2020

LibTracker--A React Native & Quarkus App to Track Your Personal Book Library

Adam Luckenbaugh
Grand Valley State University

LibTracker—A React Native and Quarkus App to Track your Personal Book Library

Adam Luckenbaugh

A Project Submitted to

GRAND VALLEY STATE UNIVERSITY

In

Partial Fulfillment of the Requirements

For the Degree of

Master of Science in Applied Computer Science

School of Computing and Information Systems

December 2020

The signatures of the individuals below indicate that they have read and approved the project of Adam Luckenbaugh in partial fulfillment of the requirements for the degree of Master of Science in Applied Computer Science.

<name of project advisor>, Project Advisor	Date
<name of GPD>, Graduate Program Director	Date
<name of unit head>, Unit head	Date

Table of Contents

Abstract *4*

Introduction *5*

Project Management *7*

Organization *8*

System Architecture **8**

React Native App **10**

Quarkus Server **16**

Reflection *19*

Conclusions *22*

Bibliography *23*

Abstract

This project consisted of developing a full-stack mobile application to track individuals' personal book libraries. Users were able to create accounts, import book data, categorize books by tags, organize books in virtual shelves and search the library. The ultimate goal of the project was to develop expertise in mobile application development and gain new experience with a modern framework, and therefore React Native [1] was chosen. To manage the app's data a server backend was implemented using the open source Quarkus [2] framework and data was stored in a Postgres [3] database. In addition to developing the mobile app and backend a small-scale publicly accessible deployment using a Raspberry Pi [4] as the host was built. This was performed in order to learn some mechanisms that later could be used in a future production cloud deployment that could support a public release of the LibTracker app. Overall the project began with merely the concept and progressed through learning React Native from scratch all the way to a fully functional proof of concept grade prototype with all the basic features necessary to track one's personal book library.

Introduction

This project consisted of designing and building a full-stack mobile application used to track one's personal library of books and included learning all of React Native. The motivation of the project sprung from a personal need to keep track of a large family collection of books a few years ago, but when the project was begun the original motivation was no longer relevant since the collection's size had decreased to an insignificant number. However mobile application development was something that I as a back-end software engineer had little experience with, and so this project idea became a viable learning opportunity through which a prototype was produced using a current mobile application development framework. Ultimately this project began with the personal library tracking concept and progressed from knowing nothing about React Native to a functional full-stack Android version of the LibTracker app backed by a Quarkus-based server to manage the data.

The core feature of this application was to record data about each book and provide the capability for a user to browse or search for if a certain book was in his or her possession. The app also supported keeping track of where in the user's home the books were located through linking books to virtual shelves. Books also were classified by tags, and users were able to view and adjust which books were marked with particular tags. Importing of book data ideally would be done with as little user effort as possible so a feature was implemented to support scanning ISBN barcodes and querying the Google Books API [5] to load book information. Users were also able to type ISBN numbers for books that could not be scanned and were able to manually enter all data by hand as a last resort. Thumbnails and an arbitrary number of supplementary images of each book also could be provided by the user in addition to any thumbnail supplied by the Google Books API.

Because the library data was stored in the server and not on the users' devices it is possible for this system to be extended later if desired for purposes like exporting the library data in order to sell books, or later implement a social networking feature to enable searching for and lending books to friends. Regardless, the ultimate goal of this project was to develop increased expertise with Quarkus but more importantly learn from the ground up a modern mobile application development framework, React Native. Quarkus, an open-source framework "crafted from the best of breed Java libraries and standards" [2] and targeting microservice deployments was a framework the developer was already professionally familiar with but this project provided additional experience using it to perform data management. Data was stored at rest in a Postgres database. React Native, a cross-platform Javascript-based framework supporting Android and IOS applications and released by Facebook was entirely unknown to me at the beginning of the project but is popular in the developer community, so the main focus consisted of learning all the concepts of React and using it to build a functional mobile app as the user interface. This ultimately resulted in finding that it was a framework that provided overall a subjectively enjoyable developer experience.

Project Management

Due to this project being implemented by a sole developer the project management approach was quite simple. Milestones were drafted at the outset of the project and refined as the project proceeded. Until midway through the project a simple Agile approach of using sticky notes in three columns of "to do," "in progress," and "done" measured progress in the spirit of a Kanban board but was later transitioned into using Github [6] issues. While not easily showing at a glance issue statuses like a Kanban board, Github issues supported more expansive descriptions and categorization along with implementation notes which proved more useful than the sticky note approach. The table below shows the final version of the milestones, illustrating the high-level progress of feature completion throughout the semester.

Table 1: Development Milestones

Milestone
Set up developer environment (Postgres, react native project for Android, Quarkus template app)
Account creation & login: username + password only with email authentication
Populate library by scanning ISBN and auto-populating info from Google Books API; also by populating manually or by typing in ISBN rather than scanning
Browse library, sorting by title, author or most recently added; support taking pictures of books for thumbnails and supplementary images
Search library by title, author or ISBN
Add/edit/delete/choose tags; add tab for tags to navigation menu
Add/edit/delete/choose/browse shelves; add tab for shelves to navigation menu
Clean up UI, fix bugs, make UX slightly better
Write more unit tests for backend to ensure proper server functionality
Deploy backend with top level domain name and make installable app that doesn't require Expo. Add SSL encryption to server.
Capture sample of personal library data to showcase app functionality

Organization

System Architecture

The LibTracker system was composed of the major components shown in the following deployment diagram. The React Native mobile app was the only user-facing component and interacted with the Quarkus server primarily but also queried the Google Books API to gather book data when adding new books. The Quarkus server stored all persistent data in a Postgres database and interacted with a dedicated Gmail account created for this project in order to send emails for the account signup process.

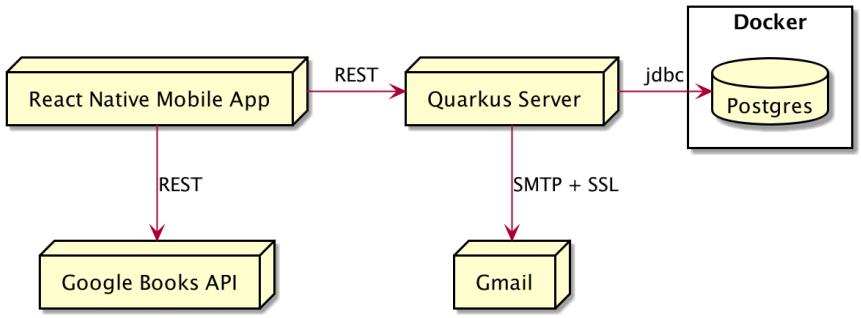

Figure 1: Development System Deployment Diagram

During development the Quarkus server and Postgres database were run on a Macbook Pro. The database was run in a standard Docker [7] container with a volume mounted for the data directory, which permitted restarting the container without losing the schema or persisted records unless the volume was intentionally removed. The Quarkus server was run in development mode with a Maven [8] command which permitted attaching a debugger and supported hot-swapping of Java code as it was written, a feature that sped up development by avoiding the entire recompilation and server restarting process as much as possible. The

interactive development environment (IDE) used for both the Quarkus project and the React Native project was IntelliJ [9].

The React Native app was developed using Expo, a "set of tools and services built around React Native and native platforms that help you develop, build, deploy, and quickly iterate on iOS, Android, and web apps from the same JavaScript/TypeScript codebase" [10]. This made it very easy to focus on writing the Javascript code to build the UI and leverage the Expo toolset to handle debugging, building native installation packages and deploying updates to production. Most of the time breakpoints did not need to be set in order to debug issues but from time to time it was helpful to use Google Chrome's [11] developer tools to hit breakpoints in the React Native codebase. The most beautiful feature was the ability to hit save after writing some Javascript and having the app on the Android development phone update in less than a second!

On the other hand, as seen in the following figure during production both Postgres and Quarkus were run in Docker containers built for a Raspberry Pi and the Quarkus backend was accessible via a top level domain name with Cloudflare [12] handling the DNS routing. HTTP traffic was encrypted with a Let's Encrypt [13] SSL certificate in order to protect users' credentials and data over the wide area network. Since it was less clear how to configure Quarkus to use the SSL certificate files, an NGINX [14] reverse proxy was configured on the Raspberry Pi to support the HTTPS connections and route traffic via HTTP on the loopback address to Quarkus. Using NGINX to handle the SSL certificates brought an added benefit of being able to leverage the certbot [15] utility to automatically renew the SSL certificates since the utility interfaced well with NGINX. Because the Raspberry Pi ran an ARM processor the docker container for Quarkus had to be built on the Pi itself rather than on the Mac; this was

because the Mac ran an Intel processor and the container build process would crash due to some binaries requiring an ARM processor executing during the container build.

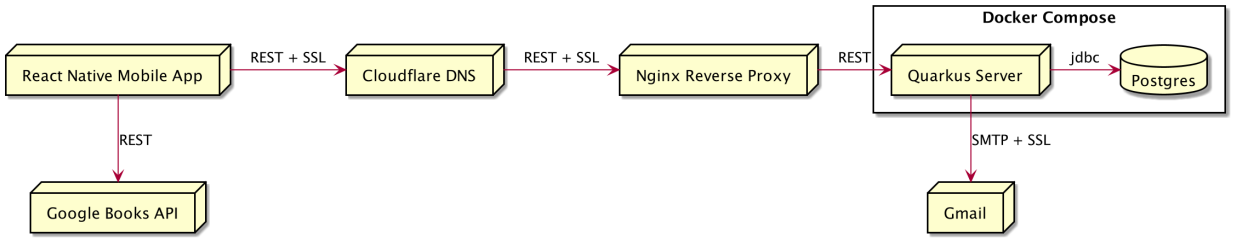

Figure 2: Production System Deployment Diagram

React Native App

As mentioned above, the React Native app was developed using Expo. This provides a framework of utilities that supports creating the project files, running the app under development in debug mode using the Expo mobile app, quickly reloading code changes during development and managing building of the native installation packages for Android (IOS is also supported but was not used in this project) and deploying over-the-air updates after users install the app. Code was structured in a typical format where each screen corresponded to a code file with a single component, and screens were composed of multiple reusable components. Examples of reusable components built for this project were validated text inputs for emails and passwords, responsively rendered images, a list refresh behavior manager, contexts to pass user and host information to all components, components to render book details, book lists and floating action buttons (FABs) to provide a selection of book import choices. The user interface was entirely manually tested.

The major libraries used in the app along with their purposes are listed below.

Table 2: Major React Native Libraries

Library	**URL**	**Purpose**
React Navigation	https://reactnavigation.org/	Navigation among screens
Expo Barcode Scanner	https://docs.expo.io/versions/latest/sdk/bar-code-scanner/	Scanning ISBN barcodes
React Native Vector Icons	https://github.com/oblador/react-native-vector-icons	Icons, such as for tab navigation and edit, save and delete icons
Expo Linking	https://docs.expo.io/workflow/linking/	Deep linking from activation email to open app and finish activation
Exp Secure Store	https://docs.expo.io/versions/latest/sdk/securestore/	Persisted user authentication info in app
React Native Elements	https://reactnativeelements.com/	Material design components
React Native Paper	https://callstack.github.io/react-native-paper/	Material design components
React Native Image Viewer	https://github.com/ascoders/react-native-image-viewer	Viewer for book pictures

The following list summarizes the main features and the screenshots after them illustrate the main screens of the app. Some of the less important screens, such as the launch splash screen, are omitted here for brevity.

- Signup: users may create an account using an email address to which will be sent an activation email that contains a deep link to open the LibTracker app and finish the account activation
- Login: users may log in using an email address and password
- Library: shows the entire contents of the user's library, allowing sorting by title, author and date added; users may also import books using the floating action button to choose manual entry, typing in an ISBN or scanning an ISBN barcode
- View book details: users may view all data stored about the book, including the list of authors, date published, page count, textual description, ISBN numbers and which shelf the book is on
- Edit/delete book: users may delete or edit any book information, also selecting tags and the current shelf the book is on
- Shelves: allows organizing groups of books by shelf and adding/deleting/renaming shelves as needed
- Tags: allows categorizing books by assigning tags to them along with adding/deleting/renaming tags as desired; these tags are automatically populated from the Google Books API data but can be manually set as well
- Search: a user may search his library for books by title, author or ISBN and may scan an ISBN instead of typing it
- Settings: provides capability to log out of the current account and while in development mode to also change the URL of the Quarkus server

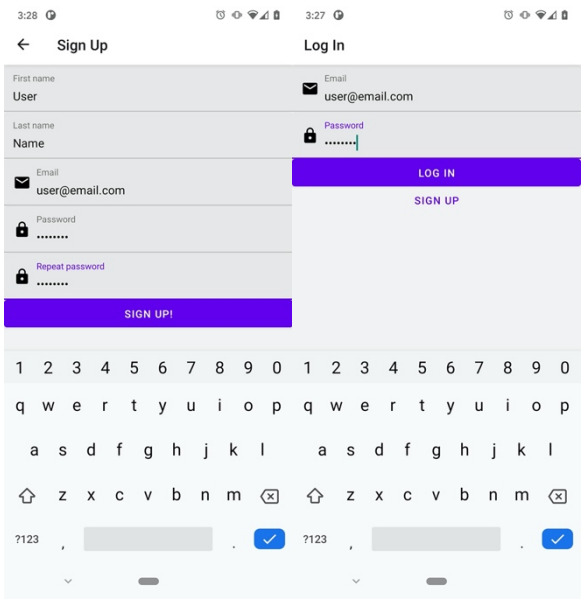

Figure 3: Signup and Login Screens

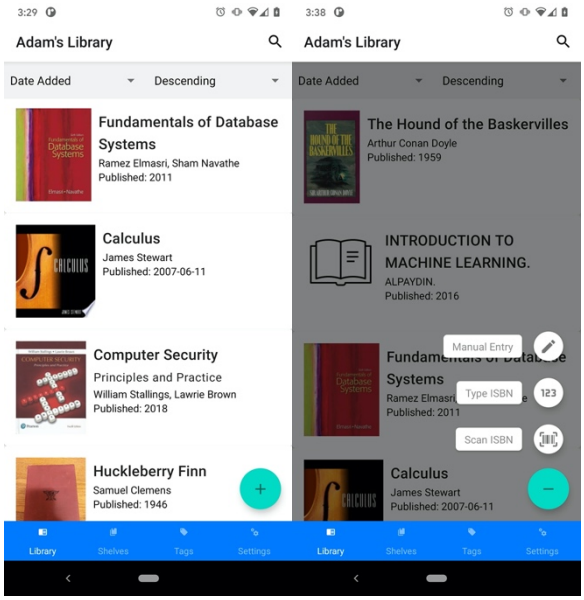

Figure 4: Library Overview & Floating Action Button for Adding New Books

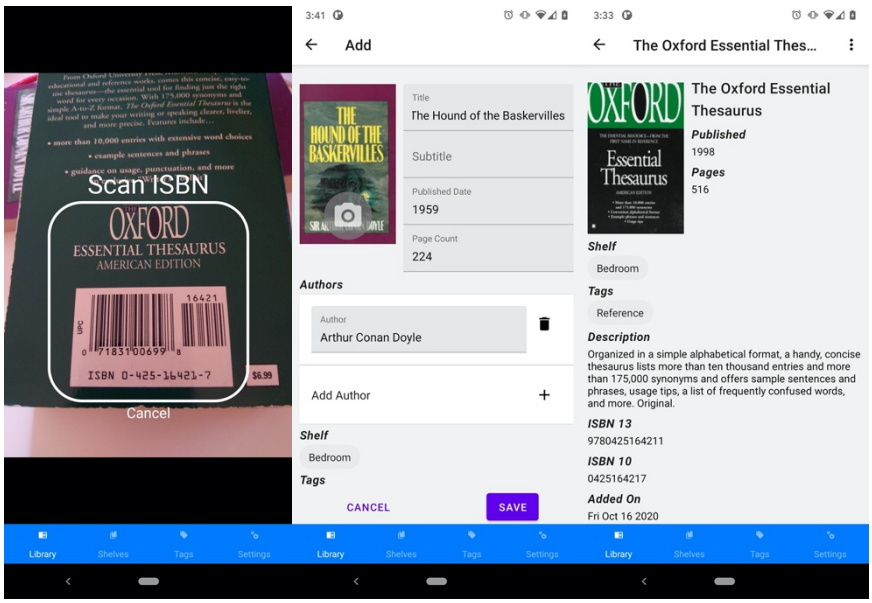

Figure 5: Scanning an ISBN, Editing a Book and Viewing Book Details

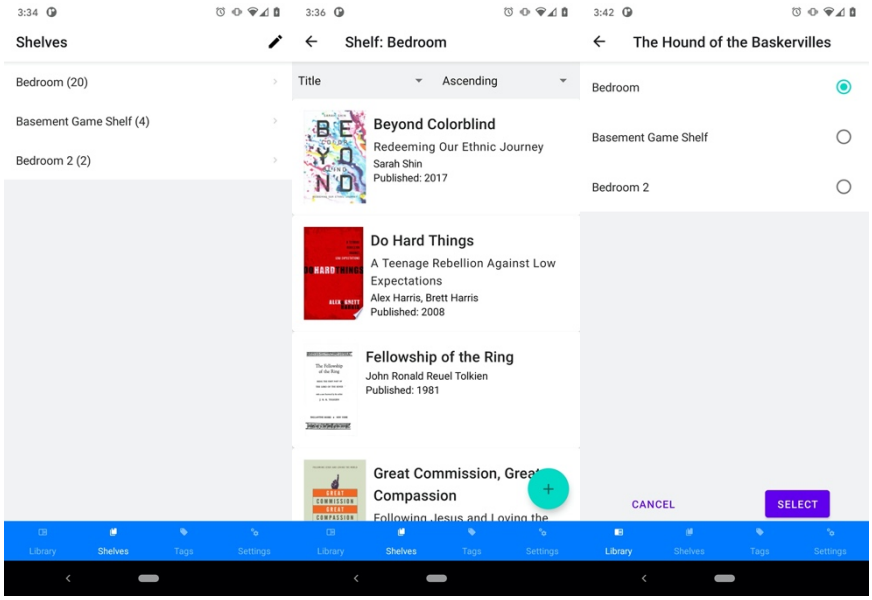

Figure 6: Shelf List, Shelf Contents and Shelf Selection when Editing a Book

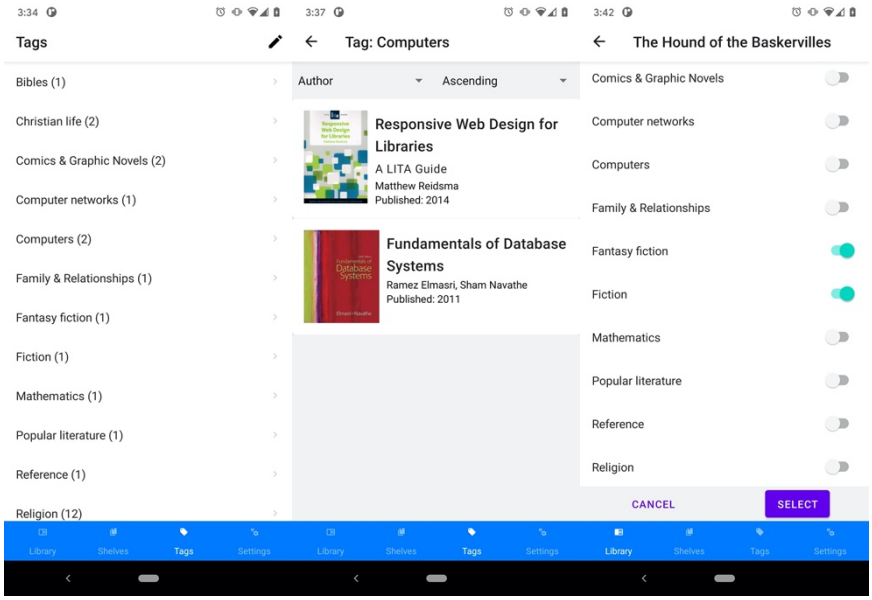

Figure 7: Tag List, Books with Tag, and Tag Selection when Editing a Book

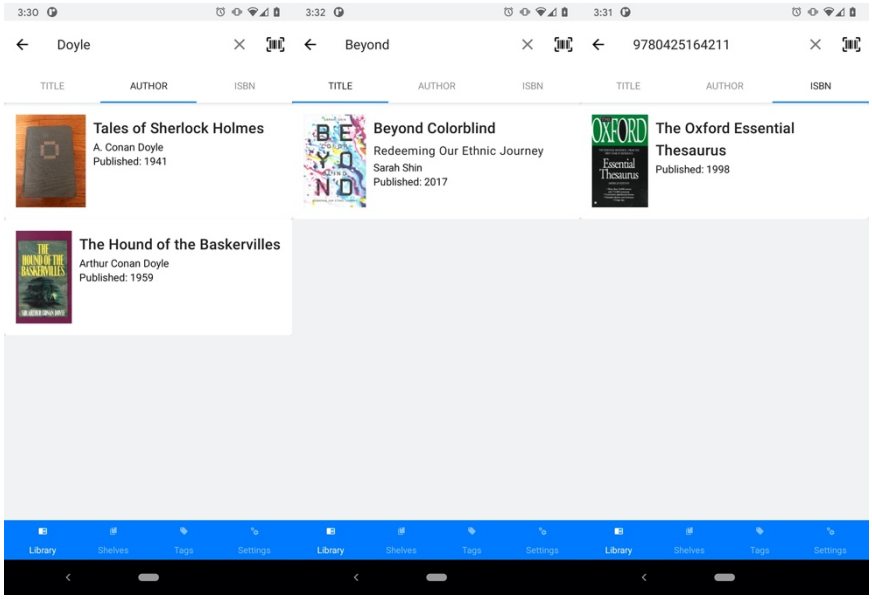

Figure 8: Searching by Author, Title and ISBN

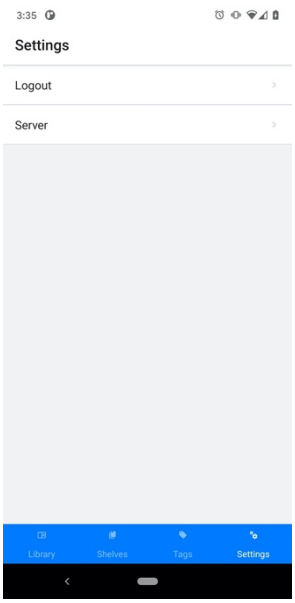

Figure 9: Settings Screen

After finishing the majority of the development of the app, Expo was used to build a native Android installable APK and installed onto the Android development phone. The publishing functionality of Expo was also used to push over-the-wire updates to the installed apps without requiring reinstallation of the APK as new code was written.

Quarkus Server

The data management for the React Native app was performed via REST calls to the Quarkus server. Essentially Quarkus is a collective of best-in-class Java libraries configured to work well together in order to support a system that is tailored for cloud deployments of microservices. While not part of the scope of this project, an element of future work would be to more fully deploy this server as a cloud-based application and leverage cloud architectures to support auto-scaling and other management features with this microservice-like architecture to support a large number of users. When creating the Quarkus app a plethora of libraries was

available to choose from to compose the project; for this one the major libraries both from Quarkus and those added from other sources are summarized in the following table with brief descriptions of their usage. All the dependencies are available in the public Maven repository [16].

Table 3: Major Quarkus App Dependencies

Maven GroupId:ArtifactId	**Purpose**
com.sun.mail:javax.mail	Send account authentication emails
org.projectlombok:lombok	Annotation-based boilerplate code generator to clean up the code
org.apache.commons:commons-lang3	Apache Commons utility functions, such as for string manipulation
org.zalando:problem	application/json+problem implementation for neatly reporting exceptions through REST
io.quarkus:quarkus-liquibase	Liquibase database schema management
io.quarkus:quarkus-resteasy	RestEasy REST functionality
io.quarkus:quarkus-container-image-docker	To build docker images of the Quarkus app
io.quarkus:quarkus-hibernate-orm-panache	To make working with the Hibernate ORM easier using Panache
io.quarkus:quarkus-jackson	Support JSON serialization
io.quarkus:quarkus-jdbc-postgresql	Postgres database support
io.quarkus:quarkus-hibernate-validator	Support database constraints with the Hibernate ORM
io.quarkus:quarkus-junit5-mockito	Mockito for mocking objects during unit testing
io.quarkus:quarkus-junit5	Unit testing with JUnit
io.rest-assured:rest-assured	Unit testing REST endpoints with RestAssured
org.testcontainers:postgresql	TestContainers Docker container for Postgres, providing a live database for unit tests

As already mentioned, a Postgres database running in a Docker container was used to persist data at rest and support querying the library data. All server actions were synchronously initiated through REST calls to the resources shown below in the table. Automated unit tests

were written to test the majority of functionality provided by these endpoints in order to reduce the amount of effort required to manually test the final system.

Table 4: REST Endpoint Definitions

Resource Class	Endpoint	HTTP Method	Description
AuthResource	/auth/signUp	POST	Register for an account
AuthResource	/auth/authCheck	POST	Check if a given authentication token corresponds to a logged in user
AuthResource	/auth/login	POST	Log in with given user credentials, returning an authentication token on success
AuthResource	/auth/activate	POST	Finish activating a user's account using the token emailed to the address provided at signup
BookResource	/book/search	GET	Search by title, author or ISBN
BookResource	/book	GET	Retrieve a list of books in the entire library, sorted by author, title or recently added and optionally constrained by a given tag or shelf
BookResource	/book/{bookId}	DELETE	Delete the given book
BookResource	/book	PUT	Create or update a given book
ImageResource	/image/{id}	GET	Retrieve an image by its ID
ImageResource	/image	POST	Save the given list of images and return the IDs once persisted
ShelfResource	/shelf	GET	Get a list of all the shelves in the user's library
ShelfResource	/shelf	PUT	Create or update a given shelf
ShelfResource	/shelf/{shelfId}	DELETE	Delete the given shelf
TagResource	/tag	GET	Get a list of all the tags for all books in the user's library
TagResource	/tag	PUT	Create or update a given tag
TagResource	/tag/{tagId}	DELETE	Delete the given tag

Reflection

The ultimate goal of this project was to make a foray into the realm of mobile application development using a modern framework while pairing it with building a backend to manage the data. All things considered, this project succeeded in that endeavor, proved by beginning with a concept of tracking one's personal library of books and no knowledge of React Native and culminating with a proof-of-concept grade mobile application to track one's personal library of books backed by a publicly accessible server.

Overall React Native was found to be an enjoyable developer experience. After spending some initial time reading through documentation and tutorials describing the basics of React and starting to write code it became easy to understand the major concepts and quickly write useful and effective code. Unfortunately it seems that in order to build a full-fledged app it is necessary to leverage multiple open-source libraries or build much from scratch. An example would be the use of React Native Paper to handle simple components like icon buttons, certain styles of text and theming. In general open source libraries run the risk of becoming unmaintained, and when the app needs to be updated to the next version of React it is possible that these libraries become out of date and must be replaced in the app; therefore it would be nice if the React ecosystem provided by Facebook contained more implementations of common components. However overall this was not found to be much of a problem while building LibTracker. Generally it was found that there was a sufficient selection of open source libraries to accomplish development with reasonable ease.

Development and debugging was much better than previous experiences I had in mobile application development where previously code changes required recompilation and redeployment of the app on the development device, a time consuming activity. However, with

React Native code changes were automatically reloaded in the app on the development device while it was running, and this in very short order, making it very quick and easy to write code and see its results. This decreased turnaround time for code changes tremendously and alone was a wonderful feature. During development it was not found to be necessary to often set breakpoints to debug issues but it was found to be useful to do this from time to time with Google Chrome's developer tools in the window that opened when the npm [17] server was launched on the development system, and this worked well.

One major shortcoming of the project is the lack of pristine user experience and graphical appeal to the user interface. In typical companies graphical and user experience designers would be available to assist developers but unfortunately this project did not have personnel with these skills. Therefore the main focus was to implement a functional app while learning React Native and pay reasonable attention to neat presentation while not placing great priority on making a beautiful interface.

Especially considering that the React Native portion of the project was the major focus the Quarkus backend proved to be less of a concern. This was unsurprising due to me already having some professional experience with this framework. Nevertheless plenty of work was required to implement the required functionality to serve the mobile app. It certainly was easier to leverage the Quarkus ecosystem to quickly integrate the necessary libraries rather than it would have been to integrate them manually from their respective open source projects.

The deployment of the Quarkus server using Docker and NGINX running on a Raspberry Pi and having DNS handled with Cloudflare and traffic encrypted with a Let's Encrypt SSL certificate was a useful learning experience. While initially it was a little troublesome to figure out how to build a docker container with the Quarkus application that would run on the

Raspberry Pi it did provide some additional experience learning how to build docker containers. This deployment approach naturally will not scale to be able to support any large number of users in a real deployment scenario, so if the LibTracker app was to be released to the public a real cloud deployment would have to be made. The lessons learned in how to build the Docker containers and configure the ingress via the NGINX reverse proxy and SSL certificates would prove useful by mapping to similar features in cloud providers like Google. While not explored in the scope of this project, the Google Kubernetes Engine [18] is used in a project in which I am professionally involved, and it would likely prove a viable solution for LibTracker.

Conclusions

Overall this project consisted of building a mobile app to track one's personal library of books and progressed from starting with this concept with no knowledge of React Native to a complete proof-of-concept grade application backed by a standalone backend server and deployed in a simple low-scale production system. This project provided a valuable learning opportunity especially regarding React Native, and left me with an impression that I would certainly recommend usage of this for professional projects.

LibTracker was developed to a proof-of-concept grade and thus several major elements remain as items of future work that would be valuable to complete before publicly deploying the app. These are listed below:

- Work with a UX and/or graphic designer to improve usability and attractiveness
- Convert Javascript to TypeScript to aid compiler-checked React Native code
- Deploy Quarkus backend (and Postgres database) to cloud, such as Google Kubernetes Engine
- Explore and implement a way to unit test the React Native components to help quality control during development rather than rely solely on manual testing
- Use Redux to manage state more cleanly
- Performance tuning when record count in tables increases (index and query tuning)
- Research and implement an analytics solution to track screen and feature usage

Bibliography

[1] Facebook, Inc, "React Native - a framework for building native apps using React," [Online]. Available: https://reactnative.dev/. [Accessed 18 November 2020].

[2] "Quarkus - Supersonic Subatomic Java," [Online]. Available: https://quarkus.io/. [Accessed 18 November 2020].

[3] "PostgreSQL - The world's most advanced open source database," [Online]. Available: https://www.postgresql.org/. [Accessed 18 November 2020].

[4] "Teach, Learn and Make with Raspberry Pi," [Online]. Available: https://www.raspberrypi.org/. [Accessed 18 November 2020].

[5] "Google Books APIs | Google Developers," [Online]. Available: https://developers.google.com/books. [Accessed 18 November 2020].

[6] "GitHub," [Online]. Available: https://github.com/. [Accessed 18 November 2020].

[7] "Empowering App Development for Developers | Docker," [Online]. Available: https://www.docker.com/. [Accessed 18 November 2020].

[8] "Maven - Welcome to Apache Maven," [Online]. Available: https://maven.apache.org/. [Accessed 18 November 2020].

[9] "IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains," [Online]. Available: https://www.jetbrains.com/idea/. [Accessed 18 November 2020].

[10] "Expo," [Online]. Available: https://docs.expo.io/versions/v36.0.0/. [Accessed 18 November 2020].

[11] "Google Chrome - Download the Fast, Secure Browser from Google," [Online]. Available: https://www.google.com/chrome/. [Accessed 18 November 2020].

[12] "Cloudflare - The Web Performance & Security Company | Cloudflare," [Online]. Available: https://www.cloudflare.com/. [Accessed 18 November 2020].

[13] " Let's Encrypt - Free SSL/TLS Certificates," [Online]. Available: https://letsencrypt.org/. [Accessed 18 November 2020].

[14] "NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy," [Online]. Available: https://www.nginx.com/. [Accessed 18 November 2020].

[15] "Certbot," [Online]. Available: https://certbot.eff.org/. [Accessed 18 November 2020].

[16] "Maven Repository: Central," [Online]. Available: https://mvnrepository.com/repos/central. [Accessed 18 November 2020].

[17] "npm | build amazing things," [Online]. Available: https://www.npmjs.com/. [Accessed 18 November 2020].

[18] "Kubernetes - Google Kubernetes Engine (GKE) | Google Cloud," [Online]. Available: https://cloud.google.com/kubernetes-engine. [Accessed 18 November 2020].