CarWiz

by

Austin Brown, Steven Glanz, Harrison Than-Win, and Trevin Wolfe

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

Austin Brown _____

Austin Brown April 12th, 2021

Steven Glanz _____

Steven Glanz April 12th, 2021

Harrison Than-win _____

Harrison Than-Win April 12th, 2021

Trevin Wolfe _____

Trevin Wolfe April 12th, 2021

Ryan Moore _____

Ryan Moore, Faculty Advisor April 12th, 2021

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

November 2020

Table of Contents

List of Illustrations

Tables:

Figures:

Abstract

CarWiz is the all-in-one tool car owners need to keep their car maintained. This android app aims to help its users eliminate the hassle of remembering when they need have their car serviced and what servicing had been done in the past. It will provide service tracking, upcoming maintenance reminders, and video tutorials for DIY maintenance. There is currently no solution that provides all three of these functions in one. Users can create an account to store their car maintenance history into our database which can be accessed from any other android device. We predict that 80% of users will have a better maintained vehicle by utilizing the application.

Introduction

Problem:

One of the best ways to prevent a car breaking down is routine maintenance. However, it is often overlooked since it is difficult to keep track of. Websites like NextEgg and companies like AAA have reported on how unprepared car owners are for routine maintenance[1,2] . Owners must keep record of each service they have completed and remember which ones they need to get completed at upcoming mileage intervals. Scheduling maintenance can also be intimidating, especially for those who are not knowledgeable with char mechanics. On top of all that, it can be expensive, and some owners do not know that they can do basic maintenance themselves. Other apps like *CarFax Car Care*, *AUTOsist*, and *Vehicle Maintenance Tracker* each provide fragmented functionality to solve these problems. For example, CarFax's app only tracks a fraction of necessary maintenance, oil changes and tire tread life. None of them offer a comprehensive solution.

Solution:

CarWiz is a mobile application that will allow users to enter their vehicle information, track past maintenance, reminders of future maintenance, and references to instructional guides. The application will also optionally allow users to connect to OBD2 scanners to their car's diagnostic port. This enables the app to track mileage in real time and provide advanced maintenance information by linking with the scanners to read detailed information about the user's vehicle. From the app, users will be able to schedule

an appointment with a participating mechanic, track the mileage and maintenance history, see estimated expenses, and track their maintenance expenses over the life of the car.

Project Goals / Brief Methodology:

CarWiz was designed to make car maintainance simple and easy by removing the complication of knowing what maintenance needs to be performed and when to do it. The app will alleviate as much work from the user by calculating the mileage of past maintenance and notifying when to schedule upcoming service.

Overview:

This report covers the process of how the CarWiz application was created from the original framework to the final design. Included are the project concept, design objectives, methodology and technical approach, user profiles, use case diagrams, testing, and even budget explanations. This covers all the project goals and deliverables that are required.

Discussion

Project Concept:

CarWiz was inspired by the project team members' own personal struggles with car maintenance. None of them had a solid way of tracking their maintenance and remembering to schedule upcoming services. Old-fashioned pen and paper or the sticker oil change shops put on the windshield were their current methods of service tracking.

They knew there had to be a more technological and modern way to do this. So, they sought to migrate this process to the device everyone has in their pocket: the smartphone.

Design Objectives:

The goals for this app were seven main features: routine maintenance tracking, mileage tracking, cloud storage, OBD2 scanner link, YouTube links for DIY maintenance, service appointment scheduling, and maintenance pricing (historical and projected). The team implemented the first three features and decided to push back the other features to next semester due to time constraints form unexpected development issues. The app also needed to be simple and like other android apps so that it would be accessible to the widest possible audience. The target audience was anyone who owns a vehicle, which in the US, is almost every adult.

Methodology and Technical Approach:

The android platform was perfect for the concept because android is the most popular smartphone platform and almost everyone has a smartphone. This was shown on www.gs.statcounter.com where it stated that Android holds 71.18% market share in the world. They were also ideal because their small form design makes it easy to enter data inside your car, as opposed to a big, clunky laptop. The team used the Android Studio IDE with Java and Kotlin as their languages. Kotlin allowed them to develop the functional side as well as the visual side.

To bring this application to the cloud, they used a MySQL database to store user and car data. The database stores basic information about the user such as first & last name, email, password, and information about the user's vehicle: year, make, model, sub

model, trim, and mileage. They plan on integrating more data into the database, such as other car part ages and past service dates. All this data is tied to a specific user account, which can be accessed from multiple android devices.

User Profile:

Mechanics will use the app to enter the services they completed into user's apps. They are benefited because they will get more business because of car owners remembering to bring in their vehicles.

Table 1: User Profile

User Profile Form 1
Application: CarWiz app
Potential Users: Mechanics
Software and Interface Experience: They will need to be familiar with Android devices and a general understanding of apps.
Experience with Similar Applications: Mechanic POS system, AndroidOS, Basic Android Apps
Task Experience: Basic Android application experience needed. Must have the ability to enter information into a field and navigate a basic menu bar.
Frequency of Use: They will need use the application when entering services into customer's accounts as well as referencing their accounts. They will also need to understand basic functionality to inform and demonstrate the app to their customers. This will be utilized as often as they have car repairs.
Key Interface Design Requirements that the Profile Suggests: A simple user interface is necessary for the mechanic to demonstrate to customer's its usability. If it is too complicated, the customer will lose interest. It also needs to be fast to enter in a service so that the mechanic can save time for the customer.

Forgetful car owners will use this app to be reminded to get their car serviced. They will need to be familiar with android smartphones.

Table 2: User Profile #2

User Profile Form 2
Application: CarWiz app
Potential Users: Forgetful Car Owners
Software and Interface Experience: They will need experience with Android and its basic functionality. We are assuming if they possess an android smartphone, they will be able to use the app.
Experience with Similar Applications: Android OS, Text messaging, settings app, Google play store.
Task Experience: Basic android navigation skills are required. They will need to know how to tap on elements and type in information into text boxes.
Frequency of Use: They will use the app after each maintenance is performed. They will also receive notifications reminding them they have an upcoming service. This app will most likely be used a handful of times a year.
Key Interface Design Requirements that the Profile Suggests: The app needs to be simple enough for a user to be able quickly pick it back up after not touching it for months. The design needs to be clear and straightforward with as little fluff as possible.

Car owners who are looking to save money will use this app to figure out what services they can complete themselves. The embedded YouTube tutorials will teach them how to complete certain maintenance tasks. This saves them money since they will not have to pay a mechanic for labor.

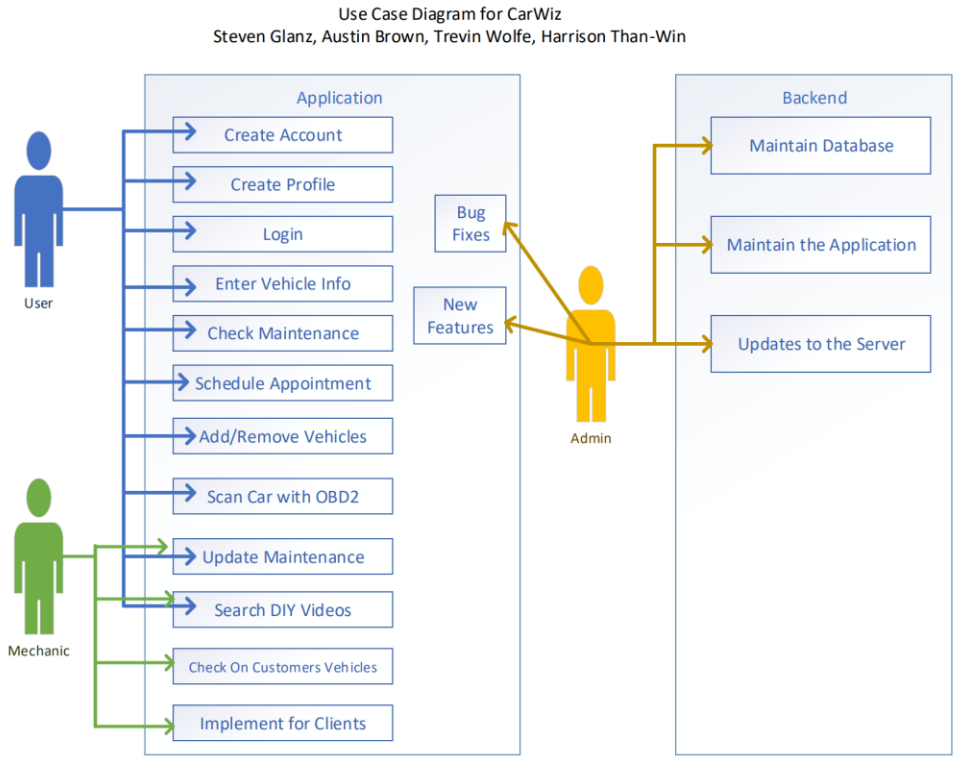
Table 3: User Profile #3

User Profile Form 3
Application: CarWiz app
Potential Users: DIY Car Owners

Software and Interface Experience: They will need experience with Android and its basic functionality. We are assuming if they possess an android smartphone, they will be able to use the app.
Experience with Similar Applications: Android OS, Text messaging, settings app, Google play store, YouTube & other video streaming platforms.
Task Experience: Basic android navigation skills are required. They will need to know how to tap on elements and type in information into text boxes.
Frequency of Use: They will use the app a little more frequently then the average user. They will need to refer to it multiple times to check the video tutorial.
Key Interface Design Requirements that the Profile Suggests: The video must be easy to access from the maintenance reminder screen. It would be frustrating for the user to navigate through many pages to return to the video.

Use case diagram:

Figure 1: Use Case Diagram



Technical Discussion:

CarWiz's foundation and core functionality is built using Android Studio for android devices. The backend of the foundation was written in Java while the front-end layout files were written in XML. On the main dashboard the user can view current mileage, view maintenance that is needed, and enter an updated mileage. The user's mile is saved securely in the database along with the vehicle information and their log in credentials. For that stats page, an API, MP Android Chart and Graph View were used to create and display a graph of the user mileage. The maintenance page displays maintenance percentages based on the user's mileage. These are calculated in the app using the most current user's mileage. Right now, most intervals are set at a generic interval, but in the future, they could be manually entered or pulled from manufacture recommendations. The app communicates with the database via an API and there is also backup PHP code that can run if the connection fails. The user's log in credentials is stored securely in the database through encryption. This is the only sensitive data that this app uses. All other information relates to the user's vehicle which is also taken care of securely. There are also checks when the user creates an account to meet certain requirements. When the user enters a mileage, the app checks both when the user enters it and when it tries to send it to the database to prevent an absurdly large number crashing the app or the database.

Testing:

Testing Overview

The testing plan for CarWiz splits the application into 3 distinct modules:

- Application Code

- Database/Web Connectivity Code
- User Interface Code

As each module is, essentially, independent of the others they can be tested separately. The Testing Methodology section details the approach taken for each module and the Scope of Testing section breaks down each module further into submodules specific to the testing plan. The Objectives section defines the testing strategy, what is being tested, and the expected end result. Finally, the Test Logs and Procedures and Review sections provide the testing results and what conclusions were drawn from them.

Testing Methodology

The contract acceptance testing methodology was used to conduct testing. At the beginning of this project, a team contract was formulated which outlined all the requirements for this application. Testing ensured that the predetermined functionalities and features were working and usable. A tester was given a task to complete within the app and recorded whether they were able to complete it along with any feedback they had. For example, the user was asked to sign up and register a vehicle (information was provided). Then they were asked to update the mileage by 10,000 miles and ensure that the correct maintenance is recommended. The tester denoted each task with a pass or fail rating depending on the outcome of the task.

Scope of Testing

The three modules were further divided into submodules for use as testing categories. The main focus was on how data is handled within the module, but also needed to ensure it was being sent and received between modules correctly.

- Application Code
- Receipt of data from the database
    - Handling of data within the application
    - Output of data to the database
- Database/Web Connectivity Code
    - Receipt of data from the application
    - Handling of data within the database
    - Output of data to the application
- User Interface Code
    - Organization and clearness of application display and flows
    - Coordination with device systems such as on-screen keyboard, screen rotation, and touchscreen
    - User input validation
    - Transfer of data to the application

Testing of the Application Code and Database/Web Connectivity Code submodules encompassed what the user doesn't see such as security or exposer of data, code optimization, and not-terminating errors. Testing of the User Interface Code included everything the user does see such as transitions between screens and user

feedback. The user's feedback during testing was the most important because it described the user experience. The project was positioned as a free app and it will not succeed if users don't like it regardless of how well everything else runs.

Objectives

- Major features and use cases need to be implemented if part of the scope
- Bugs that block the use of the app need to be resolved before IT Expo
- User profiles for "Forgetful Car Owner" and "DIY Car Owners" should be implemented
- App must perform smoothly without interruption from bugs or inefficient code

Record #	User #	Input	Expected Output	Actual Output	Pass/Fail	Reason:	Date
1	1	An excessively long username	Error Message	Error Message	P	Character length restrictions succeeded	2/8/2021
2	1	Wrong Password	Error Message	Error Message	P	Password Mismatch	2/8/2021
3	1	Account Registration	Dashboard Page	Dashboard Page	P	New User created successfully	3/1/2021
4	1	Login	Dashboard Page	Dashboard Page	P	Login succeeded	3/1/2021
5	2	Change Mileage	New Mileage	New Mileage	P	New Mileage updated	3/1/2021
6	2	Large Mileage increase	Oil change alert	Oil change alert	P	Mileage exceeded oil change limit	3/1/2021
7	3	View DIY video	YouTube App Search Results	YouTube App Search Results	P	Button click triggered app change	3/1/2021

Review

Testing revealed several points of contention in the user interface where users were having trouble navigating between pages or on the screen. Most issues were easily resolved by minor changes to the application. Overall, the testing process was very helpful in creating a minimum viable product.

Budget:

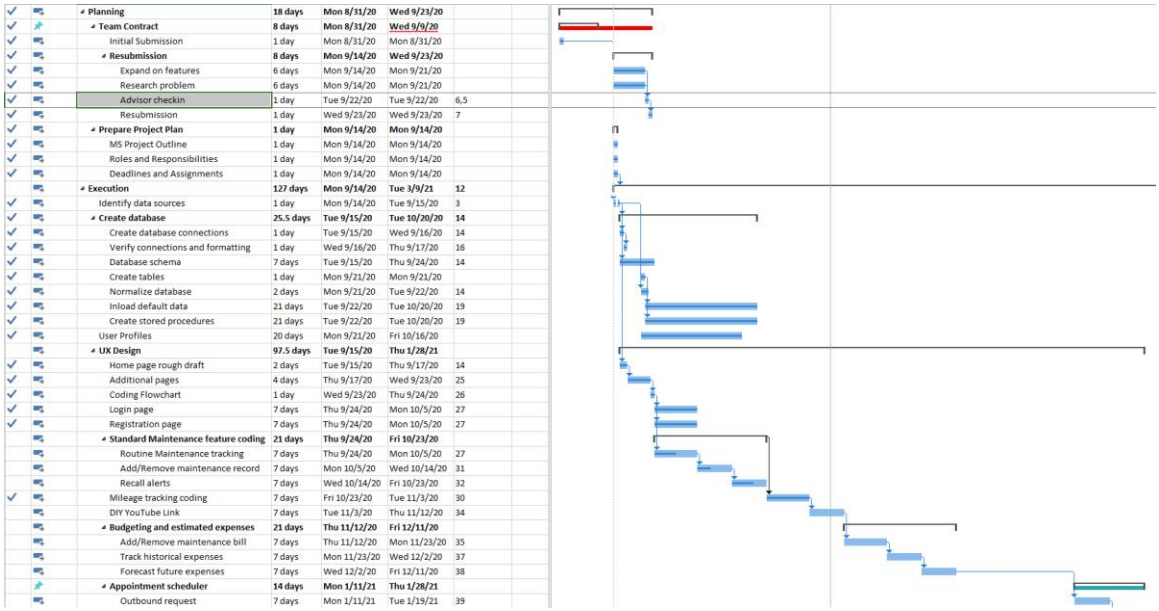
Figure 2: Project Budget, shows the cost of development and testing of CarWiz.

Figure 2: Budget

Estimated Cost Rough Order of Magnitude:							
	Rate Per/Hr	**Work Effort (Hours)**	**1 X Costs**	**Ongoing Annual**		
				Rate Per/Hr	**Work Effort (Hours)**	**1 X Support Cost**
Labor - IT	20	600	$ 12,000.00	20	60	$ 1,200.00
Labor - External	30	40	$ 1,200.00	30	20	$ 600.00
Software - External						
Hardware - External						
Misc.						
TOTAL			$ 13,200.00			$ 1,800.00
5-Year ROI Analysis						
Description		**5- Year Expected**	**Conservative (1.5)**			
Total Costs		**$ 22,200.00**	**$ 33,300.00**			
Total Benefit		**$ -**	$0			
Total Costs/Benefit Differential		**$ (22,200.00)**				
Conservative Costs/Benefit Differential		**$ (33,300.00)**				

Project Timeline:

Figure 3: Gantt Chart



	Planning	18 days	Mon 8/31/20	Wed 9/23/20	
	Team Contract	8 days	Mon 8/31/20	Wed 9/9/20	
	Initial Submission	1 day	Mon 8/31/20	Mon 8/31/20	
	Resubmission	8 days	Mon 9/14/20	Wed 9/23/20	
	Expand on features	6 days	Mon 9/14/20	Mon 9/21/20	
	Research problem	6 days	Mon 9/14/20	Mon 9/21/20	
	Advisor checkin	1 day	Tue 9/22/20	Tue 9/22/20	6,5
	Resubmission	1 day	Wed 9/23/20	Wed 9/23/20	7
	Prepare Project Plan	1 day	Mon 9/14/20	Mon 9/14/20	
	MS Project Outline	1 day	Mon 9/14/20	Mon 9/14/20	
	Roles and Responsibilities	1 day	Mon 9/14/20	Mon 9/14/20	
	Deadlines and Assignments	1 day	Mon 9/14/20	Mon 9/14/20	
	Execution	127 days	Mon 9/14/20	Tue 3/9/21	12
	Identify data sources	1 day	Mon 9/14/20	Tue 9/15/20	3
	Create database	25.5 days	Tue 9/15/20	Tue 10/20/20	14
	Create database connections	1 day	Tue 9/15/20	Wed 9/16/20	14
	Verify connections and formatting	1 day	Wed 9/16/20	Thu 9/17/20	16
	Database schema	7 days	Tue 9/15/20	Thu 9/24/20	14
	Create tables	1 day	Mon 9/21/20	Mon 9/21/20	
	Normalize database	2 days	Mon 9/21/20	Tue 9/22/20	14
	Inload default data	21 days	Tue 9/22/20	Tue 10/20/20	19
	Create stored procedures	21 days	Tue 9/22/20	Tue 10/20/20	19
	User Profiles	20 days	Mon 9/21/20	Fri 10/16/20	
	UX Design	97.5 days	Tue 9/15/20	Thu 1/28/21	
	Home page rough draft	2 days	Tue 9/15/20	Thu 9/17/20	14
	Additional pages	4 days	Thu 9/17/20	Wed 9/23/20	25
	Coding Flowchart	1 day	Wed 9/23/20	Thu 9/24/20	26
	Login page	7 days	Thu 9/24/20	Mon 10/5/20	27
	Registration page	7 days	Thu 9/24/20	Mon 10/5/20	27
	Standard Maintenance feature coding	21 days	Thu 9/24/20	Fri 10/23/20	
	Routine Maintenance tracking	7 days	Thu 9/24/20	Mon 10/5/20	27
	Add/Remove maintenance record	7 days	Mon 10/5/20	Wed 10/14/20	31
	Recall alerts	7 days	Wed 10/14/20	Fri 10/23/20	32
	Mileage tracking coding	7 days	Fri 10/23/20	Tue 11/3/20	30
	DIY YouTube Link	7 days	Tue 11/3/20	Thu 11/12/20	34
	Budgeting and estimated expenses	21 days	Thu 11/12/20	Fri 12/11/20	
	Add/Remove maintenance bill	7 days	Thu 11/12/20	Mon 11/23/20	35
	Track historical expenses	7 days	Mon 11/23/20	Wed 12/2/20	37
	Forecast future expenses	7 days	Wed 12/2/20	Fri 12/11/20	38
	Appointment scheduler	14 days	Mon 1/11/21	Thu 1/28/21	
	Outbound request	7 days	Mon 1/11/21	Tue 1/19/21	39

Problems Encountered and Analysis of Problems Solved:

The team encountered many problems, and it took a long time to figure out a solution for some of these issues. These issues were the following:

- Database driver continued to fail, and they were unable to connect to the database. This was solved by updating the database driver and no new issues have been identified.
- Needed adequate software to create a UI design. They ended up getting an education version of Figma to remedy this issue.
- Learning how to use Android Studio since they did not have any experience with it.

Future Recommendations:

If they had to redo this project, the team would've used a coding application that they were more familiar with. Specifically, one that they knew could connect to a database. The team encountered several problems with Android Studio that they would not have encountered with an application that they were all more familiar with. The team also would have better prioritized each feature to ensure that the core product idea was completed first and then work on the additional features if they had time.

For the future of CarWiz, they would like to implement features that did not make it during this project timeline. First, integration with an ODB2 scanner that would read error codes and mileage data from the vehicle. This feature would increase the app's appeal since no other maintenance tracking app uses it. However, due to the technical difficulty and time required to implement it, they were not able to include it in the final product. Second, is the ability to schedule maintenance with a mechanic through the app. Lastly, notices on vehicle recalls will be implemented. With these features, CarWiz will become an even better app that would improve user experience immensely.

Conclusion

Lessons Learned so Far:

The team learned many lessons up to this point. One of these lessons was that the team needed to remove all the workload from the main UI thread, or the app will crash.

Skills / Abilities Developed:

Many skills have been developed during this project. These include:

- Database integration (Learned how to integrate a database into Android Studio)
- Android Studio/Kotlin (Learned a new system/platform to use moving forward)
- Integration of Android Studio packages such as what we used for the charts
- MySQL Database Administration (Learned how to implement DB administration for user authentication)
- Multithreading

Android Studio was the biggest skill development area as the team was completely new to it. As part of developing that skill the team also had to learn new ways of arranging our code and how the logic flowed. One of the biggest issues with Android apps is stalling the main UI thread. Almost everything that the application needs to do had to be coded in such a way that it would run in a separate thread. There were several different challenges associated with this as our application routinely switches between static and non-static contexts. The team had to figure out how to move and read data from a static context to a non-static context and vice versa. In addition to this, the team had to figure out how to accomplish that while running everything in separate threads which included how to tell when a thread is finished, how to retrieve and perform operations on the thread, and what to do while waiting for a thread to finish. Beyond the technical limitations the team also encountered user limitations. Most of our users will not be tech-savvy and will need simplified indicators like the loading circle for circumstances such as when they need to wait for a thread to finish. In some cases, the team had to learn how to allow the application to progress back and forth between other pages while waiting for a thread to finish. This way, users wouldn't think the application was frozen and close it.

References

AAA Insurance. (n.d.). Retrieved April 12, 2021, from https://www.aaa.com

Mobile Operating System Market Share Worldwide. (n.d.). Retrieved December 05, 2020, from https://gs.statcounter.com/os-market-share/mobile/worldwide/

The Next Egg. (n.d.). Retrieved April 12, 2021, from https://www.thenextegg.org/

Appendix 1: Poster