Pothole Detection System

Group 16: Neel Patel, Revathi Dhotre, Rimsha Rizvi, Shehriar Burney

CS 440

at the

University of Illinois Chicago

Fall 2023

Table of Contents

List of Figures

I Project Description

1 Project Overview

Pothole Detection System (PDS) is an innovative mobile application designed to enhance road safety and navigation for users through urban and suburban routes. By integrating real-time data with user-generated content, PDS provides a dynamic and interactive map that alerts users to potential hazards such as potholes on their route. The application leverages Google Maps API to offer turn-by-turn navigation, avoiding areas with reported potholes and offering the safest and most comfortable travel paths. By adding to the community-driven database that powers the system, the public can validate existing potholes and report new ones while governmental officials label them as fixed, helping to maintain their local infrastructure.

2 Project Domain

Pothole Detection System's domain includes community-based reporting and urban infrastructure management. It deals with the general public's concern about road safety and maintenance of their local communities. Within the field of smart city solutions, the app works to address urban problems and enhance the quality of life using technology. A cooperative approach to addressing these issues is demonstrated by the system's effectiveness, which depends on the enthusiastic involvement of its users and the collaboration of nearby municipalities where user inputs directly influence the prioritization and response to infrastructure issues. This project domain is particularly relevant in regions with significant seasonal weather variations, where road conditions can rapidly change and impact travel safety.

3 Relationship to Other Documents

This project is the implementation of a prototype based on the design document that can be found here:

https://www.cs.uic.edu/~440/groupFiles_Fall2022/FinalSummaries/Group7Developm entProject FinalSummary.pdf

4 Naming Conventions and Definitions

4a Definitions of Key Terms

- **Pothole Detection System (PDS):** An application that integrates real-time data to alert users of potholes and road hazards during their travels.

- **Google Maps API:** A set of functions and procedures offered by Google that allows the retrieval of mapping data and can be integrated into apps or websites.
- **Backend Database:** A centralized database that the PDS app queries to fetch and store data related to potholes' locations and sizes.
- **Firebase Cloud Database:** A cloud-hosted NoSQL database that stores and synchronizes data between users in real-time, making it an ideal solution for mobile apps like PDS where timely data updates are crucial.
- **Pothole Reporting:** The process by which a user submits a new or existing pothole's location and size, contributing to the system's database.

Please refer to the glossary on page x for a complete list.

4b UML and Other Notation Used in This Document

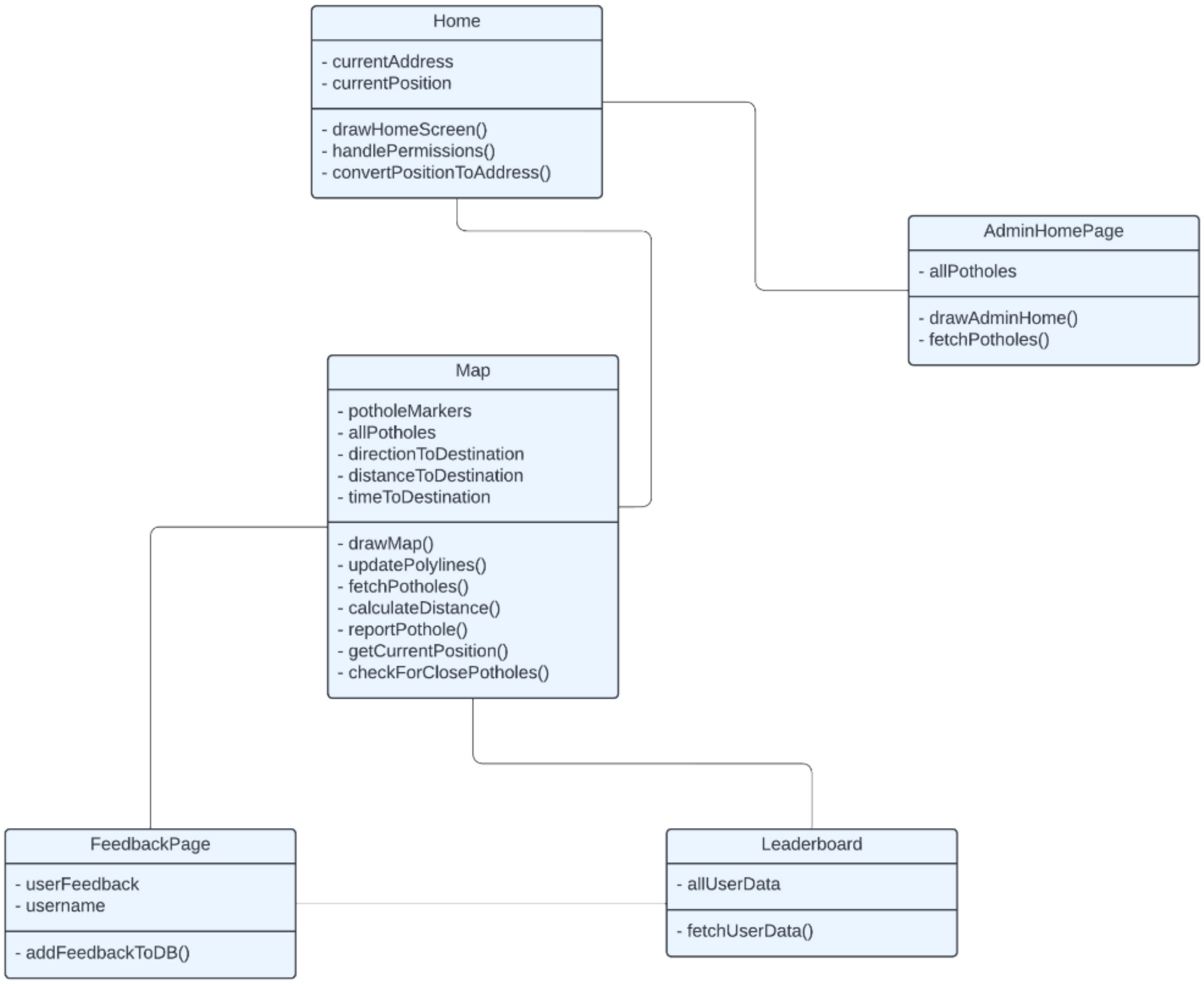


Figure 1: UML class diagram showing the structure of our application.

4c Data Dictionary for Any Included Models

Content:

The Pothole Detection System (PDS) employs a Firestore NoSQL database, part of the Firebase suite of cloud services, to manage and store various data structures essential to its functionality. The data models and their properties specific to this project are as follows:

1. User Information:

- Structure: User Record = {Username: String, Password: String, PotholesReported: Integer, Feedback: String}

    - Stored in the 'login' collection, this record maintains user authentication details along with metadata such as the volume of pothole reports and user feedback.

2. Pothole Details:

- Structure: Pothole Record = {Latitude: Double, Longitude: Double, Size: String, DateReported: Timestamp, ReportedBy: String}
- The 'potholes' collection tracks the geolocation, size classification, reporting timestamp, and identifier of the user who reported the pothole.

3. Navigation Data:

- Structure: Route Record = {PolylineCoordinates: Array of LatLng, Distance: Double, TimeEstimate: Integer}
- This data encompasses the coordinates delineating the user's route and the estimated time and distance of travel.

The Firebase API facilitates the storage, retrieval, and real-time synchronization of these data structures, ensuring that the application data is consistently up-to-date and accessible across different client devices.

Additional integration with external APIs includes:

- Google Maps API: For map rendering, navigation, and geolocation features.
- Geocoding API: To convert addresses into geographic coordinates, enabling the application to locate and report potholes accurately.

Data formats such as JSON for API responses and data serialization, along with GeoJSON for geographic data representation, are employed within the PDS.

Motivation:

The PDS's robustness is underpinned by its precise and consistent data handling. Defining the attributes of the data models ensures that the system's functionality, from user interaction to backend analytics, is coherent and reliable.

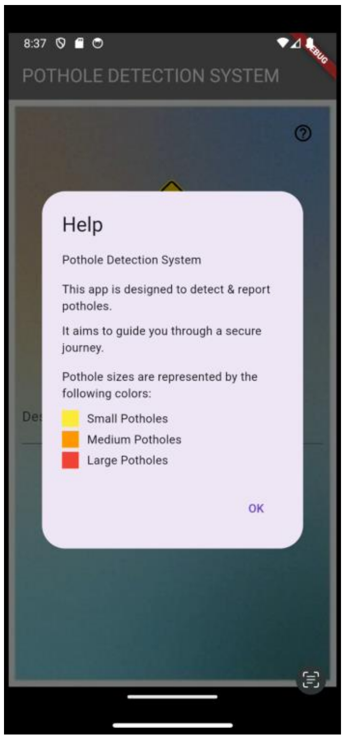
Considerations:

Implementation details added by developers to these data definitions will include specifics regarding data persistence, query optimizations, caching strategies, and API integration techniques. Such details are necessary due to the chosen technologies—

Firebase and associated cloud services—and are required to fulfill the PDS's operational requirements.

II Project Deliverables

5 First Release


8:37

POTHOLE DETECTION SYSTEM

Help

Pothole Detection System

This app is designed to detect & report potholes.

It aims to guide you through a secure journey.

Pothole sizes are represented by the following colors:

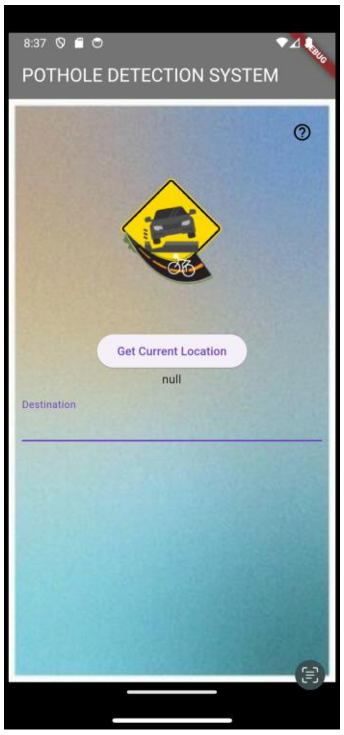
🟨 Small Potholes

🟧 Medium Potholes

🟥 Large Potholes

OK

Figure 2


8:37

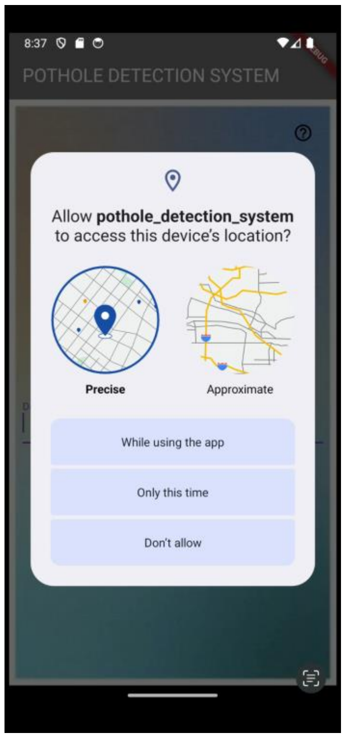
POTHOLE DETECTION SYSTEM

Get Current Location

null

Destination

Figure 3


8:37

POTHOLE DETECTION SYSTEM

Allow **pothole_detection_system** to access this device's location?

Precise

Approximate

While using the app

Only this time

Don't allow

Figure 4

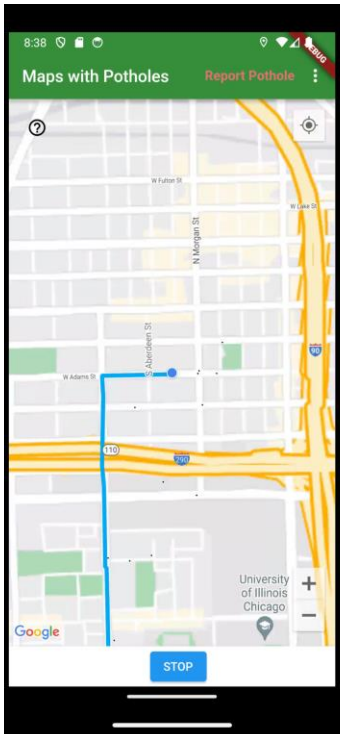


Figure 5

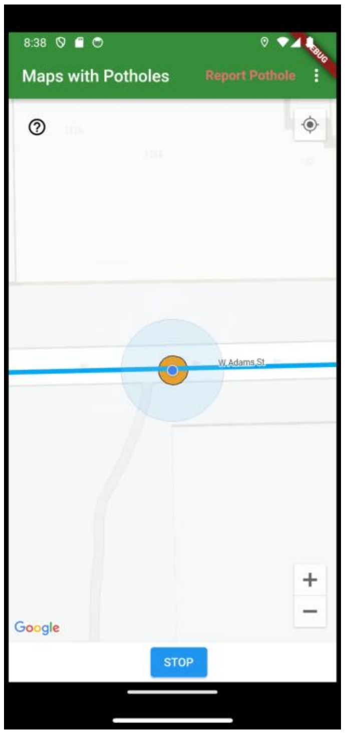


Figure 6

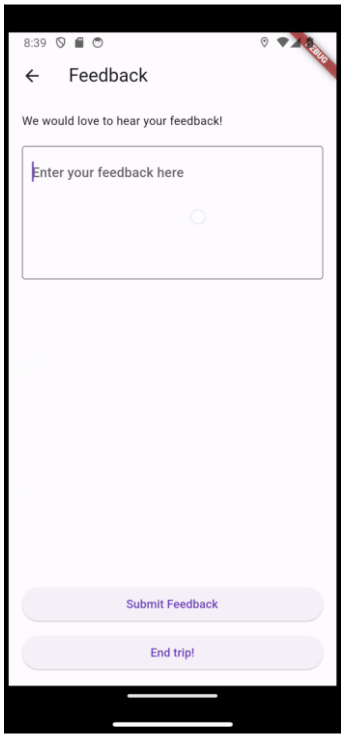


Figure 7

The following features were added for release 1 of Pothole Detection System:

Starting the Journey

A new user discovers the Pothole Detection System and downloads the application. Upon launching the application they click on the ‘Help’ button for an introduction of the application. (Figure 1)

Location Permissions

The user then clicks on the “Get Current Location” button which prompts the user to allow location permissions. (Figure 3)

Entering Destination

Now that they have given their current location, the user now enters the coordinates of their destination and clicks on the icon button of the PDS logo to start the journey. (Figure 2)

Google Maps and Polyline Interface

The user navigates to the main screen which displays a Google Maps UI. This UI also displays a directions line to guide the user on where to go. This polyline will change and update if the user makes the wrong turn. (Figure 4)

Report Pothole

While riding his bicycle, the user notices that there is a quite large pothole on the ground which could be dangerous for others on the road. They decide to report the pothole on PDS by clicking on the “Report Pothole” button and clicking “Large”. This pothole is now visible to all who are using the application. (Figure 5)

Stop and Feedback

After the user has reached their destination, they click on the “Stop” button which navigates away from the Google Maps UI to a Feedback page where users can give their opinions on the application. The user found the application very helpful so they left the comment “Awesome Application!” and close the application. (Figure 6)

6 Second Release

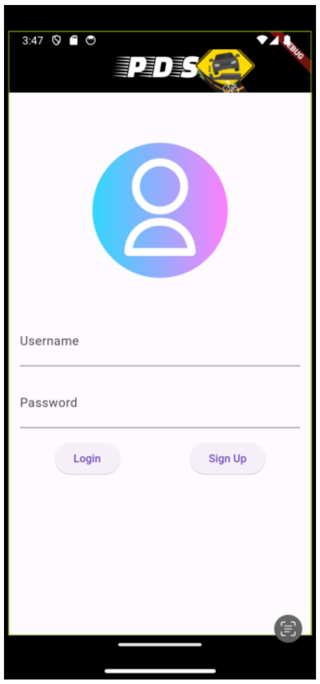
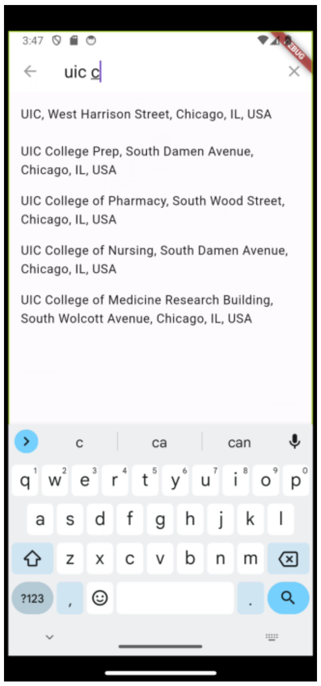


Figure 8



UIC, West Harrison Street, Chicago, IL, USA

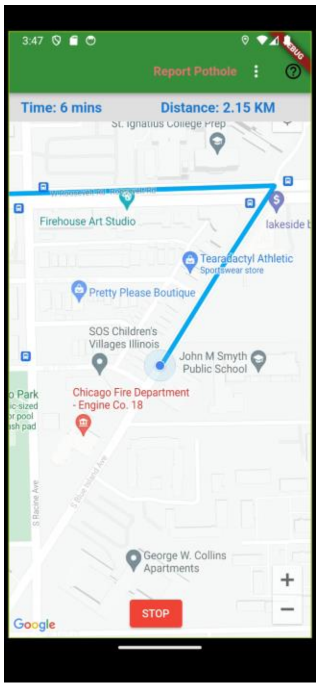
UIC College Prep, South Damen Avenue, Chicago, IL, USA

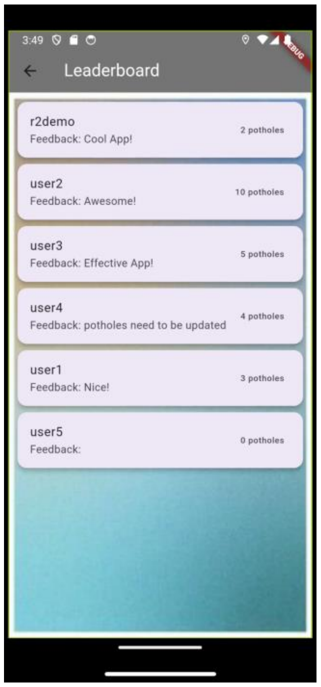
UIC College of Pharmacy, South Wood Street, Chicago, IL, USA

UIC College of Nursing, South Damen Avenue, Chicago, IL, USA

UIC College of Medicine Research Building, South Wolcott Avenue, Chicago, IL, USA

Figure 9


3:47
Report Pothole
Time: 6 mins
Distance: 2.15 KM
St. Ignatius College Prep
Firehouse Art Studio
lakeside
Tearadactyl Athletic
Sportswear store
Pretty Please Boutique
SOS Children's Villages Illinois
John M Smyth Public School
Chicago Fire Department - Engine Co. 18
George W. Collins Apartments
STOP
Google



r2demo Feedback: Cool App!	2 potholes
user2 Feedback: Awesome!	10 potholes
user3 Feedback: Effective App!	5 potholes
user4 Feedback: potholes need to be updated	4 potholes
user1 Feedback: Nice!	3 potholes
user5 Feedback:	0 potholes

Figure 10 Figure 11

This release aims to polish out the UI of our application to give it a better feel as well as release some key features of the application.

The following features were added for release 2 of Pothole Detection System:

Login System

The user can sign up to our application using a username and password. The data for this user, ie. the number of potholes they have reported, is kept secure in the backend. (Figure 7)

Destination autocomplete

Instead of inputting a coordinate string like in release 1 of PDS, the user is now able to input the address they are searching for, and our autocomplete feature will fill in the rest. (Figure 8)

Distance and Time

Once the user has inputted the address they are looking for, the application will display the distance of the route, as well as the estimated time taken of the journey. (Figure 9)

Leaderboard

After the user has completed their ride, they can check the leaderboard to see how many potholes they have reported. This screen will show the user the top contributors to out application.

7 Comparison with Original Project Design Document

The Pothole Detection System application is built on an original project idea from Group 7 of the Fall 2022 semester of CS 440 at the University of Illinois at Chicago. The group consisted of Mohammed Hisham Moizuddin, Rahul Kakarlapudi, Mohammed Hashmeh, and Surya Gowda. The original idea of the Pothole Detection System was a hardware device that is attached to a car, and while driving it is able to detect potholes in roads. There was an application that was built with this device.

In our iteration, we've taken a significant leap by transitioning from a hardware-reliant model to a purely software-based approach. Our refined application stands out as a standalone tool, designed to seamlessly integrate with users' daily commutes. It not only navigates users to their destinations but also proactively alerts them about potential potholes along their route. The most revolutionary aspect of our design is its reliance on community-driven data rather than a physical device. Potholes are identified and logged into the system through user contributions, fostering a collaborative approach to road safety.

This shift from a hardware-dependent system to a user-centric, application-focused solution not only enhances accessibility but also broadens the scope of the project. By leveraging the power of community inputs, we aim to create a constantly updating, dynamic database of road conditions. This ensures that our Pothole Detection System remains a relevant and invaluable tool for drivers, contributing to safer and more informed driving experiences.

III Testing

8 Items to be Tested

For the Pothole Detection System (PDS), the following items will undergo a series of tests to ensure reliability, functionality, and user satisfaction. The tests will be categorized under front-end and back-end to provide a structured testing approach, covering all components of the system.

1. **Front-End:**

   a. Application Launch: Verify the PDS application launches correctly on different devices and operating systems.

   b. User Interface Integrity: Test all user interface elements for responsiveness and interaction. This includes buttons, text fields, dialog boxes, and navigational components.

   c. Map Integration: Confirm the Google Maps integration is functioning as intended, with accurate map rendering and interaction capabilities.

   d. Map Functionality Tests (Marker Placement): Ensure that pothole markers are accurately placed according to geolocation data.

   e. Route Generation: Verify that the system provides a route avoiding reported potholes.

   f. Dynamic Updates: Test the application's ability to receive and display real-time updates about potholes.

   - **g.** User Feedback System: Check the functionality of the feedback submission process and its subsequent storage for further review.
   - **h.** Navigation Flow: Validate the navigation flow from the home screen to other parts of the application like reporting potholes, viewing the leaderboard, and submitting feedback.

2. **Back-End:**

   - **a.** Database Connectivity: Test the connection between the application and Firestore database for stability and data integrity.
   - **b.** Data Retrieval and Storage: Verify that pothole data is correctly retrieved and stored, with checks on data format and consistency.
   - **c.** User Authentication: Ensure the login process, including government and standard user authentication, securely manages user sessions.
   - **d.** Real-time Data Synchronization: Test the system's real-time data synchronization capabilities, ensuring that updates are propagated promptly across all clients.
   - **e.** Leaderboard Functionality: Confirm the leaderboard reflects accurate data regarding user contributions and ranks users correctly based on their reports.

9 Test Specifications

ID# FE-01 - Application Launch

Description: Confirm the PDS application starts properly across platforms.

Items covered: App startup sequence, initial load screen.

Requirements addressed: Req-101 (App Launch and Load).

Environmental needs: Various mobile devices and emulators, different OS versions.

Inter-case Dependencies: NA.

Test Procedures: Install, launch the app, and verify the initial screen is loaded.

Input Specification: NA.

Output Specifications: App loads to the main screen without errors.

Pass/Fail Criteria: The app must load successfully without crashes.

ID# FE-02 - User Interface Functionality

Description: Check responsiveness and functionality of UI elements.

Items covered: Buttons, text fields, alerts, navigation flow.

Requirements addressed: Req-102 (User Interaction).

Environmental needs: Device with touch capabilities, screen reader for accessibility.

Inter-case Dependencies: FE-01.

Test Procedures: Interact with all UI elements, navigate through the app.

Input Specification: Simulated user inputs and gestures.

Output Specifications: Elements respond correctly to inputs; navigation works.

Pass/Fail Criteria: UI functions as designed, with no unresponsive elements.

ID# FE-03 - Map Integration and Functionality

Description: Ensure Google Maps and related features operate correctly.

Items covered: Map loading, marker accuracy, route calculations.

Requirements addressed: Req-103 (Map Rendering), Req-104 (Route Calculation).

Environmental needs: Active internet connection, Google Maps API keys.

Inter-case Dependencies: FE-01.

Test Procedures: Load maps, verify pothole markers, test route suggestions.

Input Specification: Geolocation data for routes and potholes.

Output Specifications: Maps and routes display as expected with real-time data.

Pass/Fail Criteria: Maps load with accurate data, routes avoid potholes.

ID# FE-04 - Route Generation

Description: Confirm routing avoids areas with reported potholes.

Items covered: Route calculation algorithms, user interface map display.

Requirements addressed: Req-105 (Route Optimization).

Environmental needs: GPS functionality, Google Maps API access.

Inter-case Dependencies: FE-02, FE-03.

Test Procedures: Input destinations and verify that the generated route circumvents potholes.

Input Specification: Start and end points with known pothole locations.

Output Specifications: A route that navigates around the potholes.

Pass/Fail Criteria: System must provide a pothole-free route when possible.

ID# FE-05 - Dynamic Updates

Description: Assess the system's handling of live updates on pothole information.

Items covered: Real-time data fetching and display on the map.

Requirements addressed: Req-106 (Real-Time Information Display).

Environmental needs: Stable internet connection, real-time database updates.

Inter-case Dependencies: BE-03.

Test Procedures: Update pothole data and check for immediate reflection in the app.

Input Specification: Changes in pothole status (reported, fixed).

Output Specifications: Updated pothole information on the user's map.

Pass/Fail Criteria: App must reflect changes within a specified time frame.

ID# FE-06 - User Feedback System

Description: Validate the feedback submission and data storage process.

Items covered: Feedback form, database write operations.

Requirements addressed: Req-107 (User Feedback Integration).

Environmental needs: Access to Firestore, user input emulation.

Inter-case Dependencies: BE-01, BE-02.

Test Procedures: Submit feedback through the app and verify database entry.

Input Specification: Sample user feedback text.

Output Specifications: Feedback stored and retrievable from the database.

Pass/Fail Criteria: Feedback must be submitted and stored without errors.

ID# FE-07 - Navigation Flow

Description: Ensure smooth transition between different sections of the app.

Items covered: In-app navigation, button links, redirections.

Requirements addressed: Req-108 (App Navigation).

Environmental needs: Different sections of the app available for navigation.

Inter-case Dependencies: FE-01, FE-02, FE-03.

Test Procedures: Navigate through all sections and verify correct transitions.

Input Specification: User actions for navigation.

Output Specifications: Successful screen transitions without errors.

Pass/Fail Criteria: App must navigate to the correct screen on every action.

ID# BE-01 - Database Connectivity and Operations

Description: Validate the stability and integrity of database connections.

Items covered: Data reading, writing, updating.

Requirements addressed: Req-201 (Data Management).

Environmental needs: Firestore instance, network access.

Inter-case Dependencies: NA.

Test Procedures: Execute CRUD operations on database.

Input Specification: Test data sets for CRUD operations.

Output Specifications: Data is accurately manipulated in the database.

Pass/Fail Criteria: Data operations must complete without errors.

ID# BE-02 - User Authentication Process

Description: Confirm secure handling of user authentication.

Items covered: Login, session management, error handling.

Requirements addressed: Req-202 (Authentication).

Environmental needs: Secure HTTPS connection, user credentials.

Inter-case Dependencies: BE-01.

Test Procedures: Attempt logins with various user credentials.

Input Specification: Valid/invalid usernames and passwords.

Output Specifications: Access granted/denied based on credentials.

Pass/Fail Criteria: Valid credentials must provide access; invalid ones must not.

ID# BE-03 - Real-time Data Synchronization

Description: Check the immediate updating of shared data across clients.

Items covered: Real-time data streaming, update broadcasts.

Requirements addressed: Req-203 (Data Synchronization).

Environmental needs: Multiple client instances, Firestore triggers.

Inter-case Dependencies: BE-01, FE-03.

Test Procedures: Perform data updates and check for propagation.

Input Specification: Updates to pothole reports, user feedback.

Output Specifications: Updates reflected across all client instances promptly.

Pass/Fail Criteria: Real-time data sync without lag or data loss.

ID# BE-04 - Leaderboard Functionality

Description: Confirm leaderboard accuracy and user rank display.

Items covered: Leaderboard data retrieval, user rank calculations.

Requirements addressed: Req-204 (Leaderboard Accuracy).

Environmental needs: Access to Firestore, sorted data retrieval.

Inter-case Dependencies: BE-02.

Test Procedures: Verify that the leaderboard displays the correct user data and rankings.

Input Specification: Database with user contribution data.

Output Specifications: Leaderboard reflects the correct rankings.

Pass/Fail Criteria: Leaderboard must accurately reflect user reports and rankings.

10 Test Results

ID# FE-01 - Application Launch

Date(s) of Execution: November 22nd, 2023

Staff conducting tests: Shehriar Burney, Revathi Dhotre

Expected Results: PDS app launches without error across multiple devices.

Actual Results: App launched successfully on Android. Minor delay noticed on older Android device.

Test Status: Pass (Note: Investigate optimization for older devices)

ID# FE-02 - User Interface Integrity

Date(s) of Execution: Weeks 5-8 of classes - Fall 2023

Staff conducting tests: Shehriar Burney

Expected Results: Responsive UI elements with correct interaction feedback.

Actual Results: All UI components responsive.

Test Status: Pass

ID# FE-03 - Map Integration

Date(s) of Execution: Weeks 5-8 of classes - Fall 2023

Staff conducting tests: Shehriar Burney

Expected Results: Google Maps renders accurately with interactive features enabled.

Actual Results: Map rendered correctly, but zoom controls were initially but fixed afterwards.

Test Status: Pass

ID# FE-04 - Route Generation

Date(s) of Execution: Weeks 5-8 of classes - Fall 2023

Staff conducting tests: Shehriar Burney, Rimsha Rizvi

Expected Results: App provides a route that avoids reported potholes.

Actual Results: App successfully generated a route avoiding all reported potholes.

Test Status: Pass

ID# FE-05 - Dynamic Updates

Date(s) of Execution: October 5th 2023, November 2nd 2023, November 23rd 2023

Staff conducting tests: Rimsha Rizvi, Neel Patel

Expected Results: Real-time updates about potholes are received and displayed.

Actual Results: App displayed updates instantly with a delay of 1-2 seconds.

Test Status: Passed after initial fail.

ID# FE-06 - User Feedback System

Date(s) of Execution: October 5th 2023

Staff conducting tests: Neel Patel, Revathi Dhotre

Expected Results: User feedback is submitted and stored correctly.

Actual Results: Feedback submission feature worked as expected, with all entries visible in the database.

Test Status: Pass

ID# FE-07 - Navigation Flow

Date(s) of Execution: Weeks 5-13 of classes – Fall 2023

Staff conducting tests: Shehriar Burney, Revathi Dhotre

Expected Results: Seamless navigation between app sections.

Actual Results: Navigation was intuitive and without errors, though one minor UI glitch was noted and logged.

Test Status: Pass

ID# BE-01 - Database Connectivity

Date(s) of Execution: Weeks 5-8 of classes – Fall 2023

Staff conducting tests: Rimsha Rizvi

Expected Results: Stable and secure connection with Firestore database.

Actual Results: Connection established with no data leaks; latency within acceptable limits.

Test Status: Pass

ID# BE-02 - Data Retrieval and Storage

Date(s) of Execution: Weeks 5-8 and 10-13 of classes – Fall 2023

Staff conducting tests: Rimsha Rizvi, Neel Patel

Expected Results: Correct retrieval and storage of pothole data.

Actual Results: Data retrieval was 100% accurate

Test Status: Pass (Note: Add further security in storage)

ID# BE-03 - User Authentication

Date(s) of Execution: August 23, 2023

Staff conducting tests: Revathi Dhotre, Neel Patel

Expected Results: Secure management of user sessions for both standard and government users.

Actual Results: Both users managed and accessed accurately, however, for the data to be even more secure, security is necessary for both correct user management and access.

Test Status: Pass for accessing and Fail for the urgent action needed: manage security in login details.

ID# BE-04 - Leaderboard Functionality

Date(s) of Execution: Weeks 8-11 of classes – Fall 2023

Staff conducting tests: Rimsha Rizvi, Shehriar Burney

Expected Results: Accurate reflection of user contributions and rankings.

Actual Results: Leaderboard rankings matched the expected results of potholes reported in descending order.

Test Status: Pass

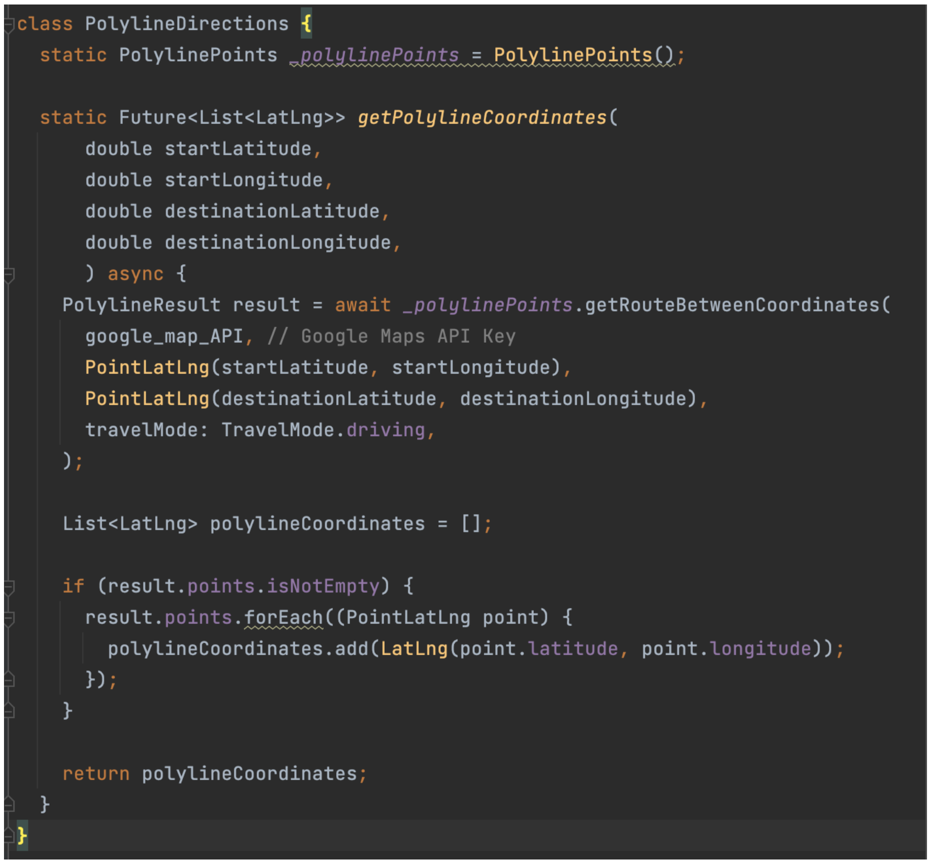
11 Regression Testing

Not required for this project because adding new features shouldn't affect the operation of the fundamental functions, like reporting potholes or marking them as fixed. These basic functions should operate correctly.

IV Inspection

12 Items to be Inspected

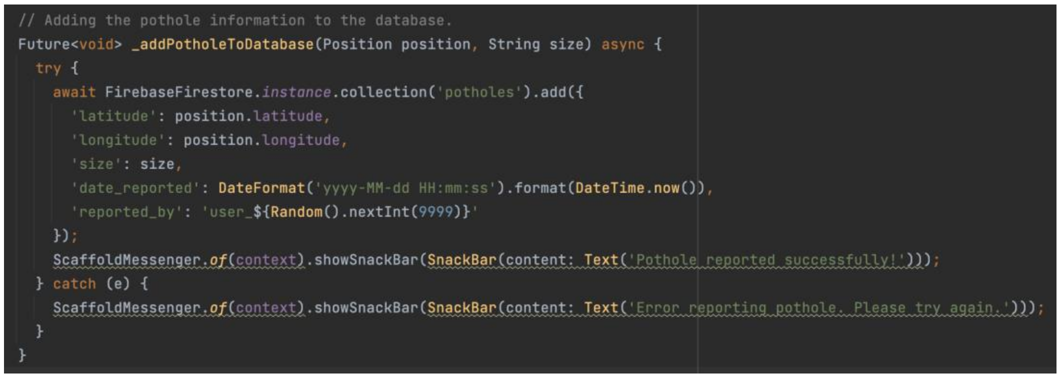
This section of code is from the polylines_directions.dart file and is significant as it is this class contains methods to interact with the Google Maps API, fetching route coordinates between two points. **getPolylineCoordinates**: Connects to Google Maps API to retrieve a set of coordinates forming a polyline between a start and destination point to give the user the most optimal route which is called on the map.dart file to provide time and distance between the route as well.



```
class PolylineDirections {
  static PolylinePoints _polylinePoints = PolylinePoints();

  static Future<List<LatLng>> getPolylineCoordinates(
      double startLatitude,
      double startLongitude,
      double destinationLatitude,
      double destinationLongitude,
      ) async {
    PolylineResult result = await _polylinePoints.getRouteBetweenCoordinates(
      google_map_API, // Google Maps API Key
      PointLatLng(startLatitude, startLongitude),
      PointLatLng(destinationLatitude, destinationLongitude),
      travelMode: TravelMode.driving,
    );

    List<LatLng> polylineCoordinates = [];

    if (result.points.isNotEmpty) {
      result.points.forEach((PointLatLng point) {
        polylineCoordinates.add(LatLng(point.latitude, point.longitude));
      });
    }

    return polylineCoordinates;
  }
}
```

Figure 12

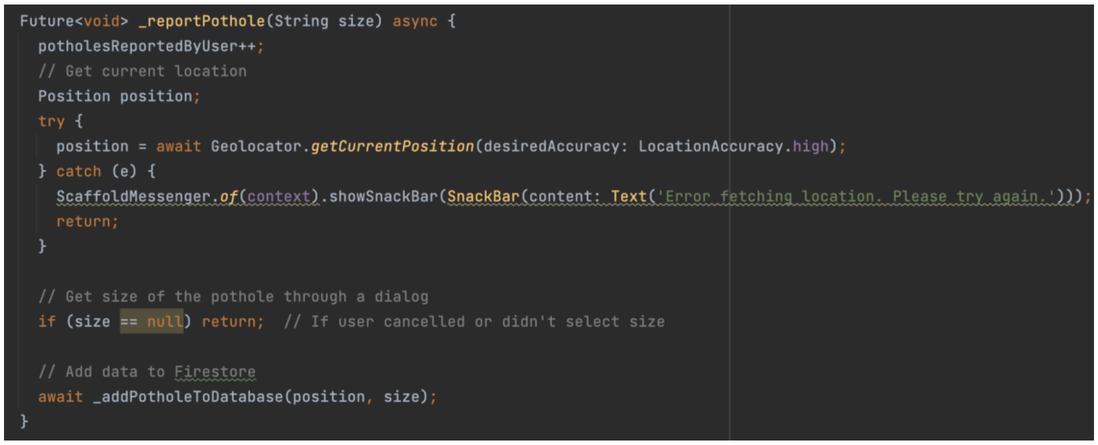
This function is responsible for adding reported pothole details into the Firestore database. The function _addPotholeToDatabase takes a geolocation position and pothole size, and adds an entry to the 'potholes' collection in Firestore.



```
// Adding the pothole information to the database.
Future<void> _addPotholeToDatabase(Position position, String size) async {
  try {
    await FirebaseFirestore.instance.collection('potholes').add({
      'latitude': position.latitude,
      'longitude': position.longitude,
      'size': size,
      'date_reported': DateFormat('yyyy-MM-dd HH:mm:ss').format(DateTime.now()),
      'reported_by': 'user_${Random().nextInt(9999)}'
    });
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Pothole reported successfully!')));
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Error reporting pothole. Please try again.')));
  }
}
```

Figure 13

This state class for ReportPothole widget, which includes methods to report a pothole and interact with the Firestore database. The function reportPothole: Initiates the process of reporting a pothole by capturing the current location and prompting for size input.



```
Future<void> _reportPothole(String size) async {
  potholesReportedByUser++;
  // Get current location
  Position position;
  try {
    position = await Geolocator.getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Error fetching location. Please try again.')));
    return;
  }

  // Get size of the pothole through a dialog
  if (size == null) return;  // If user cancelled or didn't select size

  // Add data to Firestore
  await _addPotholeToDatabase(position, size);
}
```

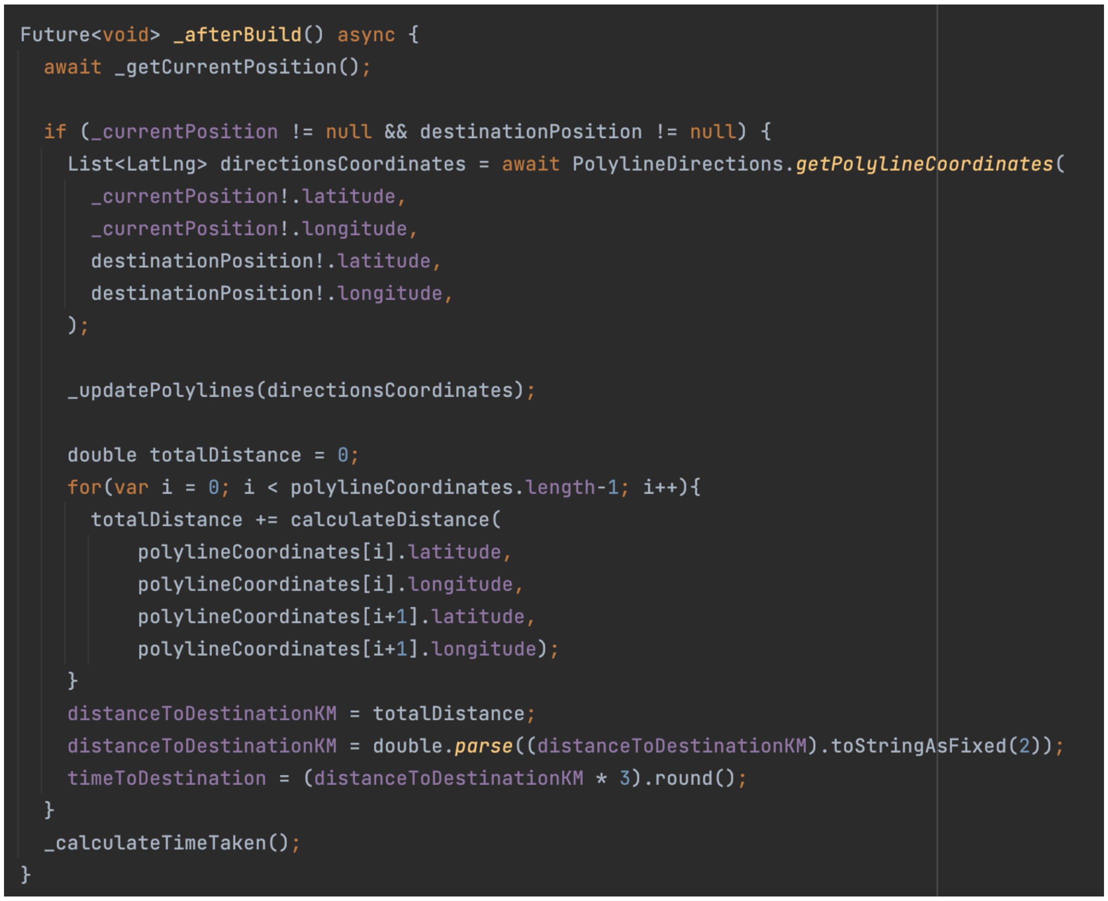
Figure 14

This is a Post-build method to calculate and update route information in the MapScreen state class. Functions:

_afterBuild: A lifecycle method that's called to calculate the route after the build phase.

_updatePolylines: Updates the polyline data on the map to represent the route

_calculateTimeTaken(): Calculates the time taken from source location to destination location



```
Future<void> _afterBuild() async {
  await _getCurrentPosition();

  if (_currentPosition != null && destinationPosition != null) {
    List<LatLng> directionsCoordinates = await PolylineDirections.getPolylineCoordinates(
      _currentPosition!.latitude,
      _currentPosition!.longitude,
      destinationPosition!.latitude,
      destinationPosition!.longitude,
    );

    _updatePolylines(directionsCoordinates);

    double totalDistance = 0;
    for(var i = 0; i < polylineCoordinates.length-1; i++){
      totalDistance += calculateDistance(
          polylineCoordinates[i].latitude,
          polylineCoordinates[i].longitude,
          polylineCoordinates[i+1].latitude,
          polylineCoordinates[i+1].longitude);
    }
    distanceToDestinationKM = totalDistance;
    distanceToDestinationKM = double.parse((distanceToDestinationKM).toStringAsFixed(2));
    timeToDestination = (distanceToDestinationKM * 3).round();
  }
  _calculateTimeTaken();
}
```

These functions are critical for achieving the PDS application's core functionalities, such as navigating while avoiding potholes, reporting new potholes, and dynamically updating route information. They interact with both the Google Maps API and Firestore database, emphasizing the importance of their proper inspection to ensure the robustness of the application.

13 Inspection Procedures

For the Pothole Detection System (PDS) project, our team, consisting of Neel Patel, Revathi Dhotre, Shehriar Burney, and Rimsha Rizvi, implemented a robust inspection procedure modeled on agile methodologies. We leveraged Jira for task management and sprint planning, with each sprint spanning a one-week period. This allowed for clear delineation of responsibilities and deadlines.

Our inspection procedures included both individual and collaborative elements. We adopted a pair programming approach, facilitated through meeting up in person and through Discord. This collaborative strategy ensured that at least two team members worked on each task, promoting knowledge sharing and immediate problem-solving.

As a result, roughly 60% of coding was performed collaboratively, enhancing code quality and team cohesion.

The remaining 40% of the development time was allocated to individual research, testing, and preliminary code reviews before embarking on pair programming sessions. These sessions were vital for laying the groundwork for efficient coding sprints.

Weekly meetings were held to assess progress, discuss challenges, and inspect the code developed during the week. Additional ad-hoc meetings were convened as needed to ensure continuous alignment with project goals. All discussions and code inspections were conducted in person during our weekly meetings and electronically via Discord and Jira, with detailed records maintained for reference and accountability.

Checklists, created and shared within the team, were used to streamline the inspection process, covering code quality, functionality, integration, and user experience aspects. These checklists were based on best practices and tailored to the specific needs of the PDS project. The checklists and outcomes from all inspections were documented and are available in the project appendix or as a separate attachment upon request.

Our inspection procedures not only ensured that individual contributions aligned with the project's standards but also fostered an environment of continuous feedback and improvement. Through this process, we were able to maintain a high level of code quality and project alignment.

14 Inspection Results

For the Pothole Detection System (PDS), our team conducted thorough inspections of our codebase and features. The following documents the results of these inspections.

Polyline Route Generation Inspection

Inspected by: Shehriar Burney, Rimsha Rizvi

Expected Results: Generation of polyline coordinates between two points.

Actual Results: Correct polyline coordinates were generated; minor discrepancies in route optimization were noted.

Test Status: Pass with notes for optimization improvements.

Database Pothole Data Addition Inspection

Inspected by: Rimsha Rizvi, Neel Patel

Expected Results: Successful addition of pothole reports to Firestore database.

Actual Results: Pothole reports were successfully added; error handling for failed network requests was implemented following inspection.

Test Status: Pass after initial re-inspection and improvement.

Report Pothole Function Inspection

Inspected by: Rimsha Rizvi, Shehriar Burney

Expected Results: Pothole reporting through UI dialog and subsequent database update.

Actual Results: Pothole reporting function operational with some UI delays.

Test Status: Pass, with UI responsiveness marked for future improvement.

MapScreen Polyline Update Inspection

Inspected by: Shehriar Burney

Expected Results: Real-time updating of route polylines based on user movement.

Actual Results: Updates functional but with occasional lags on certain devices.

Test Status: Pass, pending performance optimization.

Re-inspections occurred as part of our agile process, whenever a flaw was discovered. These were addressed in subsequent pair programming sessions, with solutions documented and shared with the team. The Jira board was updated to reflect the status of these issues, ensuring visibility and tracking of the progress made. All results and discussions were shared electronically, primarily via Discord, which facilitated immediate feedback and accelerated resolution of identified issues.

V Reccomendations and Conclusions

Upon review, the Pothole Detection System has successfully passed all of its testing and inspection processes, particularly in functionality and user interface response. However, security testing has revealed areas that require further attention. While user authentication and data encryption standards meet the current requirements, we recommend a more rigorous security audit to ensure the protection of user data and system integrity against potential breaches. Next steps include implementing enhanced security measures, such as two-factor authentication and regular vulnerability scans, followed by re-evaluation to ensure compliance with the latest security standards before final implementation.

VI Project Issues

15 Open Issues

For the Pothole Detection System (PDS), several open issues need to be addressed. The accuracy of pothole detection through user reports is one such concern, as this relies heavily on user engagement and their accurate reporting. Another issue is the app's performance across different devices, which has shown inconsistency, particularly in older models. We also face uncertainty regarding upcoming API changes by Google Maps, which could affect map functionality.

16 Waiting Room

The integration of more advanced predictive routing to avoid not just known potholes but also areas prone to potholes based on weather and traffic data is in our waiting room of features. We are also considering community-driven features, like road quality voting, which are not in the current scope due to time constraints but may significantly enhance user engagement and data accuracy in future releases.

17 Ideas for Solutions

Potential solutions for enhancing the PDS include the use of machine learning algorithms to predict pothole formation, which would require a more robust data collection and analysis framework. For improving user engagement, gamification elements could be introduced. Additionally, exploring partnerships with local municipalities for data sharing and verification could strengthen the system's reliability.

18 Project Retrospective

Reflecting on the PDS project, the use of agile methodology worked well in managing tasks and keeping the team aligned. However, the underestimation of the complexity involved in integrating various APIs led to delays. In the future, allocating more time for research and testing, particularly for new technologies, would be beneficial. Better documentation at the start could have streamlined the learning process for team members unfamiliar with certain aspects of the development stack.

VII Glossary

- **Pothole Detection System (PDS):** An application that integrates real-time data to alert users of potholes and road hazards during their travels.
- **User:** Any individual who utilizes the PDS app for navigation and pothole reporting.
- **Google Maps API:** A set of functions and procedures offered by Google that allows the retrieval of mapping data and can be integrated into apps or websites.
- **Backend Database:** A centralized database that the PDS app queries to fetch and store data related to potholes' locations and sizes.
- **Firebase Cloud Database:** A cloud-hosted NoSQL database that stores and synchronizes data between users in real-time, making it an ideal solution for mobile apps like PDS where timely data updates are crucial.
- **Pothole Reporting:** The process by which a user submits a new or existing pothole's location and size, contributing to the system's database.
- **Testing:** The process of executing the application to verify that it functions according to the specified requirements.
- **Inspection:** The process of reviewing code, design, and system architecture to identify potential issues before the testing phase.
- **UI (User Interface):** The space where interactions between humans and the PDS application occur.
- **Real-Time Alerts:** Notifications sent to the user without any delay as they navigate, informing them of upcoming potholes or road hazards.
- **Route Optimization:** The process by which the PDS application calculates the most efficient path to a destination while considering the locations of reported potholes.
- **Municipal Collaboration:** The cooperative effort between the PDS system and local government entities to address and fix reported road issues.

VIII References / Bibliography

[1] Bell, J. (2012). *Underwater Archaeological Survey Report Template: A Sample Document for Generating Consistent Professional Reports.* Chicago: Underwater Archaeological Society of Chicago.

[2] Gowda, S., Hashmeh, M., Moizuddin, M.H., Kakarlapudi, R. (2022). Pothole Detection System. University of Illinois at Chicago

[3] Fowler, M. (2004). *UML Distilled, Third Edition.* Boston: Pearson Education.

[4] Robertson, & Robertson. (n.d.). *Mastering the Requirements Process.*

[5] Silberschatz, A., Galvin, P. B., & Gagne, G. (2013). *Operating System Concepts* (Ninth ed.). Wiley.

IX Index