Building an App in Eighty Days

Raphael Wieland

Advisor Dr. Pam Cutter

Department of Computer Science

A paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Arts at Kalamazoo College.

2018

Preface

Sometimes it is difficult to land a job or internship, even in the computer science industry. Sometimes life happens and it gets in the way of work schedules, if one were to hypothetically have a job or internship. Luckily, there is a socially acceptable emerging career path for those who can't hold down a job and/or drag themselves out of the house in the morning called freelance software development. As many mobile application developers are unemployed or as they like to refer to themselves, self-employed, I figured that I could gain some valuable life and career experience by moonlighting as a full stack hybrid mobile application developer, in a language with which I was previously unfamiliar, and without outside direction.

Acknowledgements

I would like to thank my advisor, Pam Cutter, for her continued support throughout my time here at Kalamazoo College.

Table of Contents

Introduction

The purpose of writing something down is so that it isn't forgotten. Whether it be a post-it note or a text file, writing it down keeps it for posterity or until that post-it note is no longer needed. While memory usually provides a pretty accurate account of events, it is sometimes useful to have access to a recorded accounting. What would we know of our past if it were not for the written word? Each additional puzzle piece of information we have about the past allows us to make more informed decisions about the puzzle that is the world around us.

The modern computer revolutionized the storage and utilization of information and thrust us into what they are calling the Information Age. At just one byte per character, information is insanely cheap to store, copy, and transmit. Rather than having a monk tirelessly copy a manuscript over the course of a few years, in the 21st century the same manuscript can be copied and sent to the other side of the world in a matter of seconds for practically free. As a result, just about every aspect of our lives has been turned into a data set. How do we sift through all of this data to find relevant information? How do we piece together the puzzle? This can be either quantitatively or qualitatively, by human or machine learning algorithm. Once enough data points have been collected, exploratory data analysis can be performed in which trends can be identified and models can be formed. Sometimes even simpler quantitative and qualitative insights can be useful, such as summary statistics or graphical representations of information.

Just like in finance, the purpose of tracking your weightlifting 'gains' and 'losses' is so that you can see how they compound over time under different circumstances and in response to different factors. Historically, a composition notebook and pencil were

used to track such data, but in the era of big data why subjugate yourself to carrying a pencil around the gym when you carry around a fully functional computer in your pocket? After taking a course on Android mobile application development here at Kalamazoo College, I noticed a discrepancy in weightlifting logbook applications on the Google Play Store. While many of the logbook applications available on the Google Play store were functional, many lacked the features to backup your workouts or export them for analysis, features that I deem as necessary for a logbook application. Those that allowed for backup or exportation, often do so as comma separated value files which is fine for a computer scientist but isn't exactly user friendly. What if I no longer had to worry about losing my workouts because they were logged directly to the cloud? What if I could view insights of my workout history right on my phone? It is with those two thoughts in mind that set out to build my own weightlifting logbook application.

While I hope that one day this application will provide insight into my weightlifting progress, it is going to start out merely as a data collection tool tailored to my own particular needs and interests. I need an application that allows me to create different workouts, or saved lists of exercises. Next, I need to create instances of those workouts in which I can record how many sets per exercise for each exercise and the number of repetitions per set, and weight per repetition for each set. Lastly, I need to be able to view my progress over time in some sort of intelligible manner. As I intend on using this application on my phone, I will develop it as a hybrid mobile application that it can be used cross-platform on Android or iOS.

Mobile Applications

In case you've been living under a rock for the past decade, or are reading this in the Kalamazoo College archives years from now, and are unfamiliar with the term 'mobile application', let me provide some background. In the beginning, Steve Jobs conceptualized the iPhone. As Jobs would boast many time in the following years, "Apple revolutionized the cell phone industry when it introduced the iPhone" ("April 2010 Special Event"). It was sleek, fast, beautiful, powerful, and revolutionary. It integrated a cell phone, iPod, camera, text messaging, e-mail and Web browsing all into one touch-screen device. Even though originally third-party apps were not supported, the first generation iPhone was still a runaway success.

In July 2008, the iPhone 3G was released which featured higher data speeds, a GPS, and an application platform (iOS 2.0) that turned the iPhone into a general-purpose mobile computer. iOS 2.0 provided developers the ability to write their own applications for the iPhone and then share them on the marketplace that we know as the App Store. However, as third party applications can only be installed from the App Store, Apple also inadvertently revolutionized the software development and distribution industries and the Internet as a whole. As of March 2017, there were 2.2 million applications available on the Apple App Store (Android et al.) Native iOS application carry an .ipa file extension, are built using the iOS SDK through the Xcode IDE, which is only available on Macintosh computers, and are usually written in Swift or Objective-C.

Around the same time (2007), Google unveiled Android, a mobile operating system based on the linux kernel, and founded the Open Handset Alliance, a consortium to develop open standards for mobile devices. Android strives to be the antithesis to iOS.

As it is based on the Linux kernel, through a process called 'rooting' one can obtain privileged control of the device's root directory, thereby allowing almost complete control over the device. Rather requiring applications to be installed from the Google Play Store, Android allows applications to be installed via third party marketplaces or directly via an Android Application Package or APK. The Android versus iOS competition echoes the Windows versus Macintosh operating system war of the previous millennia; while Apple focuses on marketing the perfect blend of hardware and software, Google focuses entirely on producing software and leaves the hardware development in the hands of the cell phone manufacturers. As the Android source code is open source, phone manufacturers are able to use Android free of charge which drastically cuts down on development cost. As of March 2017, there were 2.8 million applications on the Google Play Store (Android et al.) Native Android Applications are built using the Android Software Development Kit and are usually written in Java though the Android Studio IDE.

HTML5 Applications

When Steve Jobs first released the original iPhone, he did so without third party applications because he envisioned that developers would write specialized web pages that could be rendered through the phone's web browser. Since first conceptualizing my application two years ago, a new type of application that is particularly well suited to the previously stated needs of this application, and true to Steve Job's vision, has become increasingly popular. HTML5 applications are web apps built using the fifth and most recent version of the HyperText Markup Language which render their user interface (UI) as a WebView in the mobile device's browser rather the operating system's native UI

framework. As such, HTML5 applications allow web developers to design mobile applications using their existing HTML, CSS, and JavaScript knowledge.

HTML5 applications offer several advantages over native applications. First off, the HTML5 applications allow developers to target multiple platforms with a single codebase; rather than having to write two separate native applications to be run on iOS and Android, you can develop a single HTML5 application that can be rendered on not only iOS or Android but any device with a modern web browser such a desktop computer running GNU/Linux. Secondly, you no longer have to appease the Apple and Google gatekeepers so that your application will be allowed on their private, centralized marketplace. In short, HTML5 applications partially restore freedom to your software locked hardware.

There are a few drawbacks to HTML5 applications as opposed to native applications. The main drawback is that, due to security risks, not all of your device's hardware functionality is available through the browser. Would you want someone to be able to wipe your hard drive if you visit a malicious website? I thought not. Hybrid mobile applications circumvent the issue by wrapping the HTML5 application as a webview in a native application. Apache Cordova is one such mobile application development framework that wraps HTML5 applications into a packaged distribution that has access to native device APIs.

While hybrid applications don't offer quite the same level of performance as native applications, for our application the difference would be negligible. Due to my project's web-interaction centric functionality, cross-platform goal, and limited hardware interactions, I concluded that it would be best written as an HTML5 application that could later be wrapped into a hybrid application later if the need arose. However, as my

existing web development skills are junior at best, this ought to be quite the learning experience.

Front-end Development

First, let's review the fundamentals of website development. The three most important tools in a front-end web developer's toolkit are HTML, CSS, and JavaScript. As mentioned earlier, HTML stands for HyperText Markup Language. It is the standard markup language used to display content on all webpages. Cascading Style Sheets (CSS) is a sheet style language that is used to describe the layout of a document written in HTML or other markup language. JavaScript is a high-level programming language that is traditionally interpreted client-side to create dynamically updating webpage content. Together these languages compose almost all of the webpages on the world wide web.

I am familiar with all three languages, but wouldn't consider myself to be an expert in any of them. HTML was the first programming language that I ever learned. The second language was of course CSS. In high school, I co-taught a web development summer bootcamp in which we covered the basics of creating static and dynamic websites as well as the use of frameworks such as bootstrap. Since being first introduced to it, the language has evolved into its fifth version bringing with it a whole list of new elements and cool new features that allow websites to do amazing new things. CSS has also added a few new tricks as it had to adjust for all the new elements in HTML5. JavaScript was the first high level programming language that I learned. Starting off with text based role playing game that I wrote at the age of 13, it exposed me to the endless possibilities of a Turing-Complete language. As this project is the first time that I

am using JavaScript in an asynchronous context, I had to familiarize myself with concepts such as callbacks and promises.

TCP/IP

Once a web page is created, how it is shared with others on the world wide web? In the client-server model, the server (backend) hosts the files of which the website consists and clients (frontend) can access the website through the Internet protocol suite. The TCP/IP model is a suite of protocols which found the conceptual model of how computers communicate over the Internet. Named after the two main protocols, the TCP/IP model defines how end-to-end communications, such as client-server communications, should be packetized, addressed, transmitted, routed and received. The TCP/IP model is broken down into four levels: the application layer, the transport layer, the Internet layer, and the network access layer which are summarized in Table 1. For something to be transferred over the Internet, it must be passed down the TCP/IP suite of protocols before it can be sent as binary over a physical medium.

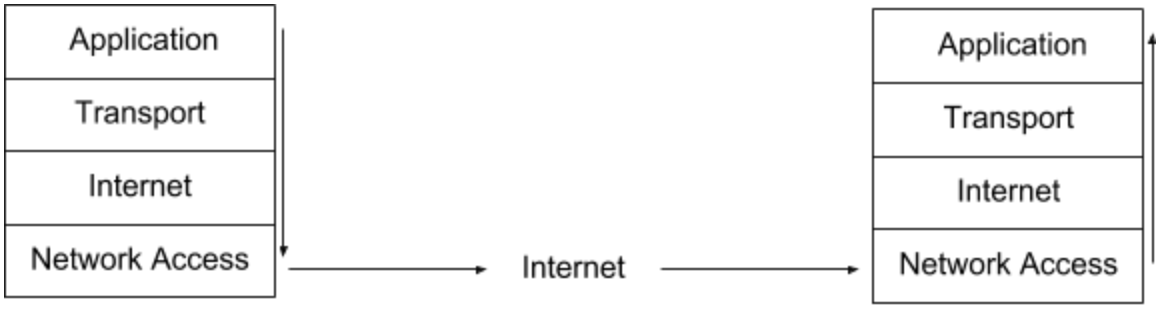
Table 1.

Protocol Layer	**Specific Protocols**	**Description**
Application	HTTP, DNS	Protocols specific to applications
Transport	TCP, UDP	Directs packets of information to the correct port
Internet	IP	Directs packets to specific machine
Network Access	Ethernet	Binary signals transmitted across a medium

When attempting to visit a website, the client submits a request to the server using the Hypertext Transfer Protocol (HTTP). If the request is too long the

Transmission Control Protocol (TCP) will break it up into smaller chunks called packets. TCP then adds its own headers and routes the packets to a specific port on the web server, usually port 80. Next, the Internet Protocol (IP) which dictates where the packets will be sent adds its headers to the packet. Every computer on the internet has a unique IP address that allows it to be identified. Once the source IP address and port and the destination IP address and port have been set, the HTTP request is now able to be converted into binary and sent over the Internet. Once the request is received by the server, TCP has to piece the message back together and route it to the correct port where the HTTP request can finally be processed by the server before it responds back to the client through the same process which is represented visually in Figure 1.

Figure 1. TCP/IP Breakdown of Two Computers Communicating

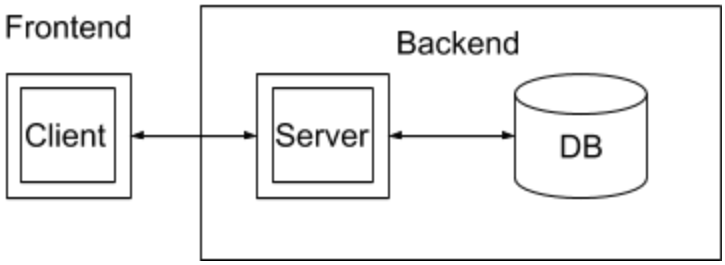


Backend Development

HTTP is a connectionless text-based protocol. The client (a web browser) sends a request to the server, the server contemplates the request, and then server sends a response. Additional requests must be made for each resource including every image, external CSS file, or linked script. This highlights that there are two types of web developing, front end development which concentrates on presentation to the client and

backend development which concentrates on everything else. This client-server or frontend-backend relationship can be observed in Figure 2.

Figure 2. The 'Frontend' and 'Backend' of the Application



An Application Programming Interface or API is a set of building block that allows the frontend to access resources controlled by the backend. A representational state transfer or RESTful API is one which conforms to a set of uniform and predefined stateless operations. The most common of those operations are create, read, update, and delete or CRUD functions and correspond to the GET, POST, PUT, and DELETE HTTP methods. It is through HTTP and this RESTful API that my application will access the Node.js server.

Competition

In the two years since I originally came up with the idea for this application, the market has evolved. Simple Workout Log and JEFIT both offer the feature to backup your workouts to the cloud. As there are already several prominent mobile applications that offer weightlifting logbook solutions with new ones being added to the Apple App Store and the Google Play Store every day, does the world need another weightlifting logbook mobile application? Below is an analysis of two applications with which my application will be in direct competition.

Simple Workout Log

Simple Workout Log is a free android application which can be used to log and track anaerobic exercise routines. Its intended audience is weightlifting enthusiasts, both professional and amateur, who want to move away from a physical paper workout logbook and into the 21st century. Rather than carry around a notebook and writing utensil, this app allows you to log your workouts locally to your phone. Simple Workout Log was one of the first exercise logs to land in the Google Play store and has since become one of the most popular applications due to its speed and simplicity. However, due to its simplistic user interface, some of the application's features are hidden behind cryptic buttons.

The application consists of several tabs, including a summary view, an exercise view, a category view, a routine view, and a tools view. The summary view provides the user with stats such as last workout date, total exercises and weight logged to date though the application, and a list of most recently logged exercises. The exercise tab allows users to create their own custom exercises or download common exercises from a server. If you click on a particular exercise, you can log an entry for just that exercise, view a history of logged instances of that exercise, or view a graphical representation of your history. The category page provides an alternative way to access exercises. You can create custom categories such as "push" exercises and "pull" exercises so that you can easily group exercises as you see fit. The routine view allows you to build your own workout routine in which you can select which exercises you want to include.

JEFIT: Workout Tracker, Gym Log & Personal Train

JEFIT is an application for Android. It is also a weightlifting logbook which allows you to create workouts, add exercises, and specify the weight, reps, and sets for each exercise. Most importantly, it allows you to save your workout locally or in the cloud. Cloud backups is a very useful feature that was missing from even Simple Workout Log, until recently.

This is a relatively new application that was honored with the editor's pick. While it does have a great, intuitive UI, useful and convenient tabbed layout, graphs to track your progress, and social connectivity, many of the views are too cluttered with very specific buttons. Perhaps there is a better way to allow users to input weight than to place a full number pad on the screen?

Designing The Application

It was clear that the world didn't need another weightlifting logbook, but that I still did. As a user, and potentially the sole user at that, I would like to be able to create a new workout routine to customize the logbook to my specific needs. This routine can be a single workout or a multiple day split. For each day, the user will be able to add exercises by either selecting them from a list of exercises grouped by body part. The database of exercises will be populated with exercises from Arnold Schwarzenegger's *The New Encyclopedia of Modern Bodybuilding*, my weightlifting bible. A stretch feature would be to allow users to submit a request for a exercise to be added to the database, if a particular exercise is not available or to implement a method for creating custom

exercises. For each exercise in your workout, you will be able to alter the number of sets, reps, and weights from the default values.

As a user, I would also like to be able to log instances of my previously created workout routine so that I can review them later. Relevant information that should be included in all logs include date, start time, end time, exercises, sets per exercise, reps per set, and weight used per rep. If I want to increase the weight or lower the amount of reps, I should be able to do so while I am logging my lift. Hopefully, I will also be able to track useful statistics such as progress per exercise, total weight lifted per workout, or average workout length.

As an administrator, I need to keep sensitive information secure. As I originally built this application as a data collection tool, I need a way to look at workout information without knowing anything about the user who created it. As at first most data will likely all be my own, there is are very few security concerns. However, as this application is built out and features are added security becomes a bigger and bigger issue. No one would likely mind if someone leaked a complete record of their workouts. However, if the application is modified to log more information about the user security becomes a huge issue. What about a self-documented record of your past two years weight? Complete pictures of your progress? I need to make sure that I am storing the information securely in the database and that I store as little personal information as possible while recording all of the workouts.

My Stack

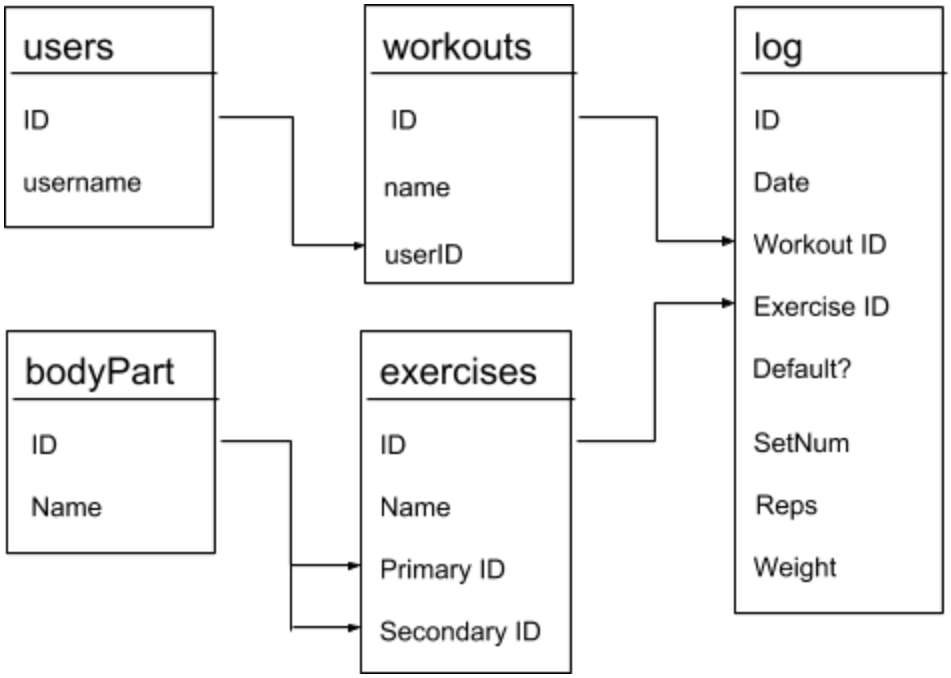
Now that I have a clear idea of what kind of application I am building, it is time to pick which technologies to utilize. As I had been thinking about it for several years, I went about creating the database first.

SQL

SQL stands for structured query language. It is a language that lets users interact with a relational database. There are three major components of SQL: DML or Data Manipulation Language, DDL or Data Definition Language, and DCL or Data Control Language. DML allows you to retrieve, update, add or delete from a database, essentially CRUD functions. DDL allows you to create and modify the database itself. DCL maintains proper database security (Rockoff 3). I was first introduced to SQL as part of the inaugural CUNY Masters in Data Science bridge course I complete prior to attending Kalamazoo College. While No-SQL databases such as MongoDB are becoming increasingly popular, I decided to stick with SQL as relational databases are particularly well suited to the kind of application I plan to build.

The structure of my database is layed out in Figure 3. There is the *bodyPart* table which links each macroscopic muscle group to an ID. There is the *exercises* tables which links a vast variety of exercises to an ID and their primary and secondary muscle groups. There is the *users* table which links a username to a userID and the *workouts* table which links workouts to an ID and a user. Lastly, there is the *log* table which links an instances of exercises by linking them to the workoutID, exerciseID, workoutDefault, setNum, reps, and weight.

Figure 3. Database Design



JavaScript

JavaScript is a high-level, dynamic, multi-paradigm interpreted programming language. If HTML provides the content for a web page and CSS dictates the page's style, then JavaScript provides website functionality. JavaScript allows for websites to be dynamic changing mediums rather than static pages of formatted text. While traditionally used for client-side scripting, the increasingly popular "JavaScript everywhere" paradigm allows web application development to be unified around this one powerful language.

Node.js is an asynchronous JavaScript runtime environment for executing JavaScript code server-side. By modifying Google Chrome's V8 JavaScript engine, Node.js allows JavaScript functionality to move from the client's browser to the application's backend. Through asynchronous callbacks and promises, Node.js is able to largely avoid blocking calls which stall the JavaScript thread. This is the first I've worked with Node.js.

Not only does Node.js replace a traditional Apache HTTP server to serve resources to visitors, but it also allows developers to use premade modules such as express.js, a web application framework for Node.js. Express provides all the necessary routing tools to create middleware and APIs as well as provides a standard boilerplate code. This is the first I've worked with express. After watching most of the MEAN stack youtube tutorials from TraversyMedia, I embarked on creating a RESTful API (TechGuyWeb). From my user stories, I knew much of what my backend needed to do. The API would take HTTP requests, process them, and then perform Create, Read, Update, and Delete (or CRUD) functions from a SQL database which would be storing all persistent user data. While I had originally stated only 'create' in the goals, what would create be without the ability to read, update or delete? Therefore, I would need to create API routes that would be able to perform CRUD functions on the the *workouts* table and the *logs* table.

Eventually as this application is scaled to multiple users, API routes would need to be created for user sign in and user authentication. Most of this is made trivial by the passport.js module. Passport.js is unobtrusive middleware for authentication for Node.Js based applications. It allows for OAuth 2.0 authentication to be required before requests would be processed. In addition to allowing local authentication strategies, passport.js

allows for single sign-on authentication from countless providers including Google, Facebook, and Twitter.

Frontend

After creating all of the CRUD API functions in the backend, I turned my sights on the frontend. As stated earlier, this project is intended to be a hybrid mobile application. Therefore, my next logical step is to build a frontend in HTML5, CSS, and JavaScript that can be then packaged into a native application through Apache Cordova or similar program. There are plenty of options when it comes picking a frontend framework. I decided to use Onsen UI due largely by the amount of documentation on their website.

Onsen UI is a front-end framework for mobile applications. It is merely a collection of UI components written in HTML5, cssnext, and JavaScript that create a near native UI experience that conforms to both iOS and Android design standards. It is framework-agnostic meaning that you can utilize vanilla JavaScript to manipulate the elements rather than a fancy modern JavaScript-based framework such as Angular.js. In addition, Onsen UI offers a tool called Monaca which is built upon Apache Cordova, the open source backbone of Adobe's proprietary PhoneGap framework. Monaca allows for simplistic hybrid mobile application development as it converts your web application into a fully functioning native application. This is the first I've worked with OnsenUI or Monaca/Cordova.

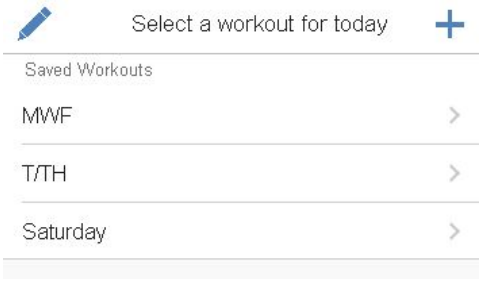
Log a workout

For this application, we will use a single-page application model in which all application logic is contained within a single HTML document. The Document Object Model (DOM)

can then be updated asynchronously via JavaScript. My application has two major information flows: logging a workout and reviewing results.

First I built out the flow to create a new workout and populate it with exercises.

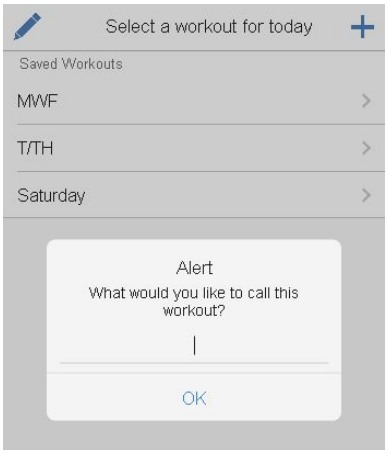
Figure 4: My Workouts



This first page allows the user to select a workout from a list of their premade workouts. Clicking on a workout allows you to log an instance of that workout as shown in Figure 4. To edit or delete a workout, the user clicks on the edit button located on the top left of the view which shows an edit and delete icon for each workout allowing the user to call the update workout or delete workout API routes. If the user hasn't created a workout yet, or if they would simply like to create a new one, the user can click on the plus button on the top right to create a new workout, as shown in Figures 5 through 7, and select exercises to be a part of that workout, as shown in Figure 6.

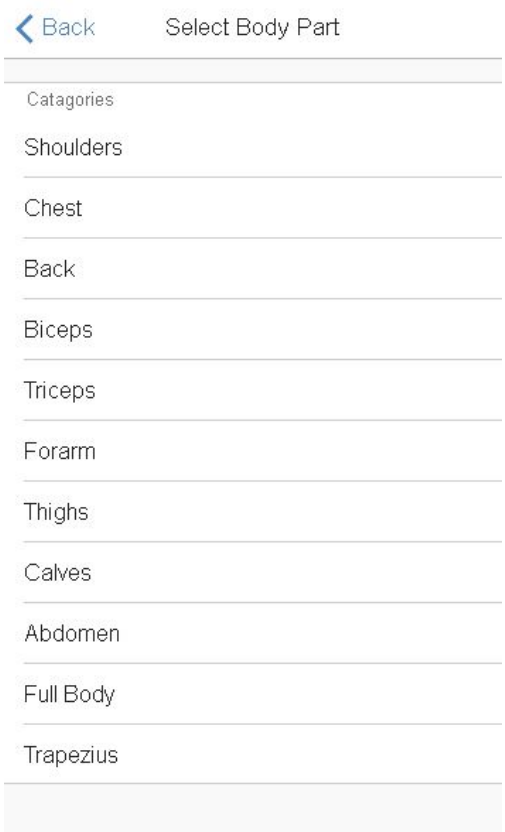
To create a new workout, the first the user must pick a name to identify their workout. This is done via prompt as shown in Figure 5.

Figure 5: Add new workout



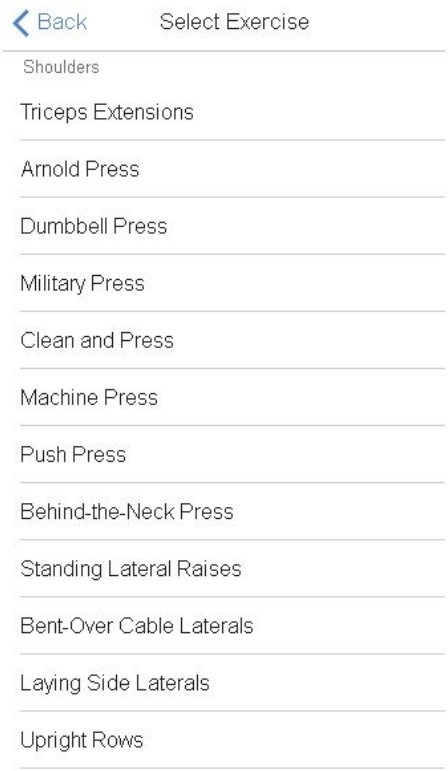
Clicking 'OK' then sends a POST request to the API to create a new workout entry. Then you are taken to a blank workout page where you can add exercises. Adding an exercise takes your first to a list of body parts which is populated with a GET request from the BodyParts route of the API, as shown in Figure 6:

Figure 6: Select Body Part



Clicking on a body part sends another GET request to the server and populates a list of exercises for that particular body part, as shown in Figure 7:

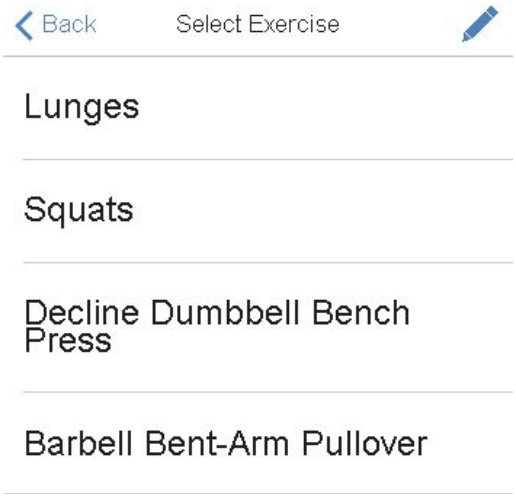
Figure 7: Select Exercise



Clicking on a particular exercise then adds that exercise to the workout. Or in more technical terms, it creates a new entry in the log table linking the workout ID to the exercise ID with the default row set to true. You can rinse and repeat for any other exercise you want to add to the workout.

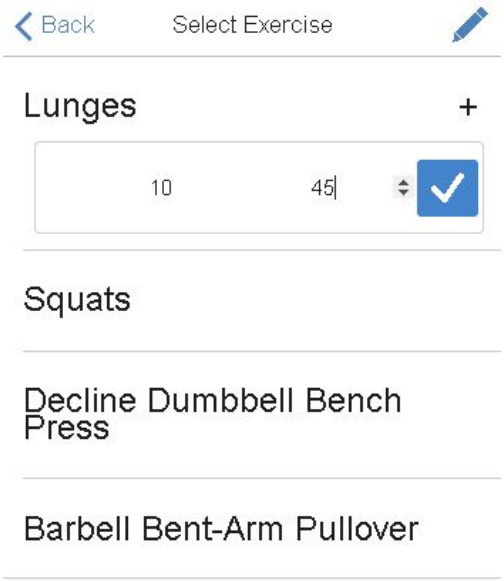
After you've created your workout with all of the exercises that you want within it, you can log an instance of that workout. Clicking on one of the workouts, as in Figure 4, will bring you to a dynamically populated list of exercises, as in Figure 8.

Figure 8: Exercise entries dynamically populated from backend



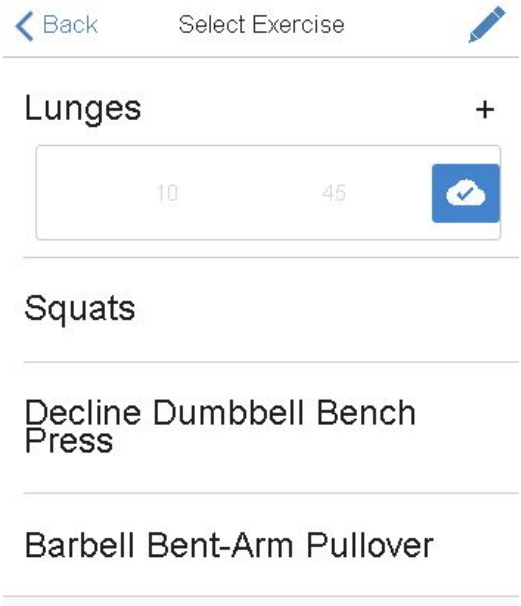
Clicking on an exercise adds a new set as shown in Figure 9.

Figure 9: Tap on an exercise to add a set



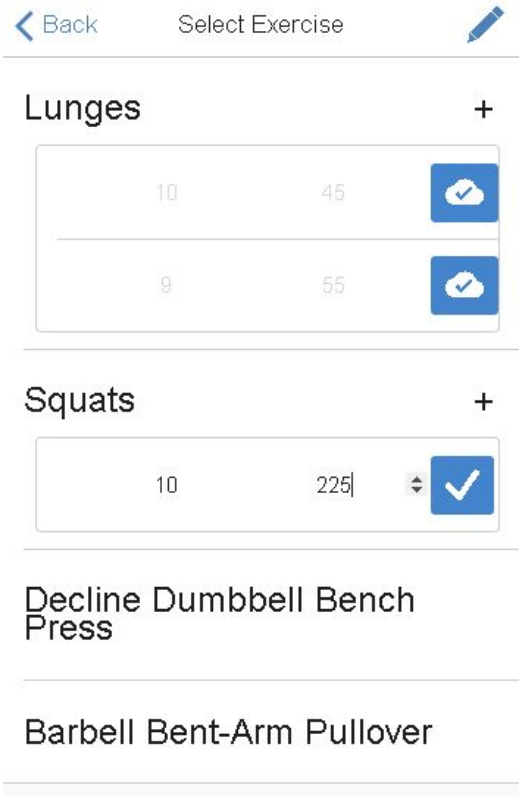
The user can then set the number of reps and the weight and then click the checkmark button to save the set, as shown in Figure 10.

Figure 10: Tap the checkmark button to save your set



Saving a set disables the two text inputs and then sends a POST request to the API which creates a new entry in the log table with the workoutID, exerciseID, reps, and weight. You cannot add a new set for the current exercise or more on to another exercise until you've saved the current set. However, if the the current set is saved, a new set can be added for that particular exercise by clicking the plus icon.

Figure 11: Click on the plus icon to add additional sets



After you're done with your workout and saved all your sets, you can return to the workouts page by clicking the back button.

Future Steps

There is still a lot of work to be done on this application. First off, I need to build out a way to review instances of logged workouts. I have yet to complete this feature because it is non-essential for my personal use as I have alternative access to logged information through the backend. However, if this application were to be released to the public, I would need a way to access such information through the front end. To do this, the front end will make a call to the existing backend to get all of the logged exercises sessions for that particular userID. Then it will create a list of each unique exercise that was

returned which, when clicked on, will show all logged instances of that exercise starting with the most recent. An eventual addition to the view will be a graph of how much weight was lifted over time. I plan to use a package called NVD3 to display the graphs.

Lastly, I need to publish my application to either an application store as a native Android or iOS application or to the Internet as a small screen focused web application. I started the process of the latter quite early on in my project by purchasing a domain name, renting some space on a linux based server, and using that as my development environment. The domain name I chose is wieghtlifter.com, a play on the misspelling of the word 'weight' and the three first letters of my name. Of course the proper spelling would be a better domain and wouldn't be pronounced as 'white lifter', but as that domain is already taken and this is only a personal application at the moment I decided to stick with that name. Perhaps one day there will be a whole suite of 'wie-' applications and the domain and application name would seem less offensive.

Bibliography

Android, and Apple, and Google, and Microsoft, and AppBrain, and BlackBerry, and Various sources (WindowsCentral.com, and International Games Week Berlin), and Amazon, and VentureBeat, and CNET. "Number of Apps Available in Leading App Stores as of March 2017." Statista - The Statistics Portal, Statista, www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, Accessed 12 Jan 2018

"API Reference." *Express.js,* https://expressjs.com/en/4x/api.html.

"Documentation." *Node.js*, www.nodejs.org/en/docs/.

Hanson, Jared. “Documentation.” *Passport.js*, www.passportjs.org/docs/.

Rockoff, Larry. *The Language of SQL*. Addison-Wesley, 2017.

TechGuyWeb. “Traversy Media.” YouTube, YouTube, www.youtube.com/channel/UC29ju8bIPH5as8OGnQzwJyA.