CHAPTER 1 : INTRODUCTION

* 1. Introduction

In today's dynamic educational landscape, fostering meaningful connections between college teachers and students is crucial for personal and academic growth. The “Mentorship” Android application is a cutting-edge solution designed to revolutionize the mentorship experience. This innovative platform leverages the power of mobile technology to create a seamless environment that offers academic guidance, and supports holistic development. Through this innovative Android application, mentors and mentees are seamlessly connected, allowing for effective communication, valuable feedback, and progress reporting. This technology-driven solution revolutionizes the manual mentorship experience, offering a secure and user-friendly environment for students to thrive.

1.2 Purpose

The purpose of Mentorship android app encompasses to create a user-friendly platform that simplifies the process of connecting mentors and students, fostering meaningful interactions, to tailor mentorship connections based on academic interests, career aspirations, and extracurricular pursuits, ensuring compatibility and relevance , Enables mentors to offer detailed reports and constructive feedback to their mentees, promoting continuous growth and development, Empowers students to set academic and personal goals, with mentors monitoring progress and offering targeted support and to develop an environment within colleges that encourages mentorship as a means of personal and academic development, fostering a sense of community and support.

1.3 Scope

Software Product Name: Mentorship Application

The scope of a mentorship app encompasses a multifaceted approach aimed at fostering meaningful connections and facilitating knowledge exchange between mentors and mentees. It involves creating a user-friendly platform that enables the seamless matching of individuals seeking guidance with experienced mentors. This includes robust user profiles, allowing detailed descriptions of skills, interests, and aspirations for effective pairing. The app's functionality extends to communication tools, offering diverse channels for mentor-mentee interactions, be it through messaging, video calls, or discussion forums. It also involves a resource-sharing repository, empowering users to exchange documents, articles, and learning materials. Progress tracking tools and personalized goal-setting features aid in monitoring mentee development, while feedback mechanisms ensure continual improvement. With a focus on community building, networking opportunities, and stringent security measures to safeguard user data, the scope of the mentorship app aims to create a holistic ecosystem that nurtures growth and fosters a supportive learning environment for all participants.

1.4 Problem Definition

The project aims to develop a Android application for college mentorship, featuring three modules (Mentor, Mentee, and Admin). The app facilitates user authentication, mentor-mentee interaction, task assignment, and administrative management, providing a comprehensive solution for efficient college mentorship programs.

1.5 Objectives

The objectives of a mentorship app converge on creating a dynamic platform that facilitates connections, knowledge exchange, and personalized guidance between mentors and mentees. This involves seamlessly matching individuals based on their shared interests, skills, or career aspirations to cultivate impactful and enduring mentorship relationships. The app aims to serve as a conduit for mentors to impart their wisdom, experiences, and insights, nurturing the growth of mentees in both technical skills and broader professional development. Personalization lies at the core, tailoring guidance to meet the unique needs of each mentee. Additionally, the app endeavours to steer mentees toward career advancement, offering advice, networking opportunities, and resources to navigate their chosen paths. Through feedback mechanisms, community engagement features, and a commitment to stringent security and accessibility, the app seeks to create a supportive, inclusive, and secure environment conducive to learning and development. Ultimately, these objectives coalesce to measure and ensure the app's impact, assessing success metrics, user satisfaction, and the tangible outcomes achieved by both mentors and mentees within the mentorship ecosystem.

1.6 Methodology

The proposed methodology for the Mentorship Connect Android app involves meticulous planning and requirement analysis, followed by designing system components and confirming the technology stack. Development encompasses frontend and, optionally, backend work, with a focus on security measures and feedback mechanisms.

1. The app starts with login prompt.
2. When you login as admin, admin can view the registered students ,mentors and can assign students to respective mentors. Admin can also view the assigned details.
3. If you are a new student register as student first by filling up the form.
4. When you login as student, Student can view the assigned teacher and can view the feedback
5. When you login as mentor, Mentor can view the allocated students and can give the feedback to the respective allocated students

The algorithm flowchart for the proposed application is as follows:



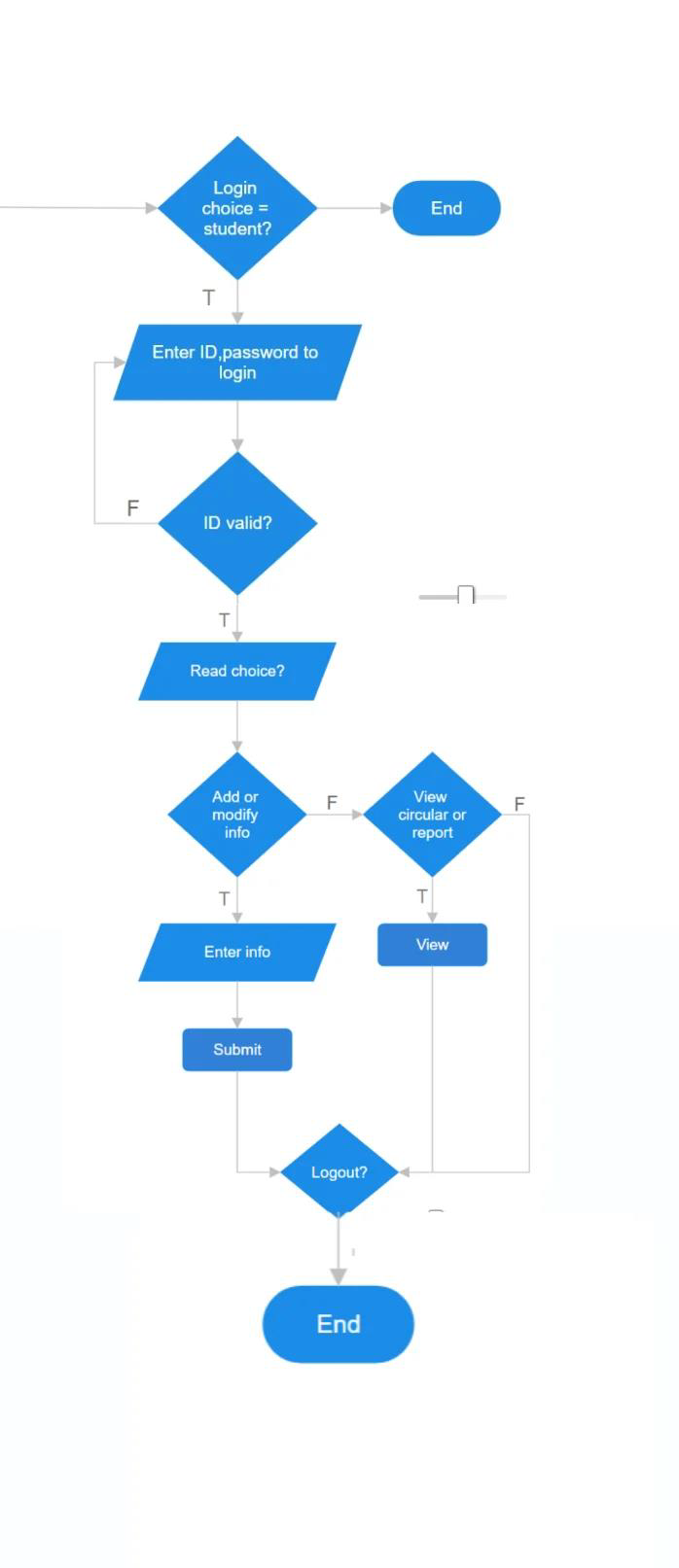


Fig-1.1(Flowchart)

1.7 Limitations

1. Limited Human Connection: Automated matching algorithms may struggle to

replicate nuanced human connections, potentially leading to superficial mentor-mentee relationships.

1. Time Constraints: Balancing schedules between mentors and mentees can be challenging, leading to limited availability for meaningful interactions, impacting the depth of guidance.
2. Quality Control: Ensuring the quality and authenticity of advice or information exchanged within the app may be difficult, potentially affecting the relevance of guidance provided.
3. Privacy Concerns: Maintaining user data privacy and confidentiality presents a significant challenge, requiring robust security measures to prevent breaches and ensure user trust.
4. User Engagement: Sustaining active user engagement over time might be challenging, potentially resulting in inactive profiles or limited participation, affecting the vibrancy of the mentorship community

CHAPTER 2 :

TECHNOLOGIES USED

Creating a Mentorship app involves using a combination of technologies. Here's a list of some used technologies:

1. Programming LanguageJava: Android apps are commonly written in Java or Kotlin, with Kotlin gaining popularity for its conciseness and expressiveness.
2. Development Framework

Android SDK (Software Development Kit): A set of tools and libraries provided by Google for Android app development, including essential components like UI elements and communication features.

1. User Interface (UI) Development

XML (Extensible Markup Language): XML is used for designing the layout of Android app screens, specifying the arrangement and appearance of elements.

Android XML Data Binding: Enhances UI development by allowing data to be bound directly to the layout, simplifying the code structure.

1. Backend Development

Firebase: Provides a comprehensive backend solution with features like real-time database, authentication, and cloud functions, streamlining backend development.

Node.js: Often used for building the server-side logic and APIs, facilitating communication between the app and the backend.

DatabaseSQLite: A lightweight relational database management system included with Android, suitable for storing app data locally.

1. Firebase Authentication: Offers secure user authentication using various methods, including email/password, social logins, and multi-factor authentication.
2. Google Cloud Platform (GCP): Offers a range of cloud services for scalable and reliable app hosting, storage, and computing resources.
3. Version Control- Git: A distributed version control system commonly used for collaborative software development, allowing multiple developers to work on the same project.
4. Integrated Development Environment (IDE)

Android Studio: The official IDE for Android app development, providing tools for coding, testing, and debugging Android applications.

CHAPTER 3 :

SOFTWARE REQUIREMENTS SPECIFICATIONS

3.1 Product Prespective

The project perspective for the Student Mentorship App involves a comprehensive approach to planning, development, and management. It encompasses various aspects to ensure the successful delivery of the software. Here's a more detailed breakdown:

1. **Objectives and Scope:**
   * Clearly define the goals and objectives of developing the app, such as improving mentor-student communication or enhancing academic tracking.
   * Establish the boundaries of the project by determining the features and functionalities to be included initially and potential future enhancements.
2. **Requirements Analysis:**
   * Conduct a detailed analysis of both functional and non-functional requirements.
   * Understand the needs of end-users and stakeholders to align the app with their expectations.
3. **Project Timeline and Resource Planning:**
   * Develop a realistic project timeline with milestones, deadlines, and deliverables.
   * Identify and allocate the necessary resources, including developers, designers, testers, and any external resources.
4. **Technology Stack and Development Methodology:**
   * Decide on the technology stack, including programming languages, frameworks, and databases.
   * Choose a development methodology (e.g., Agile, Scrum) and define development sprints, iterations, and release plans.
5. **Risk Management:**
   * Identify potential risks to the project and develop strategies for risk mitigation.
   * Address technical, operational, and business risks to ensure project success.
6. **Quality Assurance and Testing:**
   * Establish comprehensive testing procedures, including unit testing, integration testing, and user acceptance testing.
   * Ensure the quality and reliability of the software through thorough testing.
7. **User Training and Documentation:**
   * Plan for user training sessions, particularly for mentors and administrators.
   * Develop user documentation for effective usage and maintenance.
8. **Deployment and Maintenance:**
   * Define the deployment process and procedures for releasing updates.
   * Establish a maintenance plan for addressing bugs, implementing feature enhancements, and ensuring the long-term viability of the app.
9. **Collaboration and Communication:**
   * Set up effective communication channels among team members and stakeholders.
   * Use collaboration tools to facilitate communication, project tracking, and document sharing.
10. **Budgeting and Cost Management:**
    * Develop a comprehensive budget for the project, covering development, testing, and associated costs.
    * Monitor and manage expenses throughout the project lifecycle.
11. **Legal and Compliance Considerations:**
    * Ensure that the project adheres to legal and regulatory requirements.
    * Address data privacy laws, intellectual property considerations, and other legal aspects.

3.1.1 System Interfaces

System interfaces in the context of a software project refer to the points where the software interacts with external entities, systems, or components. For the Student Mentorship App, various system interfaces need to be considered to ensure smooth communication and integration. Here's an explanation of potential system interfaces:

3.1.2 User Interfaces:

* + Mentor Interface: The system should have a user interface designed specifically for mentors. This interface allows mentors to view allocated students, input performance details, and communicate with students.
  + Student Interface: Students will interact with a user interface that enables communication with mentors, viewing performance details, and accessing relevant information.
  + Admin Interface: Administrators require an interface to manage user accounts, allocate mentors to students, and perform system configurations.

3.1.3 Hardware Interfaces:

* + The app may utilize the device's hardware interfaces, such as cameras for capturing images or videos, and internet connectivity for communication.
    1. Software Interfaces:
  + Database System: The app may interface with a database system to store and retrieve information, including user data, mentor-student allocations, and performance details.
  + Authentication Service: Interaction with an authentication service is necessary to handle user logins, password recovery, and secure access to the app.
  + Notification Service: To implement push notifications for new messages or important updates, the app might interface with a notification service.
  + External APIs: If the app integrates with external services, such as academic databases or achievement tracking systems, there will be interfaces with these APIs.
    1. Communication Interfaces:
* Mentor-Student Communication: The app should provide a communication interface for real-time or asynchronous communication between mentors and students. This might involve messaging services or chat functionalities.
* Data Exchange Format: Define the format for exchanging data between the app and external systems, ensuring compatibility and seamless integration.
  + 1. Architectural Design

The architectural design of the Student Mentorship App follows a client-server model with three main tiers: presentation (client), business logic (server, including Firebase integration), and data storage (Firebase Realtime Database and Firestore).

Key features and modules are enhanced through Firebase services:

1.User Module: Firebase Authentication is used for secure user sign-up and login, ensuring a seamless and secure user experience.

2.Mentor-Student Allocation: Firebase Realtime Database facilitates the storage and retrieval of mentor-student allocation information, ensuring real-time synchronization.

3.Communication Module: Firebase Cloud Messaging (FCM) is integrated for efficient push notifications, and Firestore is used for real-time chat functionalities, offering a scalable and real-time database solution

4.Performance Tracking: Firestore is employed to store academic performance details, providing real-time tracking capabilities.

5.Admin Module: Firebase Authentication and Firestore manage admin-related configurations securely and centrally.

Firebase services handle communication protocols, security considerations, and scalability. Firebase Analytics provides insights into user behavior, and the integration supports modularity for future enhancements. This streamlined integration simplifies development, improves performance, and establishes a robust foundation for the Student Mentorship App.

### Memory Constraints

Memory constraints for the Student Mentorship App refer to limitations on the amount of memory resources that the application can use. Key considerations include:

1.Client-Side Memory (Mobile Devices):

• Mobile devices have limited RAM and storage.

• Optimize app assets and images for reduced memory footprint.

• Implement efficient data caching to minimize data retrieval from the server.

2.Server-Side Memory (Firebase Server):

• Firebase services may have usage quotas based on the pricing plan.

• Efficiently manage and organize data to stay within allotted storage limits.

• Monitor and optimize database queries to avoid excessive memory usage.

3.Real-Time Features:

• Real-time features (chat, updates) can increase memory consumption.

• Implement message purging or archiving to manage memory usage over time.

• Optimize data structures to minimize memory overhead in real-time databases.

4.Firebase Functions:

• Firebase Cloud Functions have resource limitations.

• Optimize server-side functions for efficient memory utilization.

• Consider breaking down complex tasks into smaller, manageable functions.

5.Client-Server Communication:

• Minimize data payloads in communication between client and server.

• Use compression techniques for data transmission to reduce bandwidth and memory usage.

6.Background Processes:

• Background processes or services may run continuously, affecting memory.

• Implement proper lifecycle management to release resources when not in use.

• Optimize background tasks to minimize memory impact.

7.Caching Strategies:

• Use smart caching mechanisms to store frequently accessed data.

• Implement cache eviction policies to manage memory used for caching.

8.Error Handling:

• Inefficient error handling can lead to memory leaks.

• Implement robust error-handling mechanisms to prevent memory-related issues.

9.Testing and Profiling:

• Regularly conduct memory profiling during development.

• Use testing tools to simulate memory constraints and identify potential issues early.

### Operations

Operations for the Student Mentorship App involve the day-to-day activities and functionalities that users, administrators, mentors, and students perform within the system. Here's a brief overview:

1.User Registration and Authentication:

• Operation: Users register for the app using personal information and undergo authentication.

• Purpose: Establishes secure user accounts and ensures authorized access.

2.Mentor-Student Allocation:

• Operation: Admin allocates mentors to students based on criteria.

• Purpose: Facilitates personalized mentorship relationships.

3.Communication:

• Operation: Mentors and students communicate through real-time messaging.

• Purpose: Enhances mentor-student interaction and support.

4.Performance Tracking:

• Operation: Mentors input and track students' academic performance and provide feedback.

• Purpose: Monitors and improves students' academic progress.

5.Admin Management:

• Operation: Admin manages user accounts, configurations, and mentor-student allocations.

• Purpose: Ensures smooth administration of the mentorship program.

6.Student Information Management:

• Operation: Students provide and update personal and academic information.

• Purpose: Maintains accurate and up-to-date student profiles.

7.Profile Viewing:

• Operation: Mentors view the profiles of allocated students.

• Purpose: Provides mentors with insights into students' backgrounds and achievements.

8.Performance Review:

• Operation: Mentors review and analyze students' academic performance over time.

• Purpose: Guides mentors in offering targeted support to students.

9.Notification Handling:

• Operation: Users receive notifications for new messages, updates, or important events.

• Purpose: Keeps users informed and engaged with the mentorship program.

10.Feedback and Assessment:

• Operation: Mentors provide feedback on students' progress and offer assessments.

• Purpose: Offers constructive feedback and helps in evaluating the effectiveness of mentorship.

12.Data Entry and Retrieval:

• Operation: Users input and retrieve data related to mentorship activities.

• Purpose: Maintains a comprehensive record of mentorship interactions and student progress.

13.User Training:

• Operation: Admin, mentors, and students undergo training on using the app.

• Purpose: Ensures effective utilization of the app's features.

14.Reporting:

• Operation: Generation of reports on mentor-student interactions, academic performance, and program effectiveness.

• Purpose: Provides insights for program evaluation and improvement.

15.Security Measures:

• Operation: Regular security checks, password updates, and adherence to data privacy policies.

• Purpose: Ensures the confidentiality and integrity of user data.

### Site Adaptation Requirements:

The site adaptation requirements for the Student Mentorship App encompass various considerations to ensure its seamless deployment and integration into different environments:

1.Device and OS Compatibility:

• Ensure the app works on diverse devices and major operating systems (Android, iOS).

2.Browser Compatibility:

• Guarantee compatibility with popular web browsers to provide a consistent experience.

3.Responsive Design:

• Develop a responsive user interface for different screen sizes and orientations.

4.Offline Functionality:

• Include offline capabilities for essential functions to accommodate users with limited connectivity.

5.Localization and Internationalization:

• Support multiple languages and cultural adaptations for a diverse user base.

6.Data Security and Compliance:

• Adhere to data security standards and privacy regulations to protect user information.

7.Integration with Existing Systems:

• Ensure seamless integration with educational systems or databases to streamline data exchange.

8.Scalability:

• Design the architecture to scale efficiently with an increasing user base and data volume.

9.User Training and Support:

• Provide training materials and responsive support channels for user assistance.

10.Customization for Educational Institutions:

• Allow customization to meet the specific needs of different educational institutions.

11.Feedback Mechanism:

• Implement a feedback mechanism for users to report issues or suggest improvements.

12.Accessibility Standards:

• Adhere to accessibility standards (e.g., WCAG) to ensure usability for individuals with disabilities.

13.Documentation:

• Provide comprehensive documentation for administrators, mentors, and students.

14.Performance Monitoring:

• Implement tools for monitoring app performance and identifying areas for optimization.

* 1. Product Functions:

Product functions for the Student Mentorship App encompass the core features and capabilities that cater to the needs of administrators, mentors, and students. Here's an overview:

1.User Authentication:

• Function: Allows users to register, log in, and recover passwords securely.

• Purpose: Establishes and verifies user identities for secure access

2.Admin Dashboard:

• Function: Provides administrators with a centralized interface to manage user accounts, allocations, and configurations.

• Purpose: Enables efficient administration of the mentorship program.

3.Mentor-Student Allocation:

• Function: Facilitates the allocation of mentors to students based on predefined criteria.

• Purpose: Establishes personalized mentorship relationships for academic support.

4.Real-Time Messaging:

• Function: Enables mentors and students to communicate in real-time through a messaging system.

• Purpose: Enhances mentor-student interaction and support.

5.Performance Tracking:

• Function: Allows mentors to input and track students' academic performance and provide feedback.

• Purpose: Monitors and improves students' academic progress.

6.Student Profile Management:

• Function: Enables students to provide and update personal and academic information.

• Purpose: Maintains accurate and up-to-date student profiles for effective mentorship.

7.Profile Viewing:

• Function: Allows mentors to view detailed profiles of their allocated students.

• Purpose: Provides mentors with insights into students' backgrounds and achievements.

8.Feedback and Assessment:

• Function: Allows mentors to provide feedback on students' progress and offer assessments.

• Purpose: Offers constructive feedback and helps in evaluating the effectiveness of mentorship.

9.Notification System:

• Function: Sends push notifications for new messages, updates, and important events.

• Purpose: Keeps users informed and engaged with the mentorship program.

10.Data Entry and Retrieval:

• Function: Facilitates input and retrieval of data related to mentorship activities.

• Purpose: Maintains a comprehensive record of mentorship interactions and student progress.

11.User Training Resources:

• Function: Provides training materials and resources for administrators, mentors, and students.

• Purpose: Ensures effective utilization of the app's features.

12.Reporting and Analytics:

• Function: Generates reports on mentor-student interactions, academic performance, and program effectiveness.

• Purpose: Provides insights for program evaluation and improvement.

13.Security Measures:

• Function: Implements secure authentication, data encryption, and regular security checks.

• Purpose: Ensures the confidentiality and integrity of user data.

14.Accessibility Features:

• Function: Incorporates features adhering to accessibility standards (e.g., WCAG) for users with disabilities.

• Purpose: Promotes inclusivity and usability for a diverse user base.

15.Offline Functionality:

• Function: Provides essential functionalities even when users have limited or no internet connectivity.

• Purpose: Ensures continued usability in low-connectivity scenarios.

* 1. User characteristics

User characteristics for the Student Mentorship App involve identifying the distinct roles and attributes of individuals who will interact with the system. Here are the main user roles and their characteristics:

1.Administrator:

Characteristics:

• Administrative staff responsible for overseeing the mentorship program.

• Has the authority to manage user accounts, allocations, and system configurations.

• Requires access to comprehensive reports and analytics for program evaluation.

2.Mentor:

Characteristics:

• Experienced educators or professionals volunteering to guide and support students.

• Manages a set of allocated students and provides academic guidance.

• Engages in real-time communication with assigned students through messaging features.

3.Student:

Characteristics:

• Individuals seeking academic guidance and mentorship.

• Provides personal and academic information for mentor matching.

• Utilizes the app to communicate with their assigned mentor and track academic progress.

4.System Support Staff:

Characteristics:

• Technical support personnel responsible for assisting users with app-related issues.

• Requires in-depth knowledge of the app's functionalities to provide effective support.

• Manages user training resources and documentation.

5.External System Integration Personnel:

Characteristics:

• IT professionals responsible for integrating the app with existing educational systems.

• Requires technical expertise in data exchange and system compatibility.

6.Trainers:

Characteristics:

• Individuals responsible for conducting user training sessions.

• Requires proficiency in explaining app functionalities to administrators, mentors, and students.

• Ensures users understand how to effectively use the app for mentorship activities.

7.Compliance Officers:

Characteristics:

• Personnel responsible for ensuring the app complies with data security and privacy regulations.

• Requires expertise in privacy laws and standards.

• Collaborates with administrators to maintain compliance.

8.Communication Facilitators:

Characteristics:

• Individuals responsible for ensuring effective communication within the app.

• Monitors and manages the real-time messaging system.

• Addresses any communication-related issues or conflicts.

9.Performance Analysts:

Characteristics:

• Personnel responsible for analyzing performance reports and data.

• Requires analytical skills to derive insights from mentor-student interactions and academic performance data.

• Collaborates with administrators for program improvement.

10.Accessibility Coordinators:

Characteristics:

• Individuals focused on ensuring the app adheres to accessibility standards.

• Works to make the app usable for individuals with disabilities.

• Collaborates with developers to implement and test accessibility features.

* 1. Constraints:

Constraints of a mentorship app are platform compatibility ensuring the app works well across different devices and operating systems might be challenging.Internet Connectivity dependency on internet access for both mentors and mentees can limit usage in areas with poor connectivity.Privacy and Security data protection compliance with data protection laws and ensuring the confidentiality of mentorship discussions and personal information.User authentication implementing secure login procedures and preventing unauthorized access.Resource Constraints like budget limitations development costs, maintenance, and updates might be restricted by available funding.Time Constraints limited timeframes for development, testing, and deployment due to institutional or project deadlines.Addressing these constraints during the development and ongoing management of the mentorship app is crucial to ensure its effectiveness, usability, and adherence to regulatory and user needs.

## Assumptions and Dependencies

1. User Availability:
   * Assumption: Users have regular access to internet-connected devices.
   * Rationale: Essential for real-time communication and data exchange.
2. User Training:
   * Assumption: Adequate training materials will be provided for effective app usage.
   * Rationale: Proficiency is crucial for successful mentorship interactions.
3. Compliance with Regulations:
   * Assumption: App development adheres to privacy and data security regulations.
   * Rationale: Ensures legal compliance and protects user information.
4. Mentorship Program Structure:
   * Assumption: Program has predefined criteria for mentor-student allocations.
   * Rationale: Allocation functionalities are based on program requirements.
5. Collaboration with Educational Institutions:
   * Assumption: App integrates seamlessly with existing educational systems.
   * Rationale: Facilitates data exchange and alignment with institutional processes.
6. Continuous Improvement:
   * Assumption: Ongoing feedback from users for app improvement.
   * Rationale: Essential for addressing user needs and enhancing effectiveness.

Dependencies for the Student Mentorship App:

1. Firebase Services:
   * Dependency: Successful integration with Firebase services for core functionalities.
   * Rationale: Fundamental for authentication, real-time data, and messaging.
2. Internet Connectivity:
   * Dependency: Reliable internet connection for app functionalities.
   * Rationale: Real-time communication and data operations rely on internet access.
3. Cross-Platform Compatibility:
   * Dependency: Development frameworks ensuring cross-platform compatibility.
   * Rationale: Ensures the app works seamlessly across devices and operating systems.
4. Regulatory Compliance:
   * Dependency: Collaboration with legal experts to ensure compliance with data protection laws.
   * Rationale: Mitigates legal risks and ensures ethical handling of user data.
5. User Training Sessions:
   * Dependency: Availability of trainers for user training sessions.
   * Rationale: Training is crucial for effective app utilization.
6. Educational Institution Cooperation:
   * Dependency: Cooperation with educational institutions for integration and alignment.
   * Rationale: Facilitates smooth integration and support for the mentorship program.
7. Feedback Mechanism:
   * Dependency: Implementation of a user feedback mechanism.
   * Rationale: Continuous improvement relies on user input and suggestions.
8. Accessibility Standards:
   * Dependency: Adherence to accessibility standards during app development.
   * Rationale: Ensures usability for individuals with disabilities.

3.6 Apportioning of Requirements.

The apportioning of requirements for the Student Mentorship App involves organizing functionalities into specific modules with assigned responsibilities. Here's a summary:

1. User Authentication and Onboarding:
   * Focus: Registration, login, and profile setup.
   * Responsibility: Development team implements secure authentication using Firebase.
2. Admin Dashboard and Management:
   * Focus: Admin dashboard, user management, and mentor-student allocation.
   * Responsibility: Development team designs and implements these features.
3. Communication Module:
   * Focus: Real-time messaging, notifications, and communication facilitation.
   * Responsibility: Development team implements messaging and notification features.
4. Performance Tracking:
   * Focus: Inputting and tracking academic performance, feedback, and assessments.
   * Responsibility: Development team implements these mentor-student interaction features.
5. User Profile Management:
   * Focus: Student profile management and mentor's profile viewing.
   * Responsibility: Development team designs features for user information input and profile viewing.
6. Reporting and Analytics:
   * Focus: Generating reports and analyzing performance data.
   * Responsibility: Development team implements reporting features; data analysts analyze reports.
7. Security Measures:
   * Focus: Data security and compliance.
   * Responsibility: Development team, in collaboration with compliance officers, ensures security and compliance.
8. Accessibility and Usability:
   * Focus: Implementing features adhering to accessibility standards.
   * Responsibility: Development team, collaborating with accessibility coordinators, ensures usability for all users.
9. User Training and Support:
   * Focus: Developing training materials and providing support channels.
   * Responsibility: Trainers, development team, and support staff collaborate on training resources and support.

CHAPTER 4: Specific Requirements

## External Interfaces

The external interface of a mentorship app involves seamless interaction between the application and external systems or users. Primarily, integration with user authentication services like Google, Facebook, or institutional logins streamlines the onboarding process for mentors and mentees. This external interface should enable calendar integration with platforms like Google Calendar or Outlook to synchronize meeting schedules and send reminders. Additionally, embedding communication APIs such as Twilio or Zoom facilitates seamless messaging, video calls, and virtual meetings within the app. Integration with cloud storage services like Dropbox or Google Drive allows mentors to share resources directly from these platforms. An open API architecture enables potential future integrations with educational systems, learning management systems, or CRM tools, expanding the app's functionality and utility. Moreover, compliance with industry-standard data protection regulations and interoperability with diverse devices, browsers, and operating systems ensure a smooth and accessible user experience across various external environments.

## Functions

A mentorship app encompasses a spectrum of functions designed to facilitate a dynamic and effective mentor-mentee relationship. At its core, the app provides a robust user authentication system, enabling mentors and mentees to create detailed profiles showcasing their expertise, interests, and goals. A sophisticated matching algorithm pairs individuals based on complementary skills and objectives, fostering meaningful connections. Seamless communication tools, such as real-time messaging, video calls, and discussion forums, empower mentors and mentees to engage and collaborate conveniently. Scheduling functionalities integrated with calendars enable effortless meeting management, ensuring timely sessions. A comprehensive resource-sharing repository allows mentors to impart knowledge by sharing documents, articles, or other educational materials. Progress tracking mechanisms facilitate continuous assessment by monitoring milestones and summarizing session progress. Feedback mechanisms, including surveys or rating systems, support ongoing evaluation for both mentors and mentees. An intuitive administrative dashboard empowers program managers to oversee user management, analyze program metrics, and evaluate overall effectiveness. Ensuring stringent privacy measures and compliance with data protection laws is paramount. Customization options for scalability and adaptability to various institutional needs round out the app's functionalities, promising a holistic and supportive mentorship experience.

## Performance Requirements

The performance requirements for a mentorship app are critical to ensuring its responsiveness, reliability, and scalability. Primarily, the app should exhibit rapid response times, with interactions such as messaging, scheduling, and content sharing happening seamlessly and near-instantaneously to enhance user experience. The system's reliability is paramount; it should maintain high availability, minimizing downtime and ensuring uninterrupted access for mentors and mentees. Scalability is also crucial, with the app capable of accommodating a growing user base without compromising performance. This includes handling increased concurrent users, data processing, and storage demands efficiently. Network latency should be minimal to support real-time communication features like video calls and messaging. Moreover, the app should be optimized for various devices and platforms, ensuring consistent performance across different screen sizes and operating systems. Security measures must guarantee data encryption, secure authentication, and protection against potential vulnerabilities or cyber threats. Finally, regular performance monitoring and optimization processes are essential to maintain the app's efficiency and responsiveness as user demand and usage patterns evolve over time

## Logical Database Requirements

The logical database requirements for a mentorship app are pivotal in structuring and managing the data integral to its functionality. Firstly, a robust relational database system is essential, offering structured storage for user profiles, mentorship session details, communication logs, and resource repositories. The database schema should efficiently handle relationships between mentors and mentees, capturing their profiles, preferences, and interaction history. Normalization techniques are vital to minimize data redundancy and ensure data integrity across the system. To support real-time interactions, the database should facilitate rapid data retrieval and updates, optimizing queries for fast access to mentorship details, messages, and scheduling information. Additionally, the database design must accommodate scalability to manage an increasing volume of user-generated content and interactions. Incorporating indexing strategies and query optimization techniques enhances the database's performance, especially in handling large datasets. Ensuring proper data security measures, such as encryption and access control, is imperative to protect sensitive user information. Regular database maintenance, including backups, data cleansing, and optimization, is fundamental for sustained performance and reliability of the mentorship app's data storage infrastructure.

1. Design Constraints:

4.5.1 Standards Compliance :

Several design constraints may impact the development and implementation of a mentorship app. Firstly, the app's user interface design must cater to diverse user demographics, considering various levels of technological proficiency and accessibility needs. This constraint necessitates a balance between simplicity and functionality to ensure ease of use for all users. Secondly, interoperability poses a challenge, requiring the app to seamlessly integrate with multiple devices, operating systems, and browsers. Achieving compatibility across different platforms while maintaining consistent performance can be a constraint during development. Thirdly, data privacy and security constraints demand strict adherence to regulatory standards and best practices. Ensuring robust encryption, secure authentication, and data protection mechanisms adds complexity to the app's design. Additionally, limited bandwidth or unreliable internet connectivity in certain regions may constrain the app's performance and usability, requiring strategies for offline access or optimized data transmission. Lastly, budgetary constraints may limit the incorporation of advanced features or extensive scalability, necessitating prioritization and optimization of functionalities within resource limitations. Balancing these design constraints is crucial to develop a mentorship app that is user-friendly, secure, and adaptable to diverse technological environments while staying within budgetary considerations.

## Software System Attributes

### Reliability

1. Mean Time between Failures (MTBF): Define MTBF requirements, expressing the expected time between software failures. Set realistic and achievable goals for system stability.

2. Fault Tolerance: Implement mechanisms to detect and recover from faults to ensure continuous operation. Design the system to handle unexpected errors gracefully without causing a complete failure.

3. Error Handling: Develop robust error-handling mechanisms to capture, log, and report errors effectively. Provide informative error messages for users to understand and address issues.

4. Testing and Validation: Conduct thorough testing, including unit tests, integration tests, and system tests. Perform validation with real-world data to ensure the software meets reliability expectations.

5. Performance Monitoring: Implement monitoring tools to track system performance in real-time. Set up alerts for abnormal behaviour, enabling proactive response to potential reliability issues.

6. Documentation: Maintain comprehensive documentation outlining system architecture, dependencies, and error recovery procedures. Ensure documentation is accessible and up-to-date for both development and support teams.

7. User Feedback: Incorporate user feedback during development to address usability issues and potential failure points. Encourage users to report issues promptly for quick resolution and continuous improvement

1. Availability:

1. Availability Requirements: Clearly define the required availability level for the student mentorship app(e.g., 24/7 availability).Specify any permissible downtime or maintenance windows.

2. Check pointing Mechanism: Implement a robust check pointing system to regularly save the system's state. Determine appropriate intervals for creating checkpoints based on usage patterns.

3. Recovery Procedures: Develop comprehensive recovery procedures to restore the system to a consistent state after a failure. Ensure recovery processes are well-documented and easily executable.

4. Restart Capability: Enable users to restart the application after a failure. Specify the maximum acceptable data loss during restart.

5. Fault Detection and Notification: Implement mechanisms to detect faults promptly. Provide notifications to system administrators or relevant personnel when a fault is detected.

6. Redundancy and Failover: Introduce redundancy in critical components to minimize the impact of failures. Implement failover mechanisms to seamlessly switch to backup systems when needed.

7. Load Balancing: Distribute the workload evenly across servers to prevent overload on any single component. Use load balancing to enhance system performance and availability.

1. Security:

Design the software with modular architecture, assigning specific functions to different modules. Limit access permissions based on user roles to ensure that each module only performs authorized actions.

Implement robust error handling mechanisms to handle unexpected scenarios gracefully. Validate user inputs to prevent common security vulnerabilities like SQL injection or buffer overflow attacks.

Establish regular backup procedures to safeguard critical data. Develop a comprehensive recovery plan to minimize downtime in case of security incidents.

### Maintainability

1. Modularity: Design the software with clear modular components, making it easier to understand and update individual parts without affecting the entire system.
2. Documentation: Provide comprehensive documentation for code, architecture, and any specific configurations. This aids new maintainers in understanding the system quickly.
3. Clear Code Structure: Follow a consistent and clear coding style. Use meaningful variable and function names, enhancing readability for those maintaining the code.
4. Error Handling and Logging: Implement robust error handling mechanisms and log relevant information for troubleshooting and debugging purposes.
5. Test Coverage: Develop a comprehensive suite of automated tests to ensure that changes do not introduce new issues. Include documentation on how to run tests and interpret results.
6. Performance Monitoring: Integrate performance monitoring tools to identify potential bottlenecks or areas for optimization. Document best practices for maintaining optimal system performance

### Portability

Designing the Student Mentorship App using Flutter, a cross-platform framework, ensures native-like performance on both iOS and Android devices. Flutter's widget-based architecture enables a responsive design, adapting seamlessly to various screen sizes and orientations. Standardized APIs and cloud-based storage integration are facilitated by Flutter's framework, promoting code reusability and consistent functionality across platforms

## 4.7 Organizing the Specific Requirements

1. System Mode:
2. Development Mode:
   * Description: During initial development.
   * Characteristics: Limited access, debugging tools enabled, continuous integration active.
3. Testing Mode:
   * Description: Deployment for comprehensive testing.
   * Characteristics: Test environment resembling production, testing tools active, various testing types conducted.
4. Staging Mode:
   * Description: Pre-production environment for final validation.
   * Characteristics: Mirrors production closely, final testing, limited user acceptance testing.
5. Production Mode:
   * Description: Live and operational state for end-users.
   * Characteristics: Full user access, continuous monitoring, ongoing support and maintenance.
6. Maintenance Mode:
   * Description: Activated for updates and routine maintenance.
   * Characteristics: Scheduled maintenance windows, minimal user disruption, software updates applied.
7. Offline Mode:
   * Description: Allows limited functionality without internet.
   * Characteristics: Offline access to cached data, goal setting, viewing downloaded resources.
8. Emergency Mode:
   * Description: Activated in critical situations.
   * Characteristics: Rapid response, temporary suspension of non-essential features, communication of emergency measures.
9. Upgrade Mode:
   * Description: Entered during major updates or version upgrades.
   * Characteristics: Temporary suspension of certain features, data migration, communication of new features.
10. Backup and Restore Mode:
    * Description: Activated for scheduled backups or data loss.
    * Characteristics: Regular backups, secure storage, rapid data restoration.
11. User Onboarding Mode:

* Description: Activated for new user onboarding.
* Characteristics: Guided onboarding process, in-app tutorials, emphasis on user education.

### User Class

1. Student Class:

Description:

* + Individuals enrolled in educational programs seeking mentorship.

Roles and Responsibilities:

* + Manage profiles, academic interests, skills, and career goals.
  + Set academic and career goals.
  + Engage in real-time communication with mentors.
  + Access and contribute to the resource repository.
  + Track and visualize progress toward goals.
  + Receive and provide feedback on the mentorship experience.

Access Levels:

* + Full access to goal-setting, communication, and resource-sharing features.
  + Limited access to administrative or mentor-specific functionalities.

2. Mentor Class:

Description:

* + Experienced individuals volunteering to guide and support students.

Roles and Responsibilities:

* + Manage mentor profiles, expertise, skills, and industry experience.
  + Accept or decline mentorship requests.
  + Engage in real-time communication with students.
  + Share educational resources and guidance.
  + Review and provide feedback on student-set goals.
  + Track and visualize student progress.
  + Provide mentorship to enhance educational and career paths.

Access Levels:

* + Full access to mentorship, communication, and resource-sharing features.
  + Access to mentor-specific functionalities, such as reviewing student goals.

3. Admin Class:

Description:

* + Administrators oversee and manage the overall operation of the Student Mentorship App.

Roles and Responsibilities:

* + User management: Create, modify, or deactivate user accounts.
  + System configuration: Manage application settings and configurations.
  + Oversight of mentor-student matching algorithms.
  + Access to analytics and reporting tools for monitoring app usage.
  + Handling escalated support issues and addressing user concerns.
  + Conducting periodic reviews of app data for quality and compliance.

### Objects

1. Student Object:

Attributes:

* + User profile data (name, email, etc.).
  + Academic information (program, year, etc.).
  + Set goals and progress data.
  + Communication history with mentors.
  + Preferences for mentor matching.

Functionality:

* + Goal setting and tracking.
  + Real-time messaging.
  + Access to shared resources.
  + Profile management.

2. Mentor Object:

Attributes:

* + User profile data (name, email, etc.).
  + Professional information (industry, expertise, etc.).
  + Communication history with students.
  + Shared resources and guidance provided.
  + Availability status.

Functionality:

* + Accept/decline mentorship requests.
  + Real-time communication with assigned students.
  + Review and feedback on student-set goals.
  + Profile management.

3. Admin Object:

Attributes:

* + Administrative credentials.
  + Access to user management tools.
  + Configuration settings.
  + Analytics and reporting data.
  + Support and issue resolution history.

Functionality:

* + User management (create, modify, deactivate accounts).
  + System configuration and settings.
  + Oversight of mentor-student matching.
  + Analytics and reporting.
  + Handling escalated support issues.

1. Feature

The Student Allotment and Feedback feature streamlines the mentorship pairing process, ensuring optimal matches based on preferences and expertise. Additionally, it facilitates continuous improvement through feedback loops between students and mentors.

1. Stimulus

Revolutionize mentorship experiences with the Student Mentorship App's automated pairing based on user preferences and real-time feedback loops, ensuring personalized and impactful connections.

#### Response

The proposed features, including automated mentor-student pairing and robust feedback mechanisms, align well with our objectives for an effective and personalized mentorship experience. The emphasis on usability, reliability, and security demonstrates a thorough understanding of key system attributes

1. Functional Hierarchy

Input-Driven Hierarchy

1. User Management:
   * Input: User registration details.
   * Process: Account creation.
   * Output: User authentication credentials.
2. Matching Algorithm:
   * Input: User preferences.
   * Process: Algorithmic matching.
   * Output: Automated mentor-student pairings.
3. Communication Tools:
   * Input: User messages, attachments.
   * Process: Real-time messaging.
   * Output: Sent and received messages.
4. Goal Setting and Progress Tracking:
   * Input: User-defined goals.
   * Process: Goal management.
   * Output: Visual progress representation, reports.
5. Resource Sharing:
   * Input: Uploaded resources.
   * Process: Resource repository management.
   * Output: Categorized and searchable resources.
6. System Attributes:
   * Input: User interactions with the interface.
   * Process: Usability, reliability, security checks.
   * Output: Responsive and secure user experience.

###### CHAPTER 5: Change Management Process

Applying the change management process to the Student Mentorship App's Software Requirements Specification (SRS) involves a structured approach to introducing, implementing, and sustaining the proposed changes. Here's an overview of the change management process for the given SRS:

1. **Identification of Change:**
   * The need for change may arise from user feedback, evolving technology, or emerging educational trends. Identification involves recognizing areas for improvement or features that need enhancement in the existing SRS.
2. **Assessment and Planning:**
   * The identified changes are assessed for their impact on the app. This includes analyzing the feasibility, risks, and resource requirements. A detailed plan is developed, specifying how the SRS will be modified or extended to accommodate the proposed changes.
3. **Stakeholder Communication and Engagement:**
   * Clear communication is essential to inform stakeholders about the planned changes to the SRS. This includes developers, project managers, administrators, and end-users. Engagement strategies, such as workshops or training sessions, help stakeholders understand the implications of the proposed changes.
4. **Change Approval:**
   * The proposed modifications to the SRS are presented to relevant stakeholders or a change control board for approval. This step ensures that decision-makers are aligned with the changes and authorize their implementation.
5. **Implementation:**
   * The approved changes are incorporated into the SRS. This may involve updating requirements, modifying system functionalities, or introducing new features. During this phase, developers and project teams follow the updated SRS guidelines.
6. **Monitoring and Evaluation:**
   * The impact of the changes on the application's functionality and user experience is monitored. Feedback from users and stakeholders is collected to evaluate the effectiveness of the modified SRS in meeting the app's goals.
7. **Documentation:**
   * Detailed documentation is maintained throughout the change management process. This includes the modified SRS, implementation details, and any adjustments made during the implementation phase.
8. **Feedback and Adjustment:**
   * Based on the monitoring and evaluation, feedback is collected to identify any issues or areas for improvement. Adjustments to the modified SRS may be made to address user concerns, enhance features, or resolve unforeseen challenges.
9. **Closure:**
   * Once the modified SRS has been successfully implemented and stabilized, the change management process for the SRS is officially closed. A final assessment ensures that the objectives of the SRS changes have been met.
10. **Continuous Improvement:**
    * The organization reflects on the change management process itself, identifying lessons learned and areas for improvement. This feedback loop contributes to an ongoing cycle of improving the SRS and the overall development and management of the Student Mentorship App.

CHAPTER 6: DESIGN PHASE

1. Architectural Design:

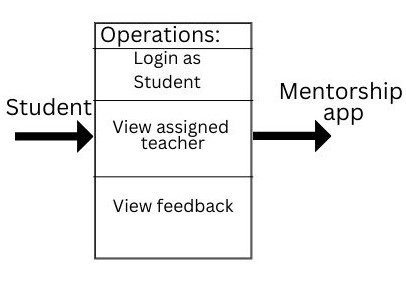
****

Fig-6.1.1

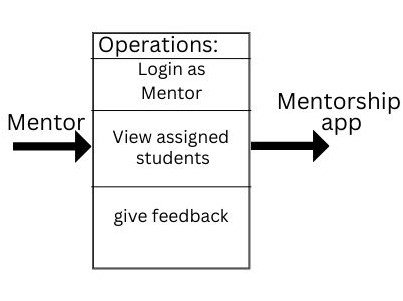
****

Fig-6.1.2

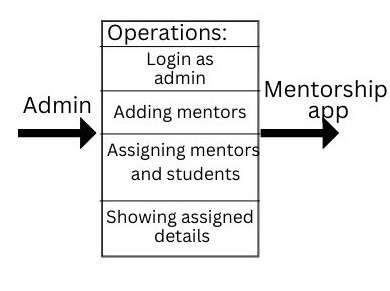
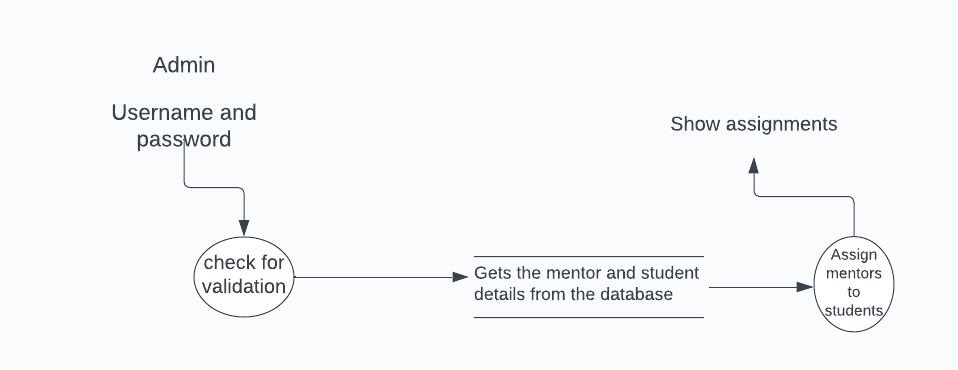
****

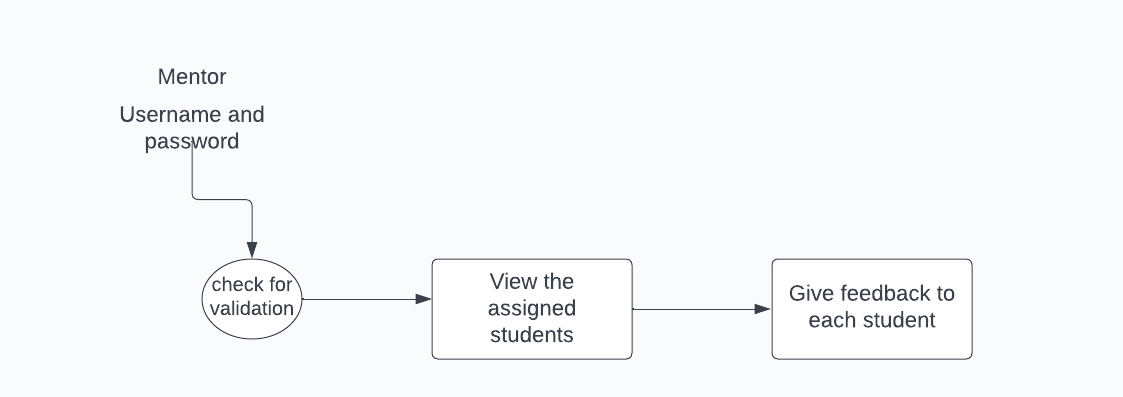
Fig-6.1.3

1. Data Flow Design:



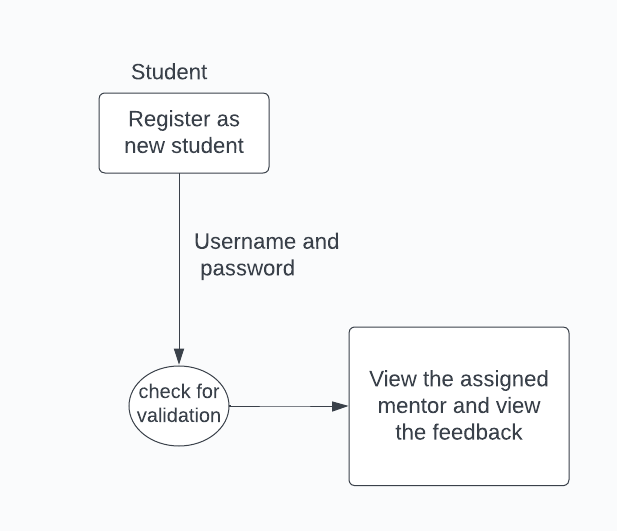
Admin

Fig-6.2.1



Mentor

Fig-6.2.2



Student

Fig-6.2.3

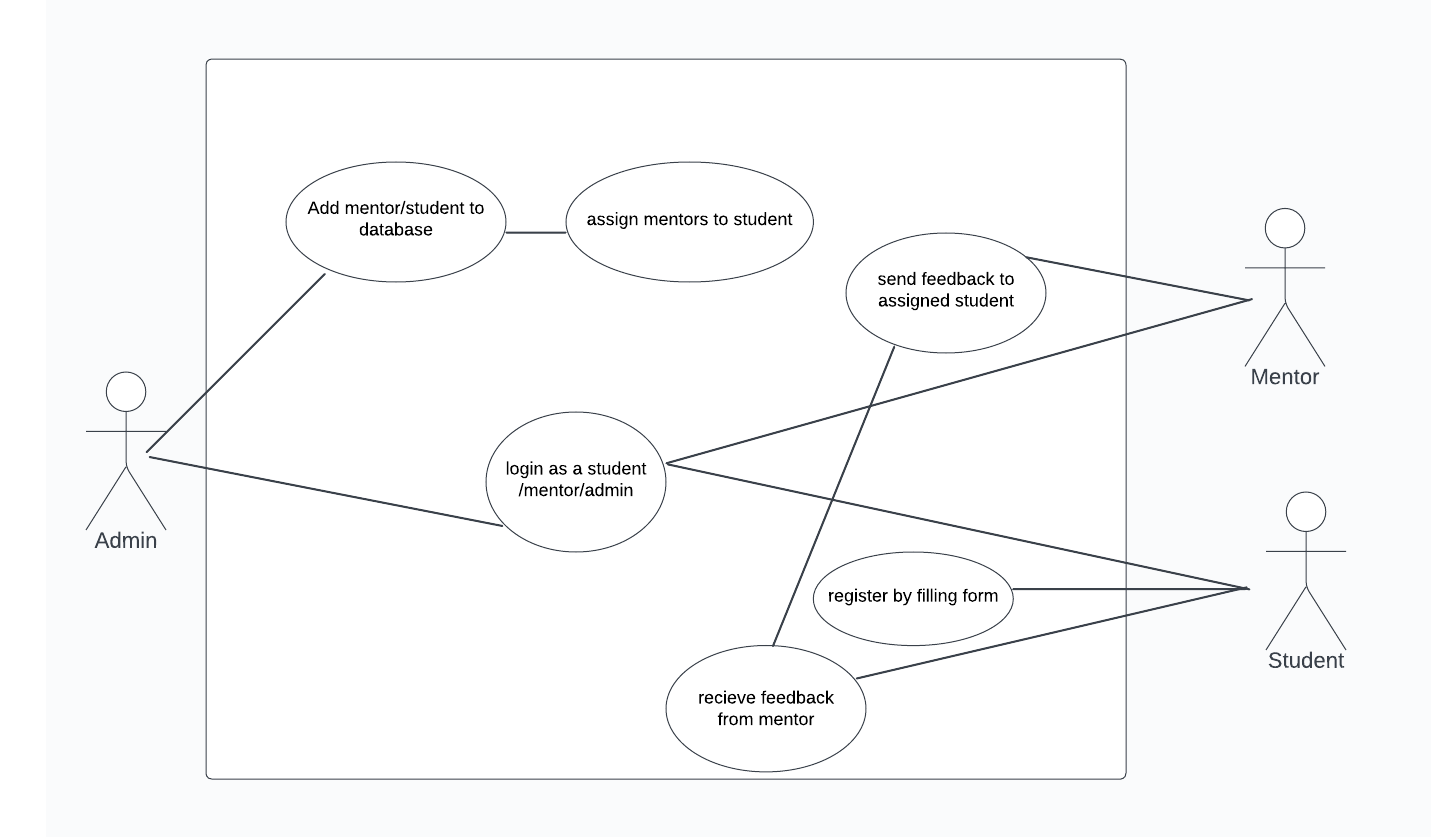
1. Class Design:

**A diagram of a program

Description automatically generated**

Fig-6.3

1. Use Case Design:

****Fig-6.4

1. Sequence Design:

A screenshot of a computer screen

Description automatically generated

Fig-6.6

CHAPTER 7 IMPLEMENTATION PHASE

1. Experimental Set up:
2. Processor:

Opt for a laptop with a powerful processor, preferably an Intel Core i5 or i7, or an AMD Ryzen equivalent. This ensures smooth compilation, running emulators, and handling resource-intensive tasks.

1. Install Android Studio:

Download and install Android Studio from the official website. Follow the installation wizard to set up the IDE on your development machine.

1. Create a New Project:

Open Android Studio and start a new project. Choose a project template, specify details such as the project name, package name, and select the minimum SDK version.

1. Set Up Emulators or Connect Physical Devices:

Android Studio provides emulators to simulate various Android devices. Set up emulators to test your app on different device configurations. Additionally, you can connect physical Android devices for testing by enabling USB debugging in Developer Options.

1. Pseudo Code

# Pseudocode for Student User Actions

# User Class Definition

class User:

userId

username

password

userType

# Student User Class Definition

class StudentUser inherits User:

major

allocatedMentor

# Mentor User Class Definition

class MentorUser inherits User:

expertise

allocatedStudents

# MentorshipApp Class Definition

class MentorshipApp:

usersList

mentorsList

studentsList

# Method to handle user login

method login(username, password):

# Find the user in the usersList

user = findUserByUsername(username)

# Check if the user exists and the password is correct

if user is not null and user.password == password:

# Perform actions based on user type

if user.userType == "Student":

showStudentMenu(user)

elif user.userType == "Mentor":

showMentorMenu(user)

elif user.userType == "Admin":

showAdminMenu(user)

else:

print("Invalid username or password")

# Method to find a user by username

method findUserByUsername(username):

for user in usersList:

if user.username == username:

return user

return null

# Method to display the student menu

method showStudentMenu(student):

print("Welcome, " + student.username + "!")

print("1. View Mentor Profiles")

print("2. Track Performance")

choice = getInput()

# Handle user choice

if choice == 1:

viewMentorProfiles(student)

elif choice == 2:

trackPerformance(student)

# Method to display mentor profiles

method viewMentorProfiles(student):

for mentor in mentorsList:

print("Mentor: " + mentor.username)

print("Expertise: " + mentor.expertise)

print("---------------")

# Method to track performance

method trackPerformance(student):

# Implement performance tracking logic

print("Tracking Performance...")

# Method to get user input

method getInput():

# Implement input logic

return input()

# Main Program

# Initialize the MentorshipApp

app = new MentorshipApp()

# Perform login

app.login("student\_username", "student\_password")

CHAPTER 8

TESTING PHASE

1. Types of test carried out

Testing is a crucial aspect of software development to ensure the reliability, functionality, and performance of your application. Here are the types of tests you should consider for your Flutter app and include in the testing report:

1.Unit Testing:

Purpose: Verify the smallest parts of your code (functions, methods, classes) work as expected in isolation.

Tools: Use the built-in test library for Dart and the flutter\_test package for Flutter.

What to Test:

Individual functions and methods.

Business logic in BLoC classes.

Data models.

2.Widget Testing:

Purpose: Ensure that individual widgets and UI components render correctly and respond appropriately to user interactions.

Tools: Use the flutter\_test package for widget testing.

What to Test:

Widget rendering and layout.

Widget interactions and state changes.

3.Integration Testing:

Purpose: Verify that different parts of your application work together as expected.

Tools: Use the integration\_test package for Flutter.

What to Test:

Integration between UI components.

Interaction between BLoC classes.

Communication with the backend (Firebase).

4.User Acceptance Testing (UAT):

Purpose: Obtain feedback from actual users to ensure the app meets their expectations and is user-friendly.

Tools: Manual testing, beta testing, or using tools like Firebase Test Lab for automated testing on real devices.

What to Test:

User workflows.

Usability and user interface design.

App performance and responsiveness.

5.Performance Testing:

Purpose: Evaluate the app's responsiveness, scalability, and resource usage.

Tools: Flutter DevTools, profiling tools.

What to Test:

Loading times.

Memory usage.

Responsiveness under various conditions.

6.Security Testing:

Purpose: Identify and fix security vulnerabilities.

Tools: Manual code review, automated scanning tools.

What to Test:

Authentication and authorization mechanisms.

Data encryption and storage.

Network security.

7.Accessibility Testing:

Purpose: Ensure your app is accessible to users with disabilities.

Tools: Manual testing, accessibility testing tools.

What to Test:

Navigation using screen readers.

Text contrast and readability.

Keyboard navigation.

8.Regression Testing:

Purpose: Ensure that new code changes do not introduce new bugs or regress existing functionality.

Tools: Automated testing tools, version control systems.

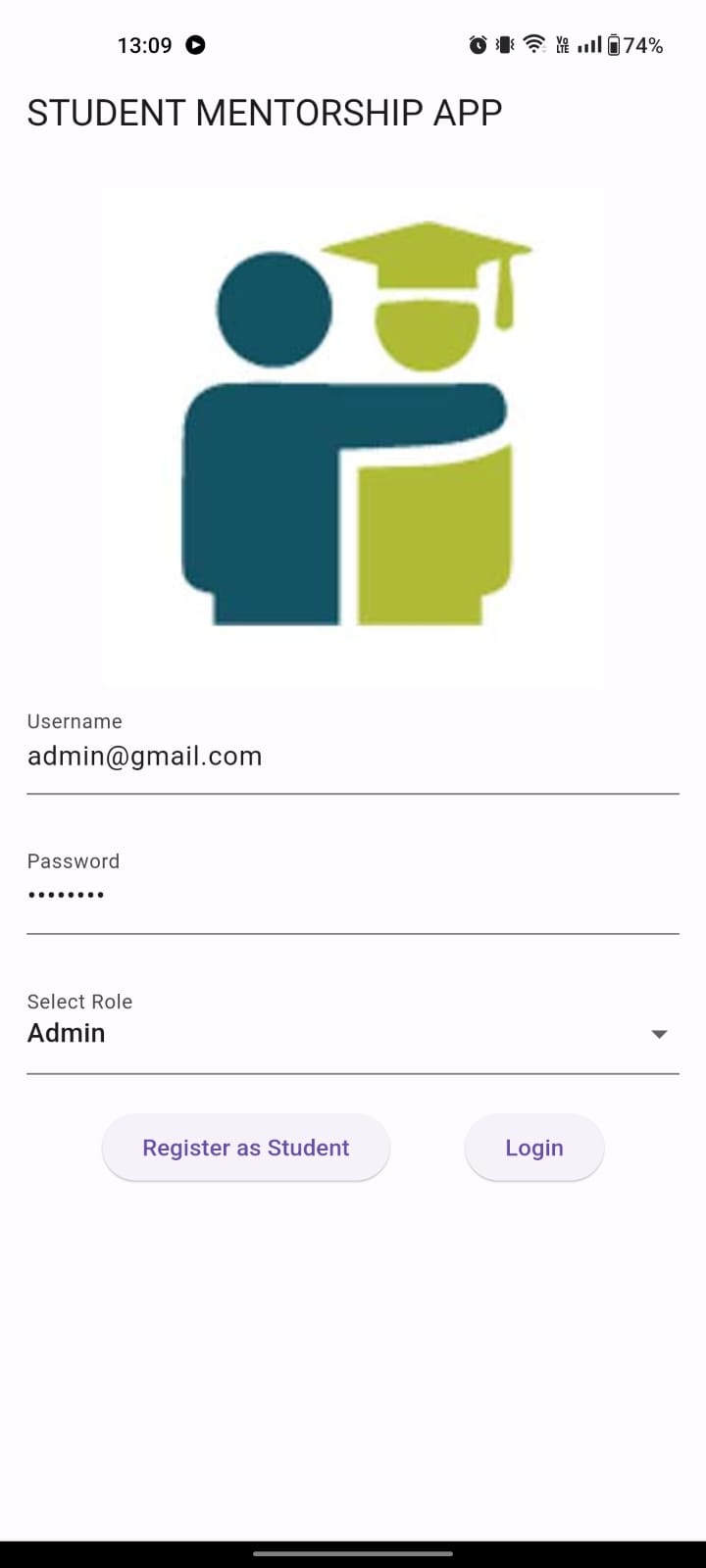
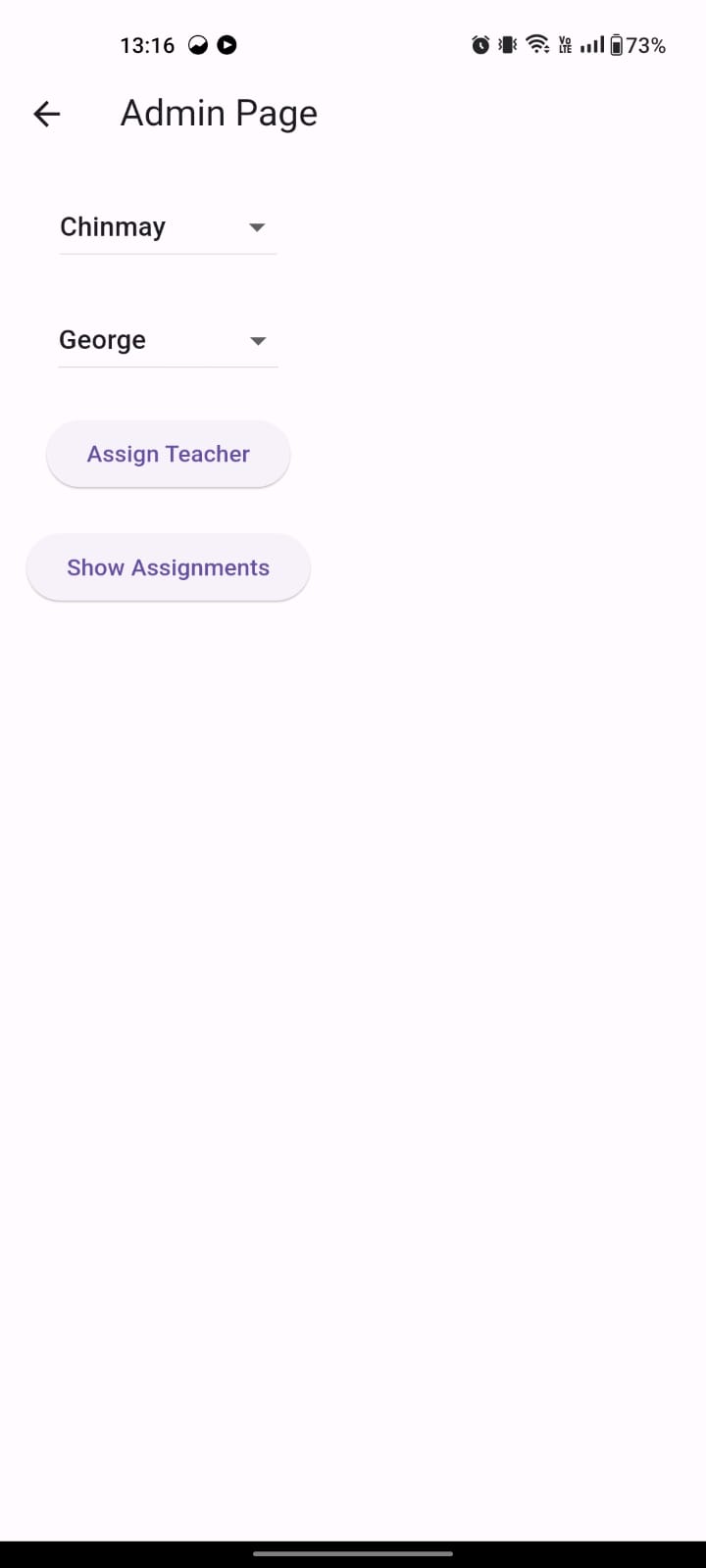
What to Test:

Key functionalities.

Previously identified issues.

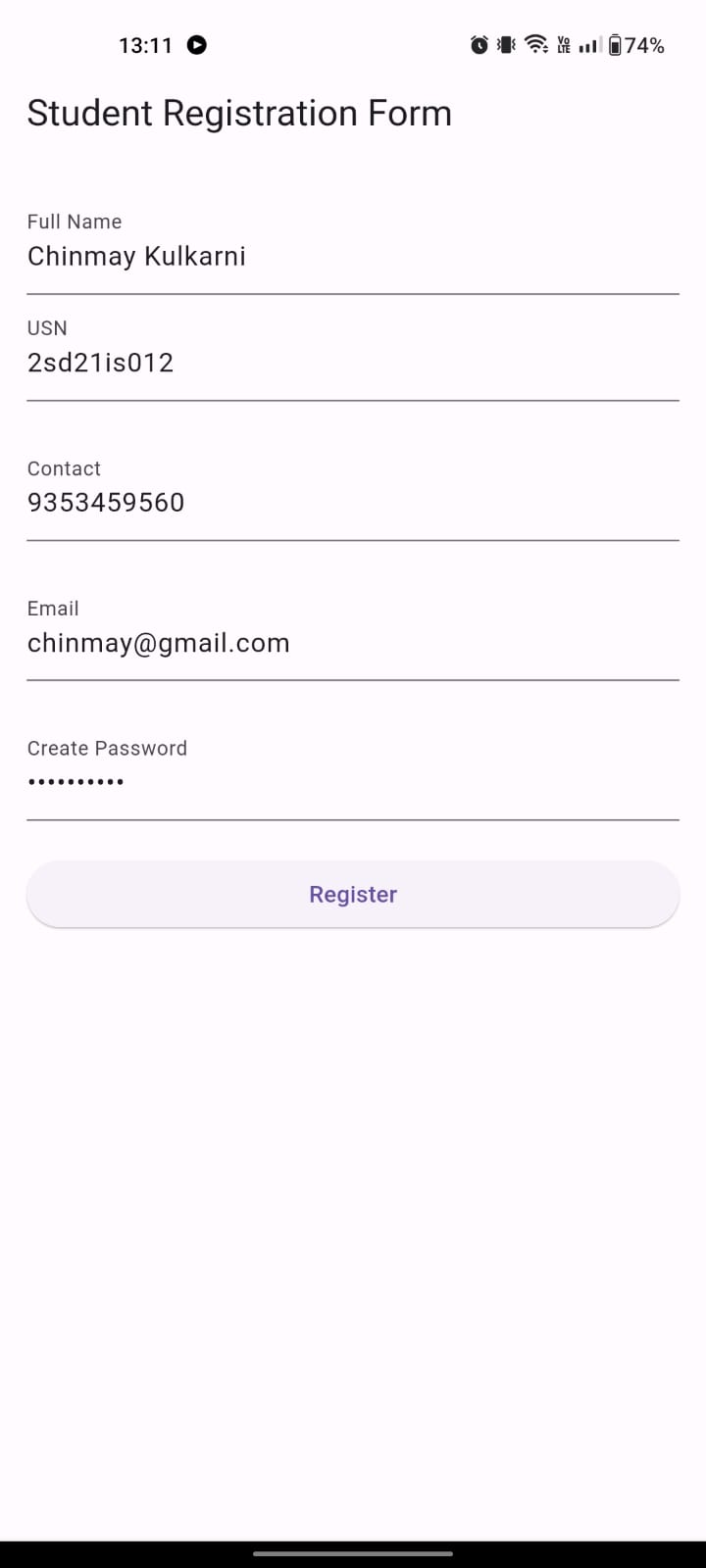
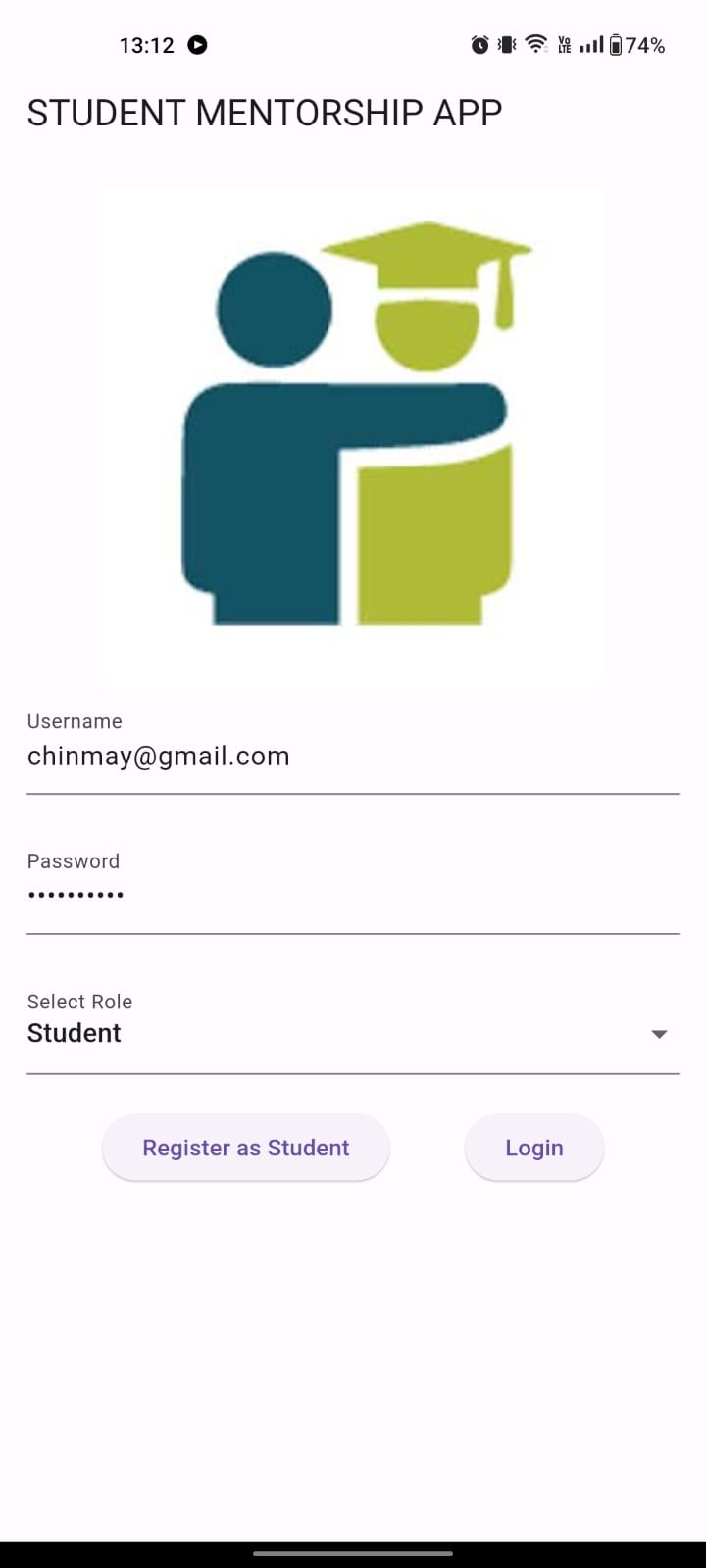
CHAPTER 9 RESULTS AND DISCUSSION

Admin:

After login, admin can view the registered students ,mentors and can assign students to respective mentors. Admin can also view the assigned details.

Student:

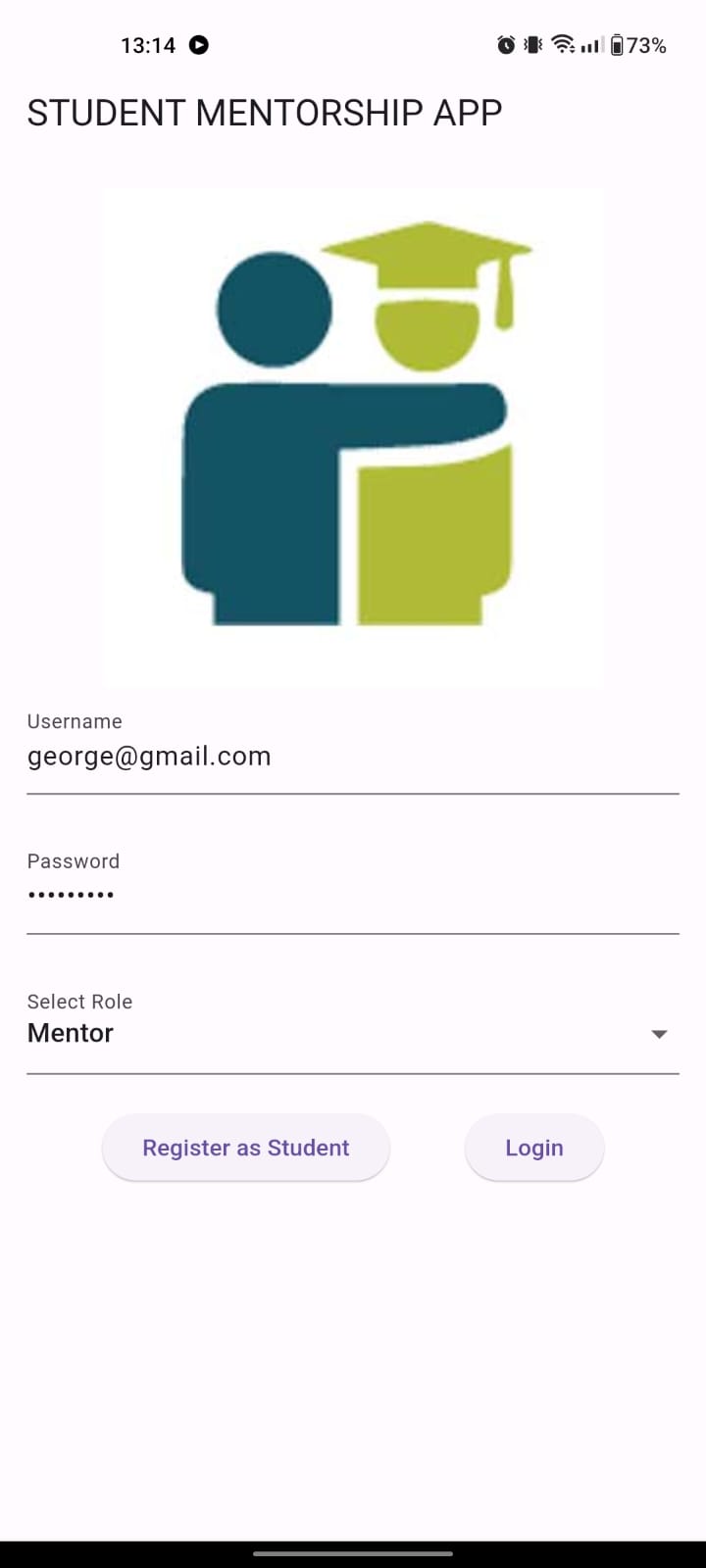
 

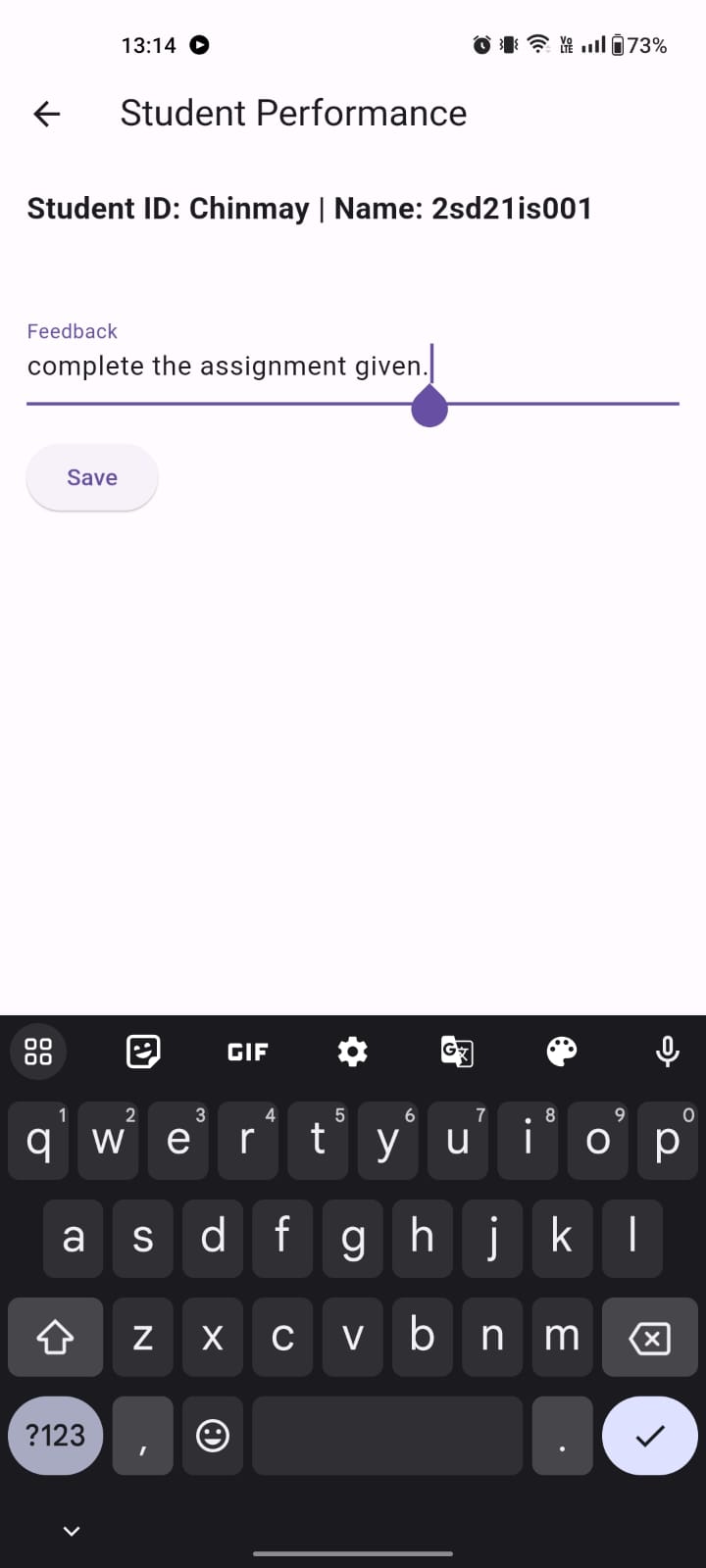
New student can register using registration form and then login as student.



After login as Student, student can view the assigned teacher and can view the feedback .

Mentor:



After login as mentor , Mentor can view the allocated students and can give feedback to the respective allocated student.

CHAPTER 10

APPLICATIONS

* Colleges and Universities: Facilitate mentorship programs within academic institutions, allowing professors to guide and support their students.
* Professional Development Programs: Implement mentorship initiatives in organizations to foster career growth and skill development among employees.
* Online Learning Platforms: Integrate mentorship features in e-learning platforms to provide personalized guidance and support to online learners.

CHAPTER 11

CONCLUSION

The Mentorship Android application is designed to enhance mentorship experiences within educational institutions. It facilitates meaningful interactions between mentors and students through features like tailored mentorship forms and feedback mechanisms. This technology-driven platform empowers mentors to provide personalized guidance while offering students access to valuable resources and goal-setting tools. Overall, it aims to create a supportive community that fosters personal and academic success among students.

**REFERENCES**

|  |
| --- |
| Tinoco-Giraldo, H., Torrecilla Sánchez, E. M., & García Peñalvo, F. J. (2020, October). Developing a design phase for a mentoring mobile app. In Eighth International Conference on Technological Ecosystems for Enhancing Multiculturality (pp. 273-282). |
| Clark, R., Sanders, M., Davidson, B., Jayaraman, S., & DiSalvo, C. (2015). The convergence innovation competition: helping students create innovative products and experiences via technical and business mentorship. In Human-Computer Interaction: Users and Contexts: 17th International Conference, HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part III 17 (pp. 144-153). Springer International Publishing. |
| DiMarzio, J. (2016). Beginning android programming with android studio. John Wiley & Sons. |
| https://jpinfotech.org/student-mentoring-system/ |