Development of an App in Android Smart Phone for Pavement Roughness Estimation

DESIGN DOCUMENT

Team

sdmay20-17

Client

Bo Yang

Advisor

Halil Ceylan

Team Members/Roles

Tanner Dempsay – Communication Leader
Christian Royston – Git Master
Kyle Eckrich – Report Manager
Justin Kuennan – Web Master
Joe Van Treeck – Data Manager
Greg Starr – Logistics Manager

Team Email

sdmay20-17@iastate.edu

Team Website

http://sdmay20-17.sd.ece.iastate.edu/

Revised

12/8/2019

Executive Summary

Development Standards & Practices Used

- Agile Project Management
- Code Documentation
- Software Testing Levels

Summary of Requirements

- Android application that records accelerometer data and location data during a vehicle's route
- Route data, including acceleration, GPS, etc. is sent to the server in real time
- Server calculates IRI value of pavement vehicle is driving on and transmit this back to phone
- All data is stored in a MongoDB or similar database structure
- User interface for application is easy to use while operating motor vehicle. Data is clearly visualized while the pavement is being measured

Applicable Courses from Iowa State University Curriculum

The following courses included content applicable to our project. These include topics such as Android development, project management, data communications, and user-interface design. Prerequisites taken earlier in the curriculum, such as MATH 165, will assist with the development and implementation of the roughness calculations.

- COMS 309: Software Development Practices
- CPRE 185: Introduction to Computer Engineering
- PHYS 221: Introduction to Classical Physics I
- CPRE 288: Embedded Systems I
- MATH 165: Calculus I

New Skills/Knowledge acquired that was not taught in courses

Since our team is entirely made up of computer engineers, the skills we have acquired in our coursework overlap significantly. This also means that most knowledge relevant to this project that is not taught in our curriculum must be learned by all members. Server implementations are one area we must acquire. The requirements for the backend specifically request a NodeJS-based backend, which is JavaScript-based. This language is not taught in the computer engineering core curriculum. This is the same situation for database management. Though some members have exposure to these platforms from their COM S 309 group projects, more research will need to be done to will understand these tools.

Our team is also researching the current methodology and tools for measuring pavement roughness. For this project to be useful to the end users, we need to build a model that can accurately measure the roughness using a smartphone. This background knowledge includes different types of pavement material, standards for measuring roughness, and use cases for measuring.

Table of Contents

List of Figures, Tables, Symbols, Definitions

Definitions:

- Department of Transportation (DOT)
- Electrical and Computer Engineering (ECE)
- Global Positioning System (GPS)
- International Roughness Index (IRI)
- JavaScript (JS)

Figures:

Tables:

1 Introduction

1.1 ACKNOWLEDGEMENT

Our team would like to thank our faculty advisors Bo Yang and Halil Ceylan for allowing us to work on this project with his team and for all future assistance. We would also like to thank our Doctoral student advisor Chen-Yeou Yu and professorial advisor Wensheng Zhang in advance for helping to guide our team. We look forward to working on this project over the next year and hope to learn from our advisors and all others who helped.

1.2 PROBLEM AND PROJECT STATEMENT

There is a large need to monitor and characterize the overall quality of roads in order to prioritize maintenance of infrastructure. The existing solution often used are class 1 profilometers, which are expensive systems that must be mounted to the vehicle. This prohibits small organizations from obtaining and is costly for very large organizations to maintain a fleet of such devices. Resultantly, there is a need for a cheaper solution for determining the roughness of pavement.

Our solution to this problem is a smartphone application that can calculate the International Roughness Index (IRI) of a given surface. The phone, mounted to the car, will collect the accelerometer data which will be used to calculate the IRI. Once the calculation is completed, the application will then store the determined IRI associated with the road in question, determined by the GPS of the device. As smartphones are ubiquitous, this will drastically decrease the cost of pavement monitoring.

IRI is a roughness standard designed to quantitively measure the roughness of paved roads. This data is used by governments and organizations to review and maintain roads to minimize driver cost and ensure their safety [1]. IRI is the ratio of a vehicle’s vertical displacement (via suspension) over the distance traveled during a unit of measurement. The result is a slope, commonly expressed in m/km or in/mi. Below is an example of road ratings with corresponding IRI ranges.

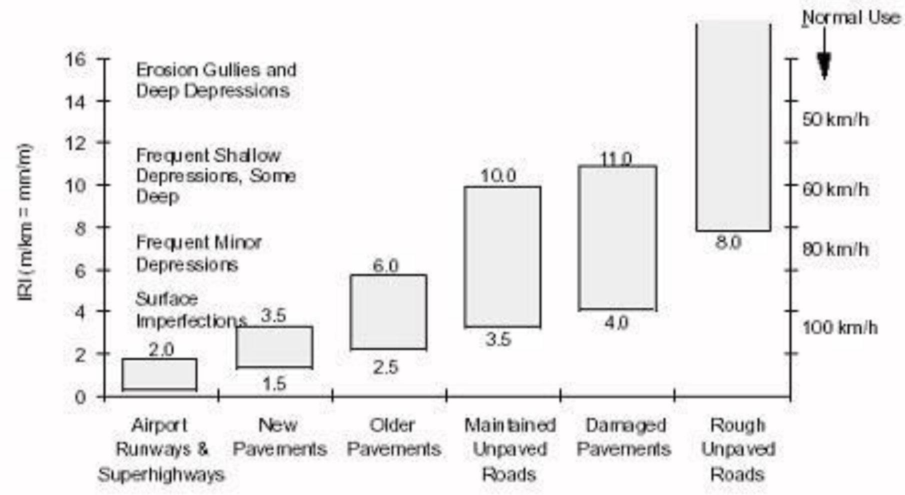


Figure 1: IRI roughness scale [2]

The optimal deliverable we hope to have by the end of this project is a platform that can be used to measure and store pavement roughness data, using a client and server model. Potential users of our project should be able to setup a multitude of client devices to measure road data, which would be able to be processed and stored in real-time. The outputs of project should be robust frontend and backend systems that allow for easy, intuitive, and useful data metrics on pavement roughness. All of this done using a low setup and operating budget from the client.

In tandem with the development of the design proposed in this document, the following materials will be created:

1. Team Meeting Notes
2. Bi-Weekly Status Report Updates
5. Design Document (Iterative Versions and Final)
6. Senior Design Website
7. Software Solution that meets all Functional Requirements
10.Lightning Talks for Senior Design

1.3 OPERATIONAL ENVIRONMENT

The app will be used on a phone that has been mounted to the dash of the car or truck. Since it will be indoors it won't need any special protection other than what the phone already offers. The car mount however should be very sturdy and stable as any unintentional movement of the device during measurement could affect the outputted data from our calculations. Since the application will be running on a smartphone, it is important to consider the battery usage during measurement and processes in our project.

1.4 REQUIREMENTS

Functional Requirements:

1. Project requires an Android app representing the front end for the user, and a server for the backend.
2. Android app must take accelerometer data from the device's onboard sensors.
3. App must log the route taken by the user during the roughness measuring.
4. Data acquired shall include GPS coordinates, accelerometer values in all axes, and IRI values among each step of the route.
5. System shall use a user-based system to identify app users.
6. The server shall be backed on NodeJS framework.
7. All relevant data relating to routes, users, etc. will be stored in a MongoDB database system.
8. Application shall have minimum API support for Android Nougat and newer versions of Android.
9. Application shall be able to calculate pavement roughness offline if the cellular connection is weak or nonexistent.
10. All data gathered during the route shall be saved on smartphone's local storage in case of connection error.

Economic Requirements:

11. Android application shall perform optimally on most phones commercially available. The platform and all its components shall not use purchased or premium software for any of its functionalities. Internet connectivity to enable connection with the application server.

Financial Requirements:

12. Team members will need Android phones for testing and development of the application.
13. A server to perform certain calculations involved in IRI calculation, and a database to store previously gathered data.
14. A vehicle with a phone holder will be required to test and verify the quality of IRI calculations.

Non-Functional Requirements:

15. Failed login attempts shall be logged for auditing.
16. Any error propagated during server runtime shall be logged to a local file for evaluation.
17. The server should be capable of handling 20 users simultaneously without affecting performance.

18. Server software should function independent of the operating system it is running on.

Environmental Requirements:

19. Application must be able to generate accurate roughness calculations in various motorized vehicle types.

UI Requirements:

20. Frontend user interface must by easy to use by the user while they are operating a motor vehicle.
21. Interface should show real-time accelerometer data, location coordinates, and IRI value during route measuring.

1.5 INTENDED USERS AND USES

This project is being designed for the future use by transportation departments within government organizations. These groups will use our product to test a large range of pavement, usually on roads, across a geographic area. In these various areas, the cell service and GPS reception quality may change and could affect the ideal functionality of the application.

The application will be run on an Android device mounted to the vehicle being driven, possibly in different orientations. The driver or passenger of the app may interface with the application during routing or use. Different vehicles could also be used to take measurements. The organization using our project will run an instance of our server application and database. Our team will communicate with the client during the development and testing of the product to ensure the solution is functional and usable for their needs.

1.6 ASSUMPTIONS AND LIMITATIONS

The concept of this project has been researched and implemented by several parties to this day (see Section 3.1 for details). Our goal is to develop a similar solution but customized for the client while creating a new roughness calculation method. As such, our project may differ from ideal or established IRI propagation methods in order to produce a more accurate solution tailored to the limited inputs from smartphones.

Assumptions:

- Accurate IRI calculations can be performed using only accelerometer data.
- IRI calculations will be performed in real-time during sensor data collection.
- The vehicle will keep a constant speed while using application, within some margin.

- Calculations are being performed from a vehicle driving on a paved surface.
- The smartphone will have a stable GPS connection during route recording.
- The smartphone being used has an accelerometer built in. This sensor is crucial in getting the acceleration data used in the roughness calculations.
- The vehicle is being driven on some type of paved surface. The model developed assumes the vehicle is on a paved surface, as unpaved ground could produce a large noise that will affect the roughness estimations.
- The vehicle features a conventional suspension system seen on most commercially available cars. The quarter model outlines a common system of forces used to generate our roughness values. Uncommon suspension systems may alter this model, producing unintended results as they deviate from the quarter car.
- The server application will be running on an operating system officially supported by NodeJS.

Limitations:

- Less accurate than high cost class 1 profilometers. Due to the accuracy of GPS receivers in smartphones and the minimal values we are measuring directly, it is unlikely that our solution’s predictions will be as accurate as hardware specifically designed for pavement roughness measurement.
- Will require the user to have a smartphone running Android OS. Since one of the client requirements for this project is to create an Android app, we will only be able to utilize smartphones running Android OS.
- May be calculation intensive, requiring server-side calculations which could potentially limit the number of concurrent users.
- Cellular signal may limit the frequency of data from client to server or vice versa. Signal strength could limit our applications communication with the server, disrupting the data sent and received in real-time on the smartphone.
- Computing hardware running server instance may affect performance of calculations. The efficiency of the calculations is partly dependent on the hardware it is on.
- GPS accuracy of smartphones will affect roughness predictions. Inaccurate coordinates during route recording could affect the estimated roughness value, as these coordinates will be used in calculating distance traveled over a length of time.

1.7 EXPECTED END PRODUCT AND DELIVERABLES

- Android application - May 2019
    - Used to collect accelerometer and GPS data while the user is driving
    - Calculates the IRI of the road driven on
- Server/Database - May 2019

  - Alternative calculation of IRI if the calculation is too intensive for a phone
  - Maintain all calculated IRI values and their associated roads

The smartphone application will simply be a tool that will allow the user to determine the roughness of a road. Before driving over a stretch of pavement, the user will attach the phone to the windshield of the vehicle, start the session, and then drive over the road. The application will then use the derived formula to generate a value to assign the roughness of the stretch of road.

There will a server and database that exist to support the application. While unknown at the moment, it is expected that the formula used to calculate the IRI value could be very time intensive. If it is, it could be too slow for the phone to perform the calculations, and we would then use the server to do these calculations. Also, the IRI values will be stored along with the road segment driven on the database to allow the user to retrieve past data.

Both tasks are intertwined with each other and require significant work to complete. As such, the deliverables are both scheduled to be completed by the month of May 2019. For a more detailed analysis of the tentative project schedule, please see Section 4.1.

2. Specifications and Analysis

2.1 PROPOSED DESIGN

The proposed design is to create an Android-based application to record accelerometer data and the geographic location of the device. The device will transmit the accelerometer data and the geographic location data to a database. Both the device and database will then calculate an International Roughness Index (IRI) value using the quarter-car model for computing IRI. The device will then display the calculated IRI values on a map of roads. Optionally, an image can be recorded at the time of accelerometer data collection and stored along with the accelerometer and geographic location data. IRI standards must be followed by the value calculation.

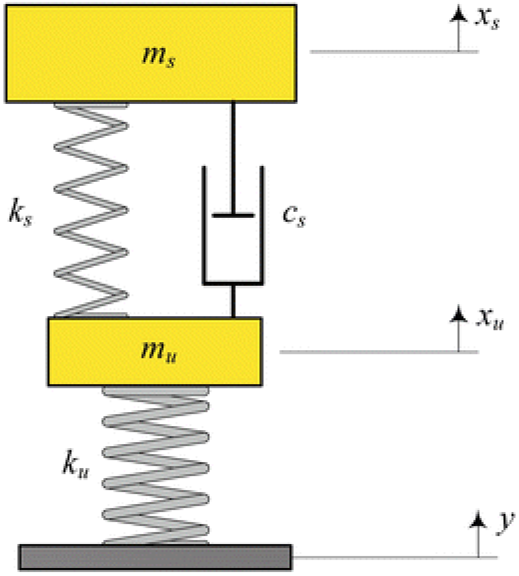


Figure 2: Quarter-car model of a vehicle suspension system [3].

Our calculation method to obtain IRI utilizes the quarter car model, a establishes a standard vehicle suspension system. This system is used to simulate the suspension system of a vehicle and the forces associated with its components. Its main components are the sprung mass (car frame), the suspension spring composed of a spring and damper, and an unsprung mass, representing the wheel hub, tire, rim, etc. [3].

All of the data recorded and produced by our application and server will be stored in a database. We will be using MongoDB as our database program, which is a cross-platform framework that organizes data into JSON-like documents [4]. Some examples of the data we will be storing include user profiles, sensor data collected from the smartphones, and the roughness calculates we calculate. Values will be linked through unique users and the routes that they record. Below is a sketch of the database structure.

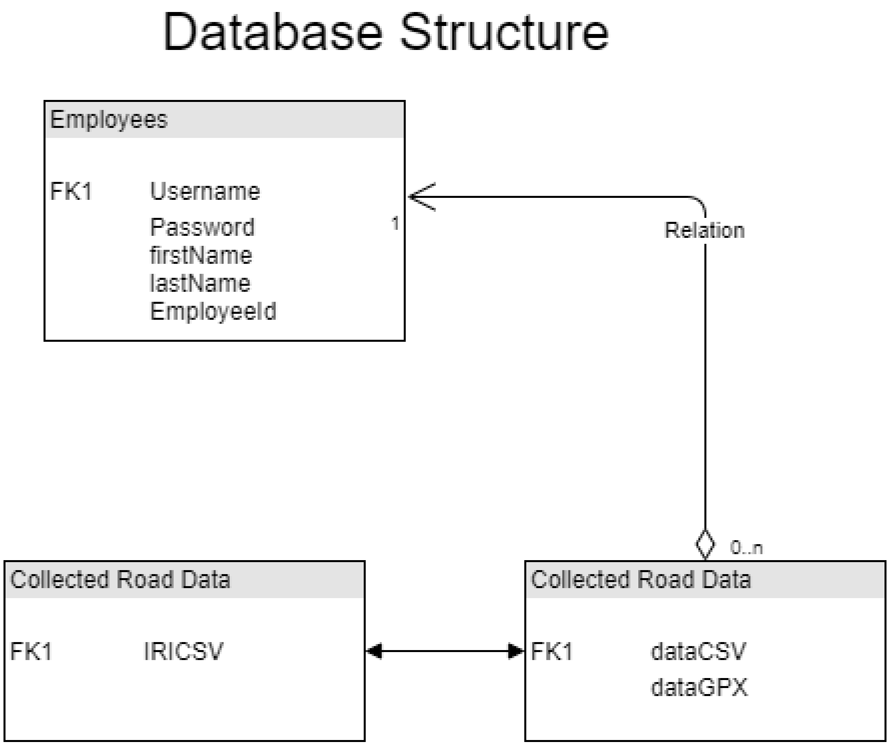


Figure 3: Database structure with types and relationships

The data will be read and written by using the MongoDB driver for NodeJS, which is maintained by the same team that develops Mongo itself. This allows us to sanitize the data received from the frontend before it is placed in the database. Figure 4 shows each major component of the system and their functions.

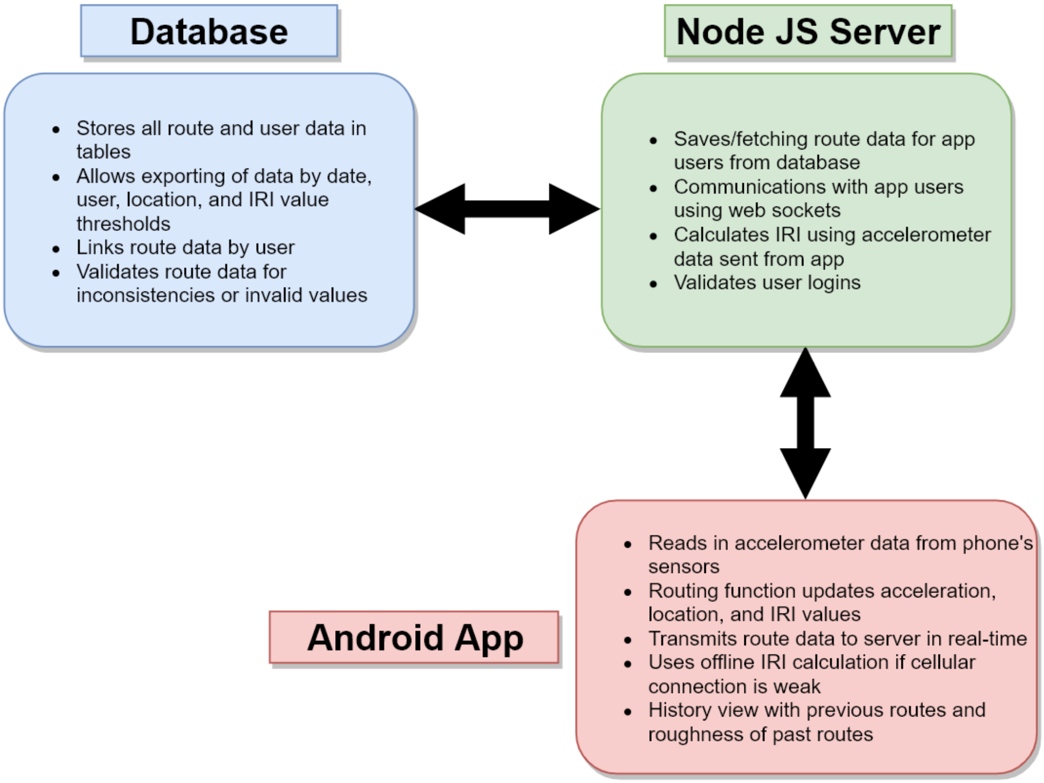


Figure 4: Block diagram of proposed solution with functionalities.

So far, a baseline server structure has been established and successfully hosted on a local machine as well as a Linux virtual machine, which has been made specially for this project. Barring any physical connection quirks or issues, clients are able to connect to the server through the host machines URL and designated port. We have also chosen to utilize Socket.IO as our primary networking library. Socket.IO is a bi-directional, real-time connectivity library and allows communication between the server and client [5]. Its API is available as in JavaScript and Android forms, allowing us to implement both sides easily and using similar syntax. We have implemented a simple test using Socket.IO between a client app and server, which is detailed in the next section.

2.2 DESIGN ANALYSIS

We have met with the professor and grad students that proposed the project and talked about what the IDOT wants in their app since they are the client. We have researched about IRI calculation as well. We have a couple of test apps that were supplied to us to build from one of the professors. Showing how to get sensor read off and to store it into a file that can be analyzed for further use. As well as a test app that gathers GPS location for a path. I work well but it is just a starting point we need to do all the back end and get the calculation working.

I think one of the big strengths of the proposed solution is how little you need to get started. All you need is a smartphone something everyone has, a car mount something that is relatively cheap, and a car or truck to make the calculations. Though I would some a weakness is the calibration for all the various cars and truck we need to find a good way to adjust the calculations for different suspenders.

To test the usability of Socket.IO, we created an example Android application to send location and acceleration data to a server periodically, which logs them. We were able so successfully view this data on the server side with nearly no delay. The server is on a standalone Linux-based virtual machine hosted within Iowa State's network.

2.3 DEVELOPMENT PROCESS

We are following an Agile development process, because most team members have experience and are comfortable with the process. Using an Agile-based process allowing us to compile our requirements into features to complete. These features will be assigned to our members which allows us to track progress for each member. A scrum board will allow members of easily see the status of the project at any moment in time. This includes features waiting to be worked on, current tasks for each member, and completed features. Our milestones will represent a release for the team, representing a group of features completed to create a significant piece of the system or final deliverable. Following standard Agile practices, our team will also be creating and executing tests for features developed throughout the project. These tests are outlined in a later section of this document.

Our frequent communication with the client will align with our Agile springs, allowing for variable input and changing requirements throughout the duration of the project. Meetings will provide individual status updates as well as project progress as a whole. Individual tasks will be evaluated on their progress compared to the project schedule, detailed in Section 4.

2.4 DESIGN PLAN

The International Roughness Index (IRI) characterizes the approximate roughness of a given surface and is used in the maintenance of roads. The application will allow for a user to instruct the application to begin IRI calculation and to transmit the data to a central database. The application will also record the GPS location of the device in this situation. In a separate use-case, a user will want to know the roughness of roads that have already been recorded. In this situation, the user will want to access a map overlay with the recorded pavement roughnesses. The application will contain a separate user interface containing the map that will allow for a user to scroll around the map and identify the roughnesses that have already been calculated.

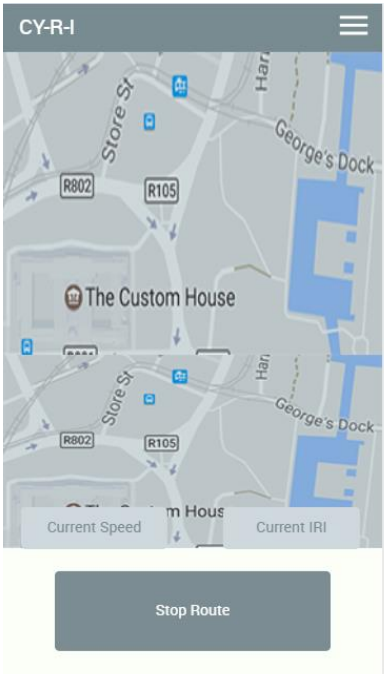
Our design plan consists of two concurrent areas of the project being developed: the frontend and backend. For the Android application, we are laying out tasks as to maximize the amount of simultaneous work that can be done. For example, one member will design and create the user interface for the application. It will be mostly non-functional skeleton of our application that will be used to place our functional elements that other members are working on. This allows multiple to contribute to its development while also minimizing dependencies on others' work, creating bottlenecks. Provided below are some of screen sketches for the application, the remainder are in Appendix I. The UI consists of: a route screen, user-login/signup page, route history page, and settings. The route screen is where the user can start and stop route recording, see the current IRI, and the current readings of acceleration and location of the device. The route history will contain all of the users previously recorded routes, with details for each step of those routes. The settings menu will be used to input the user's car information. This is used to calibrate the quarter-car model used for IRI calculations.

To reduce redundancies created by implementing similar features into the application, we will have multiple interfaces developed for modularity. First will be a location interface, that compartmentalizes all GPS initialization and management processes. It will allow the user to get current location data and setup periodic location update procedures for their activities, like callback function setters. This interface will be used in our route gathering screen as well as our route detail screen to view previous routes overlaid on a map view.

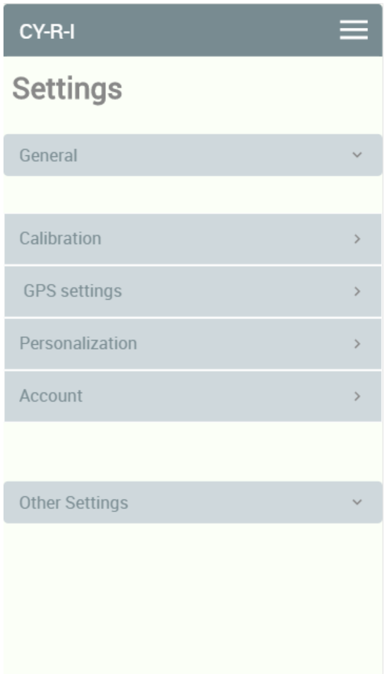
The next interface for the Android app will implement our network utilities. This set of functions will be responsible for data transmission and reception from our server. This interface will be used heavily in all activities of the frontend and therefore, it is a priority that it is completed earlier in the development cycle.

Our project plan focusses heavily on completing core requirements of our project. Some requirements, which detail specific uses or constraints for features in our platform, are inherently dependent on the initial feature being implemented in the first place. Therefore, we need to create reliable, functioning core software which we will optimize to fit all of the requirements by the end of this project.

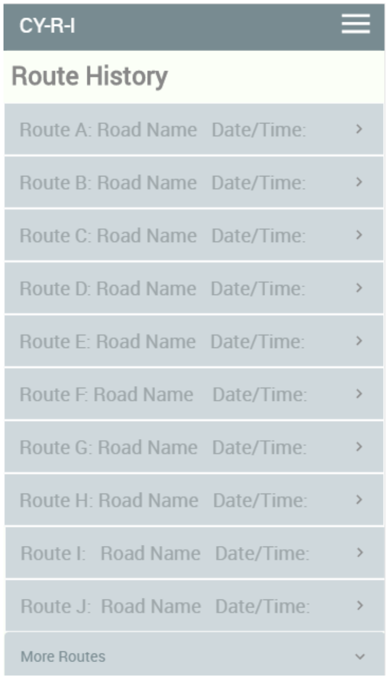
Route Screen (while measuring IRI):



Settings Screen:



Route History List Screen:



Route History View Screen:

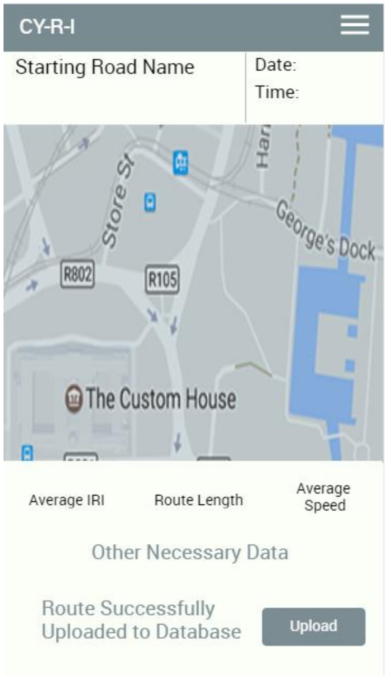


Figure 5: Screen sketches for Android application

3. Statement of Work

3.1 PREVIOUS WORK AND LITERATURE

Similar mobile applications exist with varied amounts of the features our project will contain. Some of the existing products have IRI calculation implemented alongside GPS tracking, which is part of the end goal for our product [6]. The advantage of these products is that they have been publicly available for long periods of time and have been tested by a larger audience. The disadvantage is these applications require the user to use their platform only with minimal customization. In addition, you must export the data gathered by these tools and perform analytics independently if the data they provide is not sufficient. Our project will be all inclusive, meaning that the final deliverables will include the server and database portion as well as the frontend application. This will give the client complete control over their data storage and customized analytics to fit their requirements.

Since IRI is a widely used standard of measurement, there is a wealth of existing research on methods of getting it as well as instruments used to calculate it. This includes the quarter car model that our calculation will be based on. A graduate student from the University of Illinois adapted this model to calculate the roughness using smartphone acceleration data, which is the same procedure we are using [7]. Our goal is to apply this model in the context of our Android application, and the use case of a smartphone mounted to a vehicle.

3.2 TECHNOLOGY CONSIDERATIONS

The biggest strength about the technology that is currently available is that it calculates IRI accurately, so we have a basis to test our data around. Having some examples to look at also gives us an idea about what to avoid. The biggest issue with some apps is that they do not use GPS tracking with the IRI calculations, or they do not have ways to store the data. Our goal with the project is to have an application that has the required features as well as some quality of life features. Most existing products calculate IRI on the frontend side as the accelerometer data is coming in. For our project, we will have a way to calculate IRI on the smartphone, but we will also be sending data to a server and store it in a database so that it takes some load off the phone.

Our frontend application will be built for Android using Java. All of our team members have had coursework and project experience with Java and programming for the Android environment. In addition, there is a wealth of support and examples for Android applications, including those using GPS and onboard sensors. One disadvantage is having to design a user interface that looks and performs adequately on many different screen sizes and forms. With more restrictive permissions in newer permissions in Android OS, we may be limited on the accuracy or availability of the data measured from the hardware components. The performance of the application will also vary with the computing power of the smartphone it is running on. Therefore, it is important that we develop efficient routines, minimizing CPU and memory usage on the frontend. Therefore, we are putting the primary calculation responsibility on the server.

The server will be using NodeJS, a framework based on JavaScript. Node is a hugely popular platform with many resources online, making it easy to find solutions to roadblocks during development. There is also support for many external programs and features, such as Mongo database support. It is also largely platform-independent, so our software will run on any operating system that the Node framework supports [8].

Socket.IO is the library we are basing our client-server communication modules on. This was chosen over HTTP requests because of its ease of use, consistent API, and versatility. IO's libraries are available for both Node and Android, sharing much of the same syntax and interfaces. Communications can also bi-directional, whereas in HTTP only the app could start a data transfer. This means that the server can send data to an instance of the Android app without any explicit request from the app.

3.3 TASK DECOMPOSITION

The development cycle for this project can be broken down into two main platforms: the frontend Android application, and the backend NodeJS server. Each platform has multiple tasks that are dependent on them, and some that require sufficient progress complete on both. This requires close communication between members working on either platform to complete more system-level tasks, like getting data from the database to the Android application.

1. **Create NodeJS server.** Have a local instance of a NodeJS application running on the virtual machine being used as the server for this project. Server should be reachable by other devices.
2. **Create database with formats for data types and attributes**. Test data should be inserted and relations between data types should perform as expected.
3. **Link server and database for data reading/writing functionality.** Server should be able to programmatically read and write known data to database instance using proper modules.
4. **Develop user-interface for Android app.** This includes all screens and interactive elements of application. Some functionality of these items may need to be completed in later tasks.
5. **Create networking framework for frontend to communicate with backend.** This includes Android and NodeJS network interfaces and should communicate over any data connection, not just locally/simulated.

6. **Develop IRI calculation algorithm for acceleration data from Android phones.** Create mathematic model for calculating roughness predictions based on known inputs that the smartphone will measure.
7. **Implement algorithm on backend and frontend for online and offline functionality.** Take the model developed in the previous task and implement it in Java for Android and JavaScript for the server. Test implementations for correctness and consistency.
8. **Add GPS tracking and logging to Android app.** Create interface for getting real-time coordinates of smartphone and integrate usage of interface in route recording activity.
9. **Couple GPS and IRI data to create path histories for users.** Organize and query database for entries relating to current user on frontend and be able to pull all required data.
10. **System/acceptance testing with frontend, backend, and database.** Testing procedures detailed in Section 5.

3.4 POSSIBLE RISKS AND RISK MANAGEMENT

Possible risks that could arise would be time management. With a proper schedule our team should be able to keep on pace to finish. Another risk would be lack of advanced knowledge on some of the programming platforms. If our team decides to implement functions that we are unfamiliar with, then issues could arise when learning how to work certain functions. Lastly risks could arise with debugging. We plan to have our product finished early so that we have time to debug the code and fix any issues that might arise.

3.5 PROJECT PROPOSED MILESTONES AND EVALUATION CRITERIA

Milestone 1: **Validated IRI Roughness Calculation Model.** This will be considered complete with the model is proved to be correct and outputs accurate IRI values using location and acceleration data taken from a smartphone.

Milestone 2: **Alpha Version of Platforms Complete**. The Android application and server contain implementations of all core functionalities derived from functional requirements. Unit test suites for each feature are established as outlined in Section 5 of this document.

Milestone 3; **Refine Software Solutions using Test Results.** This release will be the combination of bug-fixes and improvements to alpha software until it satisfies most functional and non-functional requirements. Integration tests should also be performed here.

Milestone 4: **System and Acceptance Testing Complete**. At this point, all aspects of solution have been validated and approved to meet all requirements outlined by client and in this document.

3.6 PROJECT TRACKING PROCEDURES

We will be making use of the Issues page on the ECE GitLab, where our Git repository for the project is located. There, we will be able to create a board similar to that of a scum-Agile project board. Requirements will be broken down into software features to be implemented, or the "issues". Issues that are in progress will be labeled as such and assigned to the team member working on that specific feature. Issues will be categorized by frontend and backend and grouped into milestones that will track the big picture progress of the project.

In addition, we will be releasing reports throughout this semester and the next on our team website. These will detail all work done on the project, including design and technical related progress being contributed by team members.

3.7 EXPECTED RESULTS AND VALIDATION

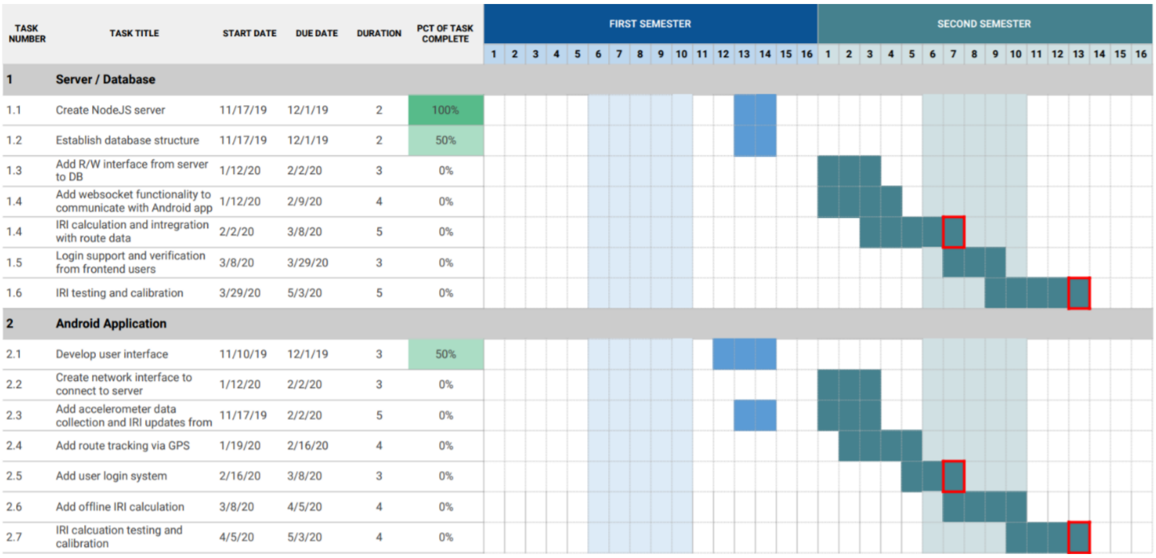
The desired outcome of the product is to have an android application that can accurately measure pavement roughness. The app will be mounted in a car or truck and will take the accelerometer data off the phone. The application will be easy to use and have relevant GPS and IRI data for the user to view. The final result should also have the appearance and accuracy of a finished product.

In order to test the accuracy of the application, we will take data from roads that have been measured by the Civil Engineering department. This department has an accurate device for measuring roughness that we can use to compare data. We will also test our application using a similar vehicle in order to get completely accurate data.

4. Project Timeline, Estimated Resources, and Challenges

4.1 PROJECT TIMELINE

This project is comprised of two major sections, the frontend and backend. To implement these components, our group will split into two smaller teams, each focusing on one of the platforms concurrently. The schedule shown below shows the tasks outlined in section 3.3 of this document with their start and end dates. The schedule shown below shows the tasks outlined in section 3.3 of this document with their start and end dates.



TASK NUMBER	TASK TITLE	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Server / Database**				
1.1	Create NodeJS server	11/17/19	12/1/19	2	100%
1.2	Establish database structure	11/17/19	12/1/19	2	50%
1.3	Add R/W interface from server to DB	1/12/20	2/2/20	3	0%
1.4	Add websocket functionality to communicate with Android app	1/12/20	2/9/20	4	0%
1.4	IRI calculation and intregration with route data	2/2/20	3/8/20	5	0%
1.5	Login support and verification from frontend users	3/8/20	3/29/20	3	0%
1.6	IRI testing and calibration	3/29/20	5/3/20	5	0%
2	**Android Application**				
2.1	Develop user interface	11/10/19	12/1/19	3	50%
2.2	Create network interface to connect to server	1/12/20	2/2/20	3	0%
2.3	Add accelerometer data collection and IRI updates from	11/17/19	2/2/20	5	0%
2.4	Add route tracking via GPS	1/19/20	2/16/20	4	0%
2.5	Add user login system	2/16/20	3/8/20	3	0%
2.6	Add offline IRI calculation	3/8/20	4/5/20	4	0%
2.7	IRI calcuation testing and calibration	4/5/20	5/3/20	4	0%

Figure 6: Project Schedule Gantt Chart

The length of each task corresponds to the estimated time to implement all of its required features. More complex tasks have longer work periods. Most of the tasks are centered around software features being implemented to each respective platform. During each "feature" task, there is a certain amount of time included in these tasks to account for basic testing of the additions, such as unit-testing.

A block outlined in red represents a deliverable or release for our project. Due to the length of development for some of our project components, we are adopting a Kanban-like release cycle, which will align with significant features working on the application and server. The server deliverable represents completion of the IRI calculation using the real-time accelerometer data from the Android application. At this point, there should be transmitting and receiving of route data and roughness measurements on the frontend, with a functional route tracking interface (first Android deliverable). The second Android release corresponds to the end of the project, where all requirements have been fulfilled and every essential feature of the frontend and backend is fully functional.

To complete this project in two semesters, we are striving to complete essential features in the first semester. Essential features are those that other aspects of the project depend on to function. For example, web socket communication between the app and server will allow the sensor data to be transmitted during routing. We will develop these using a combination of top-down and bottom-up design, where we will use our high-level requirements to create these low-level interfaces which can be adapted and reused for various features. We are also leaving substantial time for testing near the end of the project's lifecycle to test our platforms on low, high, and system levels. Since IRI measurement devices already exist, we will be able to test our calculations against existing data. We want to use this to toughly refine and prove our measurements to create product that end users will trust when they use it.

4.2 FEASIBILITY ASSESSMENT

The challenges with this project will be the dynamic environments in which the application will be used. While the IRI calculation may be a known equation, people may use our product in many different cars with different mounting systems, suspension, etc. Rigorous testing will need to be done to ensure we are producing a consistent, accurate measurement tool for people.

Another challenge will be the routing function of this application. During the measurement cycle, the application software will have to manage location data, accelerometer data, network interactions, and local storage all in real time. During the creation of this program's structure, we will have to keep in mind the time required for each of these steps and possible multi-threaded solutions to take advantage of modern phones' hardware.

Our team is adequately educated to complete the software portions of this project within the allotted. The challenges outlined above will determine the overall accuracy and efficiency of our final product.

4.3 PERSONNEL EFFORT REQUIREMENTS

The initial few tasks of the project are mainly setting up the server and database and making sure everything is connected correctly. This will not take too much time, so a few hours for each should be enough time. The next few parts are setting up the android app and making sure it can communicate with the backend. This will take a little longer than in order to make sure everything is communicating correctly with each other. The next part is developing the actual calculation for IRI. This will take a few weeks to convert data we can pull off the smartphone into a value that people can use. We will also be preforming a variety of testing procedures to make sure that the values are correct.

Following having these parts work independently of each other, we will begin to integrate all the parts into a more well-rounded solution. This includes setting up offline and online calculations, which could be tricky, so we assigned about one week to figuring this out. Task 8 will be setting up GPS data. This task has been allotted three weeks so that we can get it working accurately. The next task will be making sure everything works together, which would take about a week, mainly dealing with bugs that arise. Lastly, we need to test our product, so a few days have been allotted to make sure our app is accurate.

Tasks	Projected Members	Estimated Time
Task 1: Server Initial Setup	Tanner,	5 hours

Task 2: Database Initial Setup	Tanner, Kyle	5 hours
Task 3: Server/Database Interaction	Tanner, Kyle	15 hours
Task 4: Client/Server Communication	Tanner, Kyle, Justin	20 hours
Task 5: Android App Basic Setup	Greg, Christian, Justin	25 hours
Task 6: IRI Algorithm/Testing	Tanner, Joe	45 hours
Task 7: Online/Offline IRI Calculation	Tanner, Joe	40 hours
Task 8: User Account Login	Justin, Christian, Greg	20 hours
Task 9: IRI Calibration	Tanner, Joe	40 hours
Task 10: System/Integration Testing	Joe, Christian, Greg, Kyle, Tanner, Justin	40 hours

Table 1: Personnel Effort Requirements

4.4 OTHER RESOURCE REQUIREMENTS

- Android-based devices to test the application.
- Server space to hold database data and run instance of server application.
- LASER-based IRI measurement device and vehicle to compare the application values.
- Previous route data with GPS coordinates, acceleration data, and known IRI values in order to test correctness of designed IRI algorithm.

4.5 FINANCIAL REQUIREMENTS

There are only several additional resources required for the development of this project. The three physical components of this project will be the Android phone with the application installed, a car to drive while using and testing the application, and a web server for remote calculations and a database.

Our team requires three Android phones to be used during the development of the Android application. In addition, we will need access to a web server to be used to host our application and provide a database. However, we will be able to use team members' vehicles for environmental testing.

Required Item	**Count**	**Unit Cost**	**Total Cost**
Android Phone	3	$50+ (Cheap Android phone prices may vary)	$0-150+

Server (Development phase)	10 months	$0 (Provided by ETG)	$0
Phone Mount for Windshield	3	$20.00	$60.00
Total			**$60-$210**

Table 2: Financial Requirements

5. Testing and Implementation

5.1 INTERFACE SPECIFICATIONS

During unit-testing, we will use testing software designed for use with the target platforms. For Android, we will create Junit test suites for each feature implemented. As we move up to integration system, Junits that utilize test several components simultaneously will be created to test major features correctly. For the server software, we will be designing tests in JavaScript which will run independently of any of our platforms, like the Android application.

For integration testing and above, an Android-based device will be required. The device will utilize the accelerometers included with the phone to calculate an International Roughness Index value using the quarter-car model. Testing will be conducted with a vehicle including a LASER-based IRI calculator. The values of the applications calculation and the LASER calculation will be compared, and a determination will be made on whether the application's calculation is acceptable.

5.2 HARDWARE AND SOFTWARE

All our testing will be software testing, as there is no physical component to this project. We will be performing unit testing throughout the entirety of the development of the application, to ensure all the low-level code works exactly as intended. As we build the application from disparate pieces of code, we will then perform integration testing to ensure, for example, that the accelerometer values collected are successfully transferred to the server. Additionally, as we develop our algorithm for calculating the IRI value, we will need to be continuously testing our program in comparison to known values. Finally, we will perform user acceptance testing to ensure all functional and non-functional requirements have been adequately met.

To create consistent and repeatable system tests, we will also be using the Espresso tool built into Android Studio. This allows us to simulate UI interactions with an application as if a user was moving through it themselves. UI testing and more interactive features will be evaluated using this tool to decrease testing time and remove some human error from formal systems and integration tests.

We also want to evaluate the performance of our Android platform to minimize the use of system resources like CPU and battery. For this, Android Studio includes a profiler tool which will let us monitor resource usage of our app in real-time. Using this, we will be able to optimize our platform to better meet performance requirements.

To create unit and integration tests for the backend software, we will utilize Mocha, a JavaScript test framework built in Node.JS. Mocha is a widely used testing suite with easy to use syntax, similar to Junit tests. It includes powerful features like dynamically generated tests and performance monitoring, allows us to test our software against functional and non-functional requirements.

5.3 FUNCTIONAL TESTING

Although not limited to these, we will mainly be performing unit and integration. Unit testing is an integral part of code development, as the programmer should be aware if the functions written operate as intended. We will do this frequently in our development cycle, but examples include testing the accrual of accelerometer and GPS data and accuracy of the IRI calculations.

Integration testing becomes useful as the codebase of the project increases in complexity. When code written at different times is brought together, it must be ensured that there are no unexpected behaviors are produced. Again, this is common, but examples in our project could include testing the interaction between the collection of GPS data its association with the calculated IRI value in our database.

Below is a table of the test plan created for our functional requirements. Each test has a requirement number that corresponds to one of the requirements in Section 1.4, as well as the test's description and expected outcome. Some entries may be comprised of one or more physical tests, which together will validate each test description with its outcome.

Requirement	**Description**	**Expected Outcome**	**Test Level**	**Test Environment**
2	Evaluate data gathered from sensor interface.	Acceleration data with proper accuracy and within valid range.	Integration	Standalone Android Device
3	Simulate route start and end and evaluate local log file on Android device for accurate route data given.	Route data in log file should match simulated route given by test.	Integration	Android Emulator
4	Call location interface to get current latitude and longitude of emulated	Location data returned should be correct in location and accuracy,	Unit	Android Emulator

	device.	matching the emulator's position defined in its settings.		
4	Valid route data with known IRI values passed into calculation interface and compare outputs. Server and Android implementations tested separately.	IRI values calculated at each route step should make accepted IRI values within margin of error.	Unit	Junit (Android) / Mocha (Server).
4	Pavement data gathered by profilometer on specified route will be compared to data gathered by Android application.	IRI data calculated using our platform should match closely to that gathered by the profilometer, within a reasonable error range	Acceptance	Standalone Android Device, Pavement Route
5	Call user creation function with known username and password.	Server shall return success on insertion of new user into database.	Unit	Mocha (Server)
5	Simulate user creation though frontend application's UI.	Frontend shall report successful creation, user's entry seen in database.	Integration	Espresso (Android), Database Logs
5	Call user login function with known existing credentials. Frontend and backend calls tested separately then together.	Function shall return user object of valid user previously created.	Unit / Integration	Junit (Android), Mocha (Server)
7	Insertion of known data with corresponding functions on server. Separate tests for all data types stored.	Data passed should be seen in database.	Unit	Mocha (Server), Database Review
7	Data inserted into database from server should be immediately available to pull by corresponding get function.	Server shall immediately request data after insertion. Data shall match inserted data in values and formatting.	Integration	Mocha (Server)
8	App reviewed for compliance with Android Nougat API requirements.	App works with Android Nougat and above operating systems.	Acceptance	Standalone Android Device.

9	Simulated route with known IRI values. During route, disconnect device and watch for switch to offline calculation mode.	Device should notify user of connection error and switch to offline calculation. Calculated values should match known IRI within margin of error.	Integration	Android Emulator, Espresso (Android)
10	Record route while device is disconnected from Internet. View route history to see offline route's data.	Route shall be recorded and stored on device with all relevant data intact.	Integration / System	Android Emulator, Standalone Android Device

Table 3: Test Plan for Functional Requirements

5.4 Non-Functional Testing

The table below shows the tests associated with each non-functional requirement detailed in Section 1.4.

Requirement	Description	Expected Outcome	Test Environment
11	Application and all functionalities outlined in functional requirements should be tested on multiple Android devices with different hardware.	All functionalities of app should work and give expected values within margins of error on all devices tested.	Standalone Android Devices
11	Server and frontend shall be rebuilt on clean environments to ensure no underlying paid software was previously used.	Systems should be able to be setup and operated on clean systems with no premium software or programs.	Server Computer, Android Device
15	Test login interface with invalid user credentials.	Server should return error indicating invalid user. Corresponding error file on server should all contain error.	Junit (Android), Mocha (Server)
16	Create function on server to throw intentional error. Call function through test suite.	Log file on server should contain error info matching error thrown in function.	Mocha (Server)
17	20 concurrent users will be simulated using HTTP/Socket calls on server test suite. Server	Data received from server shall be delivered in reasonable time with all	Mocha (Server)

	performance shall be monitored through delay in data being received.	users connected.	
18	Server will be setup on hardware running different OS then primary server. Calls to server functions used to test functionalities.	All functions on server should perform identically on hardware running different OS than primary	Test Server, Primary Server, Mocha (Server)
19	Route will be recorded in various vehicle on same route with known IRI value.	IRI values calculated in each vehicle should match the known within margin of error.	Standalone Android Device
20	Client will review interface during operation.	Client is satisfied with completed user interface of Android application.	N/A
21	User interface will be evaluated during route for correctly and updated values shown on screen.	The app should display the route data in real-time as the user is driving	Android Emulator, Standalone Android Device, Junit (Android)

Table 4: Test Plan for Non-functional Requirements

5.5 Process

The IRI calculation will be tested by comparing the calculation with a separate calculation conducted by a LASER-based method. This testing will determine whether the differences in the LASER-based calculation and the Android-based calculation are within a percentage of error or not. Communication between the server and the device will be conducted through testing whether information on the server can be received by the device. Map information will be tested by comparing information stored on the database with information that the device receives from the database.

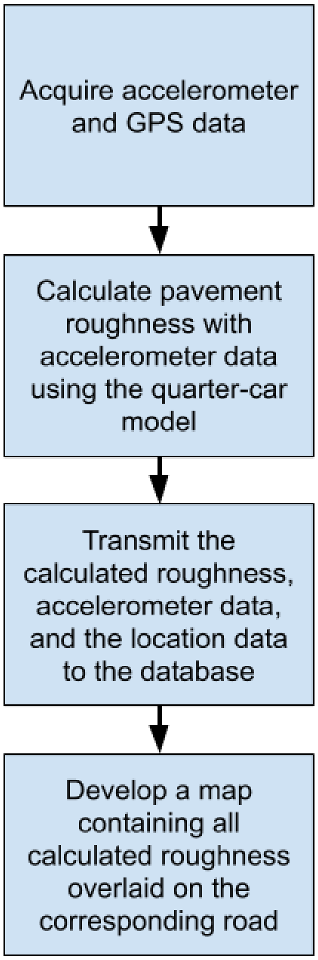


Figure 7: Process Diagram

5.6 RESULTS

No formal testing, such as those described in Section 5.3 and 5.4, have been conducted. Proof-of-concept testing has been performed on our chosen software platforms and libraries in order to verify each component will satisfy our requirements for use in our design. These tests are described in Section 2.2.

6. Closing Material

6.1 CONCLUSION

The team has researched International Roughness Index value calculations and determined tasks that each member will complete. The goal of this project is to create a platform for easily collecting road roughness on a large scale with minimal cost and operation time. Hopefully, the outcome of this project allows our client to improve the effectiveness and efficient of road maintenance, benefitting all of us who share the roads.

We seek to apply known roughness standards and calculation using the inputs given by Android smartphones. These measurements should be comparable to currently used road-profiling devices.

6.2 REFERENCES

[1] Pavement Tools Consortium, “Roughness”, *Pavement Tools Consortium,* Accessed on: Nov. 17, 2019. Available: https://www.pavementinteractive.org/reference-desk/pavement-management/pavement-evaluation/roughness/

[2] M. W. Sayers, T. D. Gillespie, & C. A. V. Queiroz, “The international road roughness experiment: A basis for establishing a standard scale for road roughness measurements.” *Transportation Research Record*, 1084, 76-85. 1986

[3] R. N. Jazar, “Quarter Car Model”, pp. 985–1026, *Springer*, New York, New York, NY, 2014.

[4] “MongoDB: The Database for Modern Applications”, MongoDB, Inc, 2019, Accessed on: Dec. 3, 2019. Available: https://www.mongodb.com/

[5] Automattic, “Socket.io”, Nov. 25, 2019, Accessed on: Dec. 3, 2019. Available: https://socket.io/

[6] A. H. Alavi and W. G. Buttlar, “An overview of smartphone technology for citizen-centered, real-time and scalable civil infrastructure monitoring,” *Future Generation Computer Systems*, vol. 93, pp. 651–672, 2019.

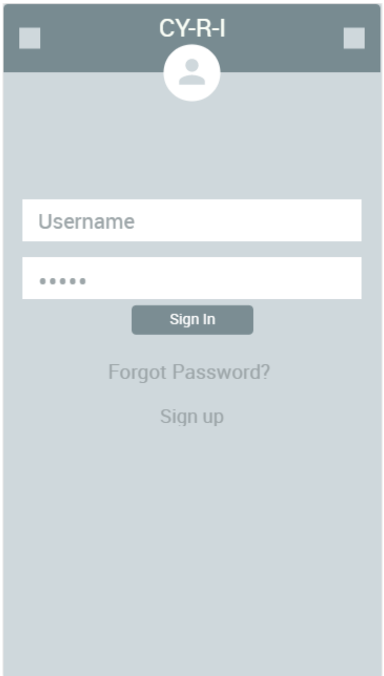
[7] S. Islam, "Development of a smartphone application to measure pavement roughness and to identify surface irregularities," PhD diss., *University of Illinois at Urbana-Champaign,* 2015.

[8] “Downloads,” *Node.js Foundation,* 2019, Accessed on: Dec. 5 2019. Available: https://nodejs.org/en/download/

6.3 APPENDICES

Appendix I: Screen sketches for Android application

Login Screen:



Home Screen: