Mobile and Web Timecard

Final Report

sdmay18-14

Client
Genova Technologies

Advisor
Neil Gong

Team Members
Nicholas Flege - Lead iOS Engineer
Andrew Hoelscher - Lead Tester
Connor McCann - Lead Android Engineer
Thomas Reins - Lead Web Engineer
Cole Stephan - Lead Server Engineer
Jason Thomas - Team Administrative Lead
Christian Wesseler - Lead Designer/Documentation Keeper

sdmay18-14@iastate.edu
sdmay18-14.sd.ece.iastate.edu

Table of Contents

1 Introduction

1.1 Acknowledgement

Genova Technologies provided significant guidance in the form of technical advice, tools, and financial aid. Bi-weekly meetings were held to review the progress of our senior design team. Occasionally we met more frequently with our technical lead from Genova, Tom Sidebottom, to ask questions specific to our code development. Genova set the senior design team up with accounts with the Agile tool, Rally, that allowed us to document our progress in a uniform way. Genova committed up to $20,000 in financial support to aid in the completion of the project. We appreciate all the time, thought, and effort that Genova put into making our senior design project possible.

1.2 Problem and Project Statement

The problem we aimed to solve with this project was Genova Technology's need for a user-friendly and well-liked method to track time that individuals spend on projects. Genova previously had a timecard application which most employees enjoyed using. However, a change in their accounting software left their old timecard incompatible. Due to this, Genova was forced to go back to an old and disliked timecard application. Genova requested that our senior design team develop a new timecard system that is compatible with their new accounting software, user-friendly, and that runs via iOS, Android, and web applications.

We approached our project with industry standard software development tools and with Agile development practices that iterated every two weeks. The Agile iterations allowed us to prioritize Genova's highest needs first and to quickly adjust to design requirements as they changed throughout the year. Our strategy required less long term planning and meant we didn't have to try to predict every possible need, all at once.

The data for the timecard is stored, updated, and retrieved from a database hosted by Microsoft Azure. The backend codebase was built on a Microsoft application server and coded in C#. The frontend will be coded differently depending on whether it is the iOS (Swift), Android (Java), or web app (Javascript). Both the backend and frontend will be version controlled on a private Github repository to allow for group collaboration.

1.3 Operational Environment

Since this is a software project, the operating environment is fairly straightforward and we don't have to worry about any physical requirements. The iOS application will run on an Apple iOS device, the Android application will run on an Android device, and the web app will run on standard web browsers such as Chrome and Firefox.

1.4 Intended Users and uses

The end users for this product are the employees of Genova Technologies, and potentially the employees of any companies Genova sells the software to. The mobile and web timecard application must provide easy and user-friendly experiences while allowing the Genova employees

to quickly and efficiently track their time spent on different projects. Additionally, it must provide a simple way for the administrators to login to the application for review and approval of timecards. If a timecard is denied, there will be an option to add a note describing why it was denied, and the timecard will go back to the employee to be changed and resubmitted. Once timecards are approved, the data must be exported to the compatible accounting software so Genova's customers can be properly billed for their time.

1.5 Assumptions and Limitations

Assumptions:

- Application will only be used by Genova employees and the companies Genova may potentially sell the software to; therefore the maximum number of concurrent connections will be a limited number of employees and consultants
- Users will not access time-card application from multiple platforms (on the same account) concurrently
- Multiple language support not necessary
- Voiceover support not necessary
- Application will not be used outside the United States

Limitations:

- Time-card application will have minimal feature parity with old system
- Time-card application must run natively on iOS and Android operating systems
- Server costs will be covered by client

1.6 Expected End Product and Deliverables

The end products are iOS, Android, and web applications that store, update, and retrieve information from a database hosted by Microsoft Azure. These applications provide easy, user-friendly ways for Genova's employees to track their time spent on different projects. Once a timecard is completed, the administrator will review it and have the option to approve or deny it. Once approved, the timecard data will be sent to the accounting software so that the customer/client can be properly billed.

2 Specifications and Analysis

2.1 Design Specifications

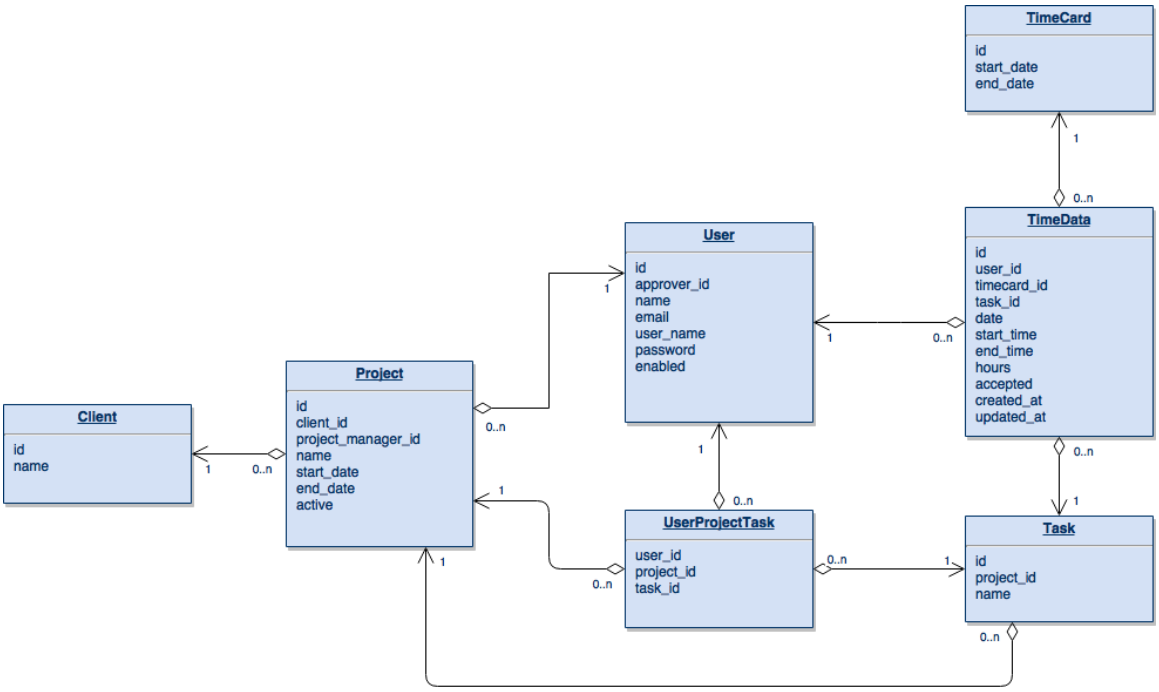
- Mobile Timecard app will function on iOS and Android devices
- Web version will support Internet Explorer, Edge, Chrome, Firefox, and Safari browsers
- Timecard backend interacts with Dynamics, Genova's accounting system
- Web and Mobile Specifications
    - User can log in with their Genova email address and password

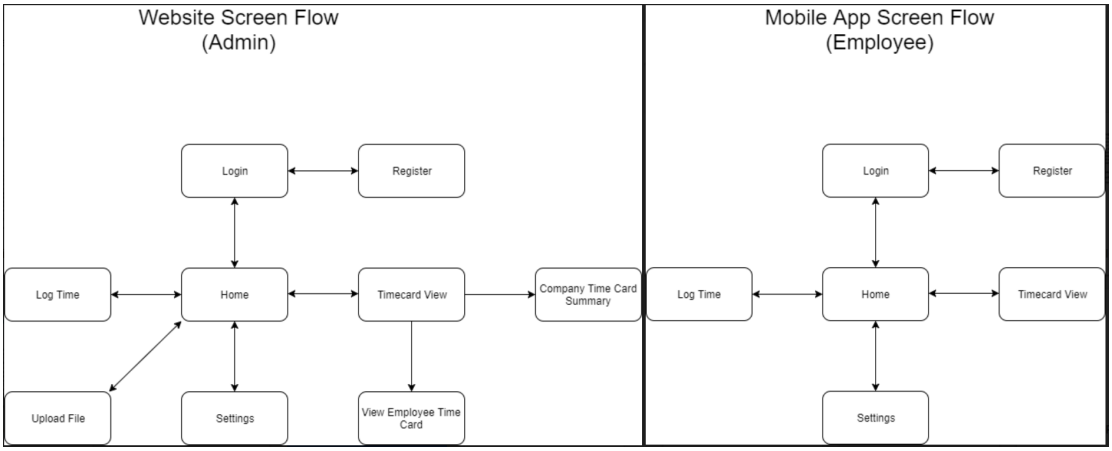
    - New users can can create accounts and passwords with their Genova email addresses
    - Mobile devices will automatically store user credentials upon first login if desired
    - Users can create a new timecard, limited to one per pay period
    - Users can record time worked for a specific client, project, and task
    - Users can record time in both block and start/end time formats
    - Users can complete and submit timecard to manager for approval
    - Users can view reports of previously submitted timecards (year-to-date or last five weeks, whichever is greater)
    - Users can log out of timecard
- Web-specific Specifications
    - Managers/Reviews can review and approve their direct reports' timecards
    - Admin can view summaries of all users in timecard system
    - Admin can create timecard for another user
    - Admin can edit/complete timecard for another user
    - Admin can submit timecards for given pay period to Dynamics accounting system

2.2 Proposed Design

The web and mobile timecard is implemented using a client-server application architecture. The application server provides the clients (Web, iOS, Android) with the data through a RESTful API. The application server communicates with the database to retrieve the data requested from the clients. The server is implemented with a Microsoft application server deployed on Microsoft Azure and a Microsoft SQL server which is also hosted on Azure. The clients make requests to the server for specific data using HTTP requests following the RESTful protocol. The data is returned to the clients in the form of JSON which is then parsed into a usable form and presented to the end user.

The database was designed following the requirements and specifications provided to us by Genova. The TimeData table is used to store a time entry for a specific user, task, and timecard. The TimeCard table is used to track the beginning and end dates for a specific pay period. The UserProjectTask table is used to manage the relationships between the three respective tables. Users can be assigned to multiple projects and each project has a number of tasks related to it, but there are some tasks on various projects that should not be available to each user. The User, Client, Project, and Task tables are used to store various properties for each. The diagram below shows fields and relationships between the tables.





These diagrams outline the screen flows for the two variations of the client apps. The mobile app flow is on the right and the web app flow on the left. Both the mobile and web apps will have features allowing the users to create timecards, log times for a specific client, project, and task, and view previously submitted timecards. The web app will have some more advanced features for managers, who are reviewing the timecards submitted by their direct reports, as well as admins, who will be able to see overall summary reports of all employees in the company and export timecard data to their accounting systems.

2.3 Design Analysis

The alternatives to this design were to use a Linux server and along with an SQL or NoSQL database. The server and database could have also been hosted with another cloud platform such as Amazon Web Services. Technologies exist to create mobile apps in a single programming language that can then be compiled into separate iOS and Android applications. A specific requirement from Genova was to build both native mobile apps, and this prevented us from

implementing such technologies. We have also found through previous experience that these technologies can be very hard to debug and learn if not already familiar. Members on the team also have previous experience with native iOS and Android development, so we were confident in being able to build separate, native mobile applications.

The advantage of using a Microsoft stack for the development of this app was its integration with other Genova systems. Genova runs Microsoft software to manage their employees and finances. Since we used a Microsoft stack, we determined using Microsoft Azure for our cloud service was the most advantageous. It has optimization for .NET web applications and integrations with Visual Studio that make deployment simple and straightforward.

We also needed to take into account that iOS development requires a Mac computer. A few team members have Macs, so they will be able to do a majority of the iOS development and there were resources available to use around campus (rentals or computer labs) in the event we need access to more Mac computers. Also, while Visual Studio was recently made available on Mac by Microsoft, many of the most useful features are only available on Windows. We are able to avoid this issue by using a virtual machine running Windows and Visual Studio. This is also an advantage for debugging/testing the mobile applications, specifically iOS, during development. For example, a local version of the server can run on a windows vm while the iOS app is running in a simulator on macOS. The iOS app can then make local API calls instead of having to deploy the server to Azure after any change is made in order to test it with the mobile applications.

2.4 Previous Work and Literature

There are hundreds of similar timecard systems that companies use. Genova had a previous system, but due to software updates it was no longer compatible. They provided the design notes on source material of their previous timecard system to help guide us. There are other options available to Genova, but they would rather have their own in-house software that they can control more and change as they please. Because of this, other software programs are inherently less efficient for them.

Some disadvantages of Genova's current timecard solution is that it is not liked by most employees and is less intuitive than their previous systems. One of the first things Genova emphasized when we began meeting with them was how well-liked the previously used timecard was. The timecard they are currently using has received wide and loud negative feedback, this is another thing Genova emphasized. They also do not have much control over their existing solution since they did not develop it themselves. The other disadvantage of the current system is that it is only accessible via the web.

Our solution is a mobile-first design approach that improves ease of access and user experience. In addition to being focused on the mobile aspect of the timecard we are developing, we are using the documentation which Genova has provided us with on the well-liked timecard they previously had. By using this documentation to guide our development we will be able to create a solution which will be more user-friendly and cause fewer complaints from those who use the timecard every day, Genova's employees.

Other applications exist to manage and track time such as Tsheets, Zenefits, or Timesheets. These applications provide subscription services that companies and pay for to provide employees with a web and mobile app to track their hours. A disadvantage of these apps is that they do not integrate with Genova's existing accounting systems. The apps also do not allow for Genova to host the required servers and databases themselves.

3. Implementation Details

3.1 Backend API

The backend was written in C# in Visual Studio. The software architecture is based on the MVC model, using models that have fields and values match their database counterparts and controllers that are the API endpoints. Requests to the endpoints are made using HTTP and JSON tokens, which are read and interpreted on the backend and make SQL requests to the database when necessary. All endpoints except for login and user registration require authentication to use, which we implemented using JSON Web Tokens. The web tokens are given to clients upon a successful login, and are currently set to last for 6 hours.

There are a number of endpoints currently implemented to serve a variety of functions. Users can login or register new accounts, which must be a valid Genova email, pull all projects, tasks, and clients, as well as get past timecard data for themselves. They can also add new time data for themselves to the database.

3.2 Web

The web application is hosted on a Microsoft Azure virtual machine which runs an Nginx web server. The web application was written in Javascript using the Bootstrap framework and communicates with the server using jQuery ajax calls. Once logged in the server sends a JSON web token and uses this web token for authentication purposes as the user navigates the site.

The web application portion of the timecard app was quite frustrating to develop at the beginning. Our original plan was to use .NET Core web api in Visual Studio. None of our team members had prior experience with .NET Core, and we found it very difficult to learn and move forward at a reasonable pace. Because of this we decided fairly late in the semester to switch to the Javascript and Bootstrap web application. We also had no prior experience using this technology, but we found it much simpler and easy to understand and were able to start moving quickly on the development of the web app.

3.3 iOS

The iOS application was created with the Swift programming language with Xcode. The app uses an Model-View-Controller (MVC) architecture. Views are created using Apple's Interface Builder within Xcode and linked to the data model using view controller classes.

The data model includes Clients, Projects, Tasks, Timecards, and TimeData. The model data is pulled into the iOS application and synchronized using a REST API. All requests to the API are authenticated with JSON Web Tokens. When the user signs in, the post a request to the API. If the login information is correct, the app will receive a token back that needs to be used to authenticate each request. When the app receives the token, it is stored in Apple's encrypted Keychain. The app will keep the user signed in as long as they have an API token that is not expired. If the app does not have a token or the token is expired, the user will be forced to sign in again.

The networking component used to handle the requests to the API uses the Singleton design pattern. This is done to ensure there is always only a single instance of the network component, and it makes the authentication process for requests simpler since they all flow through a single object. Data pulled from the API is stored locally on the iOS device. View controller classes are used to link the model and views and present the data to the user.

3.4 Android

The android application was written in Java using an Android Studio environment. The Android client utilizes the model-view-controller architecture to standardize the functionality with other components of the project. Views were created in XML using Android Studio and referenced in the corresponding code in standard Android Studio Java format.

By making RESTful API calls, any requested data is received in JSON format. Login authentication is processed in a similar manner and is passed through JSON bearer tokens after authentication is verified by the server. Communications to the server are passed through POST requests to the corresponding API.

4. Testing Procedures

4.1 Backend API

The backend API endpoints were tested manually using Postman. HTTP requests were formed by Postman that would be sent to the test server, and the results would be viewed as either a HTTP response or as a change to the database. After initial testing for correct input, we then tested edge cases for the endpoints to check the backend error checking was correct and would return the correct error message.

4.2 Web

We tested the interface of the web application by simply opening the app in a web browser and testing the functionalities. To make sure we formed our HTTP requests properly and to troubleshoot any issues we were having with the data we received we used Postman.

4.3 iOS

UI tests were created for testing login, register, and submitting time. This was done using Xcode's built in support for UI testing in Swift. Postman is a tool for making REST API calls. It can also be used to mock endpoints. If endpoints had not been created yet, Postman was used to mock the endpoint, so the iOS clients could be tested. In cases where further debugging was needed, the backend API would be run locally on the same machine as the iOS application, so breakpoints could be used on both sides of the application to trace and walk the flow of data step by step.

4.4 Android

Development of the mobile platforms began before the server was fully set up and required the implementation of mocks to simulate server responses. Although the mocks had to be configured for each system under test, they were treated like black boxes for test implementation. Testing was done manually to verify non-functional requirements of the program's performance. Standard unit testing proved insufficient for catching errors in the GUI or responsiveness of the client. Postman was also used throughout the design process as a tool to test API calls properly before implementing them into the project.

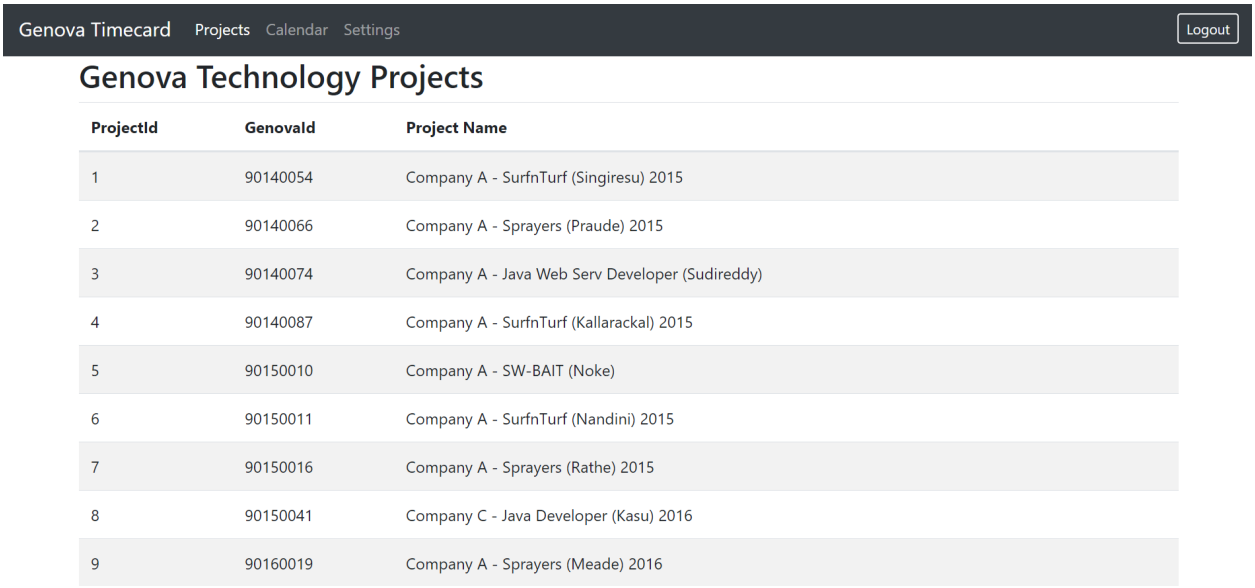
Appendix 1: Operation Manual

Web

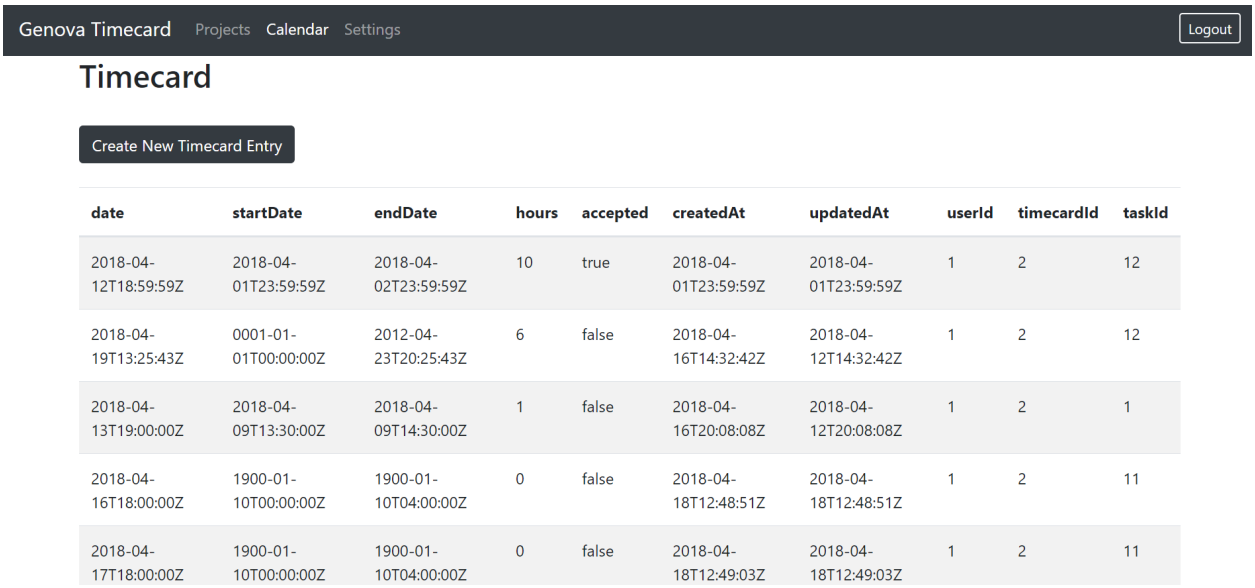
The web link to the web application is: http://genovatimecard.centralus.cloudapp.azure.com/

1. Login/Register page, from this page you can select either option and text boxes requesting the required information to either login or register will appear.

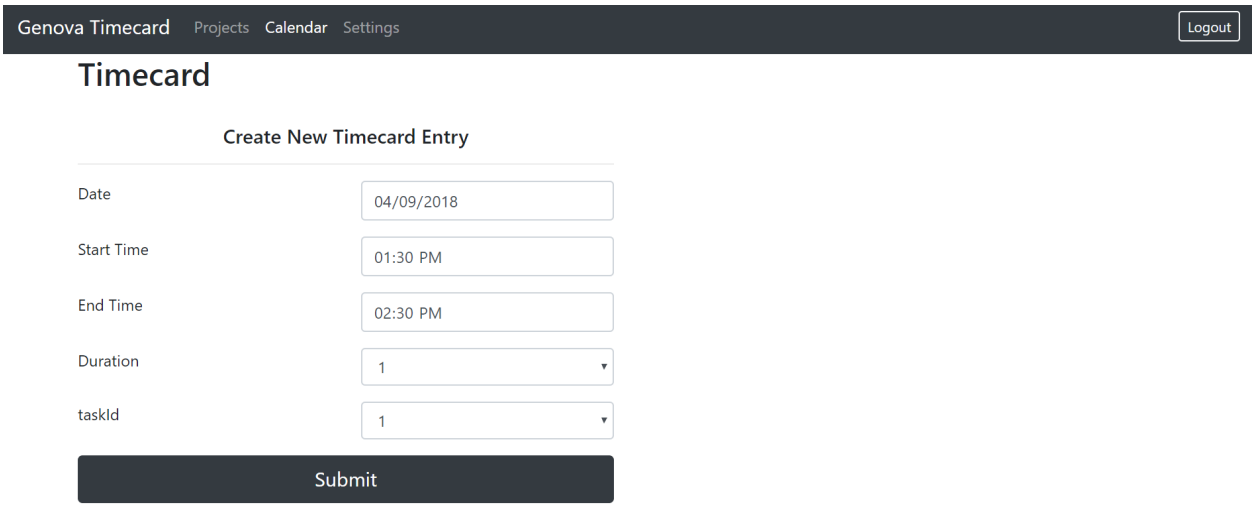
2. Projects page, once logged in the projects page is displayed and all of Genova's projects are listed in a table (dummy data shown in the screenshot)



3. Log time page, selecting the Calendar option will move you to the page below. On this page the number of hours for the logged in user is displayed (data for test user shown here). By selecting the "Create New Timecard Entry" option you can log time.



4. Log time, on this page the user will log the time they have spent on their projects and submit it. Once they have submitted their time it will be displayed in the table.



5. Selecting logout will bring the user back to the Login/Register page

Appendix 2: Alternative/other initial versions of the design

When beginning Android development there was discussion of using Kotlin instead of Java. One of our consultants at Genova recommended it as a new option to Android Studio if we thought it would be easier, but the entire team decided it would not be worth the switch. The support was there as an option but none of us knew Kotlin and while it is similar to Java it offered no real advantages over Java itself. Because of this we stuck to what we knew and went ahead with Java development.

On the website we began development using ASP .NET core. The thought was that it would mesh well with the Microsoft based backend and the Microsoft Azure cloud hosting we used. However, after slow progress and finding it to be overkill for our purposes, we made the decision to switch. The final website uses Jquery, JavaScript, and Bootstrap. The time it took us to get the site running and interacting with the backend greatly reduced. It was such a good call that, in fact, with less time using this web development strategy, we accomplished significantly more than the previous method. In addition, the documentation we were able to find online was much better and lead to more bugs being able to be fixed. Overall, this shift was very positive for our project.