Senior Project Final Report

Quentin Barnes and Ty Vredeveld

Honors Project for Ty Vredeveld

Project Vision and Overview

For our project, we sought to design an easy-to-use mobile application for users to access data and error codes generated by their vehicles. The ability to access data that vehicles generate is not a new one, but we wanted to simplify this process for users that don't need all of the overhead that some apps on the market generate. A small percentage of the people who use their vehicles are familiar with the data systems present within the vehicle, and a smaller percentage of those people actually do something meaningful with the data that the vehicle generates. Because of this, our goal was to make our application as user-friendly as possible so as to incentivize more drivers to learn about their vehicles and have access to the data that they generate. We stripped away much of the noise and focused on a smaller number of core areas where users could get a taste of the possibilities of this interface.

On top of the OBD interface, we wanted to create a place for users to keep track of maintenance that is done on the vehicle. There is a section within the app for recording this information - various tasks like oil changes and tire rotations are set up already as categories for users to enter events with. Each event is given a date when it was performed and the cost, so that this information can be tracked over time. These records and anything else relevant to a specific vehicle are all related to each other via the use of car 'profiles'. These are back-end constructs designed to make the data shown to the user only relevant for the current car that the OBD scanner is connected to. This was done to ensure consistency and avoid confusion when viewing the data on the application.

In addition to these features, Ty completed a portion of this project for honors credit. This revolved around the idea of visualizing fuel economy data. Some modern vehicles have the ability to track and record your fuel economy data over time. However, this is normally stored as a simple number and doesn't mean a whole lot when daily driving is examined. Because of this, I wanted to give users the ability to see their fuel economy and how it changes depending on different locations and patterns in driving. In doing so, the goal was to make users more aware of their driving habits and able to make changes to them in order to maximize their fuel economy - making an impact both financially and environmentally.

Background

This mobile application is based around the On-Board Diagnostics (OBD) systems integrated with modern vehicles. Beginning in 1996, the OBD-II specification was made standard for all vehicles produced in the United States. Europe and Australia/New Zealand followed shortly thereafter, implementing the specification in 2001 and 2006 respectively. This standard gives vehicles reporting capabilities. The most common use for OBD is in determining the cause of a check-engine light turning on. When the OBD spec became standard, devices were created that allowed technicians to diagnose trouble codes that vehicles were generating and recommend repairs accordingly. Not all of these trouble codes are perfect, though. For example, there's one

that signals an oxygen sensor is broken, but it doesn't specify which specific sensor is damaged - just the fact that one is.

Issues with trouble codes aside, OBD is a very powerful tool for those who know how to use it. Eventually, people started to realize that they could access the data that the vehicle is generating and using to detect its own errors. Information like engine RPM, air flow rates, and more could be made available to programmers that desired to use this data. These programmers and engineers designed devices known as OBD-II scan tools that allowed users to receive the same messages that get passed throughout the vehicle on its CAN (Control Area Network) bus.

These messages are queried using a series of PIDs (Parameter IDs) that are used to request data from the vehicle. These PIDs are divided into various services that all have different primary uses. For our application, we're primarily dealing with services 01 and 09. Service 01 is the location of all PIDs for requesting current real-time data, and service 09 is used for requesting vehicle information - data like the VIN (Vehicle Identification Number) of a vehicle.

OBD-II scanners eventually made the jump to Bluetooth. This is the innovation that makes our project possible. By interfacing with a scanner over Bluetooth, we were able to design and implement a mobile application to present this data to users that may not be completely familiar with the OBD standard. For our project, we purchased and used an OBD-II scanner that uses an ELM327 chip - a standard microcontroller used for translating and abstracting the low-level

OBD protocol into something usable by hand-held diagnostic tools or other PC/mobile applications.

System Design and Implementation

Our application was designed using the Ionic Framework. Ionic is a cross-platform development framework similar to Flutter or PhoneGap. Ionic is web-based and is based around principles of web development/design. The main logic of the application is written using TypeScript, and HTML/CSS are used to provide structure and style to the pages that are designed and implemented.

Ionic has three primary draws to its platform. The first of these is the use of custom, Ionic-made UI components. What makes these components unique is that they adapt and display differently depending on the platform on which your application is running. We're all aware of the visual differences between an app designed for Android and one designed for iOS. They have distinct styles and stand out from each other when you compare them side-by-side. When developing with Ionic, you only need to worry about using a set of Ionic UI components because they automatically adapt to the platform and display in the correct layout/theme accordingly.

Another draw to the Ionic framework is its ability to access native device functionality. Features like the camera, geolocation, local storage, and more are readily available through Ionic native device plugins. This gives programmers full access to native device functionality without

needing to dive into the full native SDK - however, that option is still available to those that desire it.

Finally, a third draw to Ionic is the sheer number of plugins you can integrate with your application. In our case, we're integrating with a number of official and community-made plugins. The Google Maps JavaScript API is used for Ty's honors page. Another plugin is used to grab stock images of vehicles based on their VINs for displaying on the home page, and yet another is used to decode a vehicle's VIN into meaningful information. These examples are just the tip of the iceberg when it comes to integrations that Ionic supports. Firebase, Apple Pay, Instagram, AWS, and more are all ready and able to be integrated into Ionic apps.

In using Ionic, we designed our app into various pages that use services for handling the data the application generates. Services are singletons that do various bits of work for your application. For example, our app has a service to handle maintenance record storage and another for connecting to/interfacing with the OBD scanner. In using services, we're able to leave most of the data processing out of the logic for the various pages we create.

Security

We want to provide a brief note on the security considerations of our application. There are 3 primary categories of OBD tools. The first is a reader that reads all hex codes traveling over the bus. The second is a scanner - what we're using. A scanner queries the car (the main

computer/hub), and it responds with the requested data. It is read-only (except for requests that get sent for data), and anything that isn't formatted as a valid request is dropped by the chip. These types of OBD devices can still be sniffed. However, there isn't any data that can be sniffed that you couldn't get by looking at the car or through its windows. There is an occasional report of finding a back door in specific devices. However, it has so far been isolated to specific device lines and not the primary OBD scanner chip (ELM327). Lastly, for anyone to even be able to interface with these devices, it would need to be either through Bluetooth or a physical wire, and the vehicle would have to be powered on. This imposes physical range limits. On top of that, Bluetooth requires a password (though manufacturers usually make it 1234 or something similarly easy). The last type of device is an OBD spoofer. This is not the technical name for it, but these aren't widely used because of obvious security risks. These are tools that allow you to actually control and manipulate your car. They pretend to be a part of your car and can be used to splice and add data to the bus.

If someone truly wanted to hack your car, they most likely wouldn't do it through the OBD port. Most cars have much greater security risks than the OBD port, since the system as a whole wasn't designed with security in mind. There are examples of cars being controlled over cellular from anywhere in the world. There are readily available wires that anyone can access on a car where all you have to do is connect a device smaller than the palm of your hand to be able to control the car. There are much simpler ways to hack a car than through an OBD scanner. The good news is that car manufacturers are starting to make security more of a priority, and currently, there are much cheaper ways to steal cars than by hacking them. Because of this, the

security risk that our app provides, by using an OBD scanner, is extremely small when compared to the risks that are already present in a car. We are also using proper data hiding methods in our app to keep it as secure as possible. While we do admit that the security isn't perfect, in our opinion the blame falls primarily on the car manufacturers for their lack of concern when it comes to these preexisting security vulnerabilities.

Testing and Results

For most of the first semester of the project, our goal was to get our app created with basic pages and be able to interface, however bluntly, with the OBD scanner. After the first few months of work, we had managed to achieve our goal, and we had an app that could retrieve the VIN from a car by communicating with the OBD scanner we had purchased.

During the second semester, we focused much more on optimizing and cleaning up our implementation of the OBD connectivity service as well as adding additional functionality like the fuel economy visualization. In coming to the end of our time working on the project, we accomplished most all of what we set out to do. Our application cleanly interfaces with the OBD scanner over a native Bluetooth connection. With this connection, various PIDs are used to request data such as air flow rates (for calculating fuel economy), VINs, error codes, and more. All data that is captured is linked together via the use of car 'profiles' to ensure data is not shared between vehicles, and users can see the most accurate picture of the data for the vehicle they're currently connected to.

Conclusions

In conclusion, we believe we have successfully designed and developed our project. We set out to create a simple mobile app to interface with a vehicle's OBD system, and our app delivers on that front. We've also added more features like maintenance record storage and fuel economy visualization that encourage users to learn more about their vehicles and driving habits. Overall, this project was very rewarding to work on and gave us both some very valuable experience. Mobile app development is only going to become a more highly requested commodity, so the experience we gained through this project is of great value.

Future Work

There are several things that we would have liked to have implemented but didn't have the time to. The first, and primary feature that we would have liked to pursue is turning our OBD connection service into an Ionic plugin. This would've been our contribution to the Ionic community. Plugins give developers a way to have easy access to new features and tools without needing to perform much of the complex development themselves. In our case, programmers would be able to install an OBD-Connection plugin that gives them full access to the OBD system over a Bluetooth connection. However, like other features, this got pushed to the side in favor of more functionality for our app.

We also would have liked to have made the service more robust and handle a larger variety of functions/data. We're currently supporting information like VIN and error codes, but there are so many more points of data we could explore and use - things like engine temperature or RPMs to name a couple. Lastly, for Ty's honors project, there are probably more accurate ways of calculating and displaying fuel economy. Currently, we're performing a lot of arithmetic for each point of data generated, but there is likely a better way to uncover this information and better ways to determine how to display it on the visualization map.

How to Reproduce Work

To prepare your machine to build this app, you will need a few pieces of software. First, you will need to install Node.js and NPM. A quick guide for doing this can be found here. After those are installed, you will need to install the Ionic Framework. A quick guide for that can be found here. Lastly, you will need to download either Android Studio or Xcode. Currently, this app is only officially supported for Android. Some of the Bluetooth functions may need tweaking in order for it to work on iOS devices. A quick guide for downloading Android Studio can be found here. Now that NPM, Ionic, and Android Studio are installed, you can clone the git repository to your computer or download it via the zip file.

After you have set up your machine, you may need to set up several API keys. This app uses three APIs. At the time of writing, the app has working API keys, but this may not always be the

case. I will quickly walk you through how to remake these API keys if it is needed. Some links that will help with this are:

- https://developers.google.com/maps/documentation/embed/get-api-key
- https://cloud.ibm.com/docs/services/HellaVentures?topic=HellaVentures-gettingstarted_HellaVentures
- https://www.carmd.com/api/

For a full guide on how to generate these keys, please go to the readme section on our GitHub page.

To run this code, you should open up the terminal that NPM is installed on, and run the following commands. First, you should run ‘npm install’. This will install all the NPM packages used in the program. After that run is finished, run ‘ionic build’. The next steps may change depending on the device you're building for. Wherever your see [device], replace it with android or ios depending on the platform you're looking to build for. You now should run ‘npx cap update [device]’ and then ‘npx cap sync [device]’. If you ever build this app again, all you will need to run is ‘npx cap copy [device]’ for this step. The final command is ‘npx cap open [device]’, which should open the app in either Android Studio or XCode for you to run and test it.

This is where we should note that we had a dependency issue at one point that was fixed by running ‘npx npm-force-resolutions’ and then following all the above steps again. We should also note here that you can run a version of the app in the browser that looks correct but doesn't have many of the functions of the app by using the command ‘ionic serve’.

Acknowledgements

We'd like to thank Professor Victor Norman for his assistance with this project. The guidance and assistance he gave us throughout the whole year have been much appreciated. His prior experience working with the Ionic Framework was also extremely valuable, as he helped us work through numerous issues we ran into while learning the framework ourselves.

Josh Morony was also very helpful through his book and tutorials we utilized. These gave us a great starting point for learning Ionic and helped us progress through the early stages of the project quickly.