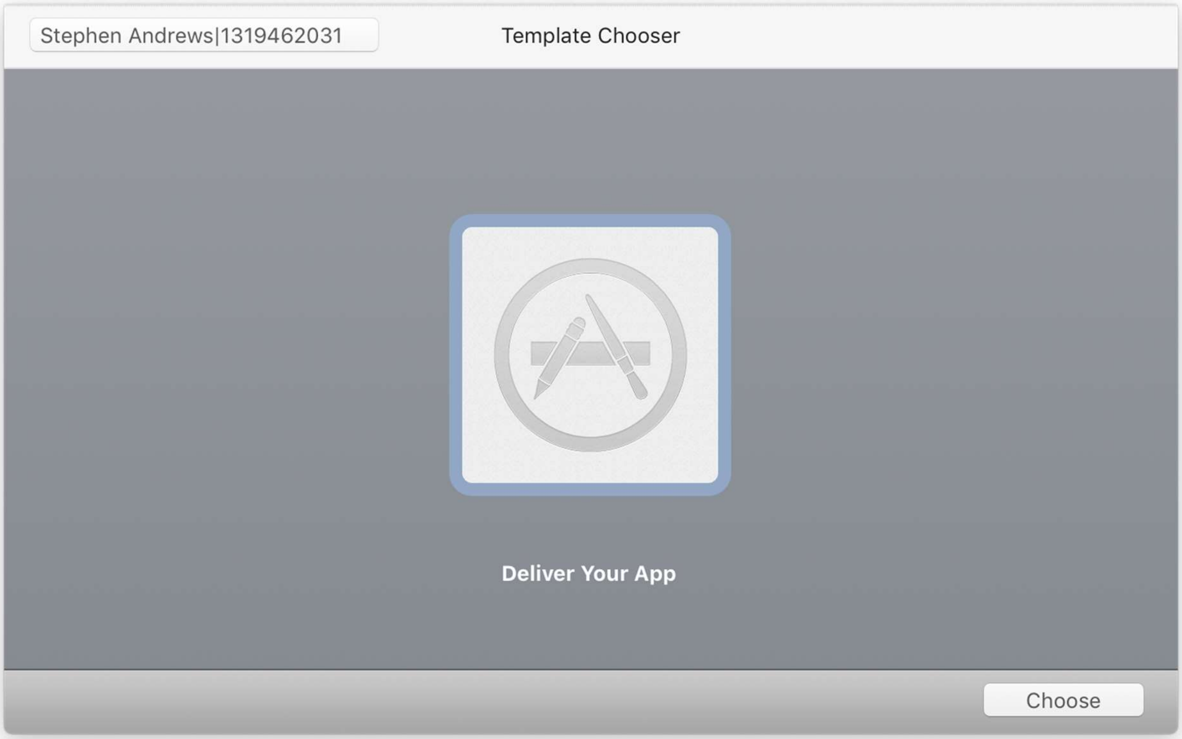


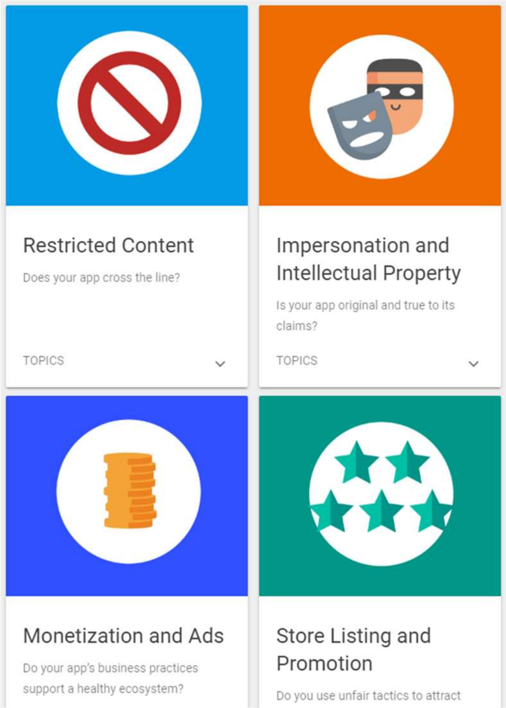
Figure 3.9: The Application Loader UI in Xcode

Once the application is uploaded through the Application Loader, it will become available in the App Store Connect dashboard to add the metadata that will ultimately be shown in the App Store.

3.3.4 Deploying to Google Play Store

The Google Play store has put out a helpful guide on launching an app on the Google Play Store.[25] This guide contains a thorough checklist, which covers everything a developer needs for developing and publishing their app.

The first recommended step is to understand the Developer Program Policies. These policies are in place to ensure that the Play store remains a trusted resource for Android users. For example, they do not want people putting malware in their apps.



Restricted Content

Does your app cross the line?

TOPICS

Impersonation and Intellectual Property

Is your app original and true to its claims?

TOPICS

Monetization and Ads

Do your app's business practices support a healthy ecosystem?

Store Listing and Promotion

Do you use unfair tactics to attract

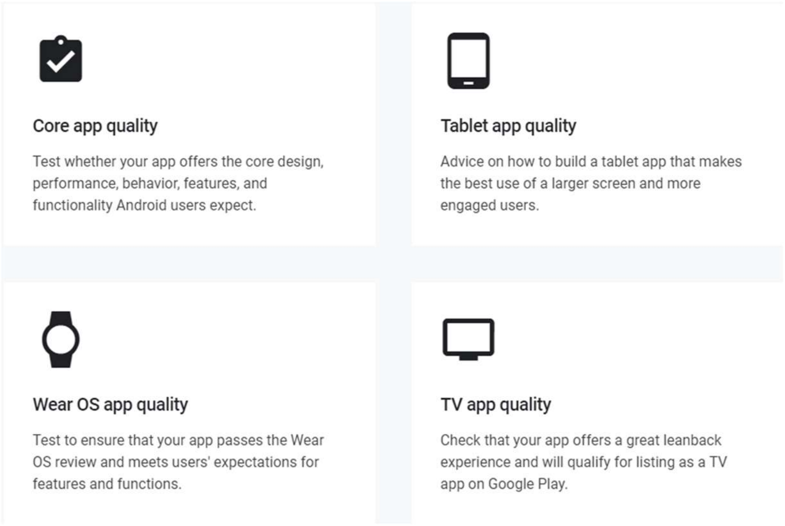
Figure 3.10: Some of the Developer Program Policies Topics

The next step is to create and prepare a developer account, and a merchant account for those who plan on selling products. Creating a developer account includes paying a one time fee of $25 at the time of this writing.[26][27]

The next recommended item is planning for localization of the app.[28] If the app is going to be deployed in different countries or cultures, there are certain traditions or practices that may vary. As an example, if the app is in Germany as well as France, it would probably be beneficial for it to have at least German and French languages available on it. Even in the United States there are places with a high population of Spanish speakers.

The checklist also suggests planning for simultaneous releases across platforms. For apps on Android and iOS, React Native is a good choice. The information specific to deploying to the iOS App Store is covered in the section “Deploying to App Store.”

Google Play has a set of quality guidelines for apps.[29] These are not difficult to abide by but are still worth researching. These guidelines cover topics such as ensuring the core functionality of the phone is not modified. An example of violating these practices would be disabling or removing the back or home buttons. Additionally, there are more specific guidelines that depend on the platform that the app is being deployed to.



Core app quality

Test whether your app offers the core design, performance, behavior, features, and functionality Android users expect.

Tablet app quality

Advice on how to build a tablet app that makes the best use of a larger screen and more engaged users.

Wear OS app quality

Test to ensure that your app passes the Wear OS review and meets users' expectations for features and functions.

TV app quality

Check that your app offers a great leanback experience and will qualify for listing as a TV app on Google Play.

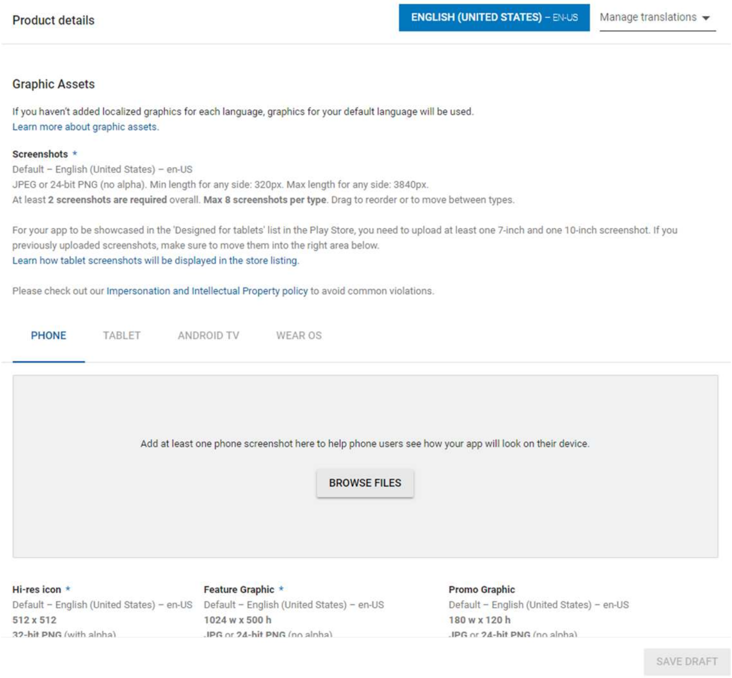
Figure 3.11: Some of the Quality Guidelines for Releasing to the Google Play Store

It is required that developers target a recent API level. As of August of 2018, Android developers are required to target at least Android 8.0 (API level 26). Apps that were already published had until November of 2018 to update to Android 8.0.

After developing the app to a satisfactory extent, the next step is to build an Android Package (APK) of the app to be uploaded to the Google Play Store on the initial release. This APK is what people will install when they download an app. With Expo CLI installed, one can simply run the command `expo build:android` to build the APK. This process should take about 10 minutes depending on the size of the app. After this is done, the APK can be downloaded from Expo’s website.

It is strongly recommended that the app is tested on multiple platforms with the built APK. Although it is good to include a bug reporting feature in the app, relying on users to discover bugs is poor practice and does not bode well for the longevity of the app. Testing the app internally is necessary for an app’s success.

The next step should come fairly naturally. Planning the app's Play Store listing should be straightforward, as a lot of this information will be taken into account during the development of the app. The Play Store listing needs screenshots and descriptions for each localized version of the app.[30] That is, if there is a French version and a Swedish version, they will each need different screenshots and information, as they may differ.



Product details

Graphic Assets

If you haven't added localized graphics for each language, graphics for your default language will be used.
Learn more about graphic assets.

Screenshots *
Default – English (United States) – en-US
JPEG or 24-bit PNG (no alpha). Min length for any side: 320px. Max length for any side: 3840px.
At least **2 screenshots are required** overall. **Max 8 screenshots per type**. Drag to reorder or to move between types.

For your app to be showcased in the 'Designed for tablets' list in the Play Store, you need to upload at least one 7-inch and one 10-inch screenshot. If you previously uploaded screenshots, make sure to move them into the right area below.
Learn how tablet screenshots will be displayed in the store listing.

Please check out our Impersonation and Intellectual Property policy to avoid common violations.

Add at least one phone screenshot here to help phone users see how your app will look on their device.

Hi-res icon *
Default – English (United States) – en-US
512 x 512
32-bit PNG (with alpha)

Feature Graphic *
Default – English (United States) – en-US
1024 w x 500 h
JPG or **24-bit PNG** (no alpha)

Promo Graphic
Default – English (United States) – en-US
180 w x 120 h
JPG or **24-bit PNG** (no alpha)

Figure 3.12: The Play Store Listing

The checklist recommends uploading the Android App Bundle to a closed or open test track. This is technically optional, but it is a great next step from testing internally. It can help discover bugs and make improvements without it being fully released yet.

After uploading the App Bundle, the prelaunch reports can be viewed to identify issues found after the app is tested automatically on different devices with different versions of Android.[31]

The next step is to set the app's price and the countries in which it is going to be distributed.[32] Once the monetization model is known, the app can be set up as free or paid.

They also suggest opting in to different distribution options.[33] This lets a developer opt their app in to specific devices and programs, like Wear OS, Android TV, and Designed for families. This promotes the app to be discovered by more users once it has been approved by Google Play.

Before publishing, the content rating of the app must be determined.[34] There is a quick and easy survey that will give an app a rating upon completion. This rating is according to laws and guidelines, and unless a mistake is made on the survey, there is nothing else to be concerned about.

The app is now ready to be published.[35] If it is an update that is being released, staged rollouts can be used to progressively push the update to more users. If there is an issue, this will limit the number of users that the update affects.

Once the app is published, there is still work to be done. One of the first things that needs to be done is to promote the app. If the app does not have users then it has failed. After the app is published, responding to reviews and fixing bugs from bug reports can help the app succeed. In addition to user bug reports and reviews, Android vitals shows important information, including download statistics and crash reports, based on the app running on users' devices.[36]

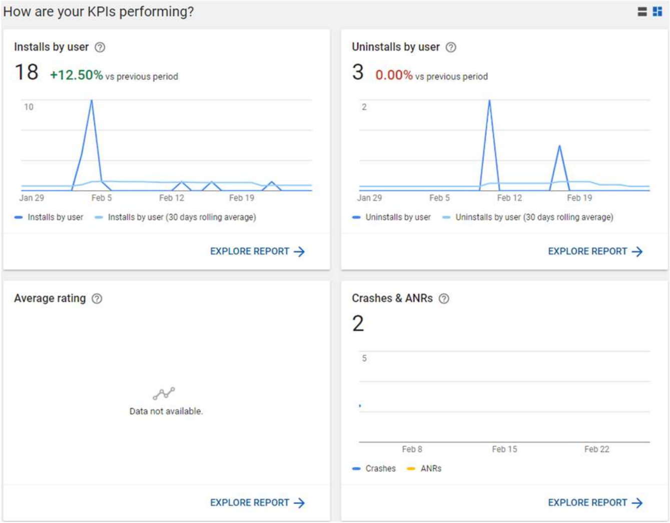


Figure 3.13: The Google Play Console Release Dashboard

Chapter 4: Services

4.1 AWS

Amazon Web Services (AWS) is a subsidiary of Amazon that provides on-demand cloud computing to its customers on a paid subscription basis.[37] Pricing is based on usage and a combination of other factors, such as the hardware and particular service that a consumer chooses. AWS offers a variety of different services, two of which are discussed below.

4.1.1 RDS

Amazon Relational Database Service (RDS) is a service that allows dedicated hosting of relational databases.[38] They offer support for Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, and Microsoft SQL Server. Pricing is based on a combination of the number of hours and rows used, as well as other factors including the number of back-ups, and the amount of data stored. Bookstand uses RDS to host its database, which contains all of the data necessary for the app to function.

4.1.2 S3

Amazon Simple Storage Service (S3) is a service that offers reliable and scalable object storage.[39] Pricing is based on the number of gigabytes of storage used each month. Bookstand uses S3 to store user-uploaded images.

4.2 Heroku

Heroku is a cloud computing platform that allows users to build, scale, and monitor web applications.[40] Heroku runs apps on what it calls “dynos,” which are basically virtual machines with certain resources allocated to them. Pricing is based on the number of dyno-hours used per month. Heroku offers many useful features, such as auto-deploying when a commit is pushed to GitHub, logging, and the ability to connect an app to other useful plugins. Heroku also offers a student discount, which greatly reduced our costs.[41]

Chapter 5: Rollout and Evaluation

5.1 Monetization

For the monetization of an app, there are certain restrictions and limitations that Bookstand faces that hinder or prevent the monetization of it in certain ways. There are a few common ways to monetize an app.[42]

Before discussing the ways an app can be monetized, it is important to understand the three models: eCPM, CPC, and CPA. eCPM means the effective cost per thousand impressions. The formula to calculate it is (Total Earnings/Total Impressions)*1000. An impression is defined as any time the app is viewed on the app store or in the search results for more than 1 second. CPC is the cost of the ad per click. It is calculated by (The Total Money Spent on the Ad/Total Measured Clicks). And CPA is cost per action. It is measured as the cost of the ad per action after the ad was clicked. These actions include downloading and launching an app, making a purchase, or subscribing to a service.

One of the popular ways to make money from apps is through in-app advertising. There are several common types of ads that are seen in most apps. Interstitial ads are one example of this. They are meant to occupy an entire screen to capture attention. These are effective at showing ads to users, however they are also frustrating to deal with from a user perspective. These ads generate about $3.50 per 1000 users on Android, and $7.20 per 1000 users on iPhone. Another type of ad is the Banner Ad. This occupies a small area on a phone's screen, and is relatively unintrusive. Per thousand users, these generate between $0.15 and $1.50 on Android, and between $0.20 and $2.00 on iPhone. Video ads are another type of ad. These are typically about 15 seconds in length and also provide a link to download the app. Video ads generate between $0.50 and $5.00 per 1000 users.

Paid apps are another way to monetize. This was especially popular back when the app stores had just launched. Over time, this model has decreased in frequency, and the advertising and in-app purchases models now make up a large portion of the monetization strategies. Paid apps work especially well for specific types of apps. Some of these app types are utilities, productivity, photo & video, and navigation.

Another monetization model is the freemium model. The freemium model gives users a certain set of functionality for no cost, and allows them to upgrade by paying in order to gain more features or remove ads.

Another way to monetize is by providing in-app purchases. There are four types of in-app purchases on the iOS App Store: consumable, non-consumable, auto-renewable subscriptions, and non-renewing subscriptions. An example of a consumable purchase would be a one time use item in a mobile game. Non-consumable purchases are purchases that only need to be made once in order to expand the set of features that an app offers. Auto-renewable subscriptions are especially nice for developers, because it provides a more consistent revenue stream. One major aspect of subscriptions is that developers can offer discounts for longer subscription periods. This incentivises the customer, and will provide an even more stable long term revenue stream.

Affiliate marketing and lead generation is another example of a monetization model. This entails either being rewarded for specific user actions that would result in another app being downloaded, or selling leads to companies looking for certain types of customers. Lead generation is an example of selling user’s data, and today it may be frowned upon. Certainly, it has a negative connotation.

Another way to monetize an app is by getting a sponsorship. If an app has a large enough user base and user engagement, there is potential for a sponsorship. Depending on the type of app, the sponsorship will vary. This is an especially good way to monetize an app because revenue generation is handled by a single advertiser.

Another way to monetize an app is through licensing. Any user-generated data that an app collects can be licensed to other companies. This allows for an app to generate revenue through non-intrusive mechanisms. This is another example of selling user’s data, and may be negatively viewed. The terms of service of an app need to outline what sensitive user data is being collected and how it is being used.

5.1.1 Restrictions on Monetizing Bookstand

Due to the use of the Google Books API, Bookstand is not eligible to be monetized in any means that directly charges the users for any aspect of the application without special approval from Google. As stated in Section 1 of the Google Books API Terms of Service, “You may not charge users any fee for the use of your application, unless you have entered into a separate agreement with Google or obtained Google's written permission.”[43] However, this is not to say that leveraging the user base for monetization infringes on this policy. For example, showing ads to users remains in compliance with these terms and conditions.

5.2 Launch

In the mobile application space, a successful launch can be crucial to the success of the project. Studies show that only 16 percent of users are willing to try a buggy app more than twice.[44] For this reason, it is crucial to carefully plan the launch of a mobile application. Since

Bookstand would only become available to WPI students, a staggered sign up rate was a major concern.

Having a staggered sign up rate could result in a dead marketplace. Without enough listings, new users may perceive the app as unpopular and avoid spending time to create a new listing. To alleviate this problem, a mailing list was hosted on bookstand.io which allowed students to be notified when the app was officially launched. In an attempt to direct students to the site, fliers were created and posted across the WPI campus and emails were sent to several mailing lists, including the CS, RBE, and IMGD departments. As a result of these efforts, 39 accounts were created within the first week of launch.

5.3 Cost Analysis

Early on in the project, the team produced a cost analysis for the database server. The goal of this cost analysis was to determine if an upgrade of storage space would be needed in the short term.

The current storage space available on Bookstand's backend is 20 GB. In order to determine if more space would be needed, the team estimated the space requirement of one thousand users, one thousand books, and one thousand listings. This was done by taking the average size of the rows in the tables that were in the database at the time and multiplying them by a thousand. The estimated size for a thousand users, a thousand books, and a thousand listings respectively are: 239 KB, 636 KB, and 500 KB.

As of 2019, WPI's total enrollment is 6,642 people.[45] Accounts for this number of people would occupy 1.6 MB of space in our database. As another example, the Library of Congress has roughly 16 million books. If the metadata of every book in the Library of Congress was stored in Bookstand's database, it would equate to 10.18 GB of storage space. Lastly, it would require only 8 GB of storage space to create listings for all of those books. Even with this level of activity, which is far beyond what we are expecting, our database would have over a gigabyte of storage space remaining. This allows us to confidently say that our database will not need to be upgraded to another tier of AWS service.

Chapter 6: Conclusion

6.1 Results

For a project that is relies heavily on users' experiences, we felt that getting feedback directly from the users was a good way to judge how successful it was. To accomplish this, we created a survey consisting of nine questions using Google Forms. Three of the questions ask about specific issues or other feedback that users may have towards the app. These responses have been useful for eliminating bugs in the app. Four of the remaining six questions aim to determine the average quality of users' experiences. They are answered on a discrete, five-point scale, ranging from "strongly disagree" to "strongly agree." The results are shown below:

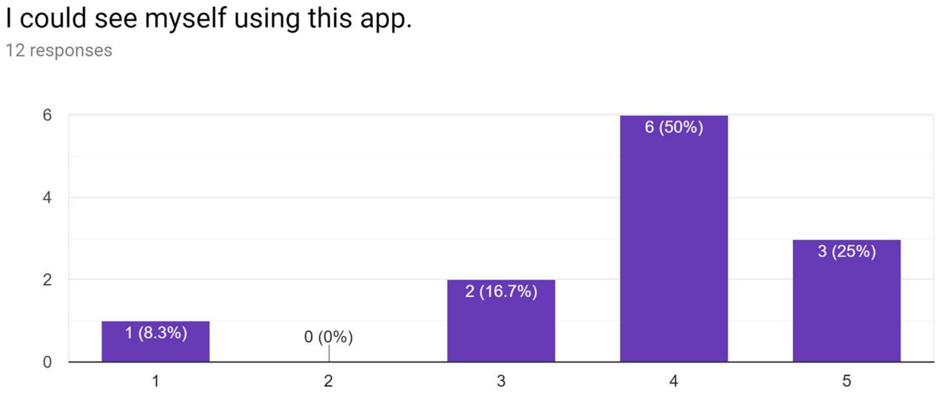


Figure 6.1: Survey results of "I could see myself using this app."

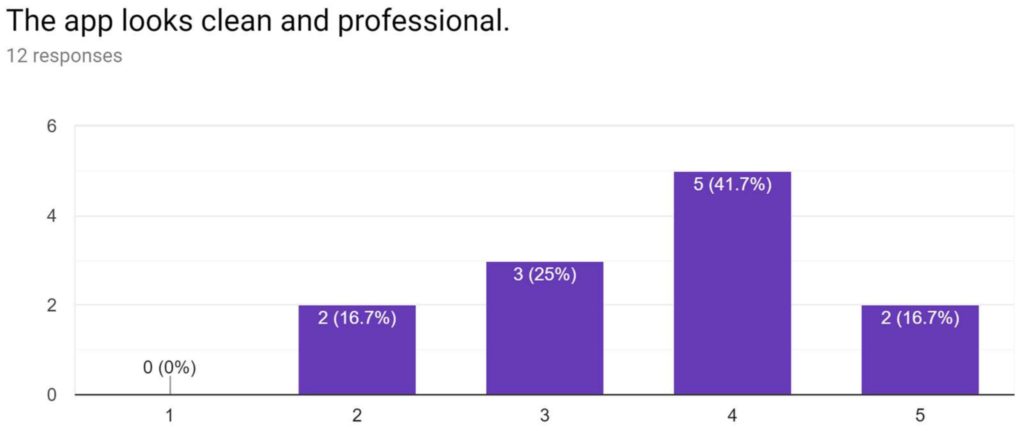


Figure 6.2: Survey results of "The app looks clean and professional."

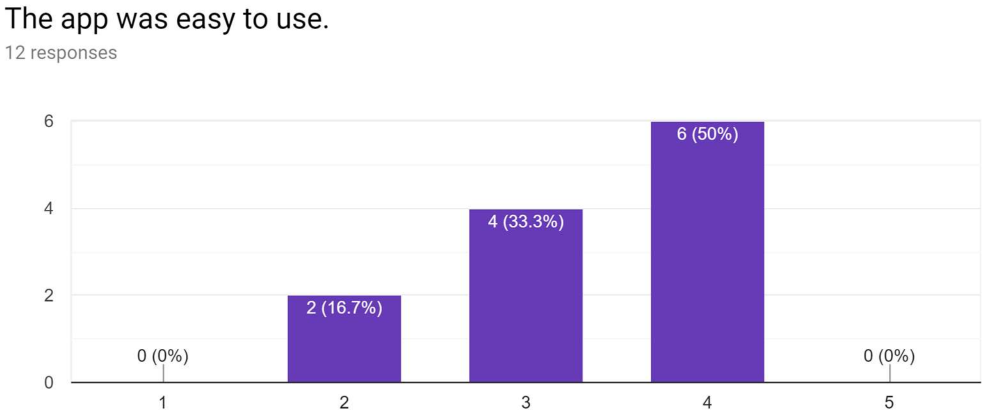


Figure 6.3: Survey results of "The app was easy to use."

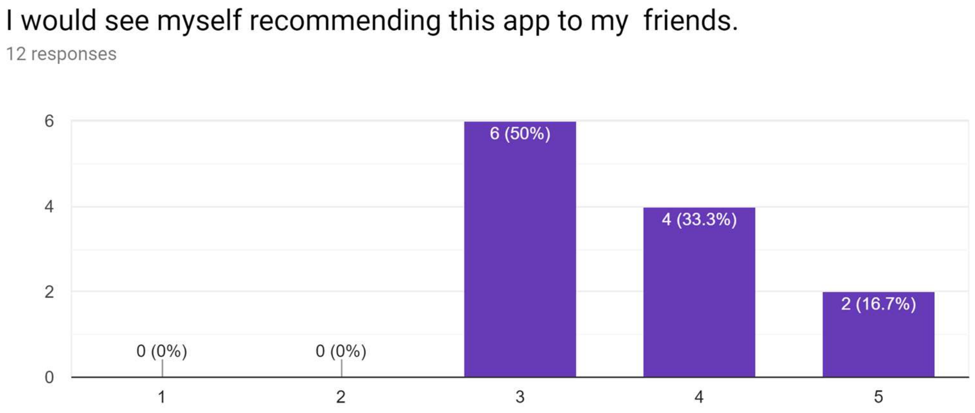


Figure 6.4: Survey results of "I would see myself recommending this app to my friends."

There were twelve responses to the questions, most of which were positive. Most of the users can see themselves using the app and recommending it to their friends. They also thought that the app looks clean and professional. The lowest overall results were on the question of whether the app was easy to use. The average response was still positive for this question, however it indicates that the app could use improvement. The remaining questions ask about students' acquisition of textbooks.

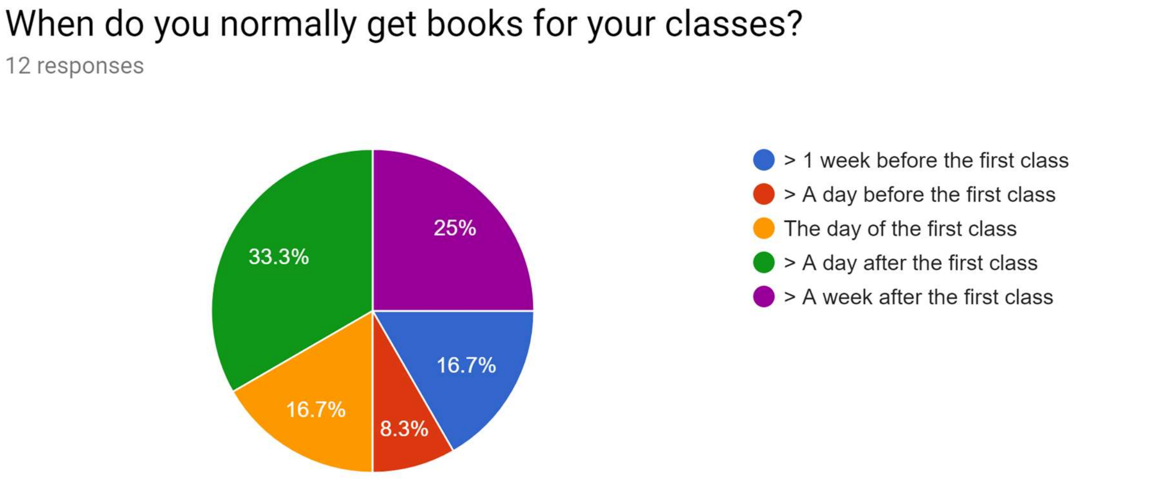


Figure 6.5: Survey results for "When do you normally get books for your classes?"

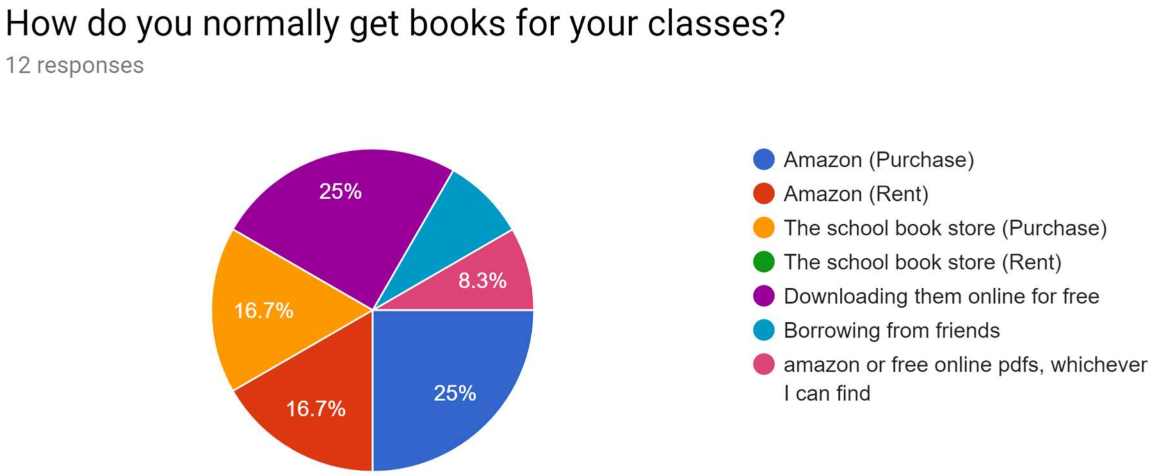


Figure 6.6: Survey results for "How do you normally get books for your classes?"

Most students acquire their textbooks after the first day of class. This is because some classes list textbooks on the syllabus that will not actually be needed and students want to avoid wasting money when so many books are overpriced. Additionally, ~41.7% of respondents attempt to get textbooks for free, whether it is by borrowing them from friends or downloading them for free. An equal number of others choose Amazon, while only 16.7% use the school bookstore.

While this project has not yet had a major impact on students' acquisition of textbooks at WPI, the results of the survey indicate that Bookstand is a needed solution. Respondents can see themselves using the app, would recommend it to their friends, and think it looks clean and professional. The app could be made easier to use, but its purpose fits perfectly with how students currently acquire textbooks: cheapest first. Despite its lack of reach, we consider this project a success; and we are confident that it has the potential for wide-reaching impact.

6.2 Unimplemented Features

Over the course of the project, there were many features that were implemented into the Bookstand app. These features all revolved around the core purpose of the app: to provide students a way to save money on their textbooks. It is safe to say that overall, the development of the app was a success. But there are still features that did not make the minimum viable product (MVP). This is unfortunate, but it is part of the development process that features get cut. Some of these features were actually items that other students suggested through survey responses.

One set of features that did not make the MVP was the ability to look up books needed by a specific class. This would have involved web crawling WPI’s course catalog and the bookstore’s website for the necessary information. Another layer to this set of features could be that students could input their schedules and it would suggest listed books to them.

Another feature that came very close to making it into the project was integration with Amazon’s bookstore. The idea was to incorporate some Amazon listings into Bookstand’s listings so that there would never be an empty search result. This would also allow the app to easily compare the listing price between a student’s listing and the Amazon listing. Additionally, with an Amazon Associates account, Bookstand would have been able to use referral links to get a small credit for purchase referrals. These features were not possible due to some unforeseen measures taken by Amazon. One of these was a preventative measure that keeps people from overusing Amazon’s API. This prevented Bookstand from querying Amazon too often, and eventually the API stopped working all together. Another circumstance that kept Bookstand from using referral links was that Associate accounts need at least three sales in order to be qualified for referrals. This means that Bookstand would have to actually sell things on Amazon in order to take advantage of the referral links.

Chapter 7: Recommendations for Future Projects

The most detrimental aspect to our project was the unrealistic timeline we set for ourselves from the start. Initially, we had planned to learn React Native, develop the app, and release it to WPI all within the first term of the project (2018 A term). Ultimately, the app was released during the first week of C term. Reflecting back on the project, we have come to better understand that the scope of the project was much more than simply learning a framework and developing an application. We severely underestimated the time it would take to structure the project, plan feature deadlines, and advertise and release to users.

Without prior knowledge of React Native, the majority of the early stages of development were spent identifying the best practices for mobile development with React Native. Initially, we had assumed that learning the framework would take only a few weeks; however, we found ourselves continually refactoring older code that was written towards the beginning of the project up until the end of the project. For this reason we had much less time to work on the core features of the app than we had initially anticipated. For example, one of the most essential features of the app, managing transactions between users, began development midway through B term. For such an important feature we would have ideally had several months to iterate and test the design.

Another setback was underestimating the amount of time we devoted to project management. Without prior knowledge of React Native, it was difficult to gauge the amount of time certain features would take. As we continued to improve our skills, our timelines more accurately reflected the amount of actual work required. However, without knowing this in advance, we lost time we had expected to use towards development. Our final advice to other projects considering developing a mobile app is to be conservative when feature planning and overestimate the amount of time development will take.

References

1. “Quick Guide: College Costs,” Pay for College - Where to Find College Scholarships. [Online]. Available: https://bigfuture.collegeboard.org/pay-for-college/college-costs/quick-guide-college-costs. [Accessed: 11-Mar-2019].
2. “How Much Does it Cost to Study in the US?,” Top Universities, 01-Jun-2018. [Online]. Available: https://www.topuniversities.com/student-info/student-finance/how-much-does-it-cost-study-us. [Accessed: 11-Mar-2019].
3. Reference: “Student2Student,” Student2Student. [Online]. Available: http://student2student.com/. [Accessed: 13-Oct-2018].
4. “StudentBookTrades,” StudentBookTrades. [Online]. Available: https://www.studentbooktrades.com/. [Accessed: 13-Oct-2018].
5. R. Hurtibise, “Peer-to-peer textbook trading app lets students be their own bookstore,” Sun-Sentinel.com, 02-Nov-2016. [Online]. Available: https://www.sun-sentinel.com/business/small-business/fl-booku-textbook-trading-app-20161102-story.html. [Accessed: 13-Oct-2018].
6. A. Perrin and A. Perrin, “10 facts about smartphones,” Pew Research Center, 28-Jun-2017. [Online]. Available: http://www.pewresearch.org/fact-tank/2017/06/28/10-facts-about-smartphones/. [Accessed: 11-Mar-2019].
7. jkielty- 1 M. 2018, “Latest report shows feature phone use still widespread,” *DeviceAtlas*, 15-May-2018. [Online]. Available: https://deviceatlas.com/blog/latest-report-shows-feature-phone-use-still-widespread. [Accessed: 17-Mar-2019].
8. “React Native · A framework for building native apps using React,” React Native Blog ATOM. [Online]. Available: https://facebook.github.io/react-native/. [Accessed: 16-Mar-2019].
9. “A Relational Model of Data for Large Shared Data Banks,” razor tie artery foundation announce new joint venture recordings | Razor & Tie. [Online]. Available: https://web.archive.org/web/20070608155955/http://www.acm.org/classics/nov95/s1p3.html. [Accessed: 11-Mar-2019].
10. “Advantages Of NoSQL,” MongoDB. [Online]. Available: https://www.mongodb.com/scale/advantages-of-nosql. [Accessed: 11-Mar-2019].
11. A. Cavale, “Why We Moved From NoSQL MongoDB to PostgreSQL - DZone Database,” dzone.com, 21-Nov-2017. [Online]. Available: https://dzone.com/articles/why-we-moved-from-nosql-mongodb-to-postgresql. [Accessed: 11-Mar-2019].

12. “Declaring Models,” Flask-SQLAlchemy - Flask-SQLAlchemy Documentation (2.3). [Online]. Available: http://flask-sqlalchemy.pocoo.org/2.3/models/. [Accessed: 11-Mar-2019].
13. “SQLAlchemy 1.3 Documentation,” Types of Mappings - SQLAlchemy 1.2 Documentation. [Online]. Available: https://docs.sqlalchemy.org/en/latest/index.html. [Accessed: 11-Mar-2019].
14. “React Native · A framework for building native apps using React,” React Native Blog ATOM. [Online]. Available: https://facebook.github.io/react-native/. [Accessed: 11-Mar-2019].
15. “React Native · A framework for building native apps using React,” React Native Blog ATOM. [Online]. Available: https://facebook.github.io/react-native/showcase. [Accessed: 11-Mar-2019].
16. “Repositories Sorted by Stars,” GitHub. [Online]. Available: https://github.com/search?q=stars:>0&s=stars&type=Repositories. [Accessed: 11-Mar-2019].
17. Robin, Chen, Fanghao, Robin, and Chen, “Why Discord is Sticking with React Native,” Discord Blog, 26-Jul-2018. [Online]. Available: https://blog.discordapp.com/why-discord-is-sticking-with-react-native-ccc34be0d427. [Accessed: 11-Mar-2019].
18. Robin, Chen, Fanghao, Robin, and Chen, “Why Discord is Sticking with React Native,” Discord Blog, 26-Jul-2018. [Online]. Available: https://blog.discordapp.com/why-discord-is-sticking-with-react-native-ccc34be0d427. [Accessed: 11-Mar-2019].
19. “Expo,” Expo. [Online]. Available: https://expo.io/. [Accessed: 11-Mar-2019].
20. “Redux · A Predictable State Container for JS Apps,” Read Me - Redux. [Online]. Available: https://redux.js.org/. [Accessed: 11-Mar-2019].
21. “React Native Vector Icons,” react-native-vector-icons directory. [Online]. Available: https://oblador.github.io/react-native-vector-icons/. [Accessed: 11-Mar-2019].
22. RedQ TeamWe are a world class software startup focusing on building scalable, “React Native Snap Carousel,” RedQ Inc, 12-Feb-2019. [Online]. Available: https://redq.io/blog/react-native-snap-carousel/. [Accessed: 11-Mar-2019].
23. Apsl, “APSL/react-native-keyboard-aware-scroll-view,” GitHub, 10-Dec-2018. [Online]. Available: https://github.com/APSL/react-native-keyboard-aware-scroll-view. [Accessed: 11-Mar-2019].
24. Apple Inc, “Certificates,” Using 2D and 3D Transforms. [Online]. Available: https://developer.apple.com/support/certificates/. [Accessed: 11-Mar-2019].

25. “Launch checklist,” Android Developers. [Online]. Available: https://developer.android.com/distribute/best-practices/launch/launch-checklist. [Accessed: 11-Mar-2019].
26. “How to use the Play Console - Play Console Help,” Google. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/6112435. [Accessed: 11-Mar-2019].
27. “Link a Google Play developer account to your payments profile - Play Console Help,” Google. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/3092739. [Accessed: 11-Mar-2019].
28. “Localization checklist,” Android Developers. [Online]. Available: https://developer.android.com/distribute/best-practices/launch/localization-checklist.html. [Accessed: 11-Mar-2019].
29. “Quality guidelines,” Android Developers. [Online]. Available: https://developer.android.com/docs/quality-guidelines/index.html. [Accessed: 11-Mar-2019].
30. “Make a compelling Google Play store listing to drive more installs,” Android Developers. [Online]. Available: https://developer.android.com/distribute/best-practices/launch/store-listing.html. [Accessed: 11-Mar-2019].
31. “Use pre-launch and crash reports to improve your app,” Android Developers. [Online]. Available: https://developer.android.com/distribute/best-practices/launch/pre-launch-crash-reports.html. [Accessed: 11-Mar-2019].
32. “Set up prices & app distribution - Play Console Help,” Google. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/6334373. [Accessed: 11-Mar-2019].
33. “Distribute your apps on the right Android platforms,” Android Developers. [Online]. Available: https://developer.android.com/distribute/best-practices/launch/distribute-apps.html. [Accessed: 11-Mar-2019].
34. “Content ratings for apps & games - Play Console Help,” Google. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/188189. [Accessed: 11-Mar-2019].
35. “Publish an app - Play Console Help,” Google. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/6334282. [Accessed: 11-Mar-2019].

36. “Use Android vitals to improve your app's performance and stability,” Android Developers. [Online]. Available: https://developer.android.com/distribute/best-practices/develop/android-vitals. [Accessed: 11-Mar-2019].
37. “Amazon Web Services (AWS) - Cloud Computing Services,” Amazon. [Online]. Available: https://aws.amazon.com/. [Accessed: 11-Mar-2019].
38. “Amazon Relational Database Service (RDS) – AWS,” Amazon. [Online]. Available: https://aws.amazon.com/rds/. [Accessed: 11-Mar-2019].
39. “Cloud Object Storage | Store & Retrieve Data Anywhere | Amazon Simple Storage Service,” Amazon. [Online]. Available: https://aws.amazon.com/s3/. [Accessed: 11-Mar-2019].
40. “Heroku,” Cloud Application Platform. [Online]. Available: https://www.heroku.com/. [Accessed: 11-Mar-2019].
41. “Heroku for GitHub students,” Cloud Application Platform. [Online]. Available: https://www.heroku.com/github-students. [Accessed: 11-Mar-2019].
42. “App Monetization Models,” Business of Apps. [Online]. Available: http://www.businessofapps.com/marketplace/app-marketing/research/app-monetization-models/#2. [Accessed: 11-Mar-2019].
43. “Terms of Service | Google Books APIs | Google Developers,” Google. [Online]. Available: https://developers.google.com/books/terms. [Accessed: 11-Mar-2019].
44. S. Perez and S. Perez, “Users Have Low Tolerance For Buggy Apps – Only 16% Will Try A Failing App More Than Twice,” TechCrunch, 12-Mar-2013. [Online]. Available: https://techcrunch.com/2013/03/12/users-have-low-tolerance-for-buggy-apps-only-16-will-try-a-failing-app-more-than-twice/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed: Techcrunch (TechCrunch). [Accessed: 11-Mar-2019].
45. “How Does Worcester Polytechnic Institute Rank Among America's Best Colleges?,” U.S. News & World Report. [Online]. Available: https://www.usnews.com/best-colleges/wpi-2233. [Accessed: 11-Mar-2019].