

# **PATHFINDE: A MOBILE-BASED REAL-TIME STUDENT TRACKING SYSTEM**

**A Major Project Report**

*Submitted in partial fulfillment for the award of the degree of*

**MASTER OF COMPUTER APPLICATIONS**

*by*

**SHREENIDHI S**

**Reg. No: 2313323037035**

*Under the guidance of*

**Mrs. A. JOSPHINE ANITHA., MCA., M.Phil., SET.**



**MCA DEPARTMENT**

**ETHIRAJ COLLEGE FOR WOMEN (AUTONOMOUS)**

**CHENNAI – 600008**

**APRIL 2025**

**ETHIRAJ COLLEGE FOR WOMEN (AUTONOMOUS)**

**BONAFIDE CERTIFICATE**



*This is to certify that the major project report titled*

**“PATHFINDE: A MOBILE-BASED REAL-TIME STUDENT  
TRACKING SYSTEM”**

Being submitted to the Ethiraj College for Women (Autonomous)

Affiliated to the University of Madras, Chennai

by

**SHREENIDHI S**

**(2313323037035)**

*in partial fulfillment for the award of the degree of*

**MASTER OF COMPUTER APPLICATIONS**

is a bonafide report of work carried out by her under my guidance and supervision.

**Signature of Guide**

**Signature of HOD**

**Place: Chennai**

**Date:**

Submitted for the viva-voice examination at Ethiraj College for Women (Autonomous)

On \_\_\_\_\_

**Examiner – 1 .....**

**(Signature of the Examiner)**

**Examiner – 2 .....**

**(Signature of the Examiner)**

## **CERTIFICATE OF ORIGINALITY**

I, hereby declare that the major project entitled “**PATHFINDE: A MOBILE-BASED REAL-TIME STUDENT TRACKING SYSTEM**”, being submitted to the **MCA DEPARTMENT, ETHIRAJ COLLEGE FOR WOMEN (AUTONOMUS)** in partial fulfilment for the award of the degree of **MASTER OF COMPUTER APPLICATIONS** is an authentic report of work carried out by me under the guidance of **Mrs. A.JOSPHINE ANITHA, MCA,, M.Phil., SET.**, and that this has not previously formed the basis for the award of any other degree.

**Place:** Chennai

**Date:**

**Signature of the Candidate**

Shreenidhi S

2313323037035

## CONTENTS

SNO		CONTENT	PAGENO
1		<b>INTRODUCTION</b>	
	1.1	Abstract	
	1.2	About the Organization	
	1.3	Existing System	
	1.4	Proposed System	
2		<b>SYSTEM ANALYSIS</b>	
	2.1	Overall Description	
	2.2	Requirement Specification	
3		<b>SYSTEM DESIGN</b>	
	3.1	ER Diagram	
	3.2	Table Description	
4		<b>SYSTEM ARCHITECTURE</b>	
5		<b>MODULE DESCRIPTION</b>	
6		<b>IMPLEMENTATION</b>	
	6.1	Coding	
7		<b>FUTURE ENHANCEMENT</b>	
8		<b>CONCLUSION</b>	
9		<b>REFERENCES</b>	
10		<b>APPENDIX -I ONLINE COURSE COMPLETION CERTIFICATE</b>	
11		<b>APPENDIX-II ATTENDANCE SHEET</b>	
12		<b>APPENDIX -III FEEDBACK FORM</b>	

# INTRODUCTION

# **PATHFINDE: A MOBILE-BASED REAL-TIME STUDENT TRACKING SYSTEM**

## **ABSTRACT**

In today's fast-paced world, ensuring the safety and security of students is of utmost importance. "PathFinde" is an innovative real-time student location tracking system designed to provide secure monitoring and location-sharing functionalities for students, parents, and administrators. The primary objective of this project is to enhance safety, provide real-time location tracking, and establish a seamless communication bridge between students and their guardians.

PathFinde operates by allowing students to automatically share their real-time location with their registered parents. This ensures that parents can always monitor their child's safety, whether they are commuting to school, attending extracurricular activities, or traveling back home. The system is built using .NET ensuring cross-platform compatibility across android, iOS, Windows, and Mac. By integrating Geolocation APIs, SQLServer for data storage, and RESTful APIs for server communication, PathFinde ensures high reliability, fast performance, and a seamless user experience.

The system consists of three main user roles: Students, Parents, and Administrators. Students register in the system and have their locations updated periodically, ensuring that their last known position is always available. Parents can securely log in and view the live location of their children, along with historical location data for added security. Administrators, on the other hand, oversee the system by managing users, monitoring suspicious activity, and ensuring the overall security of stored data.

Another significant feature of PathFinde is the "Send Location" functionality, which allows students to manually send their real-time location update to their registered parents in emergency situations. Additionally, the system is designed to automatically queue location updates every one minute, ensuring that parents receive up-to-date tracking data without requiring manual input from students.

PathFinde is a comprehensive student tracking solution that offers real-time location updates, parental monitoring, and administrative control, ensuring the highest standards of security and privacy.

## **1.2 ABOUT THE ORGANIZATION**

YaRCubes is a dynamic and innovative software development company that provides effective and creative IT solutions tailored to meet the unique needs of businesses of all sizes. With a deep understanding of the rapidly evolving technological landscape, YaRCubes is committed to delivering high-quality software solutions that drive business growth, streamline operations, and enhance customer experiences. The company's mission is to empower businesses through cutting-edge technology and strategic solutions, aiming to become a leading player in the software development industry by delivering innovative, reliable, and scalable software solutions aligned with each client's business objectives. YaRCubes offers a wide range of services, including desktop, mobile, and web application development, where the company's developers create responsive and user-friendly applications that work seamlessly across different platforms. The company also provides comprehensive quality assurance support, ensuring that all products undergo rigorous testing to eliminate bugs, enhance performance, and meet high security standards. YaRCubes delivers end-to-end software development solutions, from concept and design to development, testing, deployment, and ongoing maintenance, following agile methodologies to ensure timely delivery and adaptability to changing business needs. Its expertise in cross-platform mobile app development using frameworks like Flutter, Xamarin, and .NET MAUI enables businesses to reach a wider audience while reducing development time and costs. Performance optimization and bug fixing are also core strengths of YaRCubes, ensuring that applications run smoothly, are scalable, and offer a superior user experience. The company leverages continuous integration and deployment (CI/CD) processes to automate build, test, and deployment cycles, ensuring faster delivery of updates and new features while maintaining code quality. Additionally, YaRCubes provides dedicated technical support and maintenance to ensure the long-term success of its solutions, offering round-the-clock assistance to troubleshoot issues, provide updates, and make improvements based on client feedback. The company's customer-centric approach focuses on understanding client requirements and delivering tailored solutions that drive measurable business results. YaRCubes' strengths lie in its ability to deliver scalable and secure solutions, ensuring that businesses can grow while maintaining high performance and security. With a proven track record of successful projects and satisfied clients, YaRCubes continues to set new standards in software development and IT services, making it a trusted choice for businesses looking to leverage technology for growth and success.

### 1.3 EXISTING SYSTEM

The existing system in the Pathfinde project is a completely new project developed from scratch, starting with just a login page. The project is built using .NET MAUI for the frontend and SQLite for the backend, with Xamarin playing a key role in enhancing cross-platform compatibility. The goal of Pathfinde is to create a real-time student tracking system that allows students, parents, and administrators to manage and monitor student activity, including real-time location tracking and communication features.

.NET MAUI (Multi-platform App UI) is a cross-platform framework used for creating native mobile and desktop applications from a single codebase. It allows developers to build applications that run on Android, iOS, Windows, and macOS using C# and .NET. Pathfinde leverages .NET MAUI for the frontend development, providing a unified and consistent user experience across different platforms. The framework's flexibility enables the use of a single project structure, which simplifies development, reduces maintenance efforts, and ensures consistent behavior and appearance across platforms. The login page and other UI components in Pathfinde are built using XAML (Extensible Application Markup Language) in .NET MAUI, allowing developers to create responsive layouts and dynamic user interfaces. MAUI's inbuilt support for styling, themes, and adaptive layouts ensures that the app is visually appealing and responsive on different screen sizes and resolutions.

For backend data storage and management, SQLite is used due to its lightweight nature and ability to run efficiently on mobile and desktop devices. SQLite is an embedded relational database that stores data locally within the device, ensuring that the app can function offline and sync data when an internet connection is available. In Pathfinde, SQLite is used to store user information, student details, and activity logs securely. The data is managed using Entity Framework Core, which allows developers to define models and manage database operations such as creating, reading, updating, and deleting records with minimal code. SQLite's ability to handle structured data efficiently ensures quick data retrieval and low latency, which is essential for real-time tracking and communication.

Xamarin is integrated with .NET MAUI to enable cross-platform functionality and provide native performance. Xamarin.Forms allows developers to create shared UI components and business logic, which are then compiled into native code for each platform. This ensures that the app delivers high performance and a native-like experience on both Android and iOS.



Xamarin also provides access to platform-specific APIs, such as GPS, camera, and sensors, enabling real-time location tracking and notifications. The combination of .NET MAUI and Xamarin simplifies code management, reduces duplication, and accelerates the development process.

Overall, Pathfinde leverages the powerful combination of .NET MAUI, SQLite, and Xamarin to create a scalable and efficient student tracking system. The unified codebase, efficient data management, and native performance ensure that the app delivers a seamless and consistent user experience across platforms while handling real-time data processing and communication effectively.

## 1.4 PROPOSED SYSTEM

The proposed system for Pathfinde aims to implement advanced features using the Geolocation API and real-time alert messaging, designed to enhance student tracking and communication. Pathfinde will have three user roles within the app: Parent, Admin, and Student. Each role will have specific access and functionality tailored to their needs, ensuring a secure and efficient user experience. The system is being developed using .NET MAUI for the frontend, SQLite for the backend, and Xamarin for cross-platform compatibility, making it accessible on both Android and iOS devices.

One of the key features of the proposed system is the integration of the Geolocation API to enable real-time location tracking of students. By using the device's GPS capabilities, the system can capture and update the student's location in real-time. This data is processed and stored in the SQLite database, allowing parents and administrators to monitor student activity and movement accurately. The Geolocation API will ensure that the location data is captured securely and efficiently while optimizing battery usage to prevent excessive power consumption. The real-time tracking feature will enable parents to know when their child leaves or arrives at a specific location, providing peace of mind and increased safety.

The proposed system will also implement an alert messaging feature to keep parents informed about critical events. Notifications will be sent to parents in case of unusual activity, late arrivals, or when the student enters or leaves a predefined location. The app will use push notifications to ensure that parents receive timely updates even when the app is running in the background. The alert system will be configurable, allowing parents to customize the types of notifications they want to receive. This feature ensures that parents are always aware of their child's activity without needing to check the app continuously.

The Pathfinde system will support three roles: Parent, Admin, and Student. Each user role will have specific permissions and access levels. Parents will have access to real-time location tracking and receive alert notifications. They will also have access to the student's activity logs and communication features within the app. Students will have limited access to their own location data and the ability to send emergency alerts to their parents if needed. The Admin role will have higher access privileges, including the ability to view the location of all students, manage user accounts, and configure system settings. The Admin will also have access to

activity reports and system performance metrics to monitor the overall health and efficiency of the app.

Overall, the proposed system for Pathfinde leverages the power of .NET MAUI, Xamarin, and SQLite to create a secure, scalable, and user-friendly student tracking solution. By combining real-time geolocation tracking with alert messaging and role-based access, the app will provide parents and administrators with a comprehensive tool to monitor student activity and ensure their safety.

# **SYSTEM ANALYSIS**

## 2.1 OVERALL DESCRIPTION

**.NET MAUI** (Multi-platform App UI) is a cross-platform framework developed by Microsoft that allows developers to create native mobile and desktop applications using a single codebase written in C#. It is the successor to Xamarin.Forms and is part of the .NET ecosystem, which streamlines cross-platform development for Android, iOS, macOS, and Windows. MAUI enables developers to define the user interface using XAML (Extensible Application Markup Language) and write business logic in C#. It introduces a simplified project structure where all platform-specific code is managed within a single project, unlike Xamarin.Forms which required separate projects for each platform. This unification reduces development complexity and enhances code sharing across platforms. MAUI leverages the native capabilities of each platform, allowing developers to access device-specific features such as camera, sensors, file storage, and network connectivity through platform APIs. One of the key advantages of .NET MAUI is its performance optimization, as it compiles to native code, ensuring smooth execution and fast response times. Developers can use the .NET Hot Reload feature, which allows them to see the impact of code changes in real time without restarting the application. The framework also supports dependency injection, which simplifies service management and enhances code reusability. MAUI's layout engine supports adaptive UI design, ensuring that the interface adjusts smoothly across different screen sizes and orientations. Developers can create custom renderers to implement platform-specific UI components while maintaining a shared codebase. MAUI also integrates with Blazor, a framework that enables developers to build interactive web applications using C# instead of JavaScript, allowing the same codebase to be used for web, desktop, and mobile applications. The MAUI framework includes built-in libraries for handling navigation, resource management, and styling, making it easier to create consistent and professional user interfaces. It also provides robust support for accessibility features such as screen readers and high-contrast themes. MAUI applications can be deployed using the .NET CLI (Command Line Interface) or Visual Studio, which offers comprehensive debugging and profiling tools. The framework supports third-party libraries and plugins, allowing developers to extend functionality and integrate with external services. MAUI's modular architecture allows developers to add or remove platform-specific code without affecting the core application logic, making it highly flexible and scalable. Its support for responsive layouts and adaptive design ensures that applications perform well across a wide range of devices and screen resolutions. Overall, .NET MAUI simplifies the development process by providing a unified platform for building high-performance, cross-platform

applications with native user experiences. Its seamless integration with the .NET ecosystem, powerful UI capabilities, and real-time debugging features make it a preferred choice for modern cross-platform application development.

**Xamarin** is an open-source framework developed by Microsoft that allows developers to create native mobile applications for Android and iOS using C#. It enables code sharing across platforms while allowing access to platform-specific features, resulting in high-performance applications with a native look and feel. Xamarin was initially released in 2011 and became part of the Microsoft ecosystem in 2016. Xamarin allows developers to write a single codebase using C# and compile it into native code for Android and iOS. This approach eliminates the need to write separate code for each platform while maintaining the ability to use native APIs and UI components. Xamarin includes two main components: Xamarin.Android and Xamarin.iOS, which provide direct access to platform-specific features and allow developers to write platform-specific code when needed. Xamarin uses the Mono runtime for compiling C# code into native code, ensuring that applications perform at the same level as those written in Java or Swift. One of the key features of Xamarin is its support for Xamarin.Forms, which allows developers to create a shared UI using XAML. Xamarin.Forms maps the shared UI components to native controls on each platform, ensuring that the application looks and feels native. Developers can customize the UI for each platform using custom renderers, which allows platform-specific design adjustments while maintaining a shared codebase. Xamarin also supports platform-specific APIs, enabling developers to use native features such as GPS, camera, sensors, and file storage without sacrificing code sharing. The framework integrates with Visual Studio, providing a powerful development environment with features like code completion, debugging, and profiling. Xamarin supports the use of NuGet packages and third-party libraries, allowing developers to extend functionality and integrate with other services. It also includes built-in support for dependency injection, simplifying service management and enhancing testability. Xamarin applications are compiled ahead of time (AOT), which improves performance and reduces startup times. The framework provides access to native APIs through platform-specific projects, ensuring that developers can implement complex functionality while maintaining a shared business logic layer. Xamarin's ecosystem includes a rich set of UI controls, templates, and plugins that simplify development and improve code quality. Xamarin also supports integration with Azure, allowing developers to implement cloud-based services and real-time data synchronization. Its ability to deliver near-native performance with a shared codebase makes Xamarin a popular choice for cross-platform

development. With the introduction of .NET MAUI, Xamarin.Forms has been integrated into the new framework, providing an even more streamlined development experience. Despite the transition to MAUI, Xamarin remains a robust and mature framework for cross-platform development, particularly for developers working on legacy projects or those requiring deep platform-specific customization.

**SQLite** is a lightweight, embedded, and serverless relational database engine designed for use in mobile and desktop applications. Unlike traditional relational database systems, SQLite does not require a separate server process, as the entire database is contained within a single file stored on the device. This makes SQLite highly portable and easy to deploy. SQLite is written in C and is known for its small size, with the entire library typically under 1 MB in size. It is ACID-compliant (Atomicity, Consistency, Isolation, and Durability), ensuring that database transactions are processed reliably even in the event of system crashes or power failures. SQLite supports a wide range of SQL commands, including complex queries, joins, subqueries, and aggregate functions. It also provides support for indexing, transactions, and triggers, making it suitable for handling structured data in embedded systems and mobile applications. SQLite's architecture is based on a file-based storage model, meaning that the entire database is stored as a single file on the local filesystem. This approach simplifies data management and reduces the need for network connectivity, making it ideal for offline applications. Developers can interact with SQLite using SQL commands or ORM (Object-Relational Mapping) libraries like Entity Framework Core or sqlite-net. SQLite also supports encryption, allowing developers to secure sensitive data stored on the device. Its read and write operations are highly efficient, with minimal memory usage, ensuring that applications remain responsive even when handling large datasets. SQLite is designed to handle high concurrency, meaning that multiple threads can read from the database simultaneously without performance degradation. The database engine includes built-in support for row-level locking and conflict resolution, ensuring data consistency in multi-threaded environments. SQLite's small footprint and low resource consumption make it ideal for resource-constrained environments such as mobile phones and embedded systems. It is also widely used in desktop applications, web browsers, and IoT (Internet of Things) devices. SQLite's flexibility allows developers to create complex data models and relationships while maintaining fast query performance. The library is open-source and widely supported, with regular updates and a large developer community. Developers using .NET MAUI can easily integrate SQLite by installing the sqlite-net NuGet package and defining database models using C# classes. The combination of SQLite's

reliability, speed, and ease of use makes it a preferred choice for local data storage in mobile and desktop applications. Its ability to operate in offline environments and handle high volumes of data efficiently ensures that applications remain responsive and secure under various operating conditions.

The **Geolocation API** is a powerful tool that allows applications to access the geographic location of a device using GPS, Wi-Fi, and cellular data. It provides real-time tracking and location-based services, enabling applications to deliver personalized experiences based on the user's location. The Geolocation API supports high-accuracy tracking by combining multiple data sources, ensuring precise location updates even in challenging environments. It can function in the background, allowing continuous location tracking even when the application is not active. Developers can use the Geolocation API in .NET MAUI through the `Geolocation.Default.GetLocationAsync()` method to retrieve the current location or monitor location changes. The API also supports reverse geocoding, which converts geographic coordinates into human-readable addresses, enabling features like location-based notifications and mapping. The Geolocation API is essential for applications that require real-time location data, such as navigation, ride-sharing, fitness tracking, and emergency response services.

The **Twilio API** is a cloud-based communication platform that enables applications to send SMS, MMS, and make voice and video calls programmatically. It allows developers to integrate messaging and calling functionality into their applications using simple RESTful API calls. Twilio supports two-factor authentication (2FA), allowing secure user verification through SMS or phone calls. It also provides webhooks to handle real-time events such as message delivery status, call logs, and user responses. Developers can configure Twilio in .NET MAUI applications by installing the Twilio NuGet package and initializing it with the account SID and authentication token. The API supports sending and receiving messages globally, ensuring reliable communication across different networks and regions. Twilio's scalability allows it to handle high volumes of messages and calls, making it suitable for customer support, notifications, and automated messaging systems. Its secure infrastructure and compliance with global communication standards ensure reliable and secure message delivery.

Together, .NET MAUI, Xamarin, SQLite, the Geolocation API, and the Twilio API form a robust and scalable system capable of delivering high-performance cross-platform applications with real-time tracking and communication features. The combination of a unified



development framework, efficient data storage, real-time location services, and global communication capabilities ensures a seamless user experience across different devices and platforms.

## 2.2 REQUIREMENT SPECIFICATION

### Hardware Specification

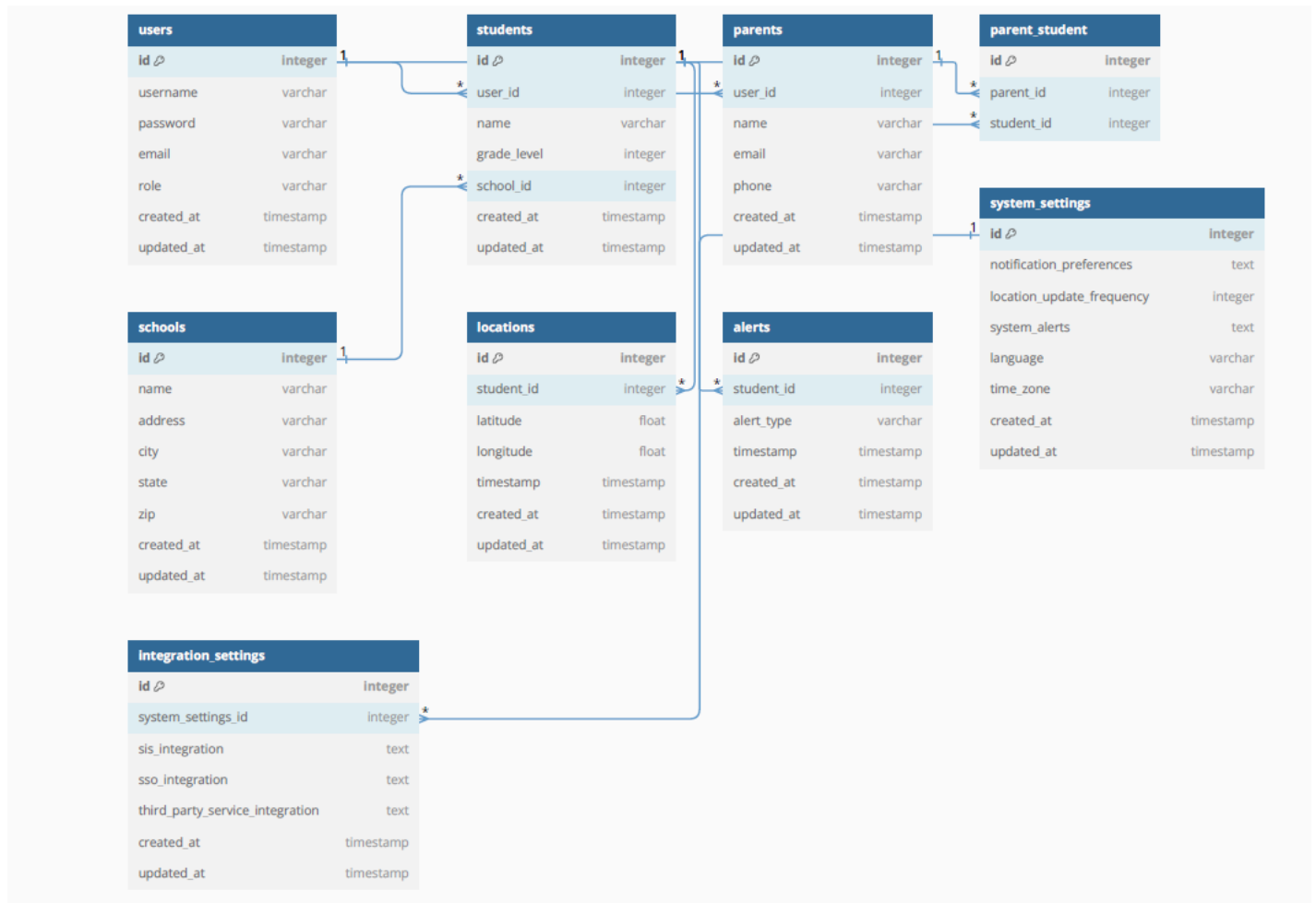
COMPONENT	SYSTEM INFORMATION
Processor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1.80 GHz
Memory (RAM)	12.0 GB (11.9 GB usable)
System Type	64-bit OS, x64-based processor

### Software Specification

PARTICULAR	INFORMATION
Platform	Visual Studio 2022
Frontend Framework	.NET MAUI
Backend Database	SQLite
Cross-Platform Support	Xamarin
Programming Language	C#
API Integration	Geolocation API

## **SYSTEM DESIGN**

### 3.1 ER DIAGRAM



The database schema is designed to manage the relationships between users, students, parents, schools, and system settings in an organized manner. The Users Table stores basic user information such as username, password, email, and role (admin, parent, or student), and serves as the foundation for other related tables. The Students Table links each student to a user account via a foreign key (user\_id) and also associates each student with a school through the school\_id. The Parents Table stores information about parents and links to the Users Table through user\_id, ensuring that each parent has a corresponding user account. The Parent\_Student Table establishes a many-to-many relationship, allowing a parent to be associated with multiple students and a student to have multiple parents.

## 3.2 TABLE DESCRIPTION

### 1. Users

This table stores user-related information, including login credentials and user roles.

- **id** – Primary key (integer)
- **username** – Unique username for the user (varchar)
- **password** – Password for user authentication (varchar)
- **email** – User's email address (varchar)
- **role** – Type of user (admin, parent, student) (varchar)
- **created\_at** – Timestamp for when the record was created
- **updated\_at** – Timestamp for the last update to the record

### 2. Students

This table stores student-related information and links students to their respective user and school.

- **id** – Primary key (integer)
- **user\_id** – Foreign key linking to the users table (integer)
- **name** – Name of the student (varchar)
- **grade\_level** – Grade level of the student (integer)
- **school\_id** – Foreign key linking to the schools table (integer)
- **created\_at** – Timestamp for when the record was created
- **updated\_at** – Timestamp for the last update to the record

### 3. Parents

This table stores parent-related information and links parents to the user table.

- **id** – Primary key (integer)
- **user\_id** – Foreign key linking to the users table (integer)
- **name** – Name of the parent (varchar)
- **email** – Parent's email address (varchar)
- **phone** – Parent's phone number (varchar)
- **created\_at** – Timestamp for when the record was created

- **updated\_at** – Timestamp for the last update to the record

#### 4. parent\_student

This table establishes a many-to-many relationship between parents and students.

- **id** – Primary key (integer)
- **parent\_id** – Foreign key linking to the parents table (integer)
- **student\_id** – Foreign key linking to the students table (integer)

#### 5. Schools

This table stores school-related information.

- **id** – Primary key (integer)
- **name** – Name of the school (varchar)
- **address** – Address of the school (varchar)
- **city** – City where the school is located (varchar)
- **state** – State where the school is located (varchar)
- **zip** – Zip code of the school's location (varchar)
- **created\_at** – Timestamp for when the record was created
- **updated\_at** – Timestamp for the last update to the record

#### 6. Locations

This table stores real-time location data of students.

- **id** – Primary key (integer)
- **student\_id** – Foreign key linking to the students table (integer)
- **latitude** – Latitude of the student's location (float)
- **longitude** – Longitude of the student's location (float)
- **timestamp** – Time when the location was recorded
- **created\_at** – Timestamp for when the record was created
- **updated\_at** – Timestamp for the last update to the record

#### 7. Alerts

This table stores alert information linked to student activities.

- **id** – Primary key (integer)
- **student\_id** – Foreign key linking to the students table (integer)
- **alert\_type** – Type of alert (e.g., late arrival, exit) (varchar)
- **timestamp** – Time when the alert was triggered
- **created\_at** – Timestamp for when the record was created
- **updated\_at** – Timestamp for the last update to the record

## 8. system\_settings

This table stores system-level settings and preferences.

- **id** – Primary key (integer)
- **notification\_preferences** – User-defined notification settings (text)
- **location\_update\_frequency** – Frequency of location updates (integer)
- **system\_alerts** – Types of system alerts (text)
- **language** – Language setting for the system (varchar)
- **time\_zone** – Time zone setting (varchar)
- **created\_at** – Timestamp for when the record was created
- **updated\_at** – Timestamp for the last update to the record

## 9. integration\_settings

This table stores information related to third-party service integrations.

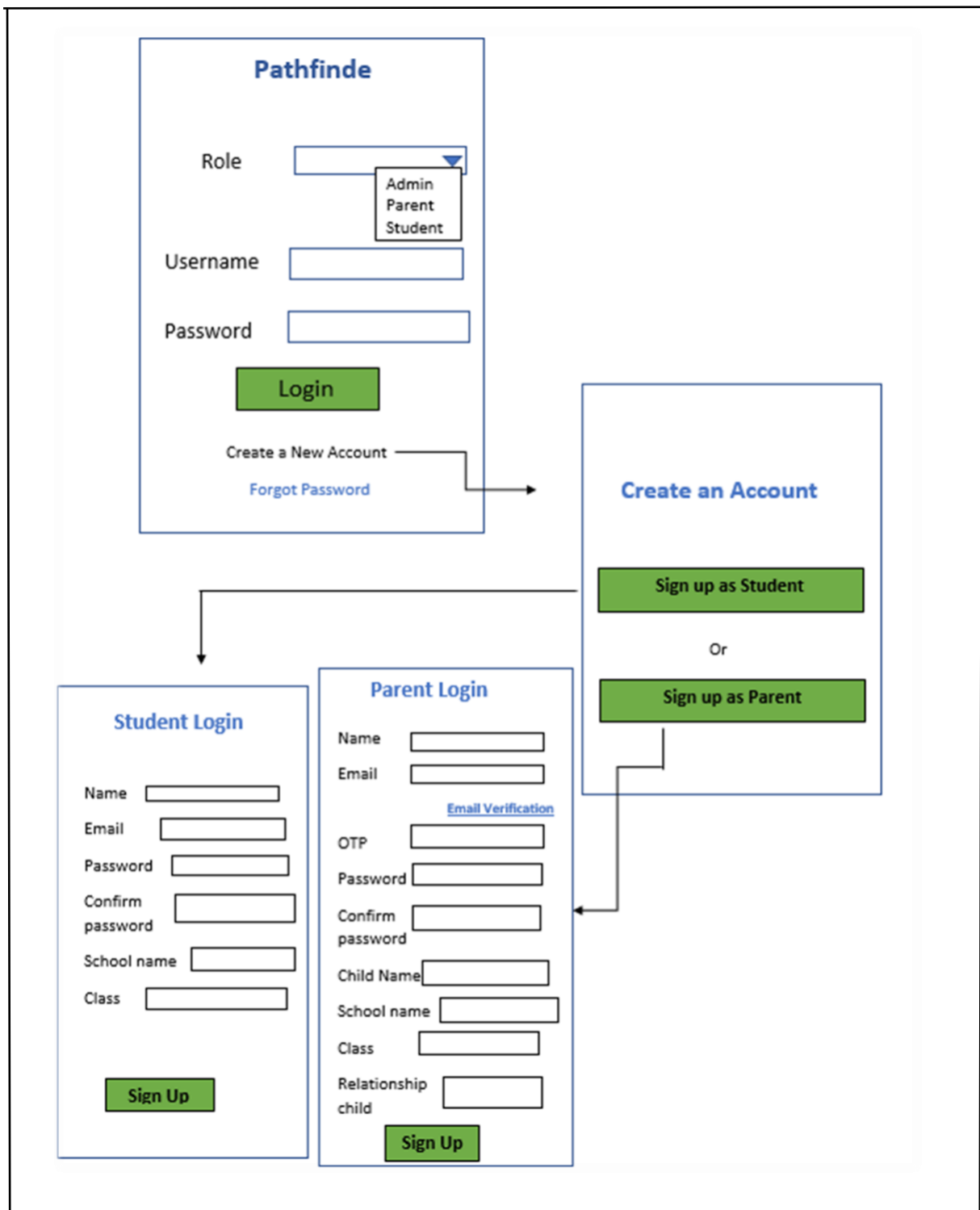
- **id** – Primary key (integer)
- **system\_settings\_id** – Foreign key linking to the system\_settings table (integer)
- **sis\_integration** – Integration with the Student Information System (text)
- **sso\_integration** – Single sign-on integration (text)
- **third\_party\_service\_integration** – Integration with other external services (text)
- **created\_at** – Timestamp for when the record was created
- **updated\_at** – Timestamp for the last update to the record.

# SYSTEM ARCHITECTURE

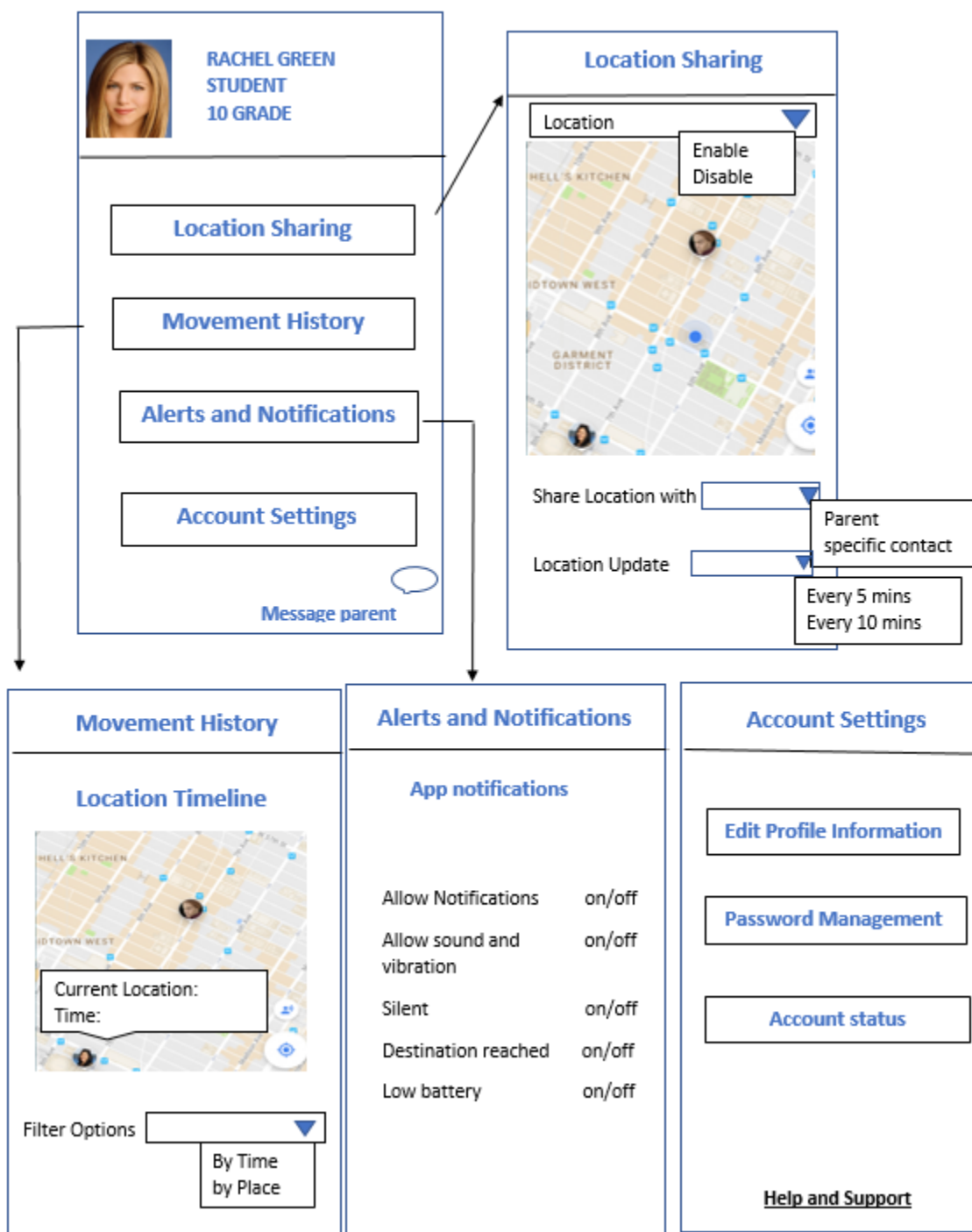


## 4 SYSTEM ARCHITECTURE

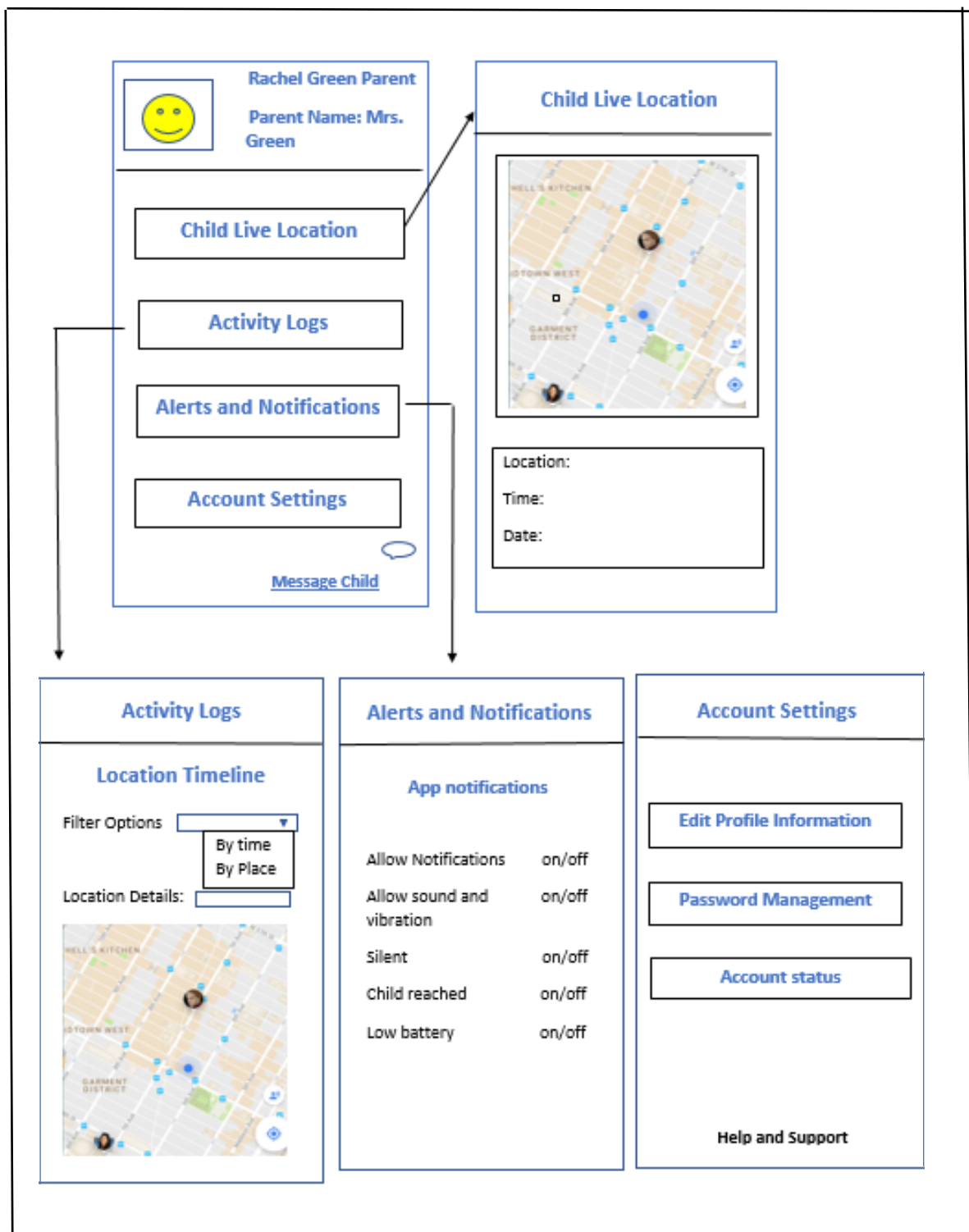
### UI Design (Login page)



## Student Dashboard



## Parent Dashboard

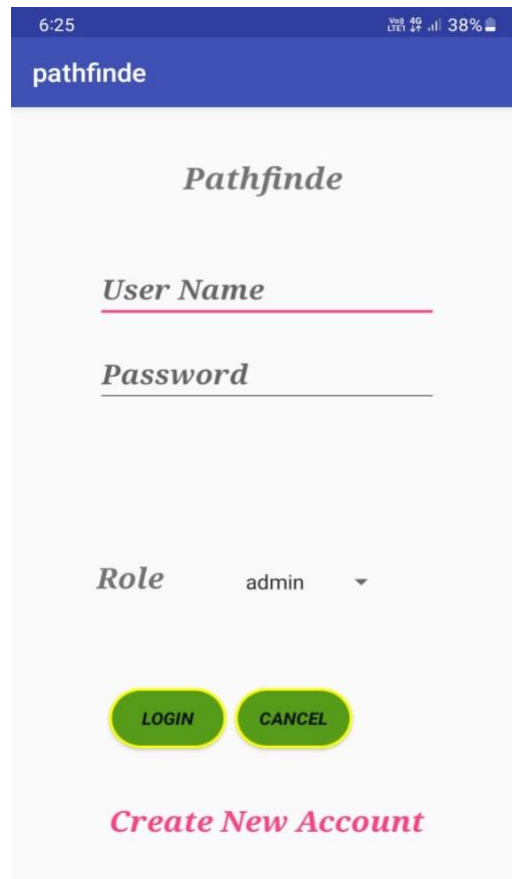


## MODULE DESCRIPTION

## 5 MODULE DESCRIPTION

The Pathfinde app is designed as a real-time student tracking system, developed using **.NET MAUI** and **Xamarin** for cross-platform compatibility, and **SQLite** for secure data storage. The app consists of multiple modules, each tailored to specific user roles, including **Admin**, **Parent**, and **Student**. These modules work together to provide a seamless and secure user experience.

### 1) Login Module :

The screenshot shows the Pathfinde login screen. At the top, there is a blue header bar with the text 'pathfinde' in white. Below the header, the word 'Pathfinde' is displayed in a large, italicized, dark blue font. Underneath, there are two input fields: 'User Name' and 'Password', both in italicized dark blue font. Below these fields is a 'Role' dropdown menu, currently set to 'admin'. At the bottom, there are two green buttons with yellow borders: 'LOGIN' and 'CANCEL'. Below the buttons, there is a pink text link that says 'Create New Account'.

The Pathfinde login module serves as the entry point for the application, enabling user authentication and role-based access. The login interface consists of fields for **User Name** and **Password**, ensuring that user credentials are validated before accessing the system. The user must select their role from the dropdown list, which includes options like **Admin**, **Parent**, and **Student**. This ensures that each user type receives tailored access to specific features and permissions. For example, an Admin can manage user accounts and monitor overall system activity, while Parents can track their child's real-time location and receive alerts. After entering the required information, the user can click the "**LOGIN**" button to authenticate or the "**CANCEL**" button to clear the fields. A "**Create New Account**" option

at the bottom allows new users to register. The module uses backend validation through SQLite, ensuring data security and proper authentication handling.

## 2) Create New Account Module:

The image displays three screenshots of the Pathfinde app's registration module. The first screenshot shows the main registration screen with a blue header 'pathfinde' and two green buttons: 'SIGN UP AS STUDENT' and 'SIGN UP AS PARENT'. The second screenshot shows the 'Student Register' form with fields for 'Enter name', 'EnterPassword', 'Email', 'School name', and 'Class', followed by a green 'SIGNUP' button. The third screenshot shows the 'Parent Register' form with fields for 'Enter name', 'EnterPassword', 'Phone', 'School name', 'Class', 'Child Name', and 'Relationship', followed by a green 'SIGNUP' button.

The Create New Account module is an essential part of the Pathfinde app, designed to allow users to register as either a Student or a Parent. This module is built using .NET MAUI and Xamarin for cross-platform support, ensuring that the app functions seamlessly on both Android and iOS devices. It is linked to the SQLite database on the backend, which securely stores user information and manages authentication data. The module includes user validation and secure handling of passwords, ensuring that sensitive information is protected during registration. The module begins with a screen where users are prompted to select their registration type "Sign up as Student" or "Sign up as Parent." Based on the selection, the app navigates the user to the respective registration form, ensuring that the fields and options are tailored to the selected role. This approach enhances user experience by simplifying the registration process and reducing input errors.

### Student Registration

In the student registration form, the user is required to provide the following details:

- **Enter Name** – The student's full name is captured and stored in the database.

- **Enter Password** – The student creates a password for authentication, which is securely encrypted and stored.
- **Email** – The student's email is used for communication and verification.
- **School Name** – The school associated with the student is selected or manually entered.
- **Class** – The student's grade or class level is recorded.

Once all fields are filled, the user can click the "SIGNUP" button to complete the registration process. The app performs backend validation to ensure that all mandatory fields are completed, the email format is correct, and the password meets security requirements. After successful registration, the student's details are saved in the users and students tables within the SQLite database, and the student is redirected to the login page.

## Parent Registration

The parent registration form includes additional fields to accommodate the relationship between the parent and the student:

- **Enter Name** – The parent's full name.
- **Enter Password** – The parent sets a password for login and authentication.
- **Phone** – The parent's contact number for communication and notifications.
- **School Name** – The school where the student is enrolled.
- **Class** – The student's grade or class level.
- **Child Name** – The parent specifies the child's name to link their account.
- **Relationship** – The parent indicates their relationship with the student (e.g., mother, father, guardian).

After submitting the form, the app validates the details and ensures that the parent-student relationship is properly established in the **parent\_student** table. The parent's information is stored in the users and parents tables, and the relationship is mapped in the **parent\_student** table.

### 3) Student Module



The Student Module in the Pathfinde app is designed to provide students with essential features such as real-time location sharing, movement history tracking, and direct communication with parents through alerts and messaging. Built using .NET MAUI and Xamarin for cross-platform compatibility, the module ensures a seamless user experience on both Android and iOS devices. The module is backed by SQLite for secure data storage and the Geolocation API for accurate real-time location tracking.

#### Key Features and Functionality

Upon successful login, the student is welcomed by name, which enhances personalization and improves user engagement. The main screen displays several key options:



### •LiveLocationShare

The "Live Location Share" feature allows the student to share their real-time location with parents and the administrator. This functionality leverages the **Geolocation API**, which captures the device's GPS coordinates and transmits them to the backend in real time. The latitude and longitude data are stored in the **locations** table within the SQLite database, and the system continuously updates the location at regular intervals. The parents and administrators can view the student's current location using the parent and admin modules, ensuring that the student's whereabouts are continuously monitored for safety and accountability.

### •MovementHistory

The "Movement History" feature provides a detailed record of the student's location history. This data is fetched from the **locations** table, which logs each location update with a timestamp. Students and parents can review past movement patterns, including timestamps and specific locations, which helps track attendance, travel routes, and any deviations from expected paths. This feature ensures transparency and allows parents to monitor their child's movements over time.

### •AlertNotification

The "Alert Notification" feature is one of the most critical components of the student module. In case of emergencies, such as low battery, the student can automatically or manually trigger an alert notification. The alert system sends a real-time SMS to the parent's registered phone number, informing them of the nature of the alert and the student's current status. The example shown in the screenshot indicates a **Low Battery Alert** sent at **14:47:05** on **2025-03-15**. The message includes the exact time and date, allowing parents to respond promptly. The system uses **Twilio** or similar messaging APIs to deliver alerts reliably. This feature enhances student safety by enabling immediate communication with parents during critical situations.

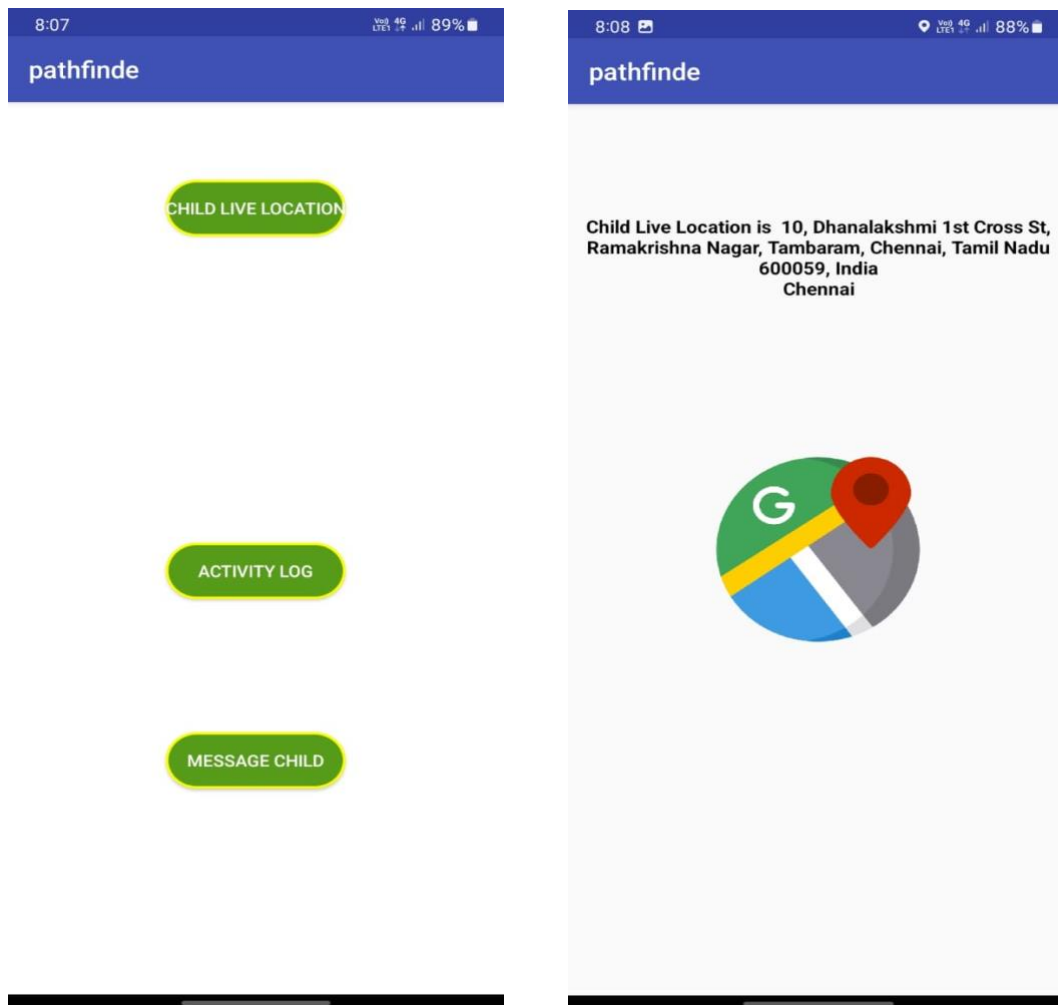
### •MessageParent

The "Message Parent" feature allows students to send direct messages to their parents in real time. This feature is integrated with a messaging service that ensures secure and fast delivery. Students can use this feature to notify parents about changes in plans, delays, or any other important information. The messaging history is also recorded in the system for future reference.

### •AddressDisplay

The bottom of the screen displays the student's registered address. This provides a quick reference for the student and serves as a validation point for parents and administrators when verifying location data.

## 4) Parent Module



The Parent Module in the Pathfinde app is designed to provide parents with real-time access to their child's location, activity history, and communication options, ensuring enhanced safety and security. Built using .NET MAUI and Xamarin, the module leverages cross-platform compatibility to work seamlessly on both Android and iOS devices. The primary goal of the Parent Module is to enable parents to monitor their child's location, receive updates on movement history, and maintain direct communication through the app. The integration of Geolocation APIs allows accurate real-time tracking of the child's movements, with the

collected location data securely stored in a SQLite database. The user interface is designed to be simple and intuitive, making it easy for parents to navigate and access key information.

The module consists of three main features: Child Live Location, Activity Log, and Message Child. The Child Live Location feature allows parents to view their child's real-time location directly on a map. By utilizing the device's GPS capabilities and the Geolocation API, the app retrieves the latitude and longitude values from the SQLite database and displays the location in an address format that includes street name, city, and postal code. When the parent taps the "Child Live Location" button, the app queries the backend for the latest location data linked to the student's ID and displays the location on a Google Map interface. This feature ensures that parents are constantly aware of their child's location, which enhances peace of mind and allows for quick intervention in case of unexpected changes in the route or potential safety threats.

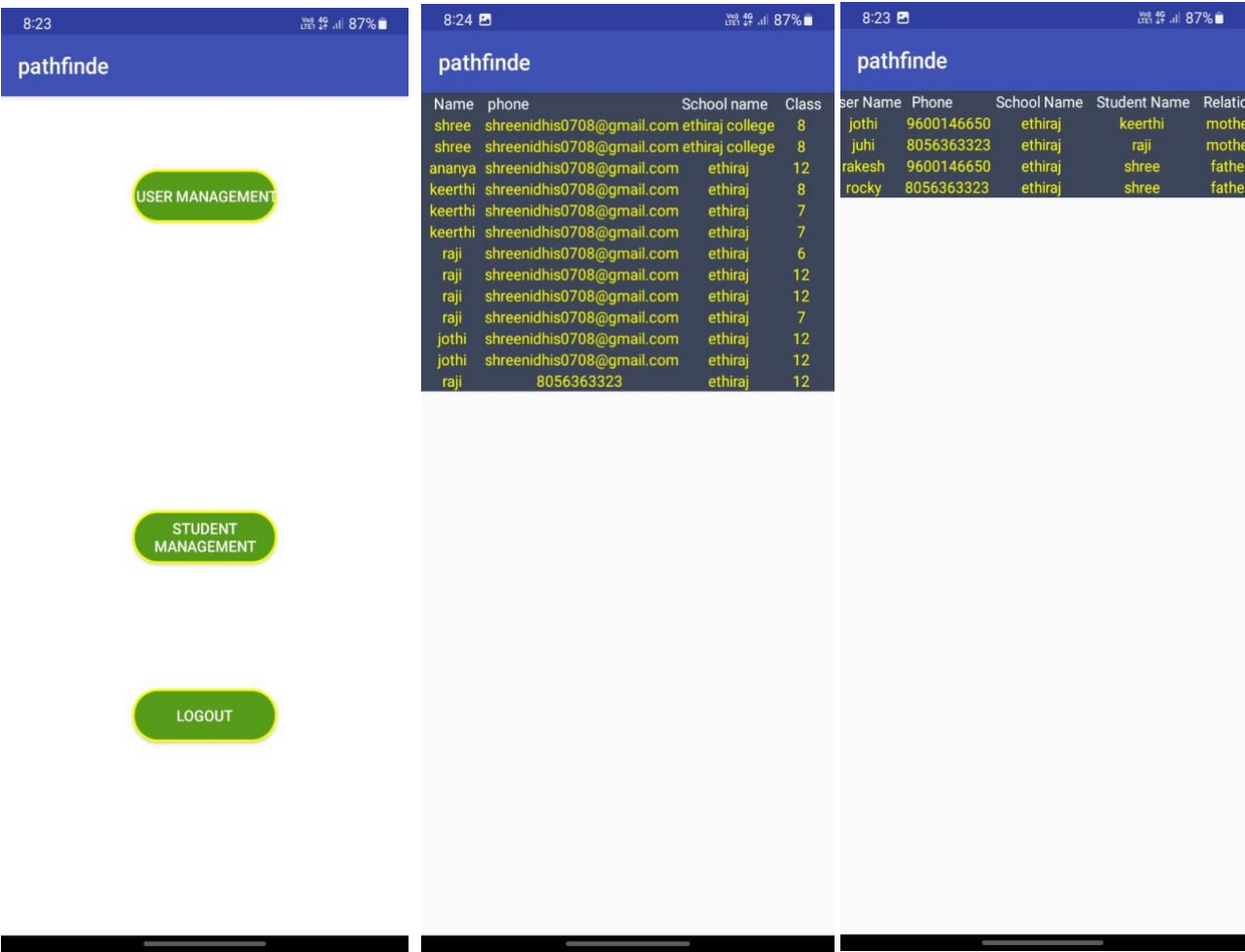
The Activity Log feature enables parents to access a detailed history of their child's movements. The location data is logged with timestamps in the SQLite database, allowing parents to review past routes and identify patterns or deviations from regular travel paths. This feature helps parents monitor attendance, track consistent travel behavior, and investigate any unusual stops or delays. The activity log ensures that all movement data is encrypted and stored securely, protecting it from unauthorized access. By having access to this detailed movement history, parents can make informed decisions about their child's safety and adjust travel routines if needed. The ability to access historical data also allows parents to provide valuable information in case of emergencies or incidents involving their child's whereabouts.

The Message Child feature allows parents to communicate directly with their child through the app. This feature supports real-time two-way communication, enabling parents to send reminders, alerts, or important updates. The messages are delivered instantly, and the child can respond within the app, ensuring a seamless exchange of information. The messaging service is secured with end-to-end encryption, ensuring that all communications remain private and protected from external access. This feature ensures that parents and children can maintain clear and immediate contact, which is particularly useful during emergencies or when a change in schedule occurs. The messaging history is stored within the app, allowing both parties to reference previous conversations if needed.

The backend infrastructure of the Parent Module is designed to handle sensitive location and communication data securely. The app uses secure authentication methods and role-based

access control to ensure that only authorized parents can access the information. Data synchronization occurs in real time, ensuring that location updates, activity logs, and messages are delivered without delays. This secure and responsive system ensures that parents have constant and reliable access to their child's location and communication history. The Parent Module enhances the overall functionality of the Pathfinde app by providing parents with a comprehensive and secure tool for monitoring their child's movements and maintaining communication. The combination of real-time tracking, historical data analysis, and direct messaging creates a robust safety net, ensuring that parents remain informed and connected with their children at all times.

### 5) Admin Module



(Admin dashboard)

(User Management)

(Student Management)

The Admin Module in the Pathfinde app is a comprehensive and powerful tool designed to give the administrator complete control over the system's functioning and user management. It is one of the most important modules in the application as it ensures smooth operation, security, and efficient user handling. The module allows the admin to manage user accounts, student details, and system settings while also providing real-time insights into the app's overall performance. Built using .NET MAUI and Xamarin, the Admin Module connects directly with the backend SQLite database, which handles data storage, retrieval, and synchronization. The admin module's interface is designed to be intuitive and responsive, ensuring that the admin can easily access and modify data as needed.

The Admin Module is divided into three key sections: Admin Dashboard, User Management, and Student Management. The Admin Dashboard serves as the main entry point for the administrator, offering quick access to essential management features. From the dashboard, the admin can navigate to different sections such as User Management and Student Management with just a single tap. The dashboard also includes a logout button to ensure secure access, allowing the admin to exit the system when their session ends. This ensures that unauthorized users cannot access the admin interface without proper credentials.

The User Management section allows the administrator to view and manage user accounts efficiently. The admin can see detailed information about each user, including their name, phone number, email, school name, and class. The admin can also search, sort, and filter users based on different parameters to simplify data retrieval. This section is particularly useful for managing large datasets, as the admin can quickly identify and resolve any user-related issues. If a user forgets their password or encounters access issues, the admin can reset the password or modify account settings directly from this section. The admin also has the authority to create new user accounts or delete existing ones, ensuring that the system remains up to date and secure.

The Student Management section focuses on handling student-related data. The admin can view detailed information about each student, including their name, phone number, school, class, and associated parent. This section allows the admin to manage student-parent relationships effectively. For instance, if a student changes schools or updates their contact details, the admin can modify the records directly. The admin can also assign new parent-student relationships, which ensures that communication between parents and students is accurately reflected within the system. This feature is particularly useful in cases where parents

have multiple children enrolled in different schools, as it allows for seamless tracking and management of each student's data.

One of the most critical functions of the Admin Module is its real-time data monitoring capability. The admin can access live location data from the student's device and monitor their movements through an integrated map interface. This ensures that students are within safe zones and allows the admin to respond quickly to any unusual activity. If a student crosses a predefined geographical boundary, the admin can trigger an alert to notify the parent immediately. This proactive approach enhances the security of the students and gives parents peace of mind regarding their child's safety.

The Admin Module also features a robust notification system. The admin can send alerts and messages to both parents and students regarding emergencies, school events, or general announcements. The notification system is integrated with the SQLite database, which ensures that all messages are logged for future reference. The admin can also set up automatic notifications based on predefined rules, such as location-based alerts or session timeouts. This ensures that critical information reaches the intended recipients without delay and enhances overall communication efficiency within the app.

Data security and privacy are central to the Admin Module. The admin has the authority to define access levels for different user roles, ensuring that sensitive data remains protected. The module supports encrypted data transmission and secure login mechanisms, which prevent unauthorized access. The admin can also monitor system logs to identify potential security threats and take corrective actions if necessary. Furthermore, the module includes an option for regular data backups, ensuring that all user data is preserved and can be restored in case of system failure or corruption.

The reporting and analytics capability of the Admin Module enables the administrator to generate detailed reports on user activity, student performance, and system health. These reports provide valuable insights into user engagement, location trends, and communication effectiveness. The admin can use this data to identify areas for improvement and optimize the overall app experience. The ability to analyze data and adjust system settings based on real-world usage patterns ensures that the Pathfinde app remains efficient and user-friendly.

In summary, the Admin Module is the backbone of the Pathfinde app, providing the administrator with comprehensive control over user and student management, real-time data monitoring, communication, and system configuration. Its intuitive interface, combined with powerful backend integration, ensures that the administrator can efficiently manage all aspects of the app's functionality. By giving the administrator the ability to monitor, control, and analyze the system's performance, the Admin Module plays a crucial role in maintaining the app's reliability, security, and user satisfaction.

## **IMPLEMENTATION**



## 6.1 CODING

### Location.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Text.Json.Serialization;
namespace Pathfinde.Models
{
    public class StartTimerRequest
    {
        public string UserId { get; set; }
        public int RoleId { get; set; }
        public int IntervalMinutes { get; set; }
    }
    public class StopTimerRequest
    {
        public string UserId { get; set; }
        public int RoleId { get; set; }
    }
    public class LocationUpdateRequest
    {
        public string UserId { get; set; } // Same for both parent and student
        public int RoleId { get; set; } = 3; // Must be 3 for students
        public double Latitude { get; set; }
        public double Longitude { get; set; }
    }
    public class LocationDataResponse
    {
        public bool RunStatus { get; set; }
        public List<LocationResponse> Locations { get; set; }
        public int IntervalSeconds { get; set; } // Parent's set interval in seconds
    }
}
```

```

        public int Status { get; set; }
    }
    public class LocationResponse
    {
        public double Latitude { get; set; }
        public double Longitude { get; set; }
        public DateTime Timestamp { get; set; }
        public string LocationName { get; set; } // if you also reverse geocode the location
    }
}

```

## Login.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Pathfinde.Models
{
    public class LoginRequest
    {
        public string UserId { get; set; }
        public string Password { get; set; }
        public int RoleId { get; set; }
    }
    public class LoginResponse
    {
        public string Message { get; set; }
        public string Token { get; set; }
        public int RoleId { get; set; }
        public string UserId { get; set; }
    }
}

```

## User.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Pathfinde.Models
{
    public class UserDetails
    {
        public int Id { get; set; } // Primary Key
        public string UserId { get; set; } // Mobile Number (Used as Username)
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int? Age { get; set; }
        public string SecondPhoneNo { get; set; }
        public string Email { get; set; }
        public string SecondEmail { get; set; }
        public string AadhaarNo { get; set; }
        public string HomeAddress { get; set; }
        public string PasswordHash { get; set; } // Hashed Password
        public string Salt { get; set; } // Salt for Password Hashing
        public bool IsActive { get; set; } = true; // Default to Active User
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
        public DateTime? UpdatedAt { get; set; }
    }
    public class CreateUserRequest
    {
        public string UserId { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int? Age { get; set; }
        public string SecondPhoneNo { get; set; }
```

```
    public String Email { get; set; }  
    public String SecondEmail { get; set; }  
    public String AadhaarNo { get; set; }  
    public String HomeAddress { get; set; }  
    public String Password { get; set; }  
}  
}
```

## Android MainActivity.cs

```
using Android.App;
using Android.Content.PM;
using Android.OS;
namespace Pathfinde
{
    [Activity(Theme = "@style/Maui.SplashTheme", MainLauncher = true,
        ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation |
        ConfigChanges.UiMode | ConfigChanges.SmallestScreenSize |
ConfigChanges.Density)]
    public class MainActivity : MauiAppCompatActivity
    {
        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            // Request runtime permissions
            RequestPermissions(new string[]
            {
                Android.Manifest.Permission.AccessFineLocation,
                Android.Manifest.Permission.AccessCoarseLocation,
                Android.Manifest.Permission.ForegroundService
            }, 0);
        }
    }
}
```

## Mainapplication.cs

```
using Android.App;
using Android.Runtime;

namespace Pathfinde
{
    [Application]
    public class MainApplication : MauiApplication
    {
        public MainApplication(IntPtr handle, JniHandleOwnership ownership)
            : base(handle, ownership)
        {
        }

        protected override MauiApp CreateMauiApp() => MauiProgram.CreateMauiApp();
    }
}
```

## Mainpage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Pathfinde.MainPage">
    <VerticalStackLayout>
        <Button Text="New+"
            HorizontalOptions="End"
            VerticalOptions="Start"
            Margin="10"
            Clicked="OnAddNewClicked"/>
        <Grid>
            <VerticalStackLayout Padding="20" Spacing="15">
                <Label Text="User ID" />
                <Entry x:Name="UserIdEntry" Placeholder="Enter User ID" />

                <Label Text="Password" />
                <Entry x:Name="PasswordEntry" Placeholder="Enter Password" IsPassword="True"
            />
        />
    />
```

```
<!-- Role Picker -->
<Label Text="Select Role" />
<Picker x:Name="RolePicker"
    Title="Select a Role"
/>

<!-- Login Button -->
<Button Text="Login"
    Clicked="OnLoginClicked"
    HorizontalOptions="Center" />
</VerticalStackLayout>
</Grid>
</VerticalStackLayout>
</ContentPage>
```

## Mainpage.cs

```
using System;
using System.Net.Http.Json;
using Microsoft.Maui.Controls;
using Pathfinde.views;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Maui.Controls;
using Pathfinde.Models;
using Microsoft.Maui.Storage;
using Microsoft.Maui;
namespace Pathfinde
{
    public partial class MainPage : ContentPage
    {
        private readonly HttpClient _httpClient;
        private readonly Dictionary<int, string> _roles = new Dictionary<int, string>
        {
            { 1, "Admin" },
            { 2, "Parent" },
            { 3, "Student" }
        };
        public MainPage()
        {
            InitializeComponent();
            _httpClient = new HttpClient();
            RolePicker.ItemsSource = new List<string> { "Admin", "Parent", "Student" };
        }
        // Event handler for the "New+" button click
        private async void OnAddNewClicked(object sender, EventArgs e)
        {

```



```

        Navigation.PushAsync(new CreateUserPage());
    }
    private async void OnLoginClicked(object sender, EventArgs e)
    {
        string userId = UserIdEntry.Text?.Trim();
        string password = PasswordEntry.Text?.Trim();
        string selectedRoleName = RolePicker.SelectedItem as string;
        if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(password))
        {
            await DisplayAlert("Error", "UserId and Password are required.", "OK");
            return;
        }
        if (string.IsNullOrEmpty(selectedRoleName))
        {
            await DisplayAlert("Error", "Please select a role.", "OK");
            return;
        }
        // Convert the selected role name to a RoleId.
        int selectedRoleId = 0;
        foreach (var kvp in _roles)
        {
            if (kvp.Value == selectedRoleName)
            {
                selectedRoleId = kvp.Key;
                break;
            }
        }
        // Build the LoginRequest.
        var loginRequest = new LoginRequest
        {
            UserId = userId,
            Password = password,
            RoleId = selectedRoleId
        };
    }

```

```

try
{
    // URL for your API's login endpoint.
    string apiUrl = "https://localhost:44397/api/Auth/login";
    var response = await _httpClient.PostAsJsonAsync(apiUrl, loginRequest);
    if (response.IsSuccessStatusCode)
    {
        var loginResponse = await
response.Content.ReadFromJsonAsync<LoginResponse>();
        // If the API requires role selection, handle it here.
        if (loginResponse != null && loginResponse.Message == "Select a Role")
        {
            await DisplayAlert("Info", "The server requested a role selection.", "OK");
            return;
        }
        // Show token for demo (optional)
        await DisplayAlert("Success", $"Login successful!\nToken:
{loginResponse?.Token}", "OK");
        // Store the token if available.
        if (!string.IsNullOrEmpty(loginResponse?.Token))
        {
            Preferences.Set("AuthToken", loginResponse.Token);
        }
        // Store the entered UserId (or one returned from the API) in Preferences.
        Preferences.Set("UserId", userId);
        // Navigate based on role.
        if (selectedRoleId == 2)
        {
            await Navigation.PushAsync(new ParentDashboard());
        }
        else if (selectedRoleId == 3)
        {
            await Navigation.PushAsync(new StudentDashboard());
        }
    }
}

```

```
        else
        {
            await DisplayAlert("Info", "Role not supported for navigation.", "OK");
        }
    }
    else
    {
        var errorContent = await response.Content.ReadAsStringAsync();
        var statusCode = response.StatusCode;
        await DisplayAlert("Login Failed", $"Status: {statusCode} \n {errorContent}",
"OK");
    }
}
catch (Exception ex)
{
    await DisplayAlert("Exception", ex.Message, "OK");
}
}
}
```

## Maui program.cs

```
using Microsoft.Extensions.Logging;
namespace Pathfinde
{
    public static class MauiProgram
    {
        public static MauiApp CreateMauiApp()
        {
            var builder = MauiApp.CreateBuilder();
            builder
                .UseMauiApp<App>()
                .ConfigureFonts(fonts =>
                {
                    fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                    fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
                });
#if DEBUG
            builder.Logging.AddDebug();
#endif
            return builder.Build();
        }
    }
}
```

## Pathfinde.csproject

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFrameworks>net8.0-ios;net8.0-maccatalyst;net8.0-
android34.0</TargetFrameworks>
    <AndroidPackageFormat>apk</AndroidPackageFormat>
    <TargetFrameworks
Condition="$([MSBuild]::IsOSPlatform('windows'))">$(TargetFrameworks);net8.0-
windows10.0.19041.0</TargetFrameworks>
    <!-- Uncomment to also build the tizen app. You will need to install tizen by
following this: https://github.com/Samsung/Tizen.NET -->
    <!-- <TargetFrameworks>$(TargetFrameworks);net8.0-
tizen</TargetFrameworks> -->
    <!-- Note for MacCatalyst:
The default runtime is maccatalyst-x64, except in Release config, in which
case the default is maccatalyst-x64;maccatalyst-arm64.
When specifying both architectures, use the plural <RuntimeIdentifiers>
instead of the singular <RuntimeIdentifier>.
The Mac App Store will NOT accept apps with ONLY maccatalyst-arm64
indicated;
either BOTH runtimes must be indicated or ONLY macatalyst-x64. -->
    <!-- For example: <RuntimeIdentifiers>maccatalyst-x64;maccatalyst-
arm64</RuntimeIdentifiers> -->
    <OutputType>Exe</OutputType>
    <RootNamespace>Pathfinde</RootNamespace>
    <UseMaui>true</UseMaui>
    <SingleProject>true</SingleProject>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <!-- Display name -->
    <ApplicationTitle>Pathfinde</ApplicationTitle>
```

```

    <!-- App Identifier -->
    <ApplicationId>com.companyname.pathfinde</ApplicationId>
    <!-- Versions -->
    <ApplicationDisplayVersion>1.0</ApplicationDisplayVersion>
    <ApplicationVersion>1</ApplicationVersion>
    <SupportedOSPlatformVersion
Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) ==
'ios'">11.0</SupportedOSPlatformVersion>
    <SupportedOSPlatformVersion
Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) ==
'maccatalyst'">13.1</SupportedOSPlatformVersion>
    <SupportedOSPlatformVersion
Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) ==
'android'">21.0</SupportedOSPlatformVersion>
    <SupportedOSPlatformVersion
Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) ==
'windows'">10.0.17763.0</SupportedOSPlatformVersion>
    <TargetPlatformMinVersion
Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) ==
'windows'">10.0.17763.0</TargetPlatformMinVersion>
    <SupportedOSPlatformVersion
Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) ==
'tizen'">6.5</SupportedOSPlatformVersion>
    <Platforms>AnyCPU;ARM32</Platforms>
</PropertyGroup>
<PropertyGroup
Condition=" '$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
android|AnyCPU'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
<PropertyGroup
Condition=" '$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
android|ARM32'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>

```

```
</PropertyGroup>
<PropertyGroup
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
ios|AnyCPU'">
  <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
<PropertyGroup
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
ios|ARM32'">
  <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
<PropertyGroup
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
maccatalyst|AnyCPU'">
  <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
<PropertyGroup
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
maccatalyst|ARM32'">
  <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
<PropertyGroup
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
windows10.0.19041.0|AnyCPU'">
  <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
<PropertyGroup
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
windows10.0.19041.0|ARM32'">
  <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
```

```

    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
android|AnyCPU'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
android|ARM32'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
ios|AnyCPU'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
ios|ARM32'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
maccatalyst|AnyCPU'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
maccatalyst|ARM32'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
windows10.0.19041.0|AnyCPU'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>

```



```

    </PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
windows10.0.19041.0|ARM32'">
        <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
    </PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
android34.0|AnyCPU'">
        <AndroidKeyStore>False</AndroidKeyStore>
        <AndroidSigningStorePass>shree123</AndroidSigningStorePass>
        <AndroidSigningKeyAlias>pathfinde-new-key</AndroidSigningKeyAlias>
        <AndroidSigningKeyPass>shree123</AndroidSigningKeyPass>
    </PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
android34.0|ARM32'">
        <AndroidKeyStore>False</AndroidKeyStore>
        <AndroidSigningStorePass>shree123</AndroidSigningStorePass>
        <AndroidSigningKeyAlias>pathfinde-new-key</AndroidSigningKeyAlias>
        <AndroidSigningKeyPass>shree123</AndroidSigningKeyPass>
    </PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
android34.0|AnyCPU'">
        <AndroidKeyStore>False</AndroidKeyStore>
        <AndroidSigningStorePass>shree123</AndroidSigningStorePass>
        <AndroidSigningKeyAlias>pathfinde-new-key</AndroidSigningKeyAlias>
        <AndroidSigningKeyPass>shree123</AndroidSigningKeyPass>
    </PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
android34.0|ARM32'">
        <AndroidKeyStore>False</AndroidKeyStore>

```

```

    <AndroidSigningStorePass>shree123</AndroidSigningStorePass>
    <AndroidSigningKeyAlias>pathfinde-new-key</AndroidSigningKeyAlias>
    <AndroidSigningKeyPass>shree123</AndroidSigningKeyPass>
</PropertyGroup>
<ItemGroup>
    <!-- App Icon -->
    <MauiIcon Include="Resources\AppIcon\appicon.svg"
ForegroundFile="Resources\AppIcon\appiconfg.svg" Color="#512BD4" />
    <!-- Splash Screen -->
    <MauiSplashScreen Include="Resources\Images\pathfinde.png"
Color="White" BaseSize="128,128" />
    <!-- Images -->
    <MauiImage Include="Resources\Images\*" />
    <MauiImage Update="Resources\Images\dotnet_bot.png" Resize="True"
BaseSize="300,185" />
    <!-- Custom Fonts -->
    <MauiFont Include="Resources\Fonts\*" />
    <!-- Raw Assets (also remove the "Resources\Raw" prefix) -->
    <MauiAsset Include="Resources\Raw\*"
LogicalName="%%(RecursiveDir)%(Filename)%(Extension)" />
</ItemGroup>
<ItemGroup>
    <MauiImage Remove="Resources\Images\pathfinde.png" />
</ItemGroup>
<ItemGroup>
    <None Remove="Resources\Images\mail.jpg" />
    <None Remove="Resources\Images\password.jpg" />
    <None Remove="Resources\Images\pathfinde.png" />
    <None Remove="Resources\Images\phone.png" />
    <None Remove="Resources\Images\role.jpg" />
    <None Remove="Resources\Images\user.png" />
</ItemGroup>
<ItemGroup>

```

```

        <PackageReference Include="Microsoft.Maui.Controls"
Version="$(MauiVersion)" />
        <PackageReference Include="Microsoft.Maui.Controls.Compatibility"
Version="$(MauiVersion)" />
        <PackageReference Include="Microsoft.Extensions.Logging.Debug"
Version="8.0.0" />
    </ItemGroup>
    <ItemGroup>
        <MauiXaml Update="views\AdminDashboard.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\CreateUserPage.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\LocationPage.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\ParentDashboard.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\SecureDataPage.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\SignInPage.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\StudentDashboard.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
    </ItemGroup>
    <PropertyGroup
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'== 'Debug|net8.0-
android|AnyCPU'">
        <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
    </PropertyGroup>

```

</PropertyGroup>

<PropertyGroup

Condition="''\$(Configuration)|\$(TargetFramework)|\$(Platform)'=='Debug|net8.0-android|ARM32''>

<ApplicationId>com.yarcubes.pathfinde</ApplicationId>

</PropertyGroup>

<PropertyGroup

Condition="''\$(Configuration)|\$(TargetFramework)|\$(Platform)'=='Debug|net8.0-ios|AnyCPU''>

<ApplicationId>com.yarcubes.pathfinde</ApplicationId>

</PropertyGroup>

<PropertyGroup

Condition="''\$(Configuration)|\$(TargetFramework)|\$(Platform)'=='Debug|net8.0-ios|ARM32''>

<ApplicationId>com.yarcubes.pathfinde</ApplicationId>

</PropertyGroup>

<PropertyGroup

Condition="''\$(Configuration)|\$(TargetFramework)|\$(Platform)'=='Debug|net8.0-maccatalyst|AnyCPU''>

<ApplicationId>com.yarcubes.pathfinde</ApplicationId>

</PropertyGroup>

<PropertyGroup

Condition="''\$(Configuration)|\$(TargetFramework)|\$(Platform)'=='Debug|net8.0-maccatalyst|ARM32''>

<ApplicationId>com.yarcubes.pathfinde</ApplicationId>

</PropertyGroup>

<PropertyGroup

Condition="''\$(Configuration)|\$(TargetFramework)|\$(Platform)'=='Debug|net8.0-windows10.0.19041.0|AnyCPU''>

<ApplicationId>com.yarcubes.pathfinde</ApplicationId>

</PropertyGroup>

```

    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
windows10.0.19041.0|ARM32'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
android|AnyCPU'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
android|ARM32'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
ios|AnyCPU'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
ios|ARM32'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>

    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
maccatalyst|AnyCPU'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
    <PropertyGroup
Condition="''$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
maccatalyst|ARM32'">

```

```
<ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
<PropertyGroup
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
windows10.0.19041.0|AnyCPU'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
<PropertyGroup>Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='
Release|net8.0-windows10.0.19041.0|ARM32'">
    <ApplicationId>com.yarcubes.pathfinde</ApplicationId>
</PropertyGroup>
<PropertyGroup>
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net8.0-
android34.0|ARM32'">
    <AndroidKeyStore>False</AndroidKeyStore>
    <AndroidSigningStorePass>shree123</AndroidSigningStorePass>
    <AndroidSigningKeyAlias>pathfinde-new-key</AndroidSigningKeyAlias>
    <AndroidSigningKeyPass>shree123</AndroidSigningKeyPass>
</PropertyGroup>
<PropertyGroup>
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
android34.0|AnyCPU'">
    <AndroidKeyStore>False</AndroidKeyStore>
    <AndroidSigningStorePass>shree123</AndroidSigningStorePass>
    <AndroidSigningKeyAlias>pathfinde-new-key</AndroidSigningKeyAlias>
    <AndroidSigningKeyPass>shree123</AndroidSigningKeyPass>
</PropertyGroup>
<PropertyGroup>
Condition="$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net8.0-
android34.0|ARM32'">
    <AndroidKeyStore>False</AndroidKeyStore>
    <AndroidSigningStorePass>shree123</AndroidSigningStorePass>
    <AndroidSigningKeyAlias>pathfinde-new-key</AndroidSigningKeyAlias>
    <AndroidSigningKeyPass>shree123</AndroidSigningKeyPass>
```

```

</PropertyGroup>
<ItemGroup>
  <!-- App Icon -->
  <MauiIcon Include="Resources\AppIcon\appicon.svg"
ForegroundFile="Resources\AppIcon\appiconfg.svg" Color="#512BD4" />
  <!-- Splash Screen -->
  <MauiSplashScreen Include="Resources\Images\pathfinde.png"
Color="White" BaseSize="128,128" />
  <!-- Images -->
  <MauiImage Include="Resources\Images\*" />
  <MauiImage Update="Resources\Images\dotnet_bot.png" Resize="True"
BaseSize="300,185" />
  <!-- Custom Fonts -->
  <MauiFont Include="Resources\Fonts\*" />

  <!-- Raw Assets (also remove the "Resources\Raw" prefix) -->
  <MauiAsset Include="Resources\Raw\**"
LogicalName="%%(RecursiveDir)%(Filename)%(Extension)" />
</ItemGroup>
<ItemGroup>
  <MauiImage Remove="Resources\Images\pathfinde.png" />
</ItemGroup>
<ItemGroup>
  <None Remove="Resources\Images\mail.jpg" />
  <None Remove="Resources\Images\password.jpg" />
  <None Remove="Resources\Images\pathfinde.png" />
  <None Remove="Resources\Images\phone.png" />
  <None Remove="Resources\Images\role.jpg" />
  <None Remove="Resources\Images\user.png" />
</ItemGroup>
<ItemGroup>
  <PackageReference Include="Microsoft.Maui.Controls"
Version="$(MauiVersion)" />

```

```
        <PackageReference Include="Microsoft.Maui.Controls.Compatibility"
Version="$(MauiVersion)" />
        <PackageReference Include="Microsoft.Extensions.Logging.Debug"
Version="8.0.0" />
    </ItemGroup>
    <ItemGroup>
        <MauiXaml Update="views\AdminDashboard.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\CreateUserPage.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\LocationPage.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\ParentDashboard.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\SecureDataPage.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\SignInPage.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
        <MauiXaml Update="views\StudentDashboard.xaml">
            <Generator>MSBuild:Compile</Generator>
        </MauiXaml>
    </ItemGroup>
</Project>
```



## Student mainpage.cs

```
using System;
using System.Net.Http.Json;
using Microsoft.Maui.Controls;
using Pathfinde.views;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Maui.Controls;
using Pathfinde.Models;
using Microsoft.Maui.Storage;
using Microsoft.Maui;
namespace Pathfinde
{
    public partial class MainPage : ContentPage
    {
        private readonly HttpClient _httpClient;
        private readonly Dictionary<int, string> _roles = new Dictionary<int, string>
        {
            { 1, "Admin" },
            { 2, "Parent" },
            { 3, "Student" }
        };
        public MainPage()
        {
            InitializeComponent();
            _httpClient = new HttpClient();
            RolePicker.ItemsSource = new List<string> { "Admin", "Parent", "Student" };
        }
        // Event handler for the "New+" button click
        private async void OnAddNewClicked(object sender, EventArgs e)
        {
            Navigation.PushAsync(new CreateUserPage());
        }
    }
}
```

```

    }
    private async void OnLoginClicked(object sender, EventArgs e)
    {
        string userId = UserIdEntry.Text?.Trim();
        string password = PasswordEntry.Text?.Trim();
        string selectedRoleName = RolePicker.SelectedItem as string;

        if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(password))
        {
            await DisplayAlert("Error", "UserId and Password are required.", "OK");
            return;
        }
        if (string.IsNullOrEmpty(selectedRoleName))
        {
            await DisplayAlert("Error", "Please select a role.", "OK");
            return;
        }
        // Convert the selected role name to a RoleId.
        int selectedRoleId = 0;
        foreach (var kvp in _roles)
        {
            if (kvp.Value == selectedRoleName)
            {
                selectedRoleId = kvp.Key;
                break;
            }
        }
        // Build the LoginRequest.
        var loginRequest = new LoginRequest
        {
            UserId = userId,
            Password = password,
            RoleId = selectedRoleId
        };
    }

```

```

try
{
    // URL for your API's login endpoint.
    string apiUrl = "https://localhost:44397/api/Auth/login";
    var response = await _httpClient.PostAsJsonAsync(apiUrl, loginRequest);
    if (response.IsSuccessStatusCode)
    {
        var loginResponse = await
response.Content.ReadFromJsonAsync<LoginResponse>();

        // If the API requires role selection, handle it here.
        if (loginResponse != null && loginResponse.Message == "Select a Role")
        {
            await DisplayAlert("Info", "The server requested a role selection.", "OK");
            return;
        }

        // Show token for demo (optional)
        await DisplayAlert("Success", $"Login successful!\nToken:
{loginResponse?.Token}", "OK");

        // Store the token if available.
        if (!string.IsNullOrEmpty(loginResponse?.Token))
        {
            Preferences.Set("AuthToken", loginResponse.Token);
        }

        // Store the entered UserId (or one returned from the API) in Preferences.
        Preferences.Set("UserId", userId);

        // Navigate based on role.
        if (selectedRoleId == 2)
        {
            await Navigation.PushAsync(new ParentDashboard());
        }
    }
}

```

```
        else if (selectedRoleId == 3)
        {
            await Navigation.PushAsync(new StudentDashboard());
        }
        else
        {
            await DisplayAlert("Info", "Role not supported for navigation.", "OK");
        }
    }
    else
    {
        var errorContent = await response.Content.ReadAsStringAsync();
        var statusCode = response.StatusCode;
        await DisplayAlert("Login Failed", $"Status: {statusCode}\n{errorContent}",
"OK");
    }
}
catch (Exception ex)
{
    await DisplayAlert("Exception", ex.Message, "OK");
}
}
}
```

## Authcontroller.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using Microsoft.IdentityModel.Tokens;
using Dapper;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;
using pathfinde.Models;
namespace pathfinde.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly IConfiguration _configuration;
        public AuthController(IConfiguration configuration)
        {
            _configuration = configuration;
        }
        /// <summary>
        /// Logs in a user (Student or Parent) using UserId and Password.
        /// </summary>
        [HttpPost("login")]
        public IActionResult Login([FromBody] LoginRequest request)
        {
            if (string.IsNullOrEmpty(request.UserId) ||
string.IsNullOrEmpty(request.Password))
            {
                return BadRequest(new { Message = "UserId and Password are required." });
            }
        }
    }
}
```

```

try
{
    using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
    {
        connection.Open();

        // ♣ Fix: Use `QueryFirstOrDefault()` instead of `QuerySingleOrDefault()`

        var userQuery = @"
SELECT TOP 1 d.UserId, d.FirstName, d.PasswordHash, d.Salt, u.RoleId
FROM UserDetails d
JOIN Users u ON d.UserId = u.UserId
WHERE d.UserId = @UserId AND d.IsActive = 1";

        var user = connection.QueryFirstOrDefault<UserDetails>(userQuery, new {
request.UserId });

        if (user == null || !VerifyPassword(request.Password, user.PasswordHash,
user.Salt))
        {
            return Unauthorized(new { Message = "Invalid UserId or password." });
        }

        // Fetch all roles assigned to the user
        var roleQuery = "SELECT RoleId FROM Users WHERE UserId = @UserId";
        var roles = connection.Query<int>(roleQuery, new { request.UserId }).ToList();
        if (roles.Count == 0)
        {
            return Unauthorized(new { Message = "User has no assigned roles." });
        }

        // If RoleId is not provided, ask user to select one
        if (request.RoleId == 0)
        {
            return Ok(new { Message = "Select a Role", UserId = request.UserId,
AvailableRoles = roles });
        }
    }
}

```

```

        // Check if RoleId exists
        if (!roles.Contains(request.RoleId))
        {
            return Unauthorized(new { Message = "Invalid role selection." });
        }

        // Generate JWT Token
        var token = GenerateJwtToken(request.UserId, request.RoleId);
        return Ok(new { Token = token, UserId = request.UserId, RoleId =
request.RoleId });
    }
}

catch (SqlException ex)
{
    return StatusCode(500, new { Message = "Database error.", Details = ex.Message
});
}

}

private bool VerifyPassword(string enteredPassword, string storedHash, string
storedSalt)
{
    using (var sha256 = SHA256.Create())
    {
        string computedHash =
Convert.ToBase64String(sha256.ComputeHash(Encoding.UTF8.GetBytes(enteredPassword
+ storedSalt)));

        return computedHash == storedHash;
    }
}

private (string hash, string salt) HashPassword(string password)
{
    byte[] saltBytes = new byte[16];
    using (var rng = RandomNumberGenerator.Create())
    {
        rng.GetBytes(saltBytes);
    }
}

```

```

    }
    string salt = Convert.ToBase64String(saltBytes);
    using (var sha256 = SHA256.Create())
    {
        byte[] hashBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(password +
salt));
        string hash = Convert.ToBase64String(hashBytes);
        return (hash, salt);
    }
}

private string GenerateJwtToken(string userId, int roleId)
{
    var jwtSettings = _configuration.GetSection("JwtSettings");
    var key = Encoding.UTF8.GetBytes(jwtSettings["Key"]);
    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub, userId),
        new Claim(ClaimTypes.NameIdentifier, userId),
        new Claim(ClaimTypes.Role, roleId.ToString()), // Assign RoleId
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
    };

    var token = new JwtSecurityToken(
        issuer: jwtSettings["Issuer"],
        audience: jwtSettings["Audience"],
        claims: claims,
        expires: DateTime.UtcNow.AddHours(1),
        signingCredentials: new SigningCredentials(new SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256)
    );

    return new JwtSecurityTokenHandler().WriteToken(token);
}

}

public class CreateUserRequest
{

```



```

    public string UserId { get; set; } // Mobile Number
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int? Age { get; set; }
    public string SecondPhoneNo { get; set; }
    public string Email { get; set; }
    public string SecondEmail { get; set; }
    public string AadhaarNo { get; set; }
    public string HomeAddress { get; set; }
    public string Password { get; set; }
}
public class LoginRequest
{
    public string UserId { get; set; } // Mobile Number
    public string Password { get; set; } // User Password
    public int RoleId { get; set; } // Role ID (Admin = 1, Parent = 2, Student = 3)
}
}

```

### Location Controller.cs

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using Dapper;
using System.Security.Claims;
using pathfinde.Models;
namespace pathfinde.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    [Authorize] // Require authentication for all endpoints
    public class LocationsController : ControllerBase
    {
        private readonly IConfiguration _configuration;
    }
}

```

```

public LocationsController(IConfiguration configuration)
{
    _configuration = configuration;
}

[HttpGet("{userId}")]
public async Task<IActionResult> GetTimerStatus(string userId)
{
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            await connection.OpenAsync();
            // Fetch active timer details
            var query = @"
SELECT
    lt.UserId,
    lt.IntervalMinutes AS interval,
    lt.Run AS run
FROM LocationTimer lt
JOIN Users u ON lt.UserId = u.Id
WHERE lt.UserId = @UserId AND lt.StopTime IS NULL;";
            var timerStatus = await connection.QueryFirstOrDefaultAsync<dynamic>(query,
new { UserId = userId });
            if (timerStatus == null)
            {
                return NotFound(new { error = "User not found.", status = 404 });
            }
            return Ok(new
            {
                userId = timerStatus.UserId,
                interval = timerStatus.interval * 60, // Convert to seconds
                run = timerStatus.run,
                status = 200
            });
        }
    }
}

```

```

        });
    }
}
catch (SqlException ex)
{
    return StatusCode(500, new { error = "Unable to fetch timer status.", details =
ex.Message, status = 500 });
}
}
[HttpPost("start-timer")]
[Authorize(Roles = "2")] // Only Parent can start
public IActionResult StartTimer([FromBody] StartTimerRequest request)
{
    if (string.IsNullOrEmpty(request.UserId) || request.RoleId != 2)
    {
        return BadRequest(new { Message = "Only Parent (RoleId 2) can start the timer."
});
    }
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            // Check if a timer is already running for this user (with RoleId=2)
            var checkQuery = @"
SELECT Id
FROM LocationTimer
WHERE UserId = @UserId AND RoleId = 2 AND StopTime IS NULL";
            var existingTimerId = connection.ExecuteScalar<int?>(checkQuery, new {
request.UserId });

            if (existingTimerId != null)
            {

```

```

        return BadRequest(new { Message = "A tracking session is already running!"
    });

    }

    // Insert the parent's interval (same userId, but RoleId=2 in DB).
    var insertTimerQuery = @"
INSERT INTO LocationTimer (UserId, RoleId, StartTime, IntervalMinutes, Run,
CreatedAt)
VALUES (@UserId, 2, GETDATE(), @IntervalMinutes, 1, GETDATE());
connection.Execute(insertTimerQuery, new { request.UserId,
request.IntervalMinutes });

return Ok(new { Message = "Tracking started successfully." });
}
}
catch (SqlException ex)
{
    return StatusCode(500, new { Message = "Database error.", Details = ex.Message
});
}
}

[HttpGet("parent-interval/{userId}")]
public IActionResult GetParentInterval(string userId)
{
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();

            // We assume there's exactly one active timer for RoleId=2.
            var query = @"
SELECT TOP 1 IntervalMinutes
FROM LocationTimer
WHERE UserId = @UserId AND RoleId = 2 AND StopTime IS NULL
ORDER BY StartTime DESC";

```

```

        var interval = connection.ExecuteScalar<int?>(query, new { UserId = userId });
        if (interval == null)
        {
            return NotFound(new { Message = "No active parent timer found." });
        }
        return Ok(new { intervalMinutes = interval.Value });
    }
}
catch (SqlException ex)
{
    return StatusCode(500, new { Message = "Database error.", Details = ex.Message
});
}
}
[HttpPost("set-interval")]
[Authorize(Roles = "2")] // Only Parents
public IActionResult SetInterval([FromBody] StartTimerRequest request)
{
    if (string.IsNullOrEmpty(request.UserId) || request.RoleId != 2)
    {
        return BadRequest(new { Message = "Only Parent (RoleId 2) can set the timer
interval." });
    }
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            connection.Execute("sp_SetParentInterval",
                new { UserId = request.UserId, IntervalMinutes = request.IntervalMinutes },
                commandType: System.Data.CommandType.StoredProcedure);
            return Ok(new { Message = "Timer interval updated successfully." });
        }
    }
}

```

```

    }
    catch (SqlException ex)
    {
        return StatusCode(500, new { Message = "Database error.", Details = ex.Message
});
    }
}

[HttpPost("share-location")]
[Authorize(Roles = "3")] // Only Students
public IActionResult ShareLocation([FromBody] LocationUpdateRequest request)
{
    // Validate that the request is from a student.
    if (string.IsNullOrEmpty(request.UserId) || request.RoleId != 3)
    {
        return BadRequest(new { Message = "Only Students (RoleId 3) can share
location." });
    }
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            // Since the parent's timer is created using the same UserId,
            // we use request.UserId to query the parent's active tracking session.
            // Note: The parent's session row has RoleId = 2.
            var checkQuery = @"
SELECT IntervalMinutes
FROM LocationTimer
WHERE UserId = @UserId
AND RoleId = 2
AND StopTime IS NULL
AND Run = 1";

```

```

        var interval = connection.ExecuteScalar<int?>(checkQuery, new { UserId =
request.UserId });
        if (interval == null)
        {
            return BadRequest(new { Message = "No active tracking session found for the
parent!" });
        }
        // Insert the location update
        var insertLocationQuery = @"
INSERT INTO LocationUpdates (UserId, Latitude, Longitude, Timestamp)
VALUES (@UserId, @Latitude, @Longitude, GETDATE());
        connection.Execute(insertLocationQuery, new
        {
            request.UserId,
            request.Latitude,
            request.Longitude
        });
        return Ok(new { Message = "Location shared successfully!", IntervalMinutes =
interval });
    }
}
catch (SqlException ex)
{
    return StatusCode(500, new { Message = "Database error.", Details = ex.Message
});
}
}
[HttpPost("stop-timer")]
[Authorize(Roles = "2")] // Only Parent can stop the timer
public IActionResult StopTimer([FromBody] StopTimerRequest request)
{
    if (string.IsNullOrEmpty(request.UserId) || request.RoleId != 2)
    {

```

```

        return BadRequest(new { Message = "Only Parent (RoleId 2) can stop the timer."
    });
    }
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            var query = @"
SELECT TOP 1 Id, StartTime
FROM LocationTimer
WHERE UserId = @UserId
AND RoleId = 2
AND StopTime IS NULL
ORDER BY StartTime DESC";

            var timer = connection.QueryFirstOrDefault<LocationTimer>(query, new {
request.UserId });
            if (timer == null)
            {
                return BadRequest(new { Message = "No active tracking session found!" });
            }
            // Calculate total duration
            TimeSpan? duration = DateTime.UtcNow - timer.StartTime;
            int totalMinutes = duration.HasValue ? (int)duration.Value.TotalMinutes : 0;
            // Update StopTime, TotalDuration, and Run=0
            var updateQuery = @"
UPDATE LocationTimer
SET StopTime = GETDATE(),
    TotalDuration = @TotalDuration,
    Run = 0
WHERE Id = @Id";
            connection.Execute(updateQuery, new { TotalDuration = totalMinutes, Id =
timer.Id });

```



```

        return Ok(new { Message = "Tracking stopped!", TotalDuration = totalMinutes
    });
    }
}
catch (SqlException ex)
{
    return StatusCode(500, new { Message = "Database error.", Details = ex.Message
});
}
}
/// <summary>
/// ✔ Get User Locations and Run Status
/// </summary>
[HttpGet("locations/{userId}")]
public IActionResult GetUserLocations(string userId)
{
    if (string.IsNullOrEmpty(userId))
    {
        return BadRequest(new { error = "User ID is required.", status = 400 });
    }
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            // ♦ Check if User Exists
            var userExistsQuery = "SELECT COUNT(*) FROM Users WHERE UserId =
@UserId";
            int userExists = connection.ExecuteScalar<int>(userExistsQuery, new { UserId
= userId });
            if (userExists == 0)
            {

```

```

        return NotFound(new { error = "User not found.", status = 404 });
    }

    // 🚩 Fetch Run Status from LocationTimer
    var runStatusQuery = @"
        SELECT TOP 1 Run
        FROM LocationTimer
        WHERE UserId = @UserId
        ORDER BY StartTime DESC";

    bool runStatus = connection.ExecuteScalar<bool>(runStatusQuery, new {
        UserId = userId });

    // 🚩 Fetch Location Data
    var locationQuery = @"
        SELECT Latitude, Longitude, Timestamp
        FROM LocationUpdates
        WHERE UserId = @UserId
        ORDER BY Timestamp DESC"; // Latest locations first

    var locations = connection.Query<LocationResponse>(locationQuery, new {
        UserId = userId }).ToList();

    return Ok(new
    {
        runStatus = runStatus,
        locations = locations,
        status = 200
    });
}

catch (SqlException ex)
{
    return StatusCode(500, new { error = "Failed to retrieve location data.", status =
500, details = ex.Message });
}
}

```

```

/// <summary>
/// ✔ Get Timer and Run Status
/// </summary>
[HttpGet("location-timer/{userId}")]
public IActionResult GetLocationTimer(string userId)
{
    if (string.IsNullOrEmpty(userId))
    {
        return BadRequest(new { error = "User ID is required.", status = 400 });
    }

    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();

            // ♦ Fetch Timer and Run Status
            var query = @"
                SELECT TOP 1 UserId, IntervalMinutes AS Interval, Run
                FROM LocationTimer
                WHERE UserId = @UserId
                ORDER BY StartTime DESC"; // Get the latest timer entry
            var timer = connection.QueryFirstOrDefault<LocationTimerResponse>(query,
new { UserId = userId });

            if (timer == null)
            {
                return NotFound(new { error = "User not found.", status = 404 });
            }

            return Ok(new
            {

```

```

        userId = timer.UserId,
        interval = timer.Interval * 60, // Convert minutes to seconds
        run = timer.Run,
        status = 200
    });
}
}
catch (SqlException ex)
{
    return StatusCode(500, new { error = "Unable to fetch timer status.", status = 500,
details = ex.Message });
}
}
[HttpGet("get-updates/{userId}")]
[Authorize(Roles = "2")] // Only Parent can get location updates
public IActionResult GetLocationUpdates(string userId)
{
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            // Get active tracking session for this user (Parent's timer)
            var timerQuery = @"
SELECT IntervalMinutes, StartTime
FROM LocationTimer
WHERE UserId = @UserId AND RoleId = 2 AND StopTime IS NULL";
            var timer = connection.QuerySingleOrDefault<LocationTimer>(timerQuery,
new { UserId = userId });

            if (timer == null)
            {
                return BadRequest(new { Message = "No active tracking session found!" });
            }
        }
    }
}

```

```

    }
    // Fetch only location updates within the last interval period
    var locationQuery = @"
SELECT UserId, Latitude, Longitude, Timestamp
FROM LocationUpdates
WHERE UserId = @UserId
AND Timestamp >= DATEADD(MINUTE, -@IntervalMinutes, GETDATE())
ORDER BY Timestamp DESC";

    var locations = connection.Query<LocationUpdate>(locationQuery, new {
        UserId = userId, IntervalMinutes = timer.IntervalMinutes }).ToList();

    return Ok(new { Message = "Location updates fetched.", Updates = locations });
}
}
catch (SqlException ex)
{
    return StatusCode(500, new { Message = "Database error.", Details = ex.Message
});
}
}
[HttpGet("get-locations/{userId}")]
public IActionResult GetStudentLocations(string userId)
{
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            var locationQuery = @"
SELECT UserId, Latitude, Longitude, Timestamp
FROM LocationUpdates
WHERE UserId = @UserId

```

```

ORDER BY Timestamp DESC";

        var locations = connection.Query<LocationUpdate>(locationQuery, new {
        UserId = userId }).ToList();

        return Ok(locations);
    }
}
catch (SqlException ex)
{
    return StatusCode(500, new { Message = "Database error.", Details = ex.Message
});
}
}
[HttpPost("student/location/update")]
[Authorize(Roles = "3")] // Only Student can send location updates
public IActionResult UpdateStudentLocation([FromBody] LocationUpdateRequest
request)
{
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();

            // Check if there is an active timer for this student
            var checkQuery = "SELECT IntervalMinutes FROM LocationTimer WHERE
        UserId = @UserId AND RoleId = 2 AND StopTime IS NULL";
            var interval = connection.ExecuteScalar<int?>(checkQuery, new {
request.UserId });

            if (interval == null)
            {
                return BadRequest(new { Message = "No active location tracking found!" });
            }

            // Insert location update

```

```

        var insertQuery = @"
INSERT INTO LocationUpdates (StudentUserId, Latitude, Longitude, Timestamp)
VALUES (@UserId, @Latitude, @Longitude, GETDATE());
        connection.Execute(insertQuery, new
        {
            request.UserId,
            request.Latitude,
            request.Longitude
        });
        return Ok(new { Message = "Location updated successfully!", IntervalMinutes =
interval });
    }
}
catch (SqlException ex)
{
    return StatusCode(500, new { Message = "Database error.", Details = ex.Message
});
}
}
// Get all locations (Authorized for Admin only)
[HttpPost("list")]
[Authorize(Roles = "Admin,User")]
public IActionResult GetAllLocations()
{
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            var locations = connection.Query<LocationResponse>("EXEC
usp_GetAllPF_Locations").ToList();

            return Ok(locations);

```

```

        }
    }
    catch (SqlException ex)
    {
        return StatusCode(500, new { Message = "An error occurred while fetching
locations.", Details = ex.Message });
    }
}

// Get location by ID (Authorized for Admin only)
[HttpPost("{id}")]
[Authorize(Roles = "Admin,User")]
public IActionResult GetLocationById(int id)
{
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();

            var location = connection.QuerySingleOrDefault<LocationResponse>(
                "EXEC usp_GetPF_LocationById @LocationId",
                new { LocationId = id }
            );
            if (location == null)
            {
                return NotFound(new { Message = "Location not found." });
            }
            return Ok(location);
        }
    }
    catch (SqlException ex)
    {

```



```

        return StatusCode(500, new { Message = "An error occurred while fetching the
location.", Details = ex.Message });
    }
}
// Create a new location (Authorized for Admin only)
[HttpPost("create")]
[Authorize(Roles = "Admin,User")]
public IActionResult CreateLocation([FromBody] CreateLocationRequest request)
{
    try
    {
        // Get UserId and Mobile from the JWT claims
        var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
        var mobileClaim = User.FindFirst("PhoneNumber")?.Value;

        if (string.IsNullOrEmpty(userIdClaim) ||
string.IsNullOrEmpty(mobileClaim))
        {
            return Unauthorized(new { Message = "User information is missing or invalid."
});
        }
        if (request.Latitude < -90 || request.Latitude > 90)
        {
            return BadRequest(new { Message = "Latitude must be between -90 and 90." });
        }
        if (request.Longitude < -180 || request.Longitude > 180)
        {
            return BadRequest(new { Message = "Longitude must be between -180 and
180." });
        }
        int userId = int.Parse(userIdClaim);
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {

```

```

        connection.Open();
        var locationId = connection.ExecuteScalar<int>(
            "EXEC usp_InsertPF_Location @UserId, @Latitude, @Longitude,
@Mobile",
            new
            {
                UserId = userId,
                request.Latitude,
                request.Longitude,
                Mobile = mobileClaim
            }
        );
        return Ok(new { Message = "Location created successfully.", LocationId =
locationId });
    }
}
catch (SqlException ex)
{
    return StatusCode(500, new { Message = "An error occurred while creating the
location.", Details = ex.Message });
}
catch (Exception ex)
{
    return StatusCode(500, new { Message = "An unexpected error occurred.", Details
= ex.Message });
}
}
// Update a location (Authorized for Admin only)
[HttpPost("update/{id}")]
[Authorize(Roles = "Admin,User")]
public IActionResult UpdateLocation(int id, [FromBody] UpdateLocationRequest
request)
{
    if (id <= 0)

```

```

        {
            return BadRequest(new { Message = "Invalid location ID." });
        }
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            var rowsAffected = connection.Execute(
                "EXEC usp_UpdatePF_Location @LocationId, @Latitude, @Longitude,
@Mobile",
                new
                {
                    LocationId = id,
                    request.Latitude,
                    request.Longitude,
                    request.Mobile
                }
            );
            if (rowsAffected == 0)
            {
                return NotFound(new { Message = "Location not found." });
            }
            return Ok(new { Message = "Location updated successfully." });
        }
    }
    catch (SqlException ex)
    {
        return StatusCode(500, new { Message = "An error occurred while updating the
location.", Details = ex.Message });
    }
}

// Soft delete a location (Authorized for Admin only)

```

```

[HttpPost("delete/{id}")]
[Authorize(Roles = "Admin,User")]
public IActionResult DeleteLocation(int id)
{
    if (id <= 0)
    {
        return BadRequest(new { Message = "Invalid location ID." });
    }
    try
    {
        using (var connection = new
SqlConnection(_configuration.GetConnectionString("YarcubesDB")))
        {
            connection.Open();
            var rowsAffected = connection.Execute(
                "EXEC usp_DeletePF_Location @LocationId",
                new { LocationId = id }
            );
            if (rowsAffected == 0)
            {
                return NotFound(new { Message = "Location not found." });
            }
            return Ok(new { Message = "Location deleted successfully." });
        }
    }
    catch (SqlException ex)
    {
        return StatusCode(500, new { Message = "An error occurred while deleting the
location.", Details = ex.Message });
    }
}
// Models
public class LocationResponse

```

```
{
    public int LocationId { get; set; }
    public int UserId { get; set; }
    public decimal Latitude { get; set; }
    public decimal Longitude { get; set; }
    public string Mobile { get; set; }
    public bool IsDelete { get; set; }
    public DateTime Timestamp { get; set; }
}

public class CreateLocationRequest
{
    public int UserId { get; set; }
    public decimal Latitude { get; set; }
    public decimal Longitude { get; set; }
    public string Mobile { get; set; }
}

public class UpdateLocationRequest
{
    public decimal Latitude { get; set; }
    public decimal Longitude { get; set; }
    public string Mobile { get; set; }
}

public class LocationTimerResponse
{
    public string UserId { get; set; }
    public int Interval { get; set; }
    public bool Run { get; set; }
}
}
```

## 7 FUTURE ENHANCEMENTS

The Pathfinde app has significant potential for future enhancements that can further improve its functionality, user experience, and overall performance. While the current system provides essential features like real-time student tracking, secure login, role-based access, and alert notifications, several advanced improvements can make the app more robust, scalable, and efficient. These enhancements aim to address potential limitations, introduce new features, and align the app with the latest technological advancements.

One of the most impactful future enhancements would be the integration of Artificial Intelligence (AI) and Machine Learning (ML) to improve real-time tracking and predictive analysis. By using ML algorithms, the app can analyze historical location data to predict patterns and potential risks. For example, if a student deviates from their usual route, the app can automatically send a warning notification to the parent and administrator. AI-based predictive models can also help in identifying unsafe zones and suggesting alternate safer routes for students. Additionally, AI can enhance the app's ability to detect anomalies in user behavior, such as unusual login attempts or unauthorized access, thereby improving overall security.

Another critical enhancement would be the implementation of geofencing technology. Geofencing allows the app to define virtual boundaries around schools or specific areas. If a student crosses these boundaries, an automatic alert can be triggered to notify both the parent and the admin. This feature would significantly enhance the safety and security of students, as it ensures that parents and school authorities are immediately informed of any potential risks. Moreover, geofencing can be customized for different times of the day or specific school events, adding flexibility to the system.

Improving the user interface (UI) and user experience (UX) is another key area of enhancement. Introducing modern design principles, such as material design, and optimizing the app's responsiveness can make the app more intuitive and visually appealing. The app can also support a dark mode option, adjustable font sizes, and customizable color themes to cater to different user preferences. Furthermore, incorporating multilingual support would allow the app to be used by a wider audience, including non-English speaking users.

Another valuable enhancement would be the addition of advanced reporting and analytics capabilities. Currently, the admin can access basic reports on user activity and student location. Expanding this feature to include detailed analytics on student performance, attendance patterns, and user engagement can provide deeper insights for both parents and school administrators. Customizable dashboards and data visualization tools can make it easier for stakeholders to interpret and act on this data.

Integrating biometric authentication, such as fingerprint and facial recognition, can further strengthen the app's security. This would ensure that only authorized users can access sensitive information and reduce the risk of password-related breaches. Additionally, incorporating two-factor authentication (2FA) using OTPs (one-time passwords) or authentication apps would provide an extra layer of security.

Furthermore, expanding the messaging and notification system would improve communication between students, parents, and school administrators. Features like group messaging, read receipts, and automated reminders for school events or attendance issues can make the app more engaging and interactive. Voice-based notifications and integration with smart assistants like Google Assistant and Alexa would add another layer of convenience for users.

Lastly, cloud-based storage and processing could significantly enhance the app's scalability and performance. Transitioning the backend from SQLite to a cloud-based database like Firebase or AWS would allow the app to handle larger datasets and support more concurrent users without performance degradation. Cloud-based storage would also enable automatic backups and faster data retrieval, improving data security and availability.

In conclusion, future enhancements to the Pathfinde app can focus on AI-driven analysis, geofencing, enhanced security, improved UX, advanced reporting, and cloud-based scalability. These upgrades will make the app more secure, responsive, and insightful, providing a more seamless experience for students, parents, and administrators while improving the overall effectiveness of the student tracking system.

## 8 CONCLUSION

The Pathfinde app is a comprehensive and innovative solution designed to address the challenges of real-time student tracking, secure communication, and efficient management within an educational environment. Developed using .NET MAUI and Xamarin, the app integrates multiple user roles, including students, parents, and administrators, ensuring a tailored experience for each user group. The app's key features, such as live location sharing, movement history tracking, and real-time alerts, provide a reliable and secure way to monitor student safety and improve communication between stakeholders. The role-based authentication system ensures that only authorized users can access sensitive data, enhancing the overall security and privacy of the platform.

The app's modular structure, comprising login, student, parent, and admin modules, ensures smooth navigation and easy scalability. The student module empowers students to share their location and movement history, while the parent module enables parents to track their child's real-time location and receive alerts in case of emergencies. The admin module facilitates user and student management, providing comprehensive control over the system's functionality. The integration of APIs, including Twilio for messaging and geolocation services, ensures seamless real-time updates and secure communication between users.

Through the successful implementation of these features, the Pathfinde app has demonstrated its potential to improve student safety, enhance parent-school communication, and streamline administrative operations. The system's user-friendly design and real-time tracking capabilities make it an effective tool for ensuring student security and parental peace of mind. Future enhancements, such as AI-based predictive analysis, geofencing, and cloud-based storage, can further strengthen the app's capabilities and broaden its scope. Overall, the Pathfinde app represents a significant step towards creating a smarter and more secure educational environment, effectively addressing the need for real-time student monitoring and improved school-parent communication.



## 9 REFERENCE

1. Cogswell, J. (2020). *SQLite for Mobile Developers: A guide to building database-driven apps*. Apress.
2. Carter, M., & Shwery, A. (2019). *Mastering Xamarin.Forms: Build rich and elegant apps using C# and .NET*. Packt Publishing.
3. MacDonald, M. (2020). *Pro ASP.NET Core 3: Develop cloud-ready web applications using MVC, Blazor, and Razor Pages*. Apress.
4. MacLean, D. (2017). *Xamarin in action: Creating native cross-platform mobile apps*. Manning Publications.
5. Harvey, J. (2022). *SQLite database programming for beginners*. Independently Published.
6. Freeman, A. (2019). *Pro ASP.NET Core MVC 2: Developing modern web applications with ASP.NET Core MVC*. Apress.
7. Kiss, T. (2021). *Mastering Xamarin: Create dynamic cross-platform apps with Xamarin.Forms and C#*. Packt Publishing.
8. Galloway, J., Wilson, B., Allen, P., & Matson, K. (2017). *Professional ASP.NET MVC 5*. Wrox.
9. Krill, R. (2021). *SQLite essentials: Learn SQLite for Android and iOS development*. O'Reilly Media.
10. Miller, J. (2019). *ASP.NET Core in action*. Manning Publications.
11. [Microsoft Xamarin](#) – Official Xamarin documentation from Microsoft.
12. [GitHub - Xamarin](#) – Xamarin's official repository on GitHub with sample projects.
13. [Stack Overflow - Xamarin](#) – A large community of developers discussing Xamarin-related issues.
14. [Microsoft ASP.NET](#) – Official ASP.NET documentation from Microsoft.
15. [Stack Overflow - ASP.NET](#) – Community-based solutions for ASP.NET issues.
16. [CodeProject](#) – Articles and tutorials on ASP.NET and Xamarin.
17. [C# Corner](#) – Articles and discussions on ASP.NET and related technologies.

