MEDVAN: A SMART AMBULANCE BOOKING APPLICATION WITH REAL-TIME TRACKING

A PROJECT REPORT

Submitted by

SOURAV YADAV

Registration No.: 231000510715

Roll No.: 10071023021

Supervised by

Mr. Sanchayan Bhaumik

in partial fulfillment for the award of the degree

of

MASTER OF COMPUTER APPLICATIONS

IN

DEPARTMENT OF COMPUTER APPLICATIONS

Year: 2023 – 2025

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

Nadia, West Bengal, India

Dedicated to my Parents

BONAFIDE CERTIFICATE

Certified that this project report **"MEDVAN: A SMART AMBULANCE BOOKING APPLICATION WITH REAL-TIME TRACKING"** is the bonafide work of **"SOURAV YADAV"** who carried out the project work under my supervision.

Dr. Pabitra Pal
HEAD OF THE DEPARTMENT
Assistant Professor
Department of Computer Applications
Maulana Abul Kalam Azad University of Technology

Mr. Sanchayan Bhaumik
SUPERVISOR
Assistant Professor
Department of Computer Applications
Maulana Abul Kalam Azad University of Technology

SIGNATURE
External Examiner

DECLARATION

I hereby declare that this submission is my own work and, to the best of my knowledge and belief, it contains no material previously published or written by any other person, nor does it contain any material which has been accepted for the award of any other degree or diploma from any university or institution of higher learning, except where due acknowledgment has been made in the text.

_____

Sourav Yadav

ACKNOWLEDGEMENT

I would like to take this opportunity to thank all the people who have helped and advised me in the course of developing my MCA final year project, which is titled, **MedVan: A Smart Real-Time Ambulance Booking and Tracking Application**. The project has been a learning experience and I am really grateful to all those who helped in its completion through their advice, encouragement, and help.

I would like to thank my project guide **Mr. Sanchayan Bhaumik**, Assistant Professor, to whom I owe my everlasting gratitude due to their continuous support, helpful comments, and advice, recommendations at each level of the project. Their advice has been very helpful in molding the direction and performance of this work.

I am also deeply grateful to **Dr. Pabitra Pal**, Head of the Department, Department of Computer Applications of Maulana Abul Kalam Azad University of Technology, WB.

I would also like to express my gratitude to all the faculty members of the department in terms of their academic assistance and positive criticism in the internal reviews and discussions.

Finally, I would like to thank my family with all my heart because of their constant support, patience, and support during my academic life and in this project.

The successful completion of this project would not have been possible without the overall support, encouragement and guidance of all the above-mentioned individuals.

Date _____ _____

With Sincere

Sourav Yadav

TABLE OF CONTENTS

ABSTRACT

In the current busy world, availability of prompt medical help in case of an emergency is very important in saving lives. Nevertheless, the time spent to find and reserve a free ambulance, inability to track in real-time, and ineffective communication between a patient and an ambulance driver remain significant issues in emergency care provision. This project, which is named MedVan: A Smart Ambulance Booking Application with Real-Time Tracking, will solve these problems by offering a stable, convenient Android-based mobile application that will allow booking an ambulance and tracking it in real-time.

MedVan is created on Java and Firebase and is intended to be used on Android devices through Android Studio. The system consists of two distinct interfaces, one of a patient and the other one of an ambulance driver. The main functionalities will be user authentication, booking of the nearest available ambulance, the real-time location tracking of the ambulance using Google Maps API, and automatic ride assignment to the nearest ambulance using location information.

The project will help to close the gap between patients and emergency medical transport services by using the power of mobile technology and cloud-based solutions. In addition to enhancing the effectiveness of the ambulance services in terms of their operations, MedVan also plays a great role in saving lives as it helps to eliminate the delays in critical cases.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
GPS	Global Positioning System
ETA	Estimated Time of Arrival
UI	User Interface
API	Application Programming Interface
SDK	Software Development Kit
FCM	Firebase Cloud Messaging
ERD	Entity Relationship Diagram
IDE	Integrated Development Environment
OTP	One-Time Password
DB	Database

CHAPTER 1: INTRODUCTION

Time is critical in case of emergency medical care. Even several minutes of delay in receiving medical assistance may have life-threatening outcomes. Even with the current technology, the system of booking an ambulance in most of India is still not efficient as it is associated with a manual system of calling, absence of real-time tracking, and poor communication between the patient and the ambulance service. This usually causes delays, confusion and sometimes lack of ambulance services.

In order to overcome these obstacles, MedVan application has been designed as an intelligent, Android-based mobile application to simplify the process of booking ambulances. It allows the users to order the closest available ambulance in real-time with minimal response time, so that the patient receives critical care as soon as possible.

1.1 Statement of Problem

Current ambulance services are usually not transparent, real-time tracked, and digital. Patients have problems with finding the nearest ambulances, calculating the time of arrival, and getting immediate assistance. Lack of a combined platform between patients and ambulance drivers leads to inefficiency, time wastage, and loss of precious time in case of an emergency.

1.2 Project objectives

The MedVan project aims at the following purposes:

- To create an Android app that will enable people to book ambulances easily and fast.
- To introduce real-time tracking of the location of users and ambulance drivers.
- To automate the process of assigning ride by locating and notifying the closest available driver.
- To incorporate Firebase to exchange data in real-time, authentication, and notification services.
- To enhance the efficiency of emergency response and shorten the time of arrival of ambulances.

1.3 Motivation

MedVan came up with the idea of the growing demand of faster and more reliable medical transport services, particularly in the urban and semi-urban regions. The delays in the arrival of the ambulance in the case of critical emergencies as reported by media and personal experiences inspired the creation of this project. The idea is to leverage the latest mobile and cloud technologies to develop a platform that will allow saving lives by decreasing the time spent on emergency medical services.

1.4 Project scope

MedVan project is concerned with:

- Android studio development in Java.
- Real time tracking through Google Maps API.
- Integration with Firebase Realtime Database, Firebase Authentication.
- Distinctive user interface between patients (seekers of ambulance) and drivers (providers of ambulance).
- Automatic choice and alert of the closest driver.
- Auto-ride acceptance by the nearest driver.

The project is now facilitating the major features of booking, tracking, ride assigning, and authentication. Future improvements can be the integration of hospitals, payment gateways and an admin dashboard.

1.5 Users Profile

The app is reserved to:

- Patients or caregivers who require emergency services of ambulances.
- Registered ambulance drivers that are ready to be dispatched.

1.6 Organization of Chapter

This thesis is divided into several chapters to provide the development process of the MedVan application in a systematic and logical way.

- Chapter 1 presents the problem, project motivation, objectives, scope and user profiles.
- Chapter 2 provides the market survey and literature review, which examines the existing systems, tracking technologies and research gaps.
- Chapter 3 describes the adopted methodology including system analysis, design, functional modules, system diagrams, user interfaces, implementation steps and software development paradigm.
- Chapter 4 explains the experimental outcomes, hardware/software needs, the performance indicators, user comments, and limitations that were encountered.
- Chapter 5 sums up the project, talks about its successes and opportunities to improve it in the future.

CHAPTER 2: MARKET SURVEY

2.1 Introduction

It assists in the comprehension of the current systems, their weaknesses, and the way to innovate. This chapter discusses the literature review, the existing technologies and the available mobile applications in the field of ambulance booking, emergency response and real-time tracking systems. The lessons learnt in this review have informed the design and development of the MedVan application.

2.2 Current Systems and Constraints

A number of ambulance booking systems have been created in the last couple of years. Some of them are such:

- 108 Emergency Service (India): It is a government service that enables users to call 108 in case of a medical emergency. Nevertheless, it does not have a booking and tracking feature that is based on a mobile.
- Private Ambulance Apps (such as StanPlus, AMB Life, Red Ambulance): These apps offer ambulance booking services in large cities only and are not very widespread; they usually need manual verification.
- Ola and Uber-style: There are startups that have implemented ride-hailing models to dispatch ambulances, although most of these solutions either need an internet infrastructure that is not necessarily reliable in rural locations or do not provide real-time updates.

Important shortcomings of current systems:

- No real time location sharing between user and driver.
- The absence of an automated driver assignment according to the proximity of location.
- Time lost because of manual communication or operator interference.
- Inadequate cloud technologies integration to synchronize data in real time.

2.3 Tracking technologies in real time

In emergency response systems, real-time tracking is important. Technologies which are widely used are:

- GPS (Global Positioning System): It allows tracking of location accurately.
- Google Maps API: It is used to show routes and ambulance movement on the map interface.
- Firebase Realtime Database / Firestore: Enables real-time transfer of location information between the patient and the driver application.

These technologies provide scalable, reliable, and cloud-based development of responsive mobile applications that need live tracking.

2.4 Related Research Work

The significance of real-time emergency services has been mentioned in a number of research papers both academic and industry-based:

- Emergency Response System Using Mobile and Cloud Technology (International Journal of Computer Applications): Stressed the need to minimize the ambulance response time with the help of mobile-based cloud-integrated applications.
- Smart Ambulance System to Monitor Patients (IEEE): The proposed is a real-time tracking and monitoring of patients health during transportation.
- A Review on Real-Time Tracking and Navigation of Ambulance Systems: Evaluated the issues of using GPS-based ambulance systems in India as a result of traffic jams and the absence of data integration.

These studies gave useful information on the problems and solutions related to the emergency response systems, most of which were put into consideration when developing MedVan.

2.5 Market research

An initial market research was done by examining:

- Reviews of other apps in Google Play Store.
- The needs of the users gathered through informal surveys of peers and local medical practitioners.
- Difficulties experienced in Tier-II and Tier-III cities where the delays in ambulances are more evident.

It was found that the users favored:

- Quick one tap ambulance booking.
- Real-time and the estimated time of arrival (ETA).
- Apps that have low configuration and user-friendly interface.
- Communication with drivers on the basis of notification.

This justified the necessity of such an intuitive and reliable application as MedVan.

2.6 Gap in Research Identified

The gaps identified with the help of the review of the existing systems and research are the following:

- Most systems do not have real-time driver-patient connection.
- Lack of smart ride assignment on the basis of least distance.
- The least use of cloud technologies to synchronize emergency services.
- There is no unified solution to user-side communication and driver-side communication.

MedVan will fill these gaps by employing Firebase to handle real-time data, Google Maps to track in real-time, and a powerful backend to guarantee immediate driver assignment and the ability to see the movement of the ambulance.

CHAPTER 3: METHODOLOGY

3.1 Introduction

This chapter explains the overall methodology followed in the MedVan application. It includes the combined content of system analysis, system design, implementation, and the software engineering paradigm used. This unified presentation provides a comprehensive understanding of how the application was conceptualized, developed, and implemented.

3.2 System Analysis

System analysis involves knowing the functional requirements of a system, what the users expect of it and whether it is possible to design and implement a system. In the framework of the MedVan application, this step was devoted to the examination of the underlying problems that can be identified in the sphere of emergency medical services and the identification of the means through which the proposed system can address these issues. The analysis also includes identification of system requirements, user roles and project feasibility.

3.2.1 Existing System

The reservation of the ambulance in majority of the regions is still through the conventional call-based system or the third-party services which lack substantial real-time tracking facilities. The current systems tend to be problematic as follows:

- Manual intervention and time delay in communication.
- The user is not able to view the location of the ambulance.
- Inability to locate the nearest ambulance at hand.
- Extreme dependence on call centers and manual coordination.

3.2.2 Proposed System

The proposed system, MedVan is a mobile-based Android application that offers:

- Real-time booking of ambulances.
- Real-time monitoring of the patient and the driver.
- The nearest available ambulance auto assigning.
- An easy user-driver interaction via Firebase.
- Send push notifications to drivers in case of a new booking request.

The system is divided into two big sections:

- User App (Patients/Caregivers): It is an application that is utilized to request an ambulance, check the available ambulances in the region, and track their arrival.
- Driver App (Ambulance Drivers): This is an app that is used to auto-accept and receive ride requests and navigate to the user with real-time directions.

3.2.3 Requirements

Functional Requirements:

- Registration and log in of users (Firebase Authentication).
- Booking request interface of ambulance.
- GPS and Google Maps real-time tracking.
- Algorithm of nearest driver detection.
- The driver side interface to input/deny requests.
- Ride status (ride in progress, completed etc.).

Non-Functional Requirements:

- User friendly and accommodating interface.
- Scalable real-time backend infrastructure.
- Safe manipulation of user information.
- Quick response time of booking and tracking facilities.

Hardware Requirement:

- Android smart phone with GPS and internet.
- Minimum of 2 GB RAM to operate the apps without any glitch.

Software Requirement:

- Android Studio IDE.
- Java (Android SDK).
- Firebase services (Authentication, Realtime Database).
- Google Maps API.

3.2.4 Feasibility Study

Technical feasibility: Technically, the system can be implemented using Android Studio, Java, Firebase, and Google Maps APIs that are easy to find and well documented. No special hardware is required other than GPS-enabled smartphones.

Operational feasibility: The patients and the drivers are also comfortable with the application. Training or technical knowledge is not required or minimal. It is easy to use and it enables error-free interaction.

Economical feasibility: Firebase offers a free plan to use authentication and database. The system is affordable to develop and implement because the cost of development and implementation is low as the system is founded on cloud-based infrastructure and open APIs, and thus, it is inexpensive to implement in both academia and practice.

3.2.5 Users and Roles

- Patient/User: Register, log in, book an ambulance, track a ride
- Driver: Sign in, receive requests, and go to user location

3.2.6 SWOT Analysis

- Strengths: On-demand reservation and monitoring, Firebase and Google Maps integration, Simple, convenient interface
- Weaknesses: Internet addiction, Android only at the moment, It can be used only by registered drivers
- Opportunities: Expansion to the rural areas and to hospitals, Possible merger with hospitals or emergency hotlines
- Threats: Competition with the other apps, Cybersecurity and data privacy risks

3.3 System Design

System design is the process of establishing the architecture, components, modules, interfaces and data flow of the system in order to meet the specified functional and non-functional requirements. In the MedVan application, the system has been properly designed to ensure the successful booking of the ambulance, real-time monitoring of the ambulance and the ease of communication with the patient and the driver using the Android smartphones.

3.3.1 Architecture System

MedVan application is a client-server application that has a cloud backend in Firebase. Both Patient App and Driver App are clients who will connect with Firebase, to exchange real-time information.

- Patient App: To reserve an ambulance and the position of the driver.
- Driver App: It is applied to receive requests and locate the route to the patient.
- Firebase Services: It is concerned with Authentication, Realtime Database and Cloud Messaging.
- Google Maps API: It allows the location / navigation to be displayed on the maps.

3.3.2 Functional Modules

Patient user Module:

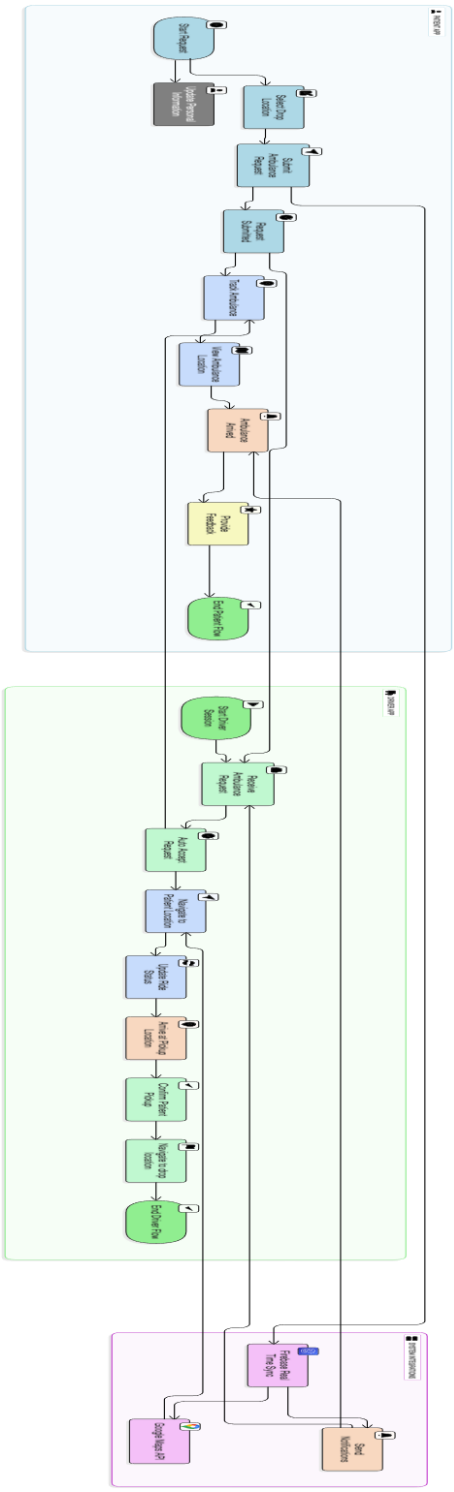
- Registration and Logging in of the User
- Order ambulance by book
- Look at the nearest ambulance
- Tracking of Ambulance location
- Check out the estimated time of arrival (ETA)

Drivers Module:

- Drivers Registration and Logging in
- Requests of Booking Incoming (see)
- Auto Acceptance of Requests
- Visit the Place of a Patient and Orientate
- Modify Ride Status (Accepted, Arrived and Completed)

3.3.3 Use Case Diagram

The Use Case Diagram shows the communication between the two primary actors Patient and Driver with the MedVan system. It emphasizes some of its major features like booking, auto-ride acceptance, location tracking, and ride status updates.

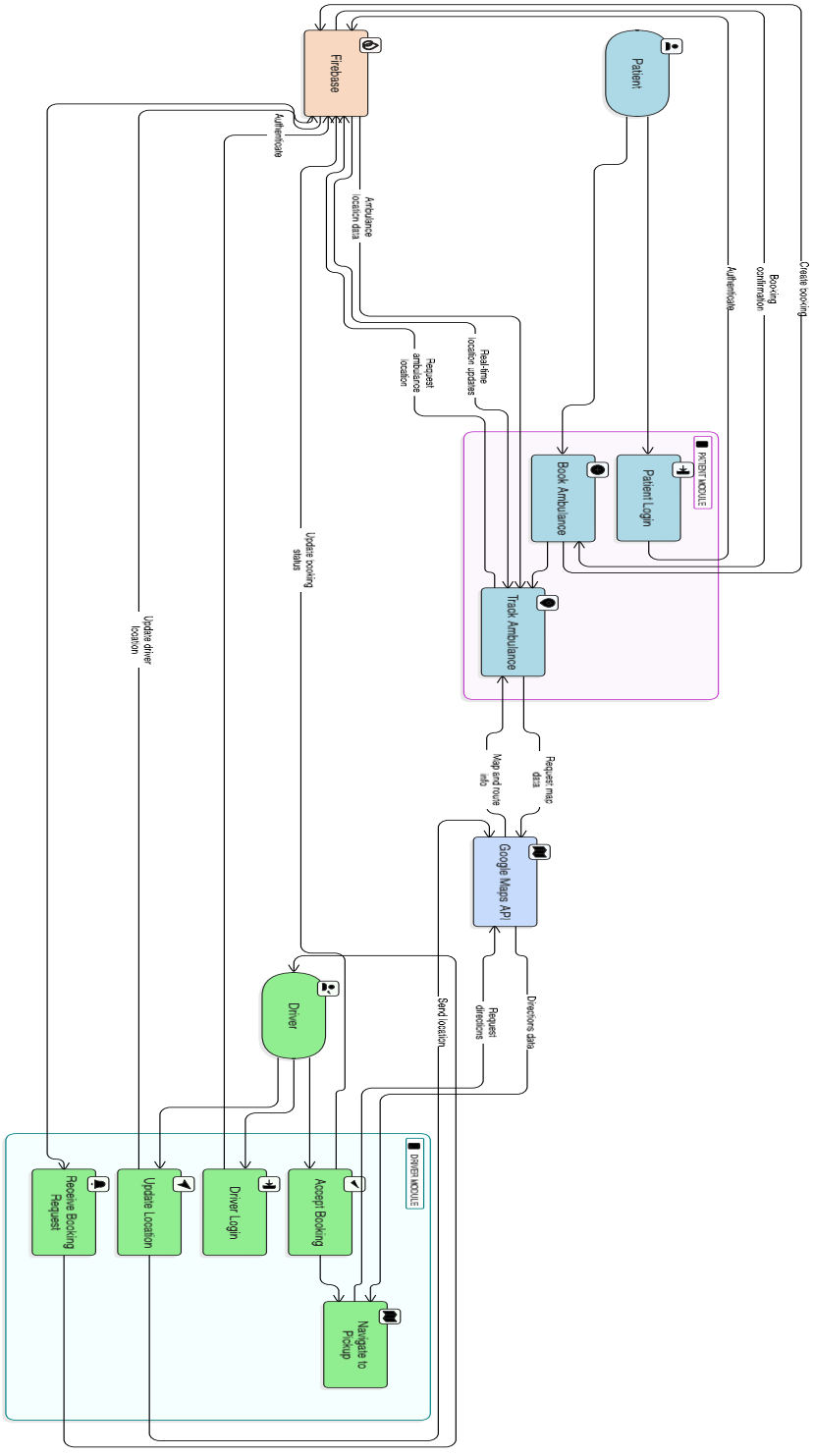


Use Case Diagram

Figure 3.1

3.3.4 Data Flow Diagram (Level 0)

Level 0 DFD is a high-level description of the system that indicates the flow of data among external entities (Patient and Driver) and MedVan system. It describes the simplest input/output procedures such as ambulance booking and answering requests.

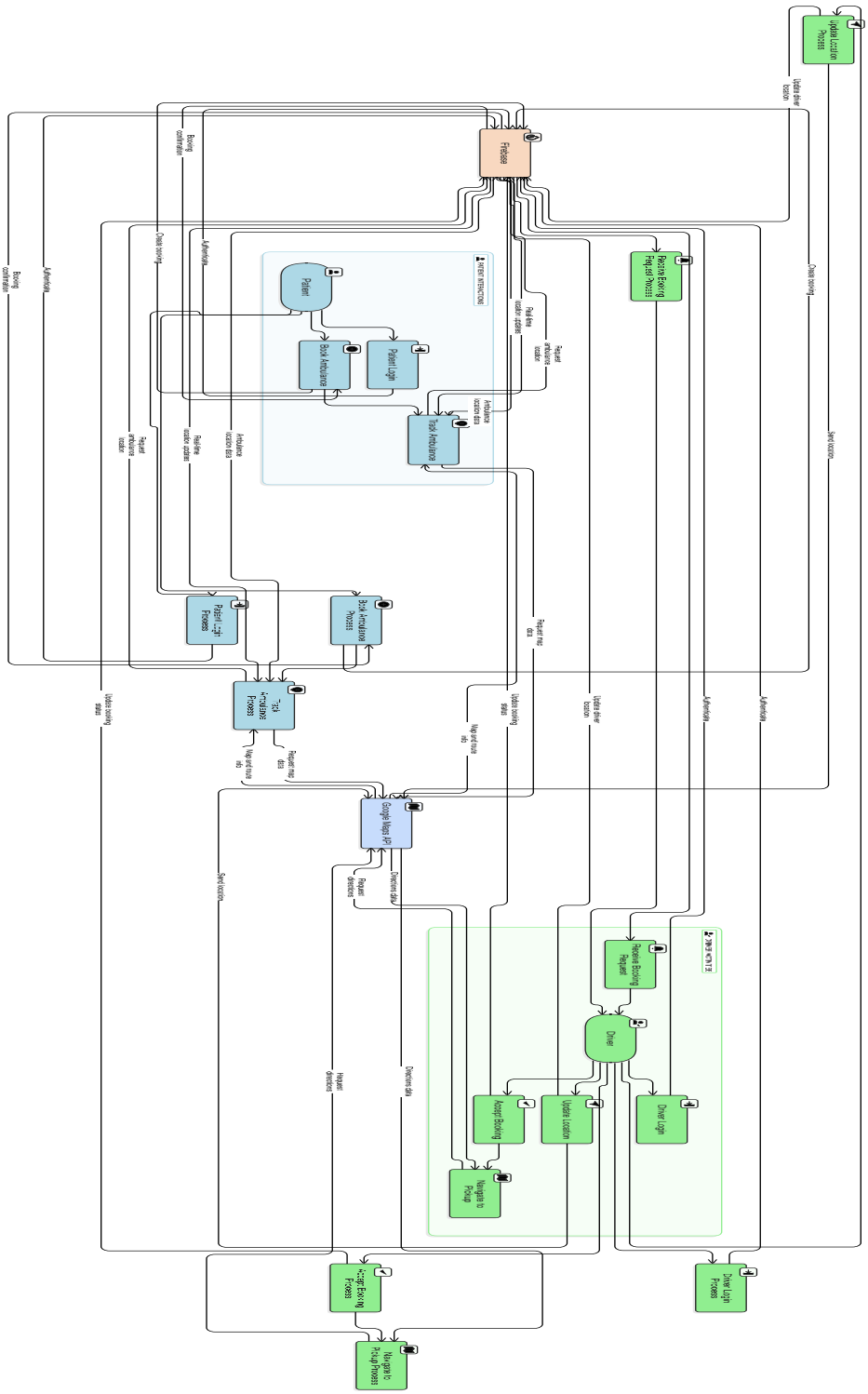


Data Flow Diagram (Level 0)

Figure 3.2

3.3.5 Data Flow Diagram (Level 1)

The Level 1 DFD provides a deeper picture of internal processes of the system. It decomposes the main functions which are booking validation, driver assignment, location tracking, and status updates into sub-processes, as well as data interactions.

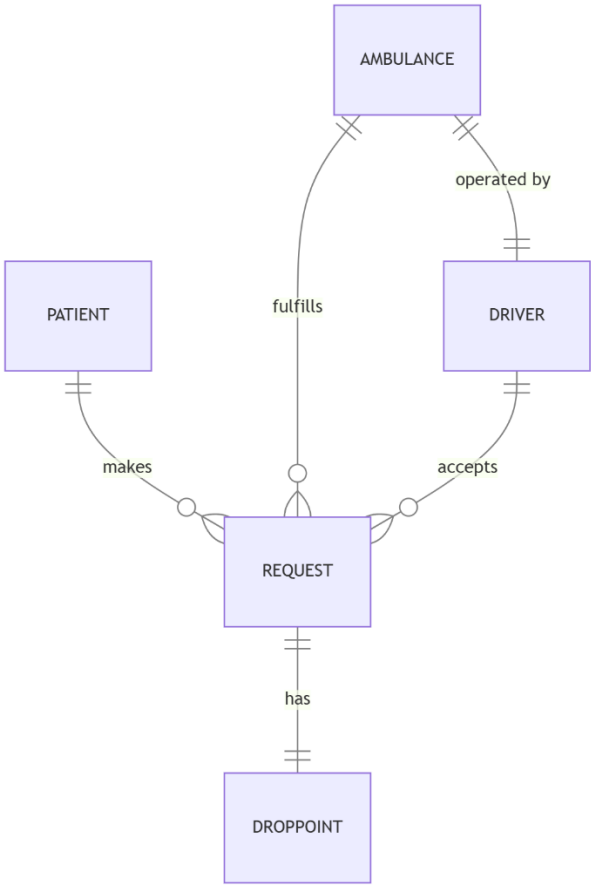


Data Flow Diagram (Level 1)

Figure 3.3

3.3.6 Entity Relationship Diagram (ERD)

The ERD graphically illustrates the most important entities of the MedVan system, including Users, Drivers, and Ride History, as well as the connection between them. It assists in learning how user data, driver profiles and ride records are organized and linked in the Firebase Realtime Database.



Entity Relationship Diagram (ERD)

Figure 3.4

3.3.7 User Interface Design

App UI Screens - Patient:

- Log in / Sign up page: Phone number authentication
- Booking Screen: live map, location picker and button Book Now
- Tracking Screen: The current location of the ambulance as well as the ETA
- Profile/History (not obligatory): see history of rides

UI App Driver Screens:

- Login/ Register Page
- Request Notification Screen - New booking pop up - location
- Navigation Screen: Google Maps to the place of the user
- Ride Status Screen: Ride update (Accepted ---- Arrived --- Completed)

3.3.8 Security Consideration

- Firebase Authentication is applied to ensure that only authenticated users can have access to the app.
- The firebase rules make sure that the information of the user and driver is not retrieved by any unauthorized person.
- The users and the drivers update their location only when they are on an active booking.

3.4 Implementation

The implementation is the stage through which the theoretical design is changed to a practical application. The chapter has described the procedure of building the MedVan system, building of the modules, integrating all of them and testing using Android Studio, Java program, Firebase services, and Google Maps API.

3.4.1 Technology Stack

- UI: Java (Android SDK)
- IDE: Android Studio
- Backend: Firebase (Realtime Database, Auth)
- Direction and Maps: Google Maps API
- Data Storage: Firebase Realtime Database
- Authentication: Firebase Authentication

3.4.2 Project Structure

On the user side: Patient App (User-Side)

- Telephone paid registration/log out
- Locally book ambulance
- View the designated driver and know his/her movement in real time
- Ride status (Accepted, Arrived, Completed)

Ambulance-Side (Driver App):

- Register/Login
- Locate the mapping of the patient in Google Maps
- Real time ride status update

3.4.3 Firebase Integration

Firebase authentication:

- It is deployed in logging in to the secure user on OTP of phone number.
- Carried out the user and driver log in with the help of the various nodes.

Realtime database of firebase:

- Profile of users, availability of drivers, and booking information are stored in stores.
- It has real time sync that implies that both applications are going to be updated in real-time.

3.4.4 Google Maps API Integration

- To display live map in both the apps.
- Shows positions in real time using the GPS of the devices.
- Pulls a route of navigation between the user and the driver once a ride has been accepted.

3.4.5 Debugging and Test Tools

- Logcat: To see the logs of the applications and to debug the errors
- Android Emulator: It is employed on the emulation of different screen size and OS
- Physical Device Testing: Final execution of app on real android smart phones to test GPS and Firebase connection

3.4.6 Challenges Faced

- The management of real-time location changes of the users and drivers simultaneously
- Firebase database rule and security management
- Ensuring that the location tracking is precise and quick even in the case of poor network
- Maximization of battery in long-time GPS tracking

3.5 Software Engineering Paradigm

A software engineering paradigm is a basic way or methodology applied in the process of software development to help in the process of requirements to deployment. To develop MedVan, a real-time ambulance booking and tracking application, a suitable paradigm was necessary in order to achieve reliability, scalability and timely delivery.

3.5.1 The Paradigm Chosen: Incremental Model

The software engineering paradigm that was selected to be used in the MedVan project is the Incremental Model. It is a highly popular model which has the advantages of both iterative and linear development. This model entails the division of the entire system into small manageable modules and their development in phases. Incremental development is based on the previous one adding new features and functionality and keeping a working version of the software during the development process.

3.5.2 Reason for selecting the Incremental Model

- Early Delivery: The core functionality of the app, i.e. login, ambulance booking, and location tracking was to be demonstrated early by a working prototype.
- Real-Time Testing: Components such as authentication based on Firebase and tracking with Google maps required a continuous testing and feedback.
- Flexibility: This model enabled the incorporation of feedback of faculty, peers and early testers after each phase easily.
- Risk Management: The modules were designed and tested separately, and there was a minimal risk of system-wide failures.
- Time Management: The model allowed the continuous work with frequent validations due to the academic schedule.

3.5.3 Stages of Incremental development in MedVan

- Increment 1: Firebase Authentication of the user and driver logins/registrations, Log in and sign-up UI
- Increment 2: GPS-based booking form and location fetching, Use of firebase database to store booking information
- Increment 3: Incorporation of Google maps API, Showing the position of the drivers and users
- Increment 4: Driver auto-accept interface, Status update of the ride using database synchronization

3.5.4 Attained Advantages

- Constant addition of features meant steady improvement
- Users were given early modules to test and therefore early bug identification was possible

- A reusable code between the user and driver modules was found and streamlined
- The documentation and reporting were made phase-wise, making it easy to track academically

CHAPTER 4: EXPERIMENTAL RESULTS

4.1 Introduction

This chapter gives the findings achieved on implementing and testing the MedVan application. It points out the way the system worked in practice, how successfully its goals were met and describes the user experience and the way of the system to behave altogether. Besides, it also comments on the strengths, limitations and areas of improvement of the application.

4.2 Hardware and Software Requirements

4.2.1 Hardware Requirements

The suggested hardware specifications of developers and end users (patients and drivers) are as follows:

Component	Minimum Requirement
Processor	Quad core 1.8 GHz and up
RAM	2 GB minimum (4 GB recommended)
Storage	The minimum free space is 500 MB
Screen size	5.0 inch or more (Android Phone/Tablet)
Network	Wi-Fi / 4G / 3G
GPS Module	Required in real-time tracking of location
Development platform	Windows/Linux/macOS, 4 GB RAM, 2.0 GHz

Hardware Requirements

Table 4.1

4.2.2 Software Requirements

Software Component	Software Specification
Operating system	Android OS (Version 8.0 Oreo and above)
Development IDE	Android Studio (the newest stable version)
Programming Language	Java (to write logic of the application and connect to Firebase)
Back End	Firebase (Realtime Database, Authentication and Cloud Messaging)
APIs	Google Maps API, Location Services API and Firebase SDK
Realtime database	Firebase Realtime Database
Version Control	Git / GitHub (optional, in case of versioning of the sources)
Build Tools	Gradle (Android Studio build system that builds APKs)
The Testing Tools	Android Emulator, JUnit, Mockito

Software Requirements

Table 4.2

These requirements ensure optimal development and execution of the MedVan system.

4.3 Results Obtained

1. On the spot booking of the ambulance

- The users would be able to order an ambulance easily by sharing their present location.
- The request of the booking was sent and stored in Firebase immediately.

2. Live Location tracking

- The movement of the ambulance could be monitored in real-time by means of a map.
- The update of locations was quick and precise (3-5 seconds delay).

3. Auto Ride Assignment

- A booking request was sent to the closest driver available automatically through the Firebase Cloud Messaging.
- In response by the driver, the ride is automatically accepted. He then navigates to the patient's location.

4. Status Synchronization

- The status of the ride (“Pending”, “Accepted”, “In Progress”, “Completed”) was reflected and also synchronized without difficulties between both applications.
- Based on every status change, push notifications were activated accordingly.

5. User Authentication

- Firebase Authentication gave the option to sign in safely and fast by phone numbers.
- Session management was also done in the right way in order to keep users logged on.

4.4 Observational Metrics of Performance

Feature	Observation
Login Time	2–3 seconds
Booking Confirmation Time	About 1–2 seconds
Delivery of Notifications	Immediately (in less than 1 second)
Delay in Update of Map Location	2–5 seconds (Network dependent)
Responsiveness of the App	Responsive and there were no lags

Observational Metrics of Performance

Table 4.3

4.5 Friendly User Feedback

Tested By: Classmates, mentors and project guide

Feedback:

- It is very handy and straightforward UI.
- The risk of live tracking is better than it is supposed.
- It would be useful to add a call button to reach the driver.
- “Easy send off feeling.”

Based on this feedback, it can be concluded that the application is effective in satisfying user requirements and provides a potential answer to real-time booking of an ambulance.

4.6 Limitations

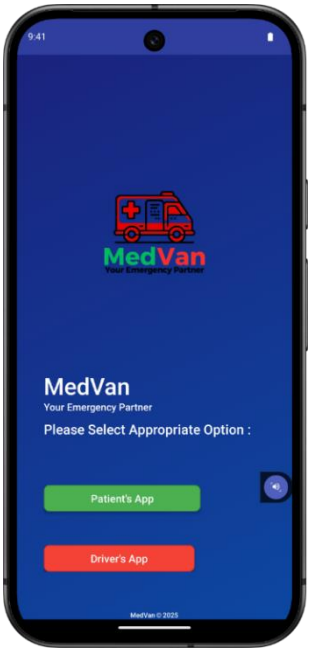
Although the general success was achieved, several restrictions were identified:

- At the given moment, the system is compatible with Android only.
- There is also no in-app calling and chat between the patient and driver.
- There was little testing under only the conditions of a poor network and rural areas.
- Driver verification process and ride monitoring is not implemented at an admin panel stage.

4.7 Screenshots of Application

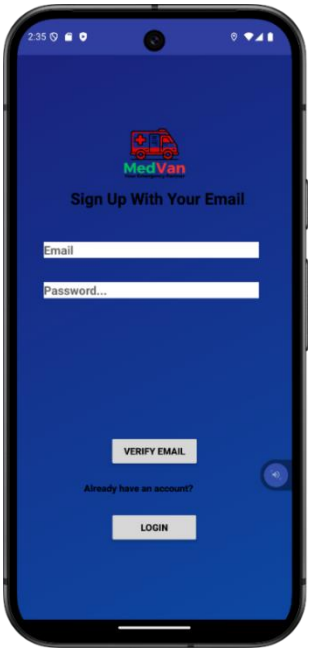
(a)

MedVan Welcome Screen



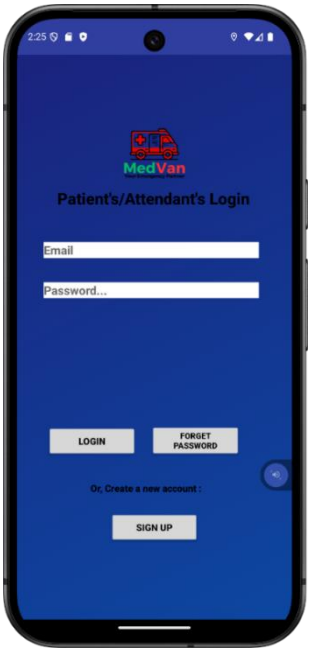
(b)

Homepage Screen



(c)

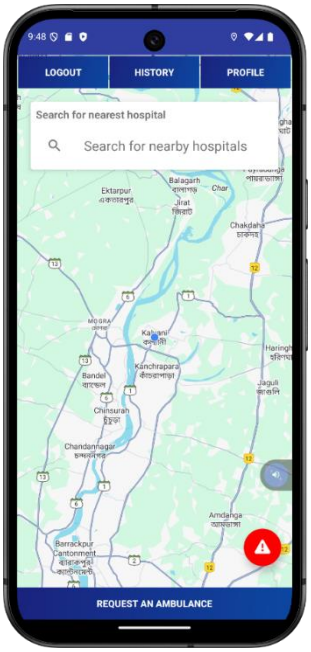
Patient Registration Screen



(d)

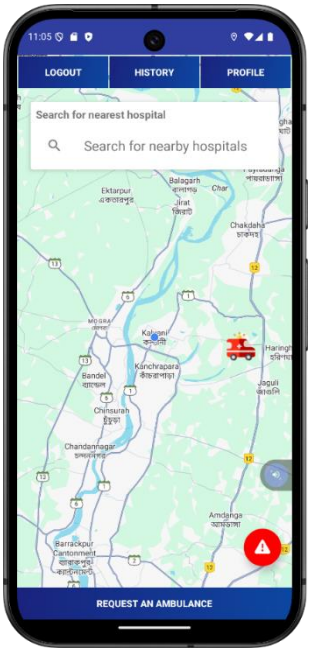
Patient Login Screen

Figure 4.1



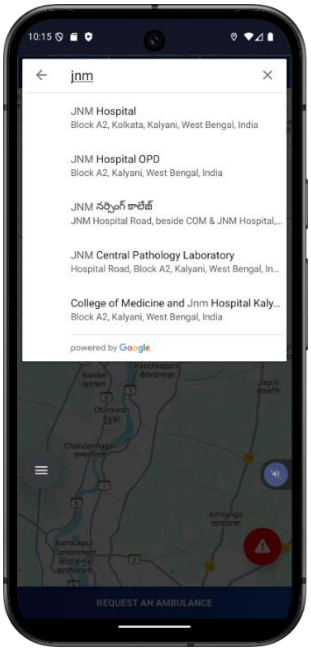
(a)

Patient Homepage Screen



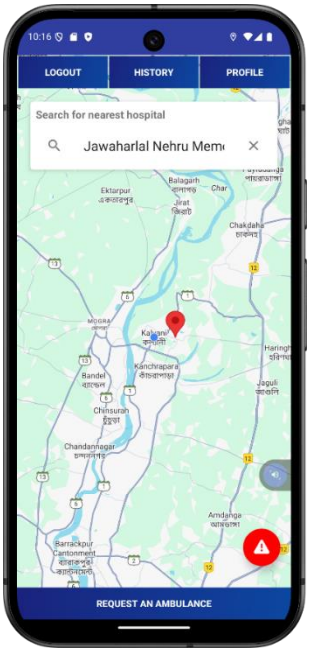
(b)

Patient Home Screen showing Ambulances



(c)

Patient Search Functionality Screen



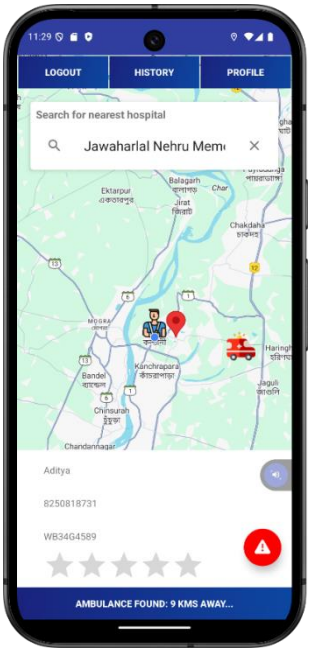
(d)

Patient Chosen Destination Screen

Figure 4.2

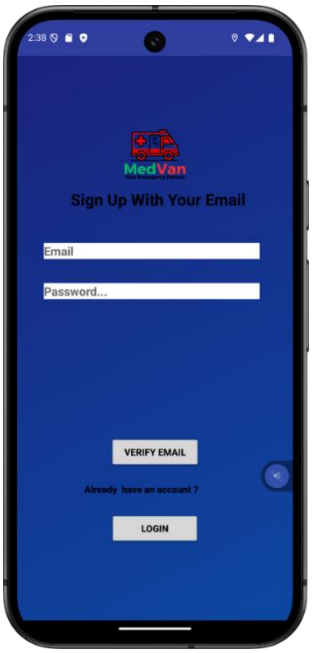
(a)

Patient Requesting for Ambulance Screen



(b)

Patient Ambulance Assigned Screen



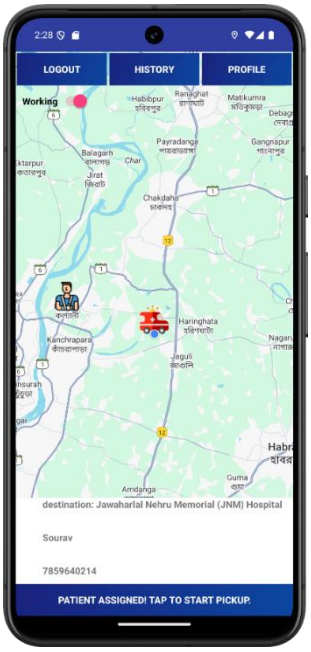
(c)

Driver Registration Screen

(d)

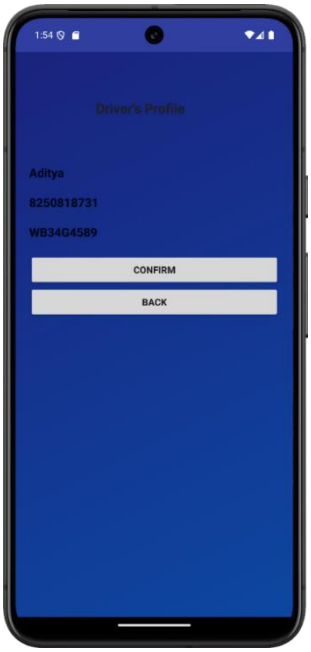
Driver Login Screen

Figure 4.3



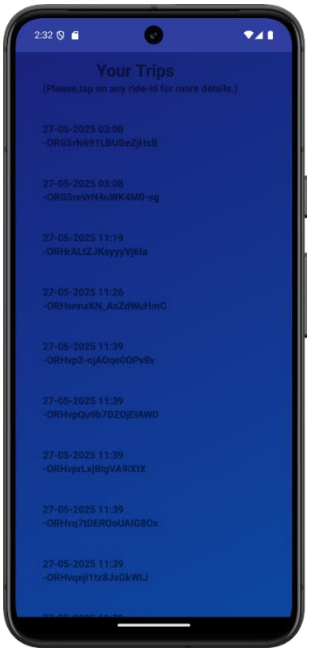
(a)

Driver Ride Assigned Screen



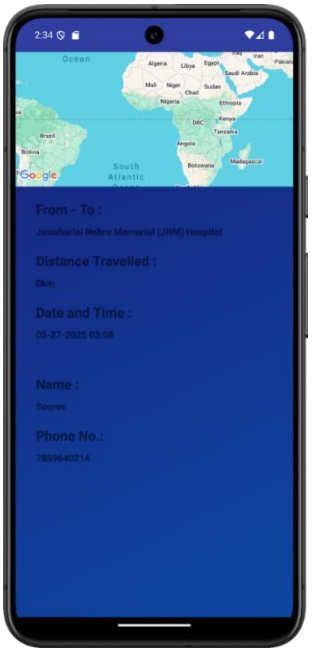
(b)

Driver's Profile Screen



(c)

Driver Ride History Screen



(d)

Driver Ride History Screen (Detailed View)

Figure 4.4

CHAPTER 5: CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

The source of the idea to develop the project was a real-life problem of getting patients a fast and efficient way of booking ambulances and monitoring their transportation process in real-time. This can be achieved by using the mobile technology Android, Firebase, and Google Maps API to achieve this goal, and it is duly achieved through the application.

The implementation of the project attests to the following accomplishments:

Online status track and booking of ambulances.

- User ability to see ambulance in real time using GPS.
- Firebase ensured that there was secure login, data synch and push notifications.
- Two distinct Android applications - a user and a driver application with simplified interface and a real-time dialogue.

Application of the Incremental Model also assisted in constructing and testing the system in a modular manner and the integration was smooth and not a dead-end progress. Unit, integration and user acceptance testing were done rigorously on the system and it was found to be robust in all the principal functions.

In summary, it is possible to declare the prototype of MedVan as a robust enterprise model applicable, besides densely populated surroundings, to semi-urban conditions. It offers interactive, responsive, and real-time user experience and indicates its viability and possible usefulness in emergency care supply chains.

5.2 Discussion

The project has been able to accomplish its main targets of: • Offering a platform to the patients to book ambulance effectively. • Guaranteeing real-time monitoring and messaging with the help of Firebase and Google Maps. • Designing of two role-specific apps (patient and driver) that are fully viable. The combination of Firebase Realtime Database and Google Maps API proved to be helpful especially in the aspect of live updating and synchronization. Application of Incremental Model of software engineering permitted the gradual development, testing and improvement.

5.3 Scope in future

Although the existing version of MedDvan encompasses the basic features of management of the ambulance booking and tracking, there is a substantial potential of improvement and extension. Among the suggested improvements there are:

5.3.1 Communication inside the App

- Introduce voice calling or chat between general and driver to make the coordination during emergency.

5.3.2 Dashboard Admin

- Create a web panel that administration can use to track ride logs, authorize drivers and to manage system data analytics.

5.3.3 Selection of type of ambulances

- Give the user freedom to select types of ambulances (i.e., basic life support, advanced cardiac life support, neonatal ICU and so on).

5.3.4 Integration to hospitals

- Link the application with local hospitals and send emergency rooms automatic notification about new patients.

5.3.5 Emergency Contact Alert

- Introduce a feature of alerting user emergency contact when an ambulance is booked.

5.3.6 Multi-Site Support

- The iOS application should be made available and convert it into a Progressive Web App (PWA).

5.3.7 Artificial Intelligence in Location Optimization

- AI algorithms should recommend the best route and the nearest available ambulance depending on the traffic and availability.

5.4 Concluding thoughts

The MedVan project has become both viable and effective, which demonstrates, on the one hand, the use of mobile and cloud technologies, and, on the other, has a significant social purpose. As it is further developed with integration and user testing, MedVan can be implemented on a larger scale that could also save lives by reducing the time of emergency response.

REFERENCES

1. The documentation of Android Developers. https://developer.android.com/docs
2. Firebase Documentation - real time database, authenticion, and cloud messaging. https://firebase.google.com/docs
3. The documentation of Google Maps Platform. https://developers.google.com/maps/documentation
4. Sommerville, Ian. Software engineering (10 th edition). Pearson education 2016.
5. Pressman, Roger S. Software Engineering A Practitioner Approach (7 th Edition). McGraw-Hill, 2010.
6. Gamma, Erich, et.al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
7. TutorialsPoint. Firebase How to and Examples. https://www.tutorialspoint.com/firebase/index.htmGeeksforGeeks/
8. *Stack Overflow. Solutions to development problems by members of the community. https://stackoverflow.com*
9. Official Instructions of Java Platform (SE 8). https://docs.oracle.com/javase/8/docs/
10. GeeksforGeeks. Android and firebase Tutorials. https://www.geeksforgeeks.org/android-tutorial/

    https://www.geeksforgeeks.org/firebase-realtime-database-in-android/
11. ISO /IEC /IEEE 29148:2018 Systems and software engineering Life cycle processes Requirements engineering.

APPENDIX

Appendix A: Firebase Database Structure

Below is an overview of the main nodes and their structure in the Firebase Realtime Database for the MedVan project.

1. Users

Customers

```
Users
  └── Customers
        └── {customerId}
              ├── Name: String
              ├── Phone: String
              ├── email: String (if stored)
              ├── history
              │     └── {rideId}: true
              └── rating
                    └── {rideId}: Number
```

Drivers

```
Users
  └── Drivers
        └── {driverId}
              ├── Name: String
              ├── Phone: String
              ├── Car: String
              ├── email: String (if stored)
              ├── location
              │     ├── lat: Double
              │     └── lng: Double
              ├── history
              │     └── {rideId}: true
              └── rating
                    └── {rideId}: Number
```

2. Ride Requests

Customer Requests (GeoFire)

```
customerRequest
  └── {customerId}
        ├── g: String (geohash)
        └── l: [latitude, longitude]
```

3. Drivers Available (GeoFire)

```
driversAvailable
   └── {driverId}
         ├── g: String (geohash)
         └── l: [latitude, longitude]

```

4. History

Each ride (completed trip) is stored under the History node:

```
History
   └── {rideId}
         ├── customer: {customerId}
         ├── driver: {driverId}
         ├── timestamp: Long (seconds since epoch)
         ├── distance: String (e.g., "5.2")
         ├── destination: String
         ├── rating: Number
         ├── location
         │     ├── from
         │     │     ├── lat: Double
         │     │     └── lng: Double
         │     └── to
         │           ├── lat: Double
         │           └── lng: Double
         │
```

Appendix B: Code by Functionality

1. User Authentication

- **Customer Login:**

```
 1. // Handles customer login using Firebase Authentication
 2. mLogin.setOnClickListener(new View.OnClickListener() {
 3.     @Override
 4.     public void onClick(View v) {
 5.         final String email = mEmail.getText().toString();
 6.         final String password = mPassword.getText().toString();
 7.         mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
 8.             @Override
 9.             public void onComplete(@NonNull Task<AuthResult> task) {
10.                 if (!task.isSuccessful()) {
11.                     Toast.makeText(CustomerLoginActivity.this, "Sign in error",
Toast.LENGTH_SHORT).show();
12.                 }
13.             }
14.         });
15.     }
16. });
17.
```

Handles customer login logic, input validation, and Firebase authentication.

- **Driver Login:**

```
 1. // Handles driver login and email verification
 2. mLogin.setOnClickListener(new View.OnClickListener() {
 3. @Override
 4. public void onClick(View v) {
 5. final String email = mEmail.getText().toString().trim();
 6. final String password = mPassword.getText().toString().trim();
 7. mAuth.signInWithEmailAndPassword(email, password)
 8. .addOnCompleteListener(DriverLoginActivity.this, new OnCompleteListener<AuthResult>() {
 9. @Override
10. public void onComplete(@NonNull Task<AuthResult> task) {
11. if (!task.isSuccessful()) {
12. Toast.makeText(DriverLoginActivity.this, "Sign in failed", Toast.LENGTH_LONG).show();
13. } else {
14. if (mAuth.getCurrentUser() != null && mAuth.getCurrentUser().isEmailVerified()) {
15. String user_id = mAuth.getCurrentUser().getUid();
16. DatabaseReference current_user_db = FirebaseDatabase.getInstance().getReference()
17. .child("Users").child("Drivers").child(user_id);
18. current_user_db.setValue(true);
19. } else {
20. Toast.makeText(DriverLoginActivity.this, "Please verify your email first",
Toast.LENGTH_LONG).show();
21. mAuth.signOut();
22. }
23. }
24. }
25. });
26. }
27. });
28.
```

Handles driver login logic and authentication.

- **Registration (Customer):**

```
 1. // Handles customer registration and email verification
 2. msignup.setOnClickListener(new View.OnClickListener() {
 3.     @Override
 4.     public void onClick(View v) {
 5.         final String email = mEmail.getText().toString();
 6.         final String password = mPassword.getText().toString();
 7.         if (awesomeValidation.validate()) {
 8.             mAuth.createUserWithEmailAndPassword(email, password)
 9.                 .addOnCompleteListener(CustomerSignup.this, new
OnCompleteListener<AuthResult>() {
10.                     @Override
11.                     public void onComplete(@NonNull Task<AuthResult> task) {
12.                         if (!task.isSuccessful()) {
13.                             Toast.makeText(CustomerSignup.this, "Sign Up Error",
Toast.LENGTH_SHORT).show();
14.                         } else {
15.                             mAuth.getCurrentUser().sendEmailVerification()
16.                                 .addOnCompleteListener(new OnCompleteListener<Void>() {
17.                                     @Override
18.                                     public void onComplete(@NonNull Task<Void> task) {
19.                                         if(task.isSuccessful()){
20.                                             Toast.makeText(CustomerSignup.this, "Registered
Successfully! Please, check your Email for verification.", Toast.LENGTH_SHORT).show();
21.                                         }
22.                                     }
23.                                 });
24.                             FirebaseAuth.getInstance().signOut();
25.                         }
26.                     }
27.                 });
28.         }
29.     }
30. });
31.
```

- **Registration (Driver):**

```
 1. // Handles driver registration and email verification
 2. msignup.setOnClickListener(new View.OnClickListener() {
 3.     @Override
 4.     public void onClick(View v) {
 5.         final String email = mEmail.getText().toString();
 6.         final String password = mPassword.getText().toString();
 7.         if (awesomeValidation.validate()) {
 8.             mAuth.createUserWithEmailAndPassword(email, password)
 9.                 .addOnCompleteListener(DriverSignup.this, new
OnCompleteListener<AuthResult>() {
10.                     @Override
11.                     public void onComplete(@NonNull Task<AuthResult> task) {
12.                         if(!task.isSuccessful()){
13.                             Toast.makeText(DriverSignup.this, "Sign Up Error",
Toast.LENGTH_SHORT).show();
14.                         } else {
15.                             mAuth.getCurrentUser().sendEmailVerification()
16.                                 .addOnCompleteListener(new OnCompleteListener<Void>() {
17.                                     @Override
18.                                     public void onComplete(@NonNull Task<Void> task) {
19.                                         if(task.isSuccessful()){
20.                                             Toast.makeText(DriverSignup.this, "Registered
Successfully! Please, check your Email for verification.", Toast.LENGTH_SHORT).show();
```

```
                                        }
                                    }
                                });
                            FirebaseAuth.getInstance().signOut();
                        }
                    }
                });
        }
    }
});

```

Handles user registration, form validation, and Firebase user creation.

- **Password Reset:**

```
// Handles password reset via email
btnResetPassword.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String email = edtEmail.getText().toString().trim();
        if (TextUtils.isEmpty(email)) {
            Toast.makeText(getApplicationContext(), "Enter your email!",
Toast.LENGTH_SHORT).show();
            return;
        }
        mAuth.sendPasswordResetEmail(email)
            .addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if (task.isSuccessful()) {
                        Toast.makeText(ResetPasswordActivity.this, "Check email to reset
your password!", Toast.LENGTH_SHORT).show();
                    } else {
                        Toast.makeText(ResetPasswordActivity.this, "Fail to send reset
password email!", Toast.LENGTH_SHORT).show();
                    }
                }
            });
    }
});

```

Handles password reset via email.

- **Password Reset:**

```
// Handles password reset via email
btnResetPassword.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String email = edtEmail.getText().toString().trim();
        if (TextUtils.isEmpty(email)) {
            Toast.makeText(getApplicationContext(), "Enter your email!",
Toast.LENGTH_SHORT).show();
            return;
        }
        mAuth.sendPasswordResetEmail(email)
            .addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
```

```
14.                     if (task.isSuccessful()) {
15.                         Toast.makeText(ResetPasswordActivity.this, "Check email to reset
your password!", Toast.LENGTH_SHORT).show();
16.                     } else {
17.                         Toast.makeText(ResetPasswordActivity.this, "Fail to send reset
password email!", Toast.LENGTH_SHORT).show();
18.                     }
19.                 }
20.             });
21.     }
22. });
23.
```

Handles password reset via email.

2. Maps and Location

- **Customer Map:**

```
 1. // Requesting an ambulance and setting pickup location
 2. mRequest.setOnClickListener(new View.OnClickListener() {
 3.     @Override
 4.     public void onClick(View v) {
 5.         if(requestBol){
 6.             endRide();
 7.         } else {
 8.             if (mLastLocation == null) {
 9.                 Toast.makeText(CustomerMapActivity.this, "Please wait for your location to
be determined", Toast.LENGTH_LONG).show();
10.                 return;
11.             }
12.             if (destinationLatlng == null || (destinationLatlng.latitude == 0.0 &&
destinationLatlng.longitude == 0.0)) {
13.                 Toast.makeText(CustomerMapActivity.this, "Please select a destination
first", Toast.LENGTH_LONG).show();
14.                 return;
15.             }
16.             requestBol = true;
17.             String userId = FirebaseAuth.getInstance().getCurrentUser().getUid();
18.             DatabaseReference ref =
FirebaseDatabase.getInstance().getReference("customerRequest");
19.             GeoFire geoFire = new GeoFire(ref);
20.             geoFire.setLocation(userId, new GeoLocation(mLastLocation.getLatitude(),
mLastLocation.getLongitude()));
21.             pickupLocation = new LatLng(mLastLocation.getLatitude(),
mLastLocation.getLongitude());
22.             pickupMarker = mMap.addMarker(new MarkerOptions()
23.                 .position(pickupLocation)
24.                 .title("Pickup Here")
25.                 .icon(BitmapDescriptorFactory.fromResource(R.mipmap.patient)));
26.             mRequest.setText("Getting Your Ambulance...");
27.             getClosestDriver();
28.         }
29.     }
30. });
31.
```

Displays map, tracks customer location, and requests rides.

- **Driver Map:**

```
// Driver toggles availability and handles ride status
mworkingSwitch.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if(isChecked){
            connectDriver();
        } else {
            disconnectDriver();
        }
    }
});
```

Displays map, shows nearby ride requests, and tracks driver location.

- **Maps Utility:**

```
// Enable location features and update location in Firebase
private void enableLocationFeatures() {
    if (mMap != null && ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
        mMap.setMyLocationEnabled(true);
        startLocationUpdates();
    }
}

private void updateLocation(Location location) {
    if (location != null && mMap != null) {
        LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude());
        mMap.clear();
        mMap.addMarker(new MarkerOptions().position(latLng).title("Your Location"));
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, 15));
        Map<String, Double> userLocation = new HashMap<>();
        userLocation.put("lat", location.getLatitude());
        userLocation.put("lng", location.getLongitude());
        driverLocationRef.setValue(userLocation);
    }
}

```

General map-related utilities and features.

3. User Settings

- **Customer Settings:**

```
// Save customer profile information
private void saveUserInformation() {
    mName = mNameField.getText().toString();
    mPhone = mPhoneField.getText().toString();
    Map userInfo = new HashMap();
    userInfo.put("Name", mName);
    userInfo.put("Phone", mPhone);
    mCustomerDatabase.updateChildren(userInfo);
    finish();
}

```

Allows customers to update their profile and preferences.

- **Driver Settings:**

```
// Save driver profile and vehicle information
private void saveUserInformation(){
    mName = mNameField.getText().toString();
    mPhone = mPhoneField.getText().toString();
    mCar = mCarField.getText().toString();
    Map userInfo = new HashMap();
    userInfo.put("Name", mName);
    userInfo.put("Phone", mPhone);
    userInfo.put("Car", mCar);
    mDriverDatabase.updateChildren(userInfo);
    finish();
}

```

Allows drivers to update their profile and vehicle information.

4. Ride History

- **History List:**

```
// Fetch and display user's ride history
private void getUserHistoryIds() {
    DatabaseReference userHistoryDatabase = FirebaseDatabase.getInstance().getReference()
        .child("Users").child(customerOrDriver).child(userId).child("history");
    userHistoryDatabase.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if(dataSnapshot.exists()){
                for(DataSnapshot history : dataSnapshot.getChildren()){
                    FetchRideInformation(history.getKey());
                }
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {}
    });
}

```

Displays a list of past rides for the user.

- **History Details:**

```
// Fetch and display details for a single ride
private void getRideInformation() {
    historyRideInfoDb.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (dataSnapshot.exists()) {
                // Extract and display ride details, user info, and route
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {}
    });
}
```

Shows detailed information about a single ride.

- **RecyclerView Adapter:**

```
1. @Override
2. public void onBindViewHolder(HistoryViewHolders holder, final int position) {
3.     holder.rideId.setText(itemList.get(position).getRideId());
4.     if(itemList.get(position).getTime()!=null){
5.         holder.time.setText(itemList.get(position).getTime());
6.     }
7. }
8.
```

Adapter for displaying ride history in a RecyclerView.

- **History Object & ViewHolder:**

```
 1. public class HistoryObject {
 2.     private String rideId;
 3.     private String time;
 4.     // getters and setters
 5. }
 6. public class HistoryViewHolders extends RecyclerView.ViewHolder implements
View.OnClickListener{
 7.     public TextView rideId;
 8.     public TextView time;
 9.     public HistoryViewHolders(View itemView) {
10.         super(itemView);
11.         itemView.setOnClickListener(this);
12.         rideId = (TextView) itemView.findViewById(R.id.rideId);
13.         time = (TextView) itemView.findViewById(R.id.time);
14.     }
15.     @Override
16.     public void onClick(View v) {
17.         Intent intent = new Intent(v.getContext(), HistorySingleActivity.class);
18.         Bundle b = new Bundle();
19.         b.putString("rideId", rideId.getText().toString());
20.         intent.putExtras(b);
21.         v.getContext().startActivity(intent);
22.     }
23. }
24.
```

Data model and view holder for ride history items.

5. App Lifecycle

- **App Killed Handler:**

```
1. @Override
2. public void onTaskRemoved(Intent rootIntent) {
3.     super.onTaskRemoved(rootIntent);
4.     String userId = FirebaseAuth.getInstance().getCurrentUser().getUid();
5.     DatabaseReference ref = FirebaseDatabase.getInstance().getReference("driversAvailable");
6.     GeoFire geoFire = new GeoFire(ref);
7.     geoFire.removeLocation(userId);
8. }
9.
```

Handles cleanup and state saving when the app is killed.

6. Welcome and Onboarding

- **Welcome Screen:**

```
// Shows splash, then reveals login options
Handler handler = new Handler();
Runnable runnable = new Runnable() {
    @Override
    public void run() {
        rellay1.setVisibility(View.VISIBLE);
        rellay2.setVisibility(View.VISIBLE);
    }
};
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_welcome_);
    rellay1 = (RelativeLayout) findViewById(R.id.rellay1);
    rellay2 = (RelativeLayout) findViewById(R.id.rellay2);
    startService(new Intent(Welcome_Activity.this,onAppKilled.class));
    handler.postDelayed(runnable, 1000); // splash timeout
}
public void sendMessage(View view) {
    Intent intent = new Intent(Welcome_Activity.this, CustomerLoginActivity.class);
    startActivity(intent);
}
public void sendMessage2(View view) {
    Intent intent = new Intent(Welcome_Activity.this, DriverLoginActivity.class);
    startActivity(intent);
}

```

Initial screen shown to users, with navigation to login/signup.