# *AttendEase*

Submitted by:

Alaa Albast (10725)

Under the Supervision of:

Dr. Hussein Joumaa

****

Department of Computer Sciences

**Lebanese University - Section 4 - Zahle**

***June 2024***

***Senior Project***

Contents

[*AttendEase* 1](#_Toc170799206)

[Chapter 1 : Introduction 8](#_Toc170799207)

[General Overview 8](#_Toc170799208)

[Current Attendance Procedure 8](#_Toc170799209)

[Problem Statement 10](#_Toc170799210)

[Chapter 2 : Proposed Solution 11](#_Toc170799211)

[For Doctors 12](#_Toc170799212)

[For Administration 12](#_Toc170799213)

[For Students 12](#_Toc170799214)

[Chapter 3: Technology Choices 13](#_Toc170799215)

[Development Platform 13](#_Toc170799216)

[Why Android Studio? 13](#_Toc170799217)

[Why Kotlin over Java? 13](#_Toc170799218)

[Cloud Platform 13](#_Toc170799219)

[Why Firebase and Cloud Firestore? 13](#_Toc170799220)

[Where is the Server Location? 14](#_Toc170799221)

[Why Firestore Test Mode? 14](#_Toc170799222)

[Chapter 4 : Technical Implementation 15](#_Toc170799223)

[Database Design 15](#_Toc170799224)

[Entities 16](#_Toc170799225)

[Relationships 16](#_Toc170799226)

[Benefits of this Design 17](#_Toc170799227)

[user interface 17](#_Toc170799228)

[Admin 17](#_Toc170799229)

[Login: 17](#_Toc170799230)

[Functions: 18](#_Toc170799231)

[Management: 19](#_Toc170799232)

[New Student 19](#_Toc170799233)

[New Doctor 20](#_Toc170799234)

[New Material 21](#_Toc170799235)

[Edit Doctor Materials 22](#_Toc170799236)

[Edit Student Materials 24](#_Toc170799237)

[Update Curriculum 25](#_Toc170799238)

[Archiving Data: 29](#_Toc170799239)

[Check Attendances: 31](#_Toc170799240)

[Current Attendances 32](#_Toc170799241)

[Previous Attendances 34](#_Toc170799242)

[Doctor 37](#_Toc170799243)

[Select Semester and Material 37](#_Toc170799244)

[Functions 38](#_Toc170799245)

[Show attendance 38](#_Toc170799246)

[take attendance 38](#_Toc170799247)

[Student 40](#_Toc170799248)

[Show Attendances 40](#_Toc170799249)

[Chapter 5:use cases scenario 41](#_Toc170799250)

[Admin 41](#_Toc170799251)

[Management 42](#_Toc170799252)

[Update Curriculum 43](#_Toc170799253)

[New Department 44](#_Toc170799254)

[New academic Year 45](#_Toc170799255)

[New Language 46](#_Toc170799256)

[New Semester 47](#_Toc170799257)

[Add material 48](#_Toc170799258)

[Add Student 49](#_Toc170799260)

[add doctor 50](#_Toc170799261)

[edit doctor materials 53](#_Toc170799262)

[remove materials 54](#_Toc170799263)

[Add Materials 54](#_Toc170799264)

[edit student materials 55](#_Toc170799265)

[remove materials 56](#_Toc170799267)

[Add Materials 56](#_Toc170799268)

[Archiving Data 57](#_Toc170799269)

[check attendance 58](#_Toc170799271)

[After selecting the Check Attendance option, the administrator will check either the Current Attendance or the Previous Attendance (archived). 58](#_Toc170799272)

[Current Attendance 59](#_Toc170799273)

[Previous Attendacnes 60](#_Toc170799274)

[Doctor 61](#_Toc170799275)

[take Attendance 62](#_Toc170799276)

[Check attendance 63](#_Toc170799277)

[Student 64](#_Toc170799278)

[chapter 6: Final Thoughts and Future Scope 65](#_Toc170799279)

[Conclusion 65](#_Toc170799280)

[Future enhancment 65](#_Toc170799281)

**ACKNOWLEDGEMENT**

We would like to gratefully acknowledge the kind guidance and expert

supervision of **Dr. Hussein Joumaa** throughout this work. Their consistent support has always been an immense source of motivation and encouragement.

We sincerely express our thankfulness to **Dr. Mouhamad Dandash** , Head of Department, Computer Science .

Finally, we extend our thanks to everyone who has contributed to our success, both directly and indirectly.

Alaa Al Bast

# Chapter 1 : Introduction

## General Overview

In the dynamic environment of academic institutions, the efficient management of student attendance is a critical aspect of ensuring a high-quality and smooth learning experience. This report aims to address a prevalent issue faced by universities: the difficulty of tracking and managing attendance records for both faculty and students.

## Current Attendance Procedure

In the current system, each student is required to write their name and file number on a paper during each lecture. The lecturer then verifies the presence of all the students whose names are listed on the paper. Following this, the faculty administrator checks all the attendance papers for all lectures across all courses and departments. This manual process is not only time-consuming but also prone to errors.

Université Libanaise

Faculté des Sciences 4-Zahlé

Département de Langues

Fiche de presence

|  |  |
| --- | --- |
| Nom de l’enseignant(e): |  |
| Signature de l’enseignant(e): |  |
| Matière : |  |
| Cursus : |  |
| Durée de la séance : |  |

|  |  |  |
| --- | --- | --- |
| Nom de l’étudiant | Numéro de dossier | Signature |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Problem Statement

The first challenge lies in the **attendance tracking of faculty members** for each lecture. The current manual methods are time-consuming, prone to human error, and lack the necessary precision and reliability. This inefficiency not only hampers the smooth functioning of the academic schedule but also impacts the overall productivity of the faculty.

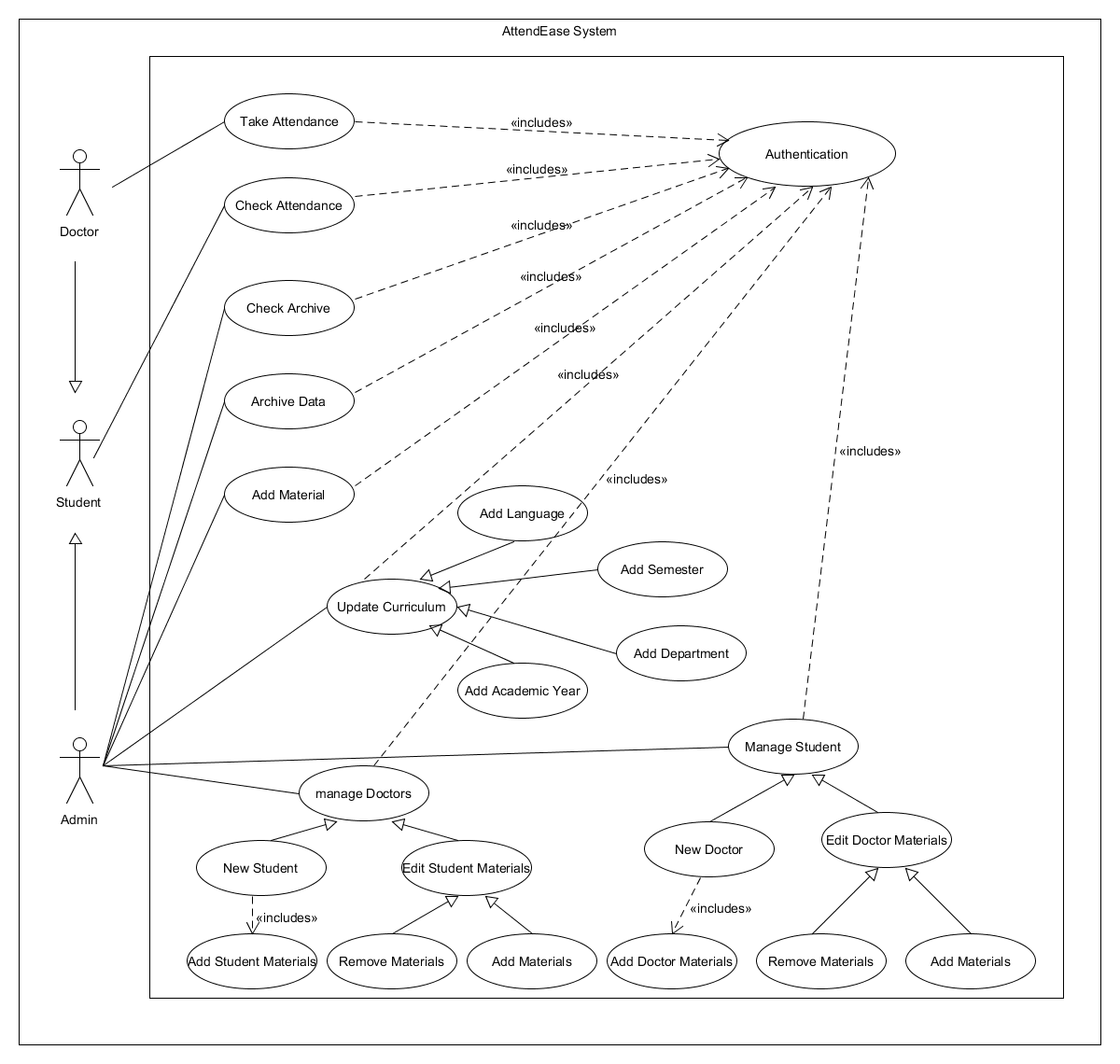
The second issue pertains to the **monitoring of student attendance** across different courses throughout the academic year. The administration faces the arduous task of accurately determining the attendance percentage of each student for each course. This data is crucial as it directly influences the eligibility of a student to appear for examinations.

Lastly, from the student's perspective, there is a significant difficulty in **keeping track of their attendance percentage** for each course. This lack of transparency and accessibility to their attendance records can lead to confusion and uncertainty, thereby affecting their academic planning and performance.

In conclusion, these challenges highlight the need for a robust, automated solution that can streamline the process of attendance tracking and management in universities. This report will delve into these issues in detail and propose an innovative application project aimed at resolving them.

# Chapter 2 : Proposed Solution

AttendEase app comes to solve the problem of time-consuming and error-prone attendance tracking methods in classrooms and professional settings. AttendEase, a mobile application, caters to doctors, students, and administrators, eliminating the need for paper sign-in sheets and manual recording. It offers a streamlined solution with benefits for everyone: doctors can quickly mark student attendance electronically, students can conveniently track their records, and administrators gain automated data collection, reduced errors, and secure archiving of attendance for each year. AttendEase saves valuable time, improves data accuracy, and fosters a more efficient and user-friendly experience for all.



## For Doctors

AttendEase offers a doctor interface. Doctors can effortlessly log in and select the semester they're teaching. From there, they can choose a specific course and create a new attendance session with just a few clicks. The intuitive interface allows doctors to seamlessly take attendance during lectures, while also enabling them to access and review student attendance records for a particular course throughout the semester. This streamlined approach empowers doctors to efficiently manage attendance and gain valuable insights into student participation within their courses.

## For Administration

The application serves as a powerful tool for the administration. It aggregates attendance data from all courses across all academic years, providing a holistic view of student attendance. The application automatically calculates the attendance percentage of students in each course, thereby determining their eligibility for examinations. This eliminates the need for manual calculations and potential errors, saving significant time and resources for the administration.

## For Students

The application acts as a personal attendance tracker for students. It provides real-time updates on their attendance percentage in all their courses. This feature not only keeps students informed about their current attendance status but also helps them plan their academic activities to meet the attendance criteria set by the university.

# Chapter 3: Technology Choices

## Development Platform

### Why Android Studio?

Our mobile app was developed using Android Studio, which provides a comprehensive range of tools to speed the process. It has intelligent code completion, powerful debugging tools, and a visual layout editor for simple app design.

### Why Kotlin over Java?

Despite Java's strong background in Android programming, Kotlin was chosen for its modern language, built-in null safety, and better developer productivity. Kotlin uses current programming techniques, resulting in clearer, more concise, and easier to maintain code. It also enforces null safety by default, avoiding common problems caused by null pointer exceptions in Java applications. Kotlin features like as extension functions and data classes encourage clearer code organization by removing boilerplate code, resulting in quicker development cycles and increased developer productivity. Despite both languages being viable.

## Cloud Platform

### Why Firebase and Cloud Firestore?

Every project requires a secure storage location for its data, and for us, Google's Firebase proved to be the best option. Firebase is a comprehensive set of tools that address many areas of mobile and online app development. Here's a breakdown:

* **Firebase Storage :** is a reliable data storage solution for user-generated content such as images, music, and video. It provides scalability, security, and ease of use for massive files.
* **Firebase Realtime Database :** is perfect for real-time data sync among clients, such as chat applications and collaboration features.
* **Firebase Cloud Firestore :** is a document-oriented database with offline capabilities, ideal for storing application data or user profiles. These NoSQL databases meet a variety of data storage demands.

For our project, capabilities such as user identification, analytics, and, most significantly, data storage were critical.Within Firebase, we chose Cloud Firestore, a NoSQL database solution. Traditional SQL databases or local storage on the device itself were examined, however Cloud Firestore offered specific features that exactly matched the needs of our app:

* **Scalability:** Cloud Firestore supports automatic scalability. As our user base expands, Cloud Firestore can handle the additional data load without requiring extra configuration or intervention.
* **Flexibility:** Unlike traditional databases, which have a strict, predefined structure, Cloud Firestore takes a flexible, document-oriented approach. This enables us to store many data kinds efficiently, which is critical for our app's requirements.
* **Real-time Updates:** Cloud Firestore enables real-time data syncing between devices. Consider a collaborative effort in which several users are involved. With Cloud Firestore, everyone sees the newest changes right away, ensuring that everyone is on the same page and has access to the most up-to-date information.

### Where is the Server Location?

GCPing.com was employed in the study to carefully choose a server location for data storage, guaranteeing the best possible user experience and low latency. By measuring ping times to multiple locations, the tool makes it possible to choose the server location that has the least delay for the intended audience.

**Europe West 6 (Zurich)** was selected as the best location because, depending on the type of data being stored, it may satisfy regional data residency rules and decrease latency for European consumers.

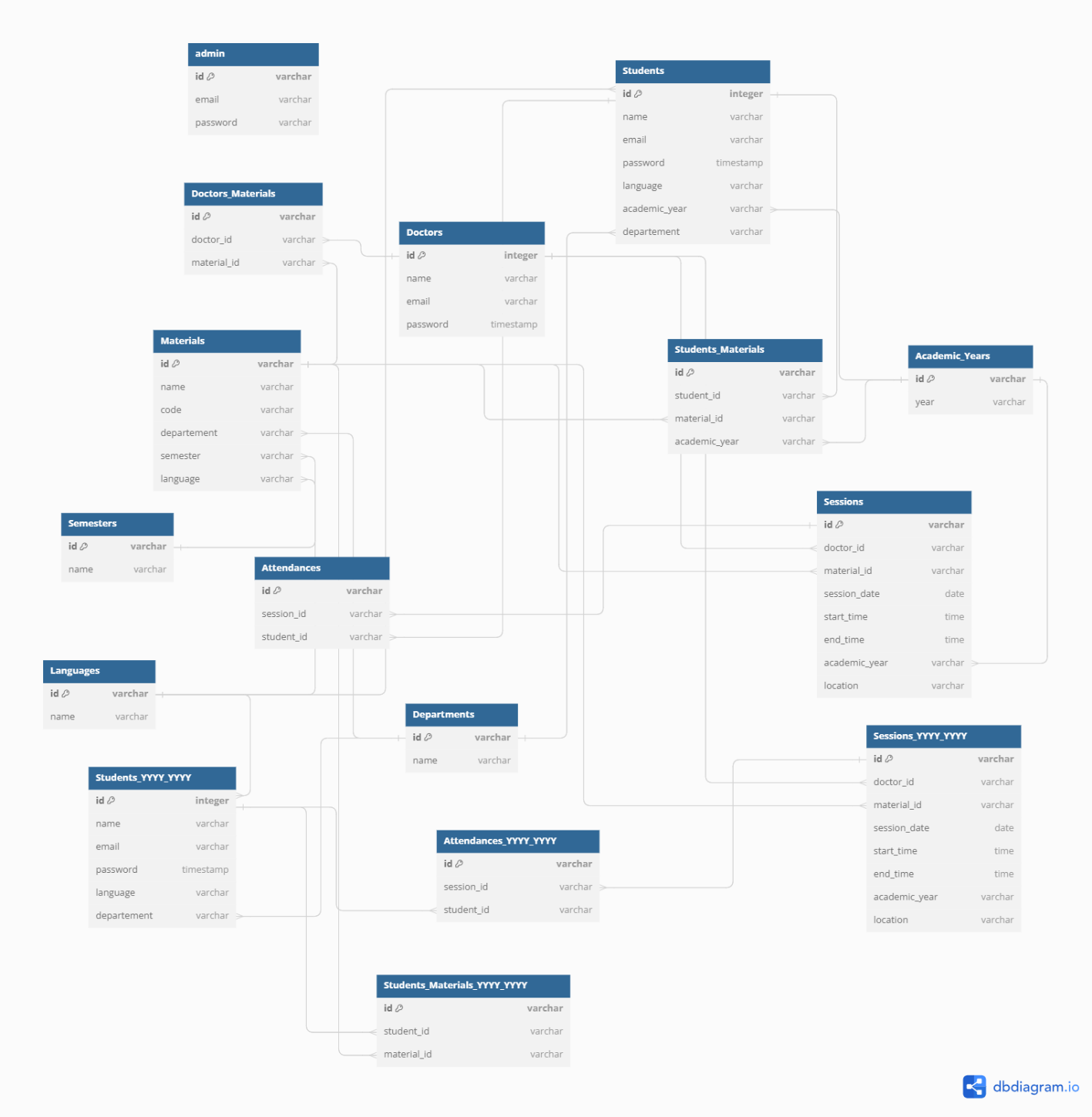
### Why Firestore Test Mode?

Test mode allows us to freely add, read, update, and delete attendance data without setting up security rules. This makes it much faster and easier to test our app's functionality during development.

# Chapter 4 : Technical Implementation

This chapter delves into the technological features of the AttendEase mobile application, which supports three user roles: administrators, doctors, and students. It describes the architecture that underpins these roles, the database design for data storage and management, and the user interface design for each position. The UI design provides a user-friendly experience for all three categories. The chapter provides a full knowledge of the AttendEase app's design and functionality, with the goal of providing a streamlined attendance management experience to all users.

## Database Design



This data model represents a university management system for mobile applications. It utilizes a relational database structure with tables to store information about admins, doctors students, materials ,departments, semesters, languages, academic years, sessions ,and attendances.

### Entities

* Admins: Manage the system.
* Doctors: Instructors who conduct sessions.
* Students: Users who enroll in courses.
* Materials: Course content assigned to students.
* Departments:Academic divisions offering courses.
* Semesters: Divisions of the academic year.
* Languages: Languages materials are offered in.
* Academic Years: Years of study for students.
* Sessions: Lectures conducted by doctors using specific materials.
* Attendances: Records of students' presence in sessions.

### Relationships

* **Many-to-Many:**
* Doctors and Materials (Doctors\_Materials): A doctor can be assigned many materials, and a material can be used by many doctors.
* Students and Materials (Students\_Materials): A student can enroll in many materials, and a material can be taken by many students.
* **Many-to-One:**
* Doctors and Sessions: A doctor can conduct many sessions, but a session has one doctor.
* Students and Sessions (Attendances): A student can attend many sessions, but a session has attendances from multiple students.
* Materials and Sessions: A material can be used in many sessions, but a session uses one material.
* Students, Materials, and Academic Years: Each entity relates to a specific academic year.
* Students and Departments: A student belongs to one department.
* Materials and Departments: A material belongs to one department.
* Materials and Languages: A material is offered in one language.
* Sessions and Academic Years: A session belongs to a specific academic year.

### Benefits of this Design

**Clear Data Organization:** Well-defined tables and relationships facilitate data management and understanding.

**Efficient Queries:** Relationships enable efficient retrieval of information based on specific criteria.

**Scalability:** The structure can accommodate future growth in data volume and complexity.

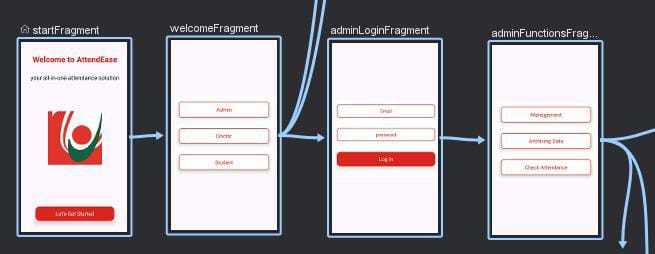
**Data Integrity:** Relationships enforce data consistency and prevent inconsistencies.

## user interface

The Attendease mobile application improves communication and management in the education industry. This chapter delves into the application's architecture, focusing on user interfaces for administrators, doctors, and students. Each part focuses on the design, UX principles, programming languages, frameworks, and libraries required to produce an intuitive interface. The database section goes into the data structure and management procedures used to save and retrieve information for each user type. The goal is to create a user-friendly and efficient platform for educational purposes.

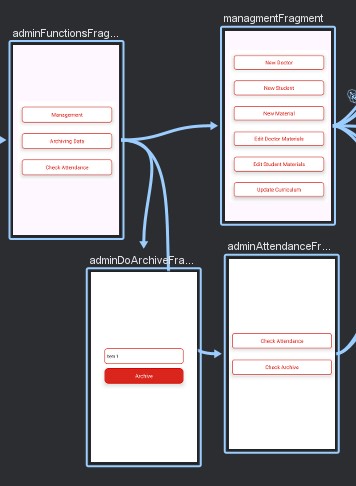
### Admin

#### Login:



The admin login functionality utilizes a two-step verification process. Upon clicking the admin's CardView, the application navigates to the *adminLoginFragment.* This fragment first validates user input by checking if both email and password fields are filled. If valid, it then queries the 'Admins' collection to verify if the entered email exists. If the email is found, the fragment proceeds to confirm if the corresponding password stored in the database matches the user's input. Successful authentication grants access to the *adminFunctionFragment*.

#### Functions:

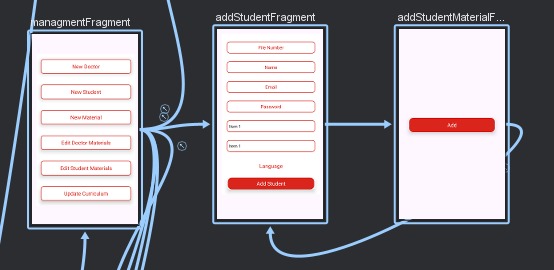


Upon successful login, the admin is directed to the *adminFunctionsFragment*. This fragment displays three card views, each corresponding to a specific administrative function. Clicking on a card view triggers navigation to a dedicated fragment for that function. For instance, clicking the "Management" card view navigates to the *managementFragment*, the "Archiving Data" card view directs to the *adminDoArchiveFragment*, and the "Attendances" card view leads to the *adminAttendancesFragment*. This approach promotes a modular and organized user interface for the admin functions.

#### Management:

Upon selecting the "Management" cardview, which navigates to the managementFragment, the administrator gains access to a comprehensive suite of administrative functionalities.

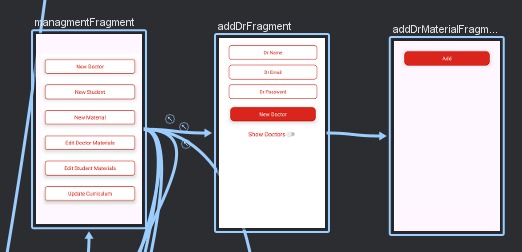
##### New Student



**The***ManagementFragment* **handles adding new students. Clicking the "New Student" card navigates to the** *AddStudentFragment***. This fragment fetches department names, IDs, academic years, languages, and IDs on creation from the database. Department and academic year names populate spinners, while languages dynamically create radio buttons within a radio group, each tagged with the language ID. Upon clicking "Add Student," the fragment validates all fields. If valid and the file number is unique, it retrieves department and academic year IDs based on their selected names. Finally, a new document is added to the "Students" collection with the student's details (including the chosen department, academic year, and language IDs). Safe Args are then used to navigate to** *AddStudentsMaterialsFragment***, passing the student ID, department ID, academic year ID, and language ID.**

The *addStudentsMaterialsFragment* leverages the onCreateView lifecycle method to retrieve relevant materials for assignment. It queries the "Materials" collection, filtering results based on department ID (received from the previous fragment) and language ID. The retrieved materials are then displayed within the layout. For each material, a checkbox is dynamically generated, with its name displayed and a unique identifier (tag) set based on the material's ID. Upon clicking the "Add" button, the application iterates through checked checkboxes, extracting their associated tags. It then creates documents within the "Students Materials" collection for each selected tag. These documents reference the student ID and academic year received from the previous fragment, effectively linking students to their assigned materials.

##### New Doctor

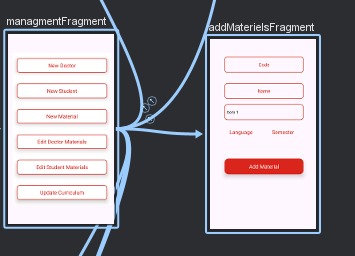


Clicking the "New Doctor" card within the *ManagementFragment* transitions the user to the *AddDoctorFragment*. This fragment retrieves all doctor names and emails from the "Doctors" collection in the database and stores them in a variable.We put this variable in a textView if the switch is on,else the textView is empty.

Upon clicking the "New Doctor" button, the fragment validates if all fields are filled. If valid, it verifies email uniqueness by comparing it against the retrieved list. If unique, a new document is added to the "Doctors" collection with auto-incremented ID, containing name, email, and password fields. Following a successful addition, navigation proceeds to the *AddDrMaterialsFragment*, passing the doctor ID using Safe Args.

The *addDrMaterialFragment* leverages the onCreateView lifecycle method to retrieve data from the firestore database. It first fetches all material IDs from the 'Doctors Materials' collection. Subsequently, it retrieves all material data from the 'Materials' collection. A nested loop is then employed to group materials by department and language. Within each group, checkboxes are dynamically generated for each material, with their IDs assigned as tags. However, only materials not found in the 'Doctors Materials' collection are included.Upon clicking the 'Add' button, the selected tags (material IDs) are retrieved, and a new document is created for each selected material within the 'Doctors Materials' collection. This document links the chosen material to the doctor ID received from the previous fragment.

##### New Material

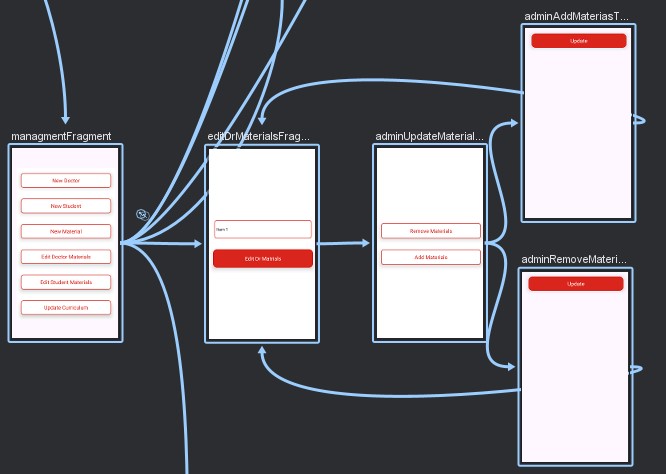


When the admin clicks on the "New Material" CardView in the *ManagementFragment,* it takes them to the *AddMaterialFragment*. In this fragment, we first retrieve all the materials from the database and save their codes, languages, and departments in three separate lists. Then we collected all of the departments' names and IDs, stored them in two separate lists, and displayed the department names in a spinner. Then, all languages' names and ids were obtained from the database and stored in two lists, before each language's name was shown as a radio button in a RadioGroup. It's the same for semesters.

The selected language, the languages name list, and the languages id list are passed to the function findString, which receives two lists and a string, and it returns the language id for the selected language name.

and by using this function, we were also able to obtain the department and semester ids. Next, if everything is checked, we used the Materials department list, Materials language list, and Materials code list to determine whether this material was unique. If it was, we added a new document to the materials collection with the following fields : Material Name, Material Code, Department (as id), Language (as id), Semester (as id).

##### Edit Doctor Materials



The EditDrMaterialsFragment will be displayed to the admin when they click on the "edit doctor materials" CardView. In this section, we retrieve every Doctor from the database and store their names and IDs in two distinct lists. Next, a spinner with the names of the doctors is displayed. We verify whether the admin has chosen a doctor when they click the "edit doctor materials" CardView.Then, the selected doctor's ID is obtained using the findString function and sent to the following fragment AdminUpdateMaterialsDrFragment using saveArgs.

This fragment has two CardViews. The first, "Remove Materials", directs the administrator to the fragment “AdminAddMaterialsToDrFragment“ for removing materials from the specified doctor . The second, "Add Materials", takes the administrator to the fragment “AdminRemoveMaterialsFromDrFragment” where they can add more materials to a doctor. Of course, both CardViews sends the doctor ID (obtained from the bundle) to the following fragment.

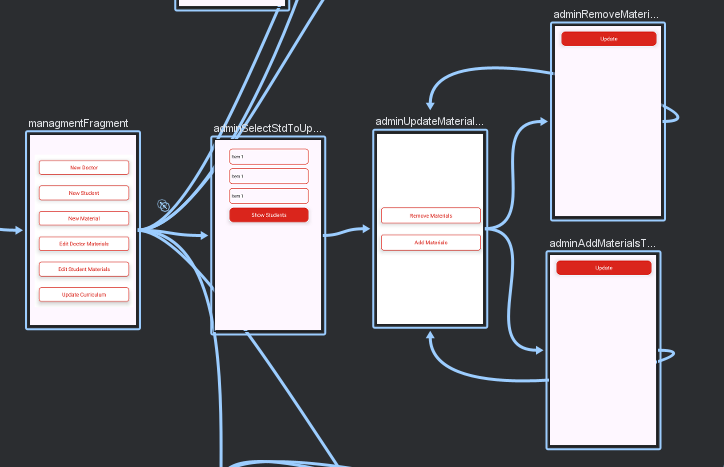
In the AdminRemoveMaterialsFromDrFragment, the administrator should uncheck the materials that they want to remove from the doctor. So, we take the selected doctor ID from the bundle, retreive all of their materials from the doctor's Materials collection, and save them in the materialsDrId list. Then we get all the language names and IDs from the languages collection, all the department names and IDs from the departments collection, and all the material names, languages, IDs, and departments from the materials collection. When everything has been retrieved, a nested loop is used to group all of the doctor materials by departments and languages, with each material shown as a checkbox.

When the admin clicks on the “remove material” CardView, we save all the IDs of the unchecked materials in a list, and we use this list to remove all the documents that contain the specified doctor ID and those materials' IDs from the collection "doctor's materials".

In the “AdminAddMaterialsToDrFragment”, the admin can select materials to assign to a chosen doctor. The code first retrieves the doctor's ID from the fragment's bundle. It then fetches all material IDs currently assigned to doctors and retrieves separate lists of language names/IDs and department names/IDs for reference. Next, it filters materials from the `materials` collection to include only those not assigned to any doctor. Finally, nested loops group these available materials by department and language, displaying each material as a checkbox for the admin's selection. This process ensures the admin can efficiently assign relevant materials to the chosen doctor.

When the administrator confirms the modification by clicking the CardView, the system collects the IDs of all checked materials. It then combines the doctor's ID (which was likely obtained previously) with each specified material ID to provide a unique identity for each assignment. Finally, new documents are created, each with the doctor's ID and a single material ID. These documents are subsequently added to the 'doctorsMaterials' collection, thereby attributing the materials to the appropriate doctor. This approach simplifies the assignment of several materials to a doctor in a single step.

##### Edit Student Materials



In the management fragment, clicking the "edit student material" CardView leads the administrator to the '*AdminSelectStudentToUpdateMaterialsFragment*'. The administrator can refine their student search based on three criteria: department, language, and academic year. Each criterion is selectable from a spinner that contains data retrieved from the database.

When the administrator selects the "show students" button, the system first confirms that all three filters (department, language, and academic year) are selected. If the selection is complete, the system retrieves all students who belong to the specified department, speak the chosen language, and are enrolled in the chosen academic year. These filtered students are then provided for the administrator to select using radio buttons.

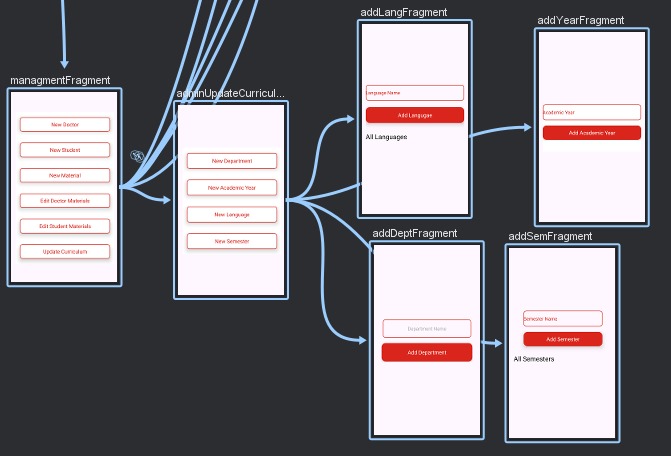
After selecting a student, the admin is sent to the *'AdminUpdateMaterialsStdFragment*', where the selected student's ID , the selected department’s ID, and the selected academic year’s ID are given along as navigation arguments ('saveArgs') for subsequent operations in the next fragment. This step-by-step filtering approach makes it easier to discover the exact student whose materials need to be updated.

This fragment has two CardViews. The first, "Remove Materials", directs the administrator to the fragment “*AdminAddMaterialsToStdFragment*“ for removing materials from the specified student . The second, "Add Materials", takes the administrator to the fragment “*AdminRemoveMaterialsFromStdFragment*” where they can add more materials to a student. Of course, both CardViews sends the student ID,department ID, and academic year ID (obtained from the bundle) to the following fragment.

In the *AdminAddMaterialsToStdFragment*, the admin can add materials to a chosen student. The system first retrieves all material IDs the student is already registered for from the students Materials collection. This helps filter out duplicates. Then, it fetches all materials that share the same department and language as the student. These are the potential materials the student can be assigned. Finally, the admin selects the desired materials using checkboxes. Clicking the "update" button triggers the creation of new documents, each combining the student's ID with a chosen material's ID. These documents are then added to the students Materials collection, effectively assigning the selected materials to the student. This streamlined process ensures efficient and targeted material assignment.

In the AdminRemoveMaterialsFromStdFragment, the admin can remove materials from a student. First, the system retrieves all material IDs the student is currently registered for from the students Materials collection. Then, it fetches the names of those materials from the materials collection. These material names are displayed as checkboxes for the admin's selection. Any unchecked checkbox corresponds to a material that the admin wants to remove. Clicking the "update" button triggers the identification of all unchecked materials . Finally, the system removes the corresponding documents from the students Materials collection, each document containing the student's ID and a material ID(from the unchecked materials) slated for removal.

##### Update Curriculum

******

The updateCurriculumFragment is displayed to the administrator upon clicking the “Update Curriculum” CardView. This fragment contains four CardViews: “New Language”, “New Department”, “New Semester”, and “New Academic Year”. Each CardView, when selected, navigates to a different fragment (*AddLangFragment, AddDeptFragment, AddSemFragment,* and *AddYearFragment* respectively) allowing the administrator to add new languages, departments, semesters, and academic years to the curriculum, thereby ensuring the system’s adaptability to the evolving needs of the institution.

1. ***New Department***

Upon interacting with the “Add Department” cardview, the user is navigated to the *AddDeptFragment*. In the onCreateView method of this fragment, a TextView is populated with the names of all departments. This is accomplished by invoking the getAllDepartments function.

The getAllDepartments function retrieves all documents from the “Departments” collection in Firestore. Each document corresponds to a department, and the name of the department is appended to the TextView.

The addDeptButton is configured with an OnClickListener. When clicked, the entered department name is retrieved, converted to lowercase, and validated for non-emptiness. If valid, a unique ID is generated for a new document in the “Departments” collection in Firestore, and the department name is stored in this document. Subsequently, the input field is cleared and the getAllDepartments function is invoked again to refresh the list of departments displayed.

If the input field is empty when the button is clicked, a toast message prompts the user to enter a department name. This ensures that no department is added without a valid name.

1. ***New Academic Year***

The "Add Academic Year" cardview directs the user to the *AddYearFragment*. Upon the initialization of this view, the getAllYears function is invoked to retrieve and display all the academic years from the Firestore collection named "Academic Years". The years are presented in descending order and appended to the TextView.

The addYearButton is equipped with an OnClickListener. When this button is clicked, the text input in the EditText is extracted and verified to ascertain if it adheres to the required format (YYYY\_YYYY). If the format is validated, a unique ID is generated and a new document bearing this ID is established in the "Academic Years" collection, with the entered year as the value. Subsequently, the EditText is cleared and the getAllYears function is invoked once more to refresh the displayed list of years.

In the event that the entered year does not conform to the required format, a toast message is displayed to guide the user to input the year in the correct format, and the EditText is cleared. This ensures the integrity of the data entered into the system.

1. ***New Language***

The "Add Language " cardview serves as a navigation point, directing the user to the *AddLangFragment.*

The languagesRef variable is a reference to the "Languages " collection in the Firestore database. An addSnapshotListener is attached to languagesRef, which retrieves all documents from the "Languages "collection whenever there is a change. Upon successful retrieval, it clears the LanguagesName list, iterates over the documents, and adds the name of each language to the LanguagesName list and LangNameString string. The allLang TextView is subsequently updated with the names of all languages.

The AddLang CardView has a click listener attached to it. Upon being clicked, it verifies if the LangName input field is not empty and the entered language does not already exist in the LanguagesName list. If both conditions are satisfied, it creates a new language document with the entered name and adds it to the “Languages” collection in the Firestore database. If the addition is successful, it clears the LangName input field and displays a success message. If the addition fails, it displays a failure message. If the LangName input field is empty or the entered language already exists, it displays an appropriate error message.

The checkLangExists function is a helper function that checks if a given language name exists in the LanguagesName list. It returns true if the name exists and false otherwise. This function is utilized to prevent the addition of duplicate languages to the database.

1. ***New Semester***

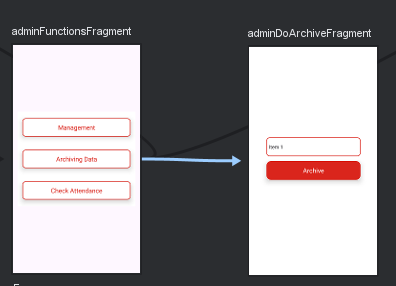
The “Add Semester” CardView serves as a navigation element that directs the user to the *AddSemFragment.* The SemestersRef object is a reference to the “Semesters” collection in the Firestore database. The addSnapshotListener method retrieves all documents from this collection. Upon successful retrieval, the SemestersName list is cleared and updated with the semester names fetched from the database. These names are subsequently displayed in the allSem TextView.

The “AddSem” CardView also functions as a button. Upon clicking, it validates the entered semester name based on three conditions: it is not empty, it does not already exist in the SemestersName list, and it matches the required format (“semester\_n”). If these conditions are satisfied, a new document is added to the “Semesters” collection in the Firestore database with the entered semester name. A success message is displayed and the SemName TextView is cleared if the addition is successful. Conversely, a failure message is displayed if the addition fails.

The checkSemExists function is used to verify if a given semester name already exists in the SemestersName list. It returns true if the name exists and false otherwise. This function aids in preventing the duplication of semester names in the database.

#### Archiving Data:

Upon selecting the "Archiving" cardview in the *adminFunctionsFragment*, which navigates to the *adminDoArchiveFragment*, the administrator gains access to archive all the data of a selected year.



Upon the instantiation of the view, the layout for this fragment is inflated using the ‘fragment\_admin\_do\_archive’ layout file. A connection to the Firestore database is established, and several variables are initialized, including a spinner for the academic year and a button for executing the archive operation.

The code retrieves a list of academic years from the Firestore database, ordered in descending order. This data populates the year spinner. This is achieved by iterating over the result set of the Firestore query, extracting the academic year and its ID from each document, and adding them to the respective lists. An ArrayAdapter is then created with the academic years list and set to the year spinner.

Upon clicking the archive button, the selected academic year is retrieved. If no year is selected, a toast message is displayed and the function returns early. If a year is selected, the corresponding academic year ID is retrieved using the ‘findString’ helper function.

The code then performs several operations to move data related to the selected academic year to the archive and delete the original data. This is done for the "Students", "Students Materials", "Sessions", and "Attendances" collections. Each operation involves a Firestore query to retrieve the data, a for loop to iterate over the result set, and Firestore commands to move the data to the archive and delete the original data.

Firstly, all the data from the collection "Students" where the academic year id equals the selected academic year is retrieved. Then, a new Collection "Students YYYY\_YYYY" (where YYYY\_YYYY is the selected year) is created. All the students of academic year YYYY\_YYYY are moved from the collection "Students" to the collection "Students YYYY\_YYYY" using a for loop to delete from the first and add to the second.

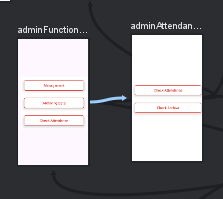
The same operation is performed with the "Students Materials" table by creating a collection "Students Materials YYYY\_YYYY" and moving the data from the first to the second.

For the "Sessions" and "Attendances" collections, the data from the collection "Sessions" where the academic year equals the selected academic year is first retrieved. In a for loop, each retrieved document from the "Sessions" is moved to the "Sessions YYYY\_YYYY" collection. For each moved document from the collection "Sessions", a for loop is used to move all the documents in the "Attendances" collection, where the session id equals the moved session id, to the collection "Attendances YYYY\_YYYY".

In the case of the “Sessions” collection, there is an additional nested for loop to handle the “Attendances” collection. This loop iterates over the attendances of each session, checks if the session ID is in the sessionsList, and if it is, moves the attendance data to the archive and deletes the original data.

After each operation, a toast message is displayed to inform the user that the operation was successful. This ensures that the user is kept informed about the progress of the archiving process.

#### Check Attendances:

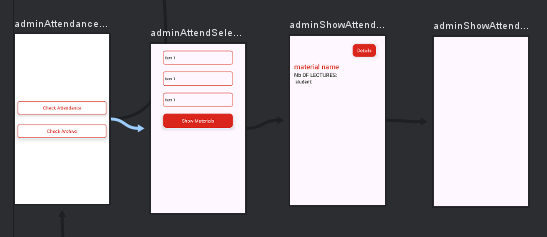


The administrator interface features a "Check Attendances" option,in the *adminFunctionsFragment.* This option is presented as a CardView element, and upon selection, it directs the user to the *adminAttendanceFragment.* This fragment serves as a gateway for administrators to review faculty attendance records across all academic years.

Within the *adminAttendanceFragment*, there are two additional CardView elements: "Current Attendances" and "Previous Attendances". These elements facilitate navigation to their respective fragments, *adminAttendSelectMaterialFragment a*nd *adminCheckArchiveSelectMaterialFragment.*

The “Current Attendances” option leads to the *adminAttendSelectMaterialFragment*, where administrators can inspect the attendance records for the ongoing academic period. Conversely, the “Previous Attendances” option redirects to the *adminCheckArchiveSelectMaterialFragment,* providing a historical view of attendance records from past academic terms. This structure ensures a comprehensive and organized review of faculty attendance data.

##### Current Attendances



The adminAttendSelectMaterialFragment is displayed to the administrator upon clicking the "Current Attendances" CardView.

1. ***Select Material***

The fragment contains several mutable lists of strings (departments, deptId, languages, langId, semesters, semId) which are used to store data retrieved from a Firestore database.

The code retrieves data from three Firestore collections: "Departments", "Semesters", and "Languages". For each collection, it clears the corresponding lists, adds a default value, and then iterates over the documents in the collection. For each document, it retrieves the name and ID, and adds them to the corresponding lists. It then creates an ArrayAdapter for each list and sets it as the adapter for the corresponding Spinner.

The show button’s setOnClickListener method is then defined. When the button is clicked, it retrieves the selected items from the spinners and checks if valid selections have been made. If not, it shows a Toast message and returns. If valid selections have been made, it retrieves the corresponding IDs from the lists using the function "findString".

The code then retrieves documents from the "Materials" collection where the "Semester", "Language", and "Department" fields match the selected values. For each document, it creates a RadioButton, sets its text and tag(the material id), and adds it to the RadioGroup. It also sets an OnCheckedChangeListener for the RadioGroup that navigates to the *AdminShowAttendFragment* when a radio button is selected with sending the material id using safe args.

1. ***Show Attendances***

The onCreateView function inflates the fragment layout, initializes the TextViews, and sets up a button click listener. It also retrieves the materialId from the arguments passed to the fragment.

The db.collection("Materials").document(materialId).get() function retrieves the name and code of the selected material from the Firestore database and displays it in the materialDesc TextView.

The db.collection("Students").get() function retrieves all student IDs and names from the Firestore database and stores them in AllStudentsId and AllStudentsName lists respectively.

The getAllSessionId function retrieves all session IDs associated with the current material from the Firestore database and stores them in the sessionsIds list. It also updates the nbOfLectures TextView with the number of lectures.

The getAllStdIdFromAtt function retrieves all student IDs from the attendance records for each session and stores them in the AllStdAttIds list. It ensures that the getDistinctStdId function is called only after all sessions have been processed.

The getAllStdBySession function retrieves all student IDs from the attendance records for a specific session and adds them to the AllStdAttIds list.

The getDistinctStdId function retrieves distinct student IDs associated with the current material from the Firestore database and stores them in the stdAttIds list. For each student, it calculates the attendance percentage and displays the student’s name, ID, number of attendances, and attendance percentage in a new TextView which is added to the checkboxContainer LinearLayout. The text color is set based on the attendance percentage:red for the student how has a percentage smaller then 50%,and green for the percentage bigger then 50%.

1. ***Show Attendances Details***

In the *AdminShowAttendFragment,*the details button navigates to the fragment *AmdinShowAttendDetailsFragment.*

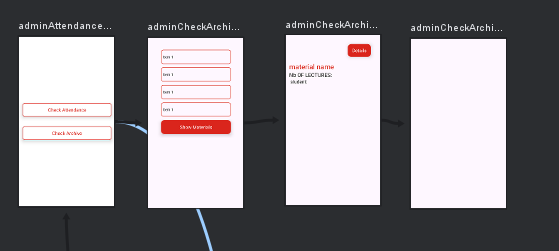
At the beginning , two mutable lists drIdList and drNameList are declared. These lists are designed to store the IDs and names of doctors, respectively.The materialId is also retrieved from the arguments passed to this fragment.

Next, a connection is established with the Firestore database to fetch the collection of "Doctors". For each doctor in the collection, the doctor’s ID and name are retrieved and added to drIdList and drNameList, respectively.

The fetchSessions function is launched in a coroutine on the main thread. This function fetches "Sessions" from the Firestore where the "Material Id" matches the materialId. For each session, it retrieves various details like date, start time, end time, doctor ID, location, and session ID. It then finds the corresponding doctor’s name using the findString() function and adds this information to the UI in the form of TextViews.

The fetchAttendances function is called for each session. It fetches "Attendances" from the Firestore where the "Session Id" matches the sessionId. For each attendance, it retrieves the student ID, fetches the corresponding student’s name from the "Students" collection, and adds this information to the UI in the form of a TextView.

##### Previous Attendances



The *adminCheckArchiveSelectMaterialFragment* is displayed to the administrator upon clicking the "Previous Attendances" CardView.

1. ***Select Material***

The fragment contains several mutable lists of strings (departments, deptId, languages, langId, semesters, semId,academicYears,yearId) which are used to store data retrieved from a Firestore database.

The code retrieves data from three Firestore collections: "Departments", "Semesters","Academic Years", and "Languages". For each collection, it clears the corresponding lists, adds a default value, and then iterates over the documents in the collection. For each document, it retrieves the name and ID, and adds them to the corresponding lists. It then creates an ArrayAdapter for each list and sets it as the adapter for the corresponding Spinner.

The show button’s setOnClickListener method is then defined. When the button is clicked, it retrieves the selected items from the spinners and checks if valid selections have been made. If not, it shows a Toast message and returns. If valid selections have been made, it retrieves the corresponding IDs from the lists using the function "findString".

The code then retrieves documents from the "Materials" collection where the "Semester", "Language", and "Department" fields match the selected values. For each document, it creates a RadioButton, sets its text and tag(the material id), and adds it to the RadioGroup. It also sets an OnCheckedChangeListener for the RadioGroup that navigates to the *AdminCheckArchiveFragment* when a radio button is selected with sending the material id and academic year id using safe args.

1. ***Show Archive***

The fragment contains several mutable lists and variables to store data such as session IDs, student IDs, student names, and attendance IDs. It also contains TextViews to display material descriptions, the number of lectures, and student information.

In the onCreateView method, the layout for this fragment is inflated and the material ID and selected year are retrieved from the bundle. The TextViews are initialized and an OnClickListener is set on a button to navigate to the AdminCheckArchiveDetailsFragment.

The code then retrieves the name and code for the material from the "Materials" collection in Firestore and sets the text of materialDesc TextView. It also retrieves all student IDs and names from the "Students $selectedyear" collection in Firestore, clears the existing lists, and adds the retrieved data to the lists. After this, it calls the getAllSessionId method.

The getAllSessionId method retrieves all session IDs where the material ID matches the current material ID from the "Sessions $selectedyear" collection in Firestore. It clears the sessionsIds list and adds the retrieved session IDs to the list. It then sets the text of nbOfLectures TextView to display the number of lectures and calls the getAllStdIdFromAtt method.

The getAllStdIdFromAtt method retrieves all student IDs from the attendance data for each session ID from the "Attendances $selectedyear" collection in Firestore. It clears the AllStdAttIds list and adds the retrieved student IDs to the list. It then calls the getDistinctStdId method.

The getDistinctStdId method retrieves distinct student IDs from the "Students Materials $selectedyear" collection in Firestore and displays the student name, ID, number of attendances, and attendance percentage in a TextView. The color of the text changes based on the attendance percentage. If the percentage is greater than or equal to 50, the text color is green; otherwise, it’s red. The TextView is then added to the checkboxContainer LinearLayout.

1. ***Show Archive Details***

The *AdminCheckArchiveDetailsFragment* is used to display detailed information about archived sessions.

In the onCreateView() method, the layout for this Fragment is inflated, which sets up the user interface for this Fragment. The arguments passed to this Fragment are retrieved, which include the materialId and year. These are likely identifiers for the specific material and academic year that this Fragment will display information about.

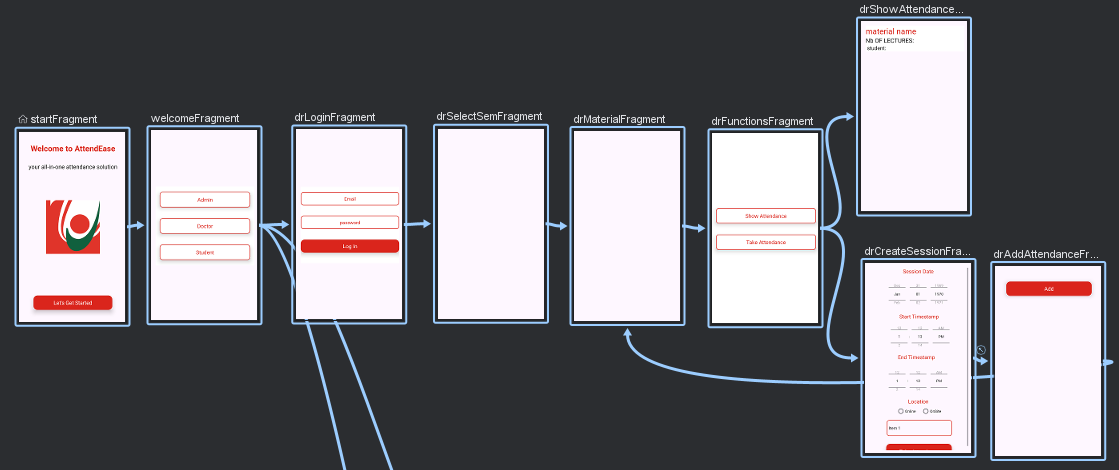
The Firestore database instance is then initialized.The list of doctors is fetched from the Firestore database and their IDs and names are stored in drIdList and drNameList, respectively.

A coroutine is then launched on the main thread to fetch the sessions related to the given materialId. Coroutines are a way of writing asynchronous code in a sequential manner.

The fetchSessions() function is a suspend function that fetches the sessions related to the given materialId from the Firestore database. For each session, it retrieves the session details, finds the name of the doctor who conducted the session, and adds a TextView to the LinearLayout to display the session details.

The fetchAttendances() function is another suspend function that fetches the attendances for a given session from the Firestore database. For each attendance, it retrieves the student’s ID, fetches the student’s details from the Firestore database, and adds a TextView to the LinearLayout to display the student’s ID and name.

### Doctor



The doctor login functionality utilizes a two-step verification process. Upon clicking the doctor's CardView, the application navigates to the *DrLoginFragment.* This fragment first validates user input by checking if both email and password fields are filled. If valid, it then queries the 'Doctors' collection to verify if the entered email exists. If the email is found, the fragment proceeds to confirm if the corresponding password stored in the database matches the user's input. Successful authentication grants access to the *DrSelectSemFragment.*

#### Select Semester and Material

In the *DrSelectSemFragment*, the doctor can select their semester. The code retrieves a list of semesters from a collection named 'Semesters'. The code iterates through this list, creating a card with the semester name in the center. Tapping this CardView takes the doctor to a new fragment DrMaterialFragment, passing along both the doctor's ID and the selected semester ID.

First, it collects information transmitted from another fragment. This contains the doctor's ID and the semester you've chosen. Then it pulls a list of material IDs associated with that doctor from a  collection called "Doctors Materials".

Next, it creates a list of all available languages in the  "Languages" collection. Finally, it searches the "Materials" collection for all materials available in the given semester.

The code loops through each material in the specified semester. If a material's ID matches one associated with the chosen doctor, a card is generated. This card shows the material's name, code, and language. Clicking this card takes you to the *DrFunctionsFragment*, passing along the doctor's ID and the material's ID .

Functions

The *DrFunctionsFragment* displays two card views, each corresponding to a specific doctor function. Clicking on a card view triggers navigation to a dedicated fragment for that function. For instance, clicking the "Check Attendance" card view navigates to the *DrShowAttendanceFragment*, and the "Take Attendance" card view leads to the *DrCreateSessionFragment*. This approach promotes a modular and organized user interface for the doctor functions.

##### Show attendance

This fragment retrieves and visualizes student attendance data for a specific material chosen by a doctor. It begins by acquiring the material's ID and then fetches its name and code from the database. This information is displayed on the screen for reference.

Next, the fragment gathers a complete list of student IDs and their corresponding names. It then identifies all lectures or sessions associated with the chosen material and stores their IDs. The total number of lectures is then displayed.

To determine student attendance, the fragment iterates through each session ID, asynchronously retrieving a list of attending students for each session. These are accumulated into a separate list. Once all sessions are processed, the fragment retrieves a distinct list of enrolled students for the chosen material.

The fragment then calculates attendance percentages for each enrolled student by checking if their ID appears in the attendance list and dividing the number of attended sessions by the total number of lectures. This percentage is displayed alongside the student's name and ID, with the text color indicating good attendance (green) or needing improvement (red). Finally, this attendance information for each student is presented in a clear and organized manner on the user interface.

##### take attendance

The *DrCreateSessionFragment* allows a doctor to create a new session. The doctor can first select an academic year from a dropdown menu populated with available years retrieved from the database, ordered by year in descending order. They can then choose a start date and time using a DatePicker and TimePicker, respectively. Additionally, the doctor can select an end time using another TimePicker. Finally, they must choose a location from a RadioGroup containing location options.

To create the session, the doctor clicks the "Add Session" button. The code performs several validation checks to ensure all required fields are filled. If any field is missing (start date, start time, end time, location, or academic year), a toast message is displayed prompting the doctor to complete the missing information.

Upon successful validation, the code retrieves the selected location text and the corresponding academic year ID (by matching the selected year string with the internal arrays). It then formats the chosen date and time into strings.

Finally, a new document is created in the "Sessions" collection of the Firestore database. This document contains details about the new session, including the doctor's ID, material ID (likely passed from another fragment), location, date, start time, end time, and academic year ID.

If the session creation is successful, a toast message confirms the addition and the doctor is automatically navigated to a new fragment *DrAddAttendanceFragment* likely for managing attendance for the newly created session. The navigation arguments passed along include the material ID and the newly created session ID.

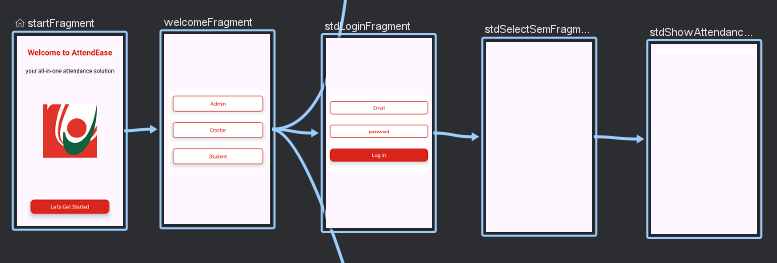
In the *DrAddAttendanceFragment*, a doctor can mark attendance for students enrolled in a particular material.

First, the fragment receives the material ID and session ID from the previous fragment. It then fetches existing attendance records and retrieves a list of student IDs associated with the chosen material.

Next, student data is retrieved, ordered by name, for students enrolled in the material. For each student, a checkbox is created and added to the UI, displaying the student's name and storing the student ID as a tag. Ideally, these checkboxes would be pre-checked if attendance already exists for a student in the current session.

Clicking the "Add Attendance" button prompts the code to iterate through all checkboxes. For each checked box, the student ID is retrieved, and attendance existence for that student in the current session is verified. If attendance isn't recorded yet, a new document with the session ID and the student ID is created and added to the attendance collection to mark the student's attendance for the created session.

### Student



The student login functionality utilizes a two-step verification process. Upon clicking the student's CardView, the application navigates to the *StdLoginFragment.* This fragment first validates user input by checking if both email and password fields are filled. If valid, it then queries the 'Students' collection to verify if the entered email exists. If the email is found, the fragment proceeds to confirm if the corresponding password stored in the database matches the user's input. Successful authentication grants access to the *StudentSelectSemFragment*.

In the *StudentSelectSemFragment,* the student can select their semester. The code retrieves a list of semesters from a collection named 'Semesters'. The code iterates through this list, creating a card with the semester name in the center. Tapping this CardView takes the student to a new fragment *StdShowAttendanceFragment*, passing along both the student's ID and the selected semester ID.

#### Show Attendances

The StdShowAttendanceFragment is responsible for displaying a student's attendance for a chosen semester. It achieves this by first retrieving the student's ID and the selected semester from the arguments passed to the fragment.

Then, it gets all the materials associated with the student in general. Next, it retrieves a list of materials offered in the chosen semester. For each material in the chosen semester, it checks if the student is enrolled in that material. If enrolled, it retrieves the number of lecture sessions offered for that material. Finally, it queries the attendance records to determine how many of those sessions the student attended.

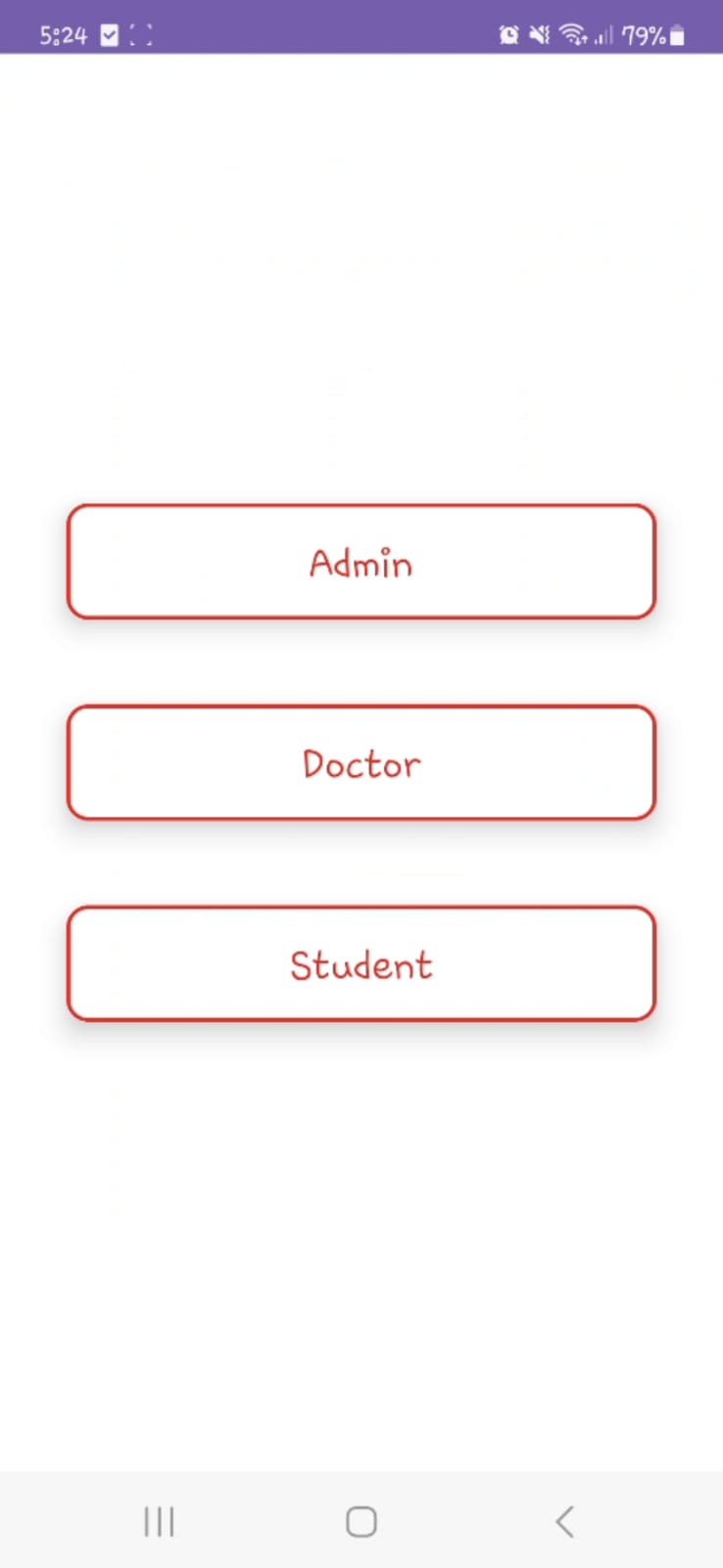
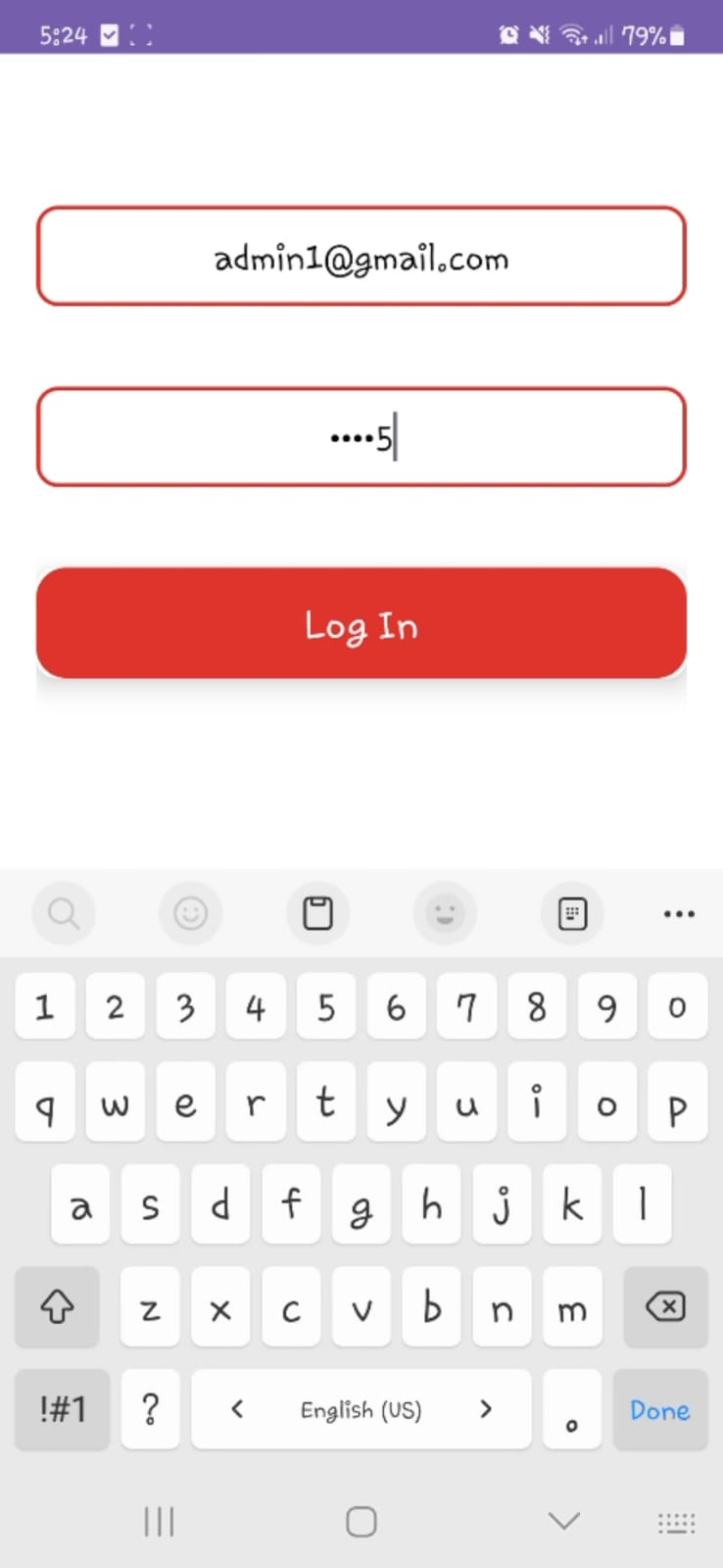
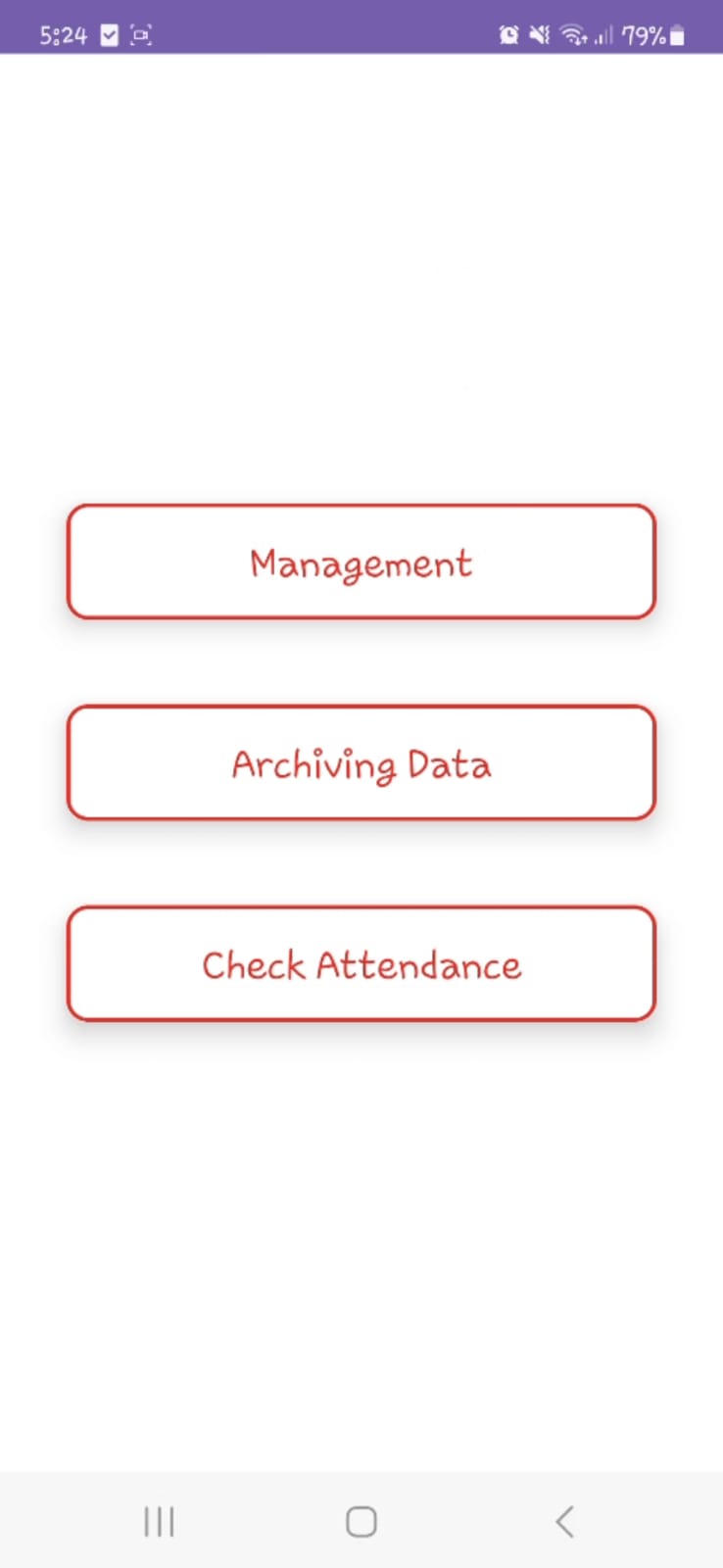
The code dynamically creates text views to display information about each material, including the name, code, number of offered sessions, and the student's attendance record . The text color is set green if attendance is over 50% and red otherwise.

# Chapter 5:use cases scenario

In this chapter, we will explore the use cases of our mobile application. Our application is designed to cater to three types of users: Admins, Doctors, and Students. Each user type has unique needs and interacts with the app in different ways, which we have carefully considered during the design and development process.

## Admin

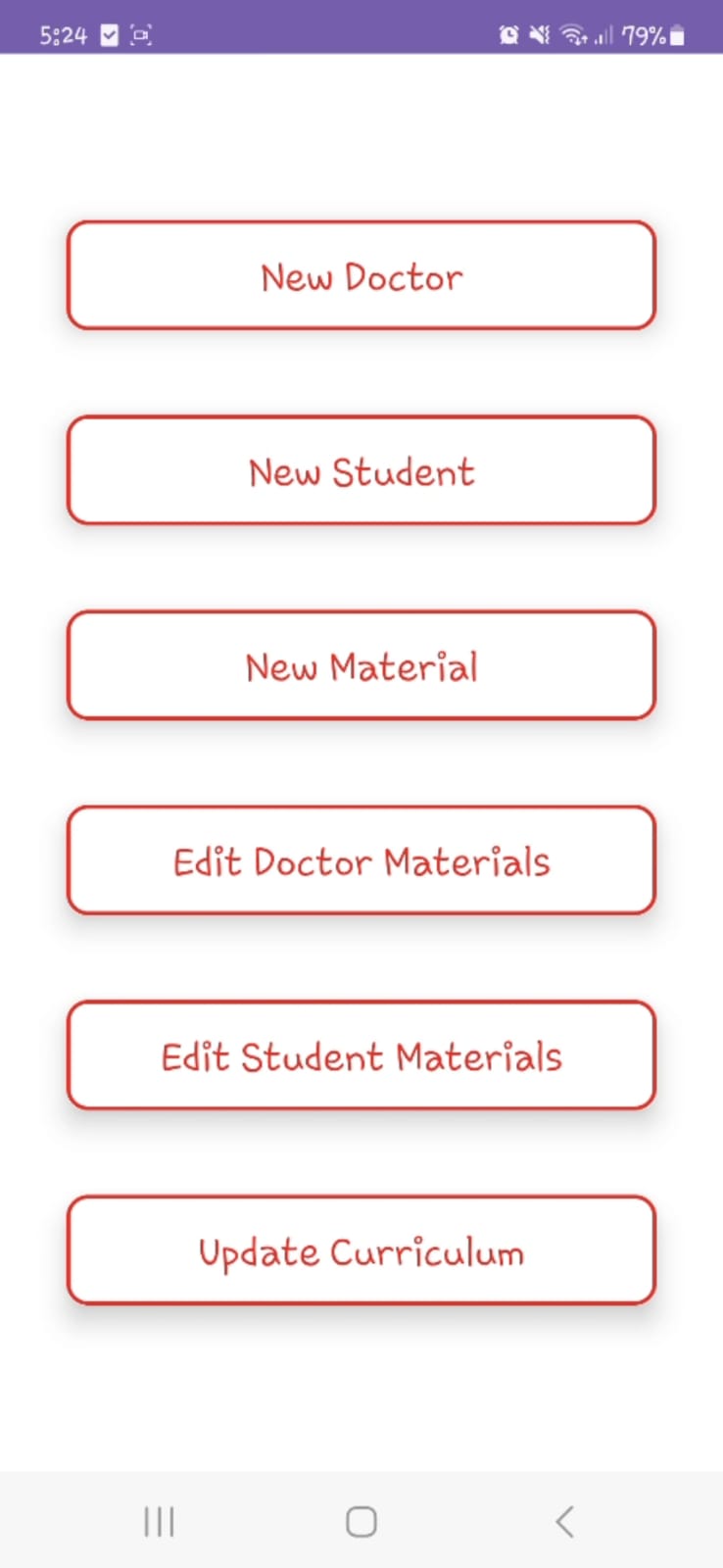
Admins are responsible for managing the system,archiving data,show attendances .

Upon successful authentication, the administrator gains access to the application. The authentication process involves the administrator entering their correct email and password. Once authenticated, the administrator is granted the ability to perform the following tasks:manage,archive,check attendances.

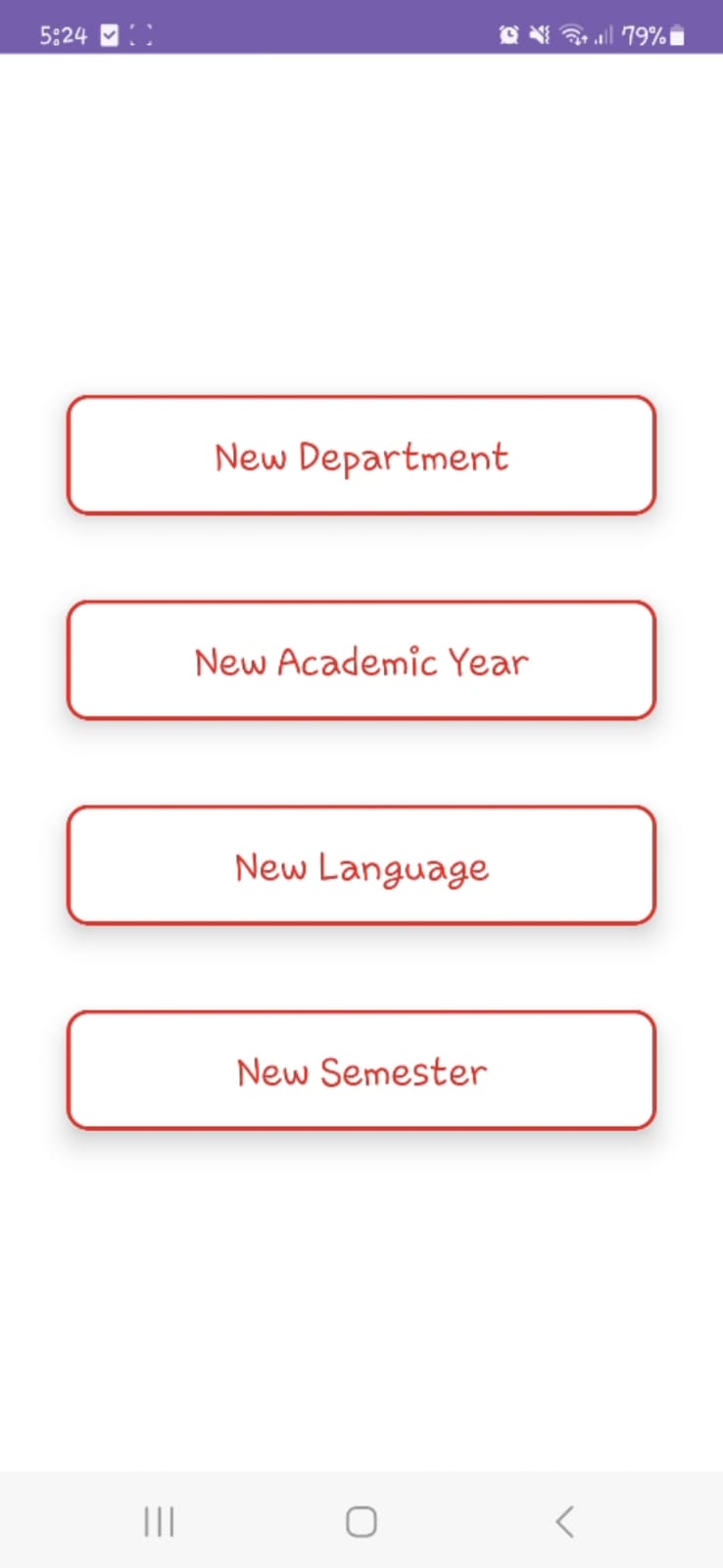
### Management

Upon selecting the Management option, the administrator is presented with a comprehensive suite of functionalities to effectively manage the system’s resources and users. These functionalities include:Adding a Student,Adding a Doctor,Adding Material,Editing Doctor Materials,Editing Student Materials,Updating Curriculum.



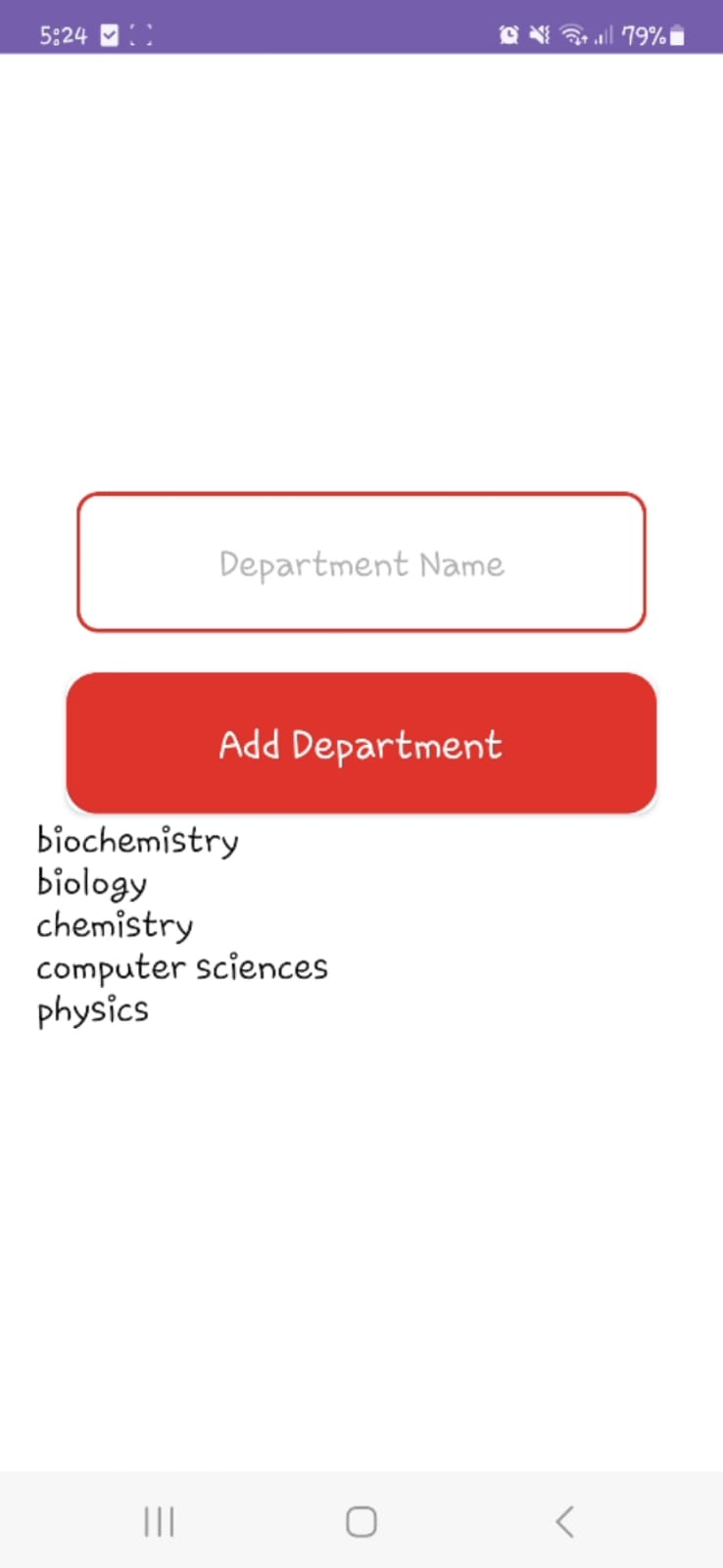
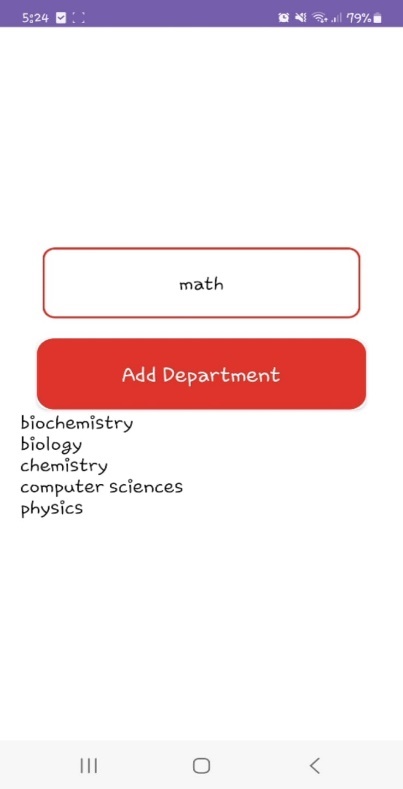
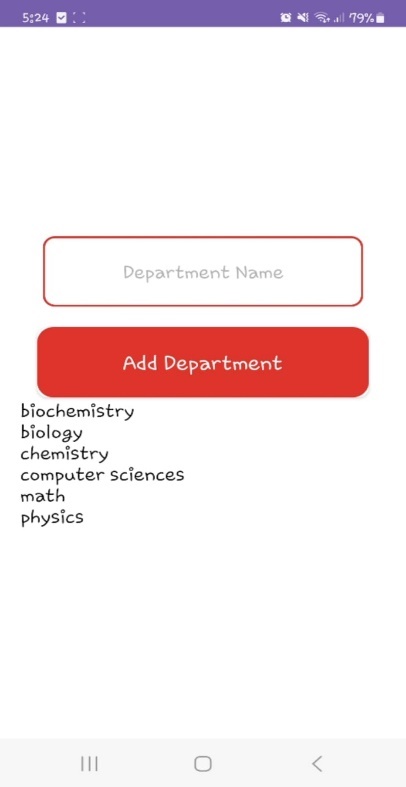
#### Update Curriculum

In this fragment the admin has many options:New Department,New Academic Year,New Lmaguage,New Semester.



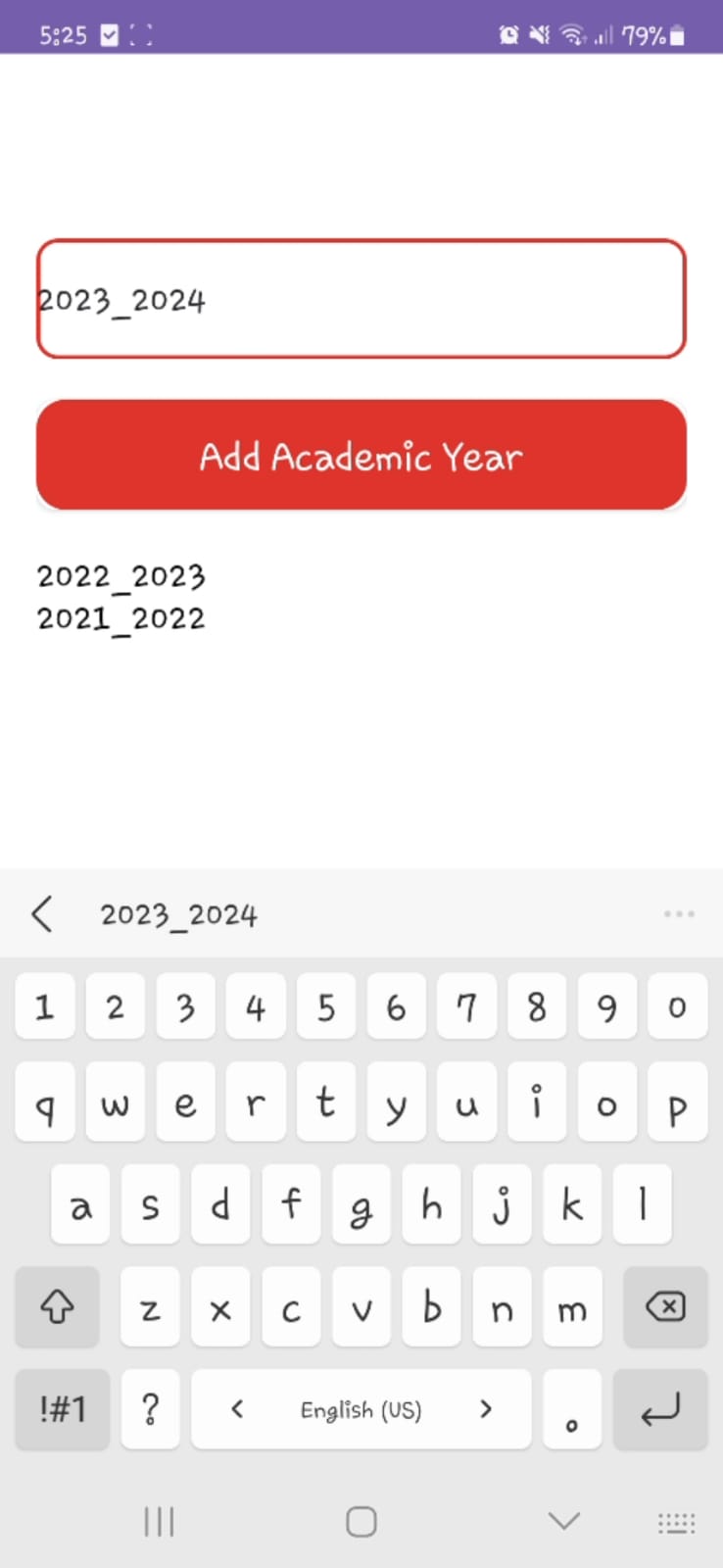
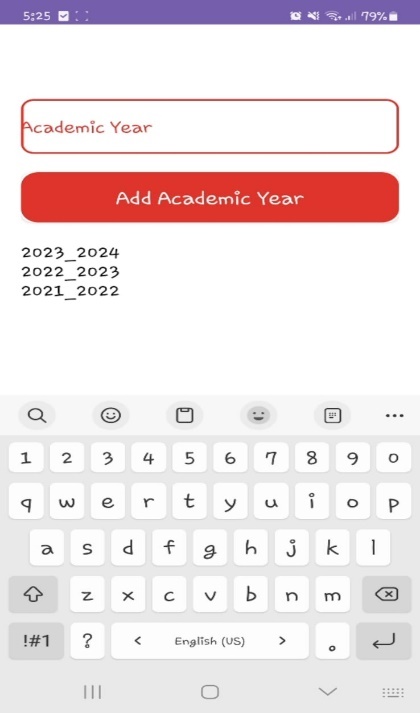
##### New Department

The admin navigates to the department management section of the application.In this section, there is an input field (EditText) provided for adding new departments.The admin enters the name of the new department, “Math”, into this EditText field.Upon entering the department name, the admin proceeds to confirm the action by clicking on the ‘Add Department’ button.The system then processes this input, it adds “Math” to the department list.

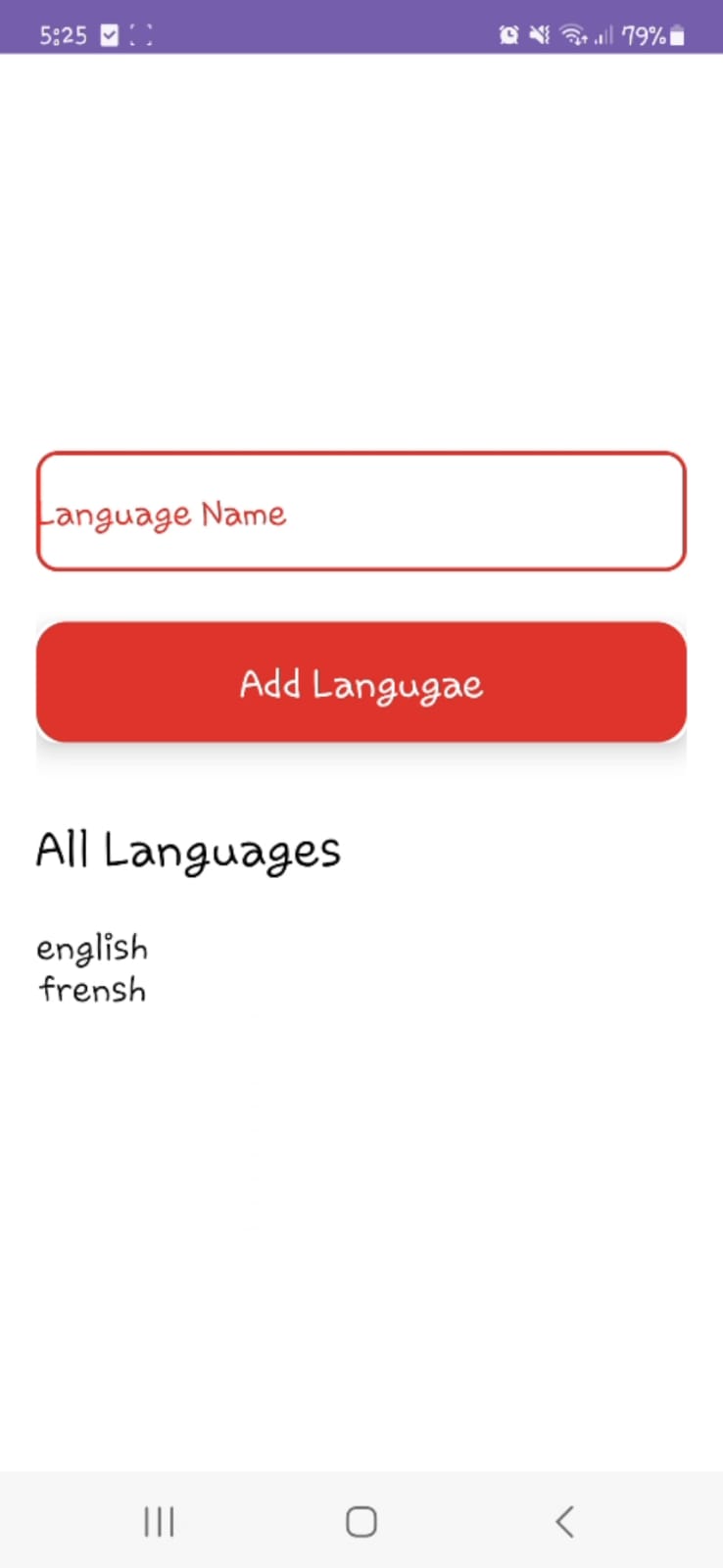
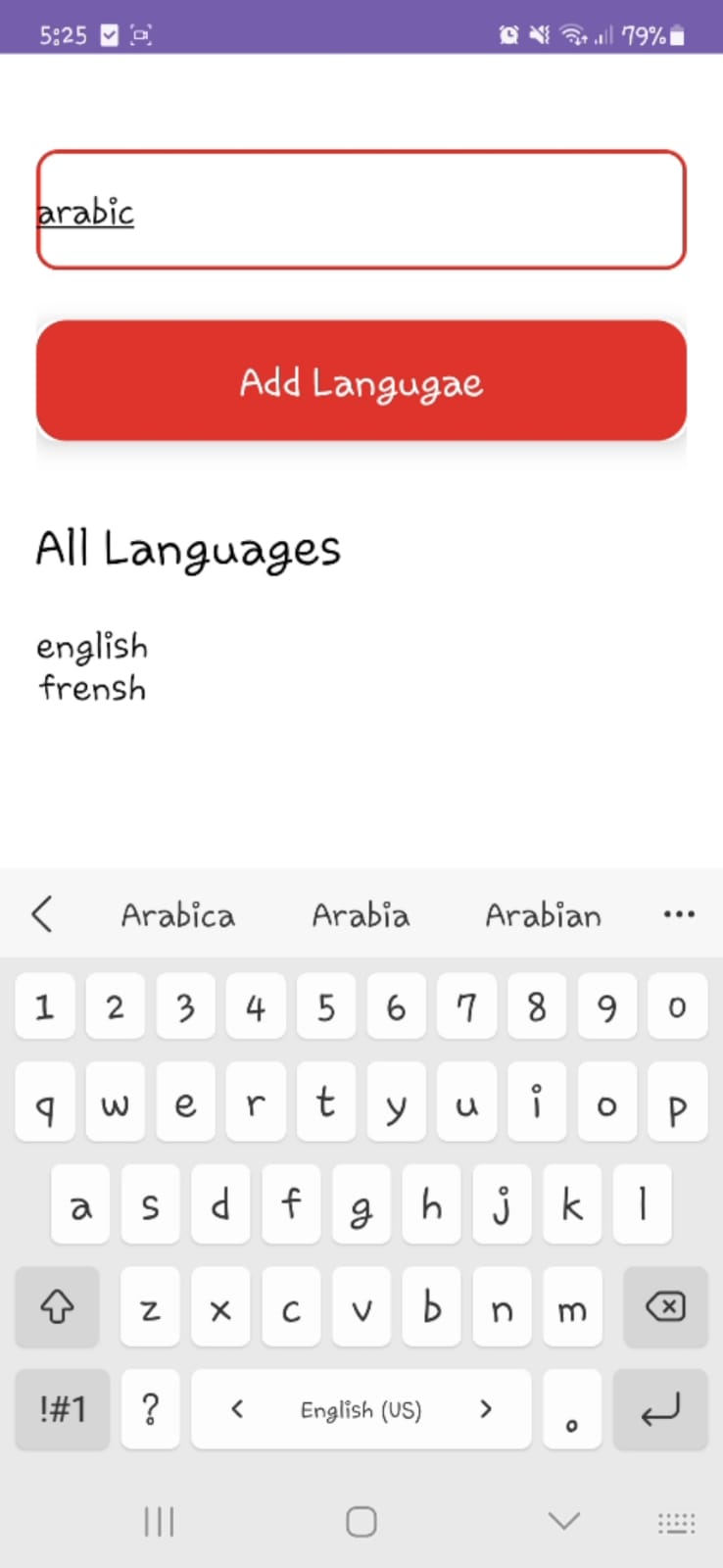
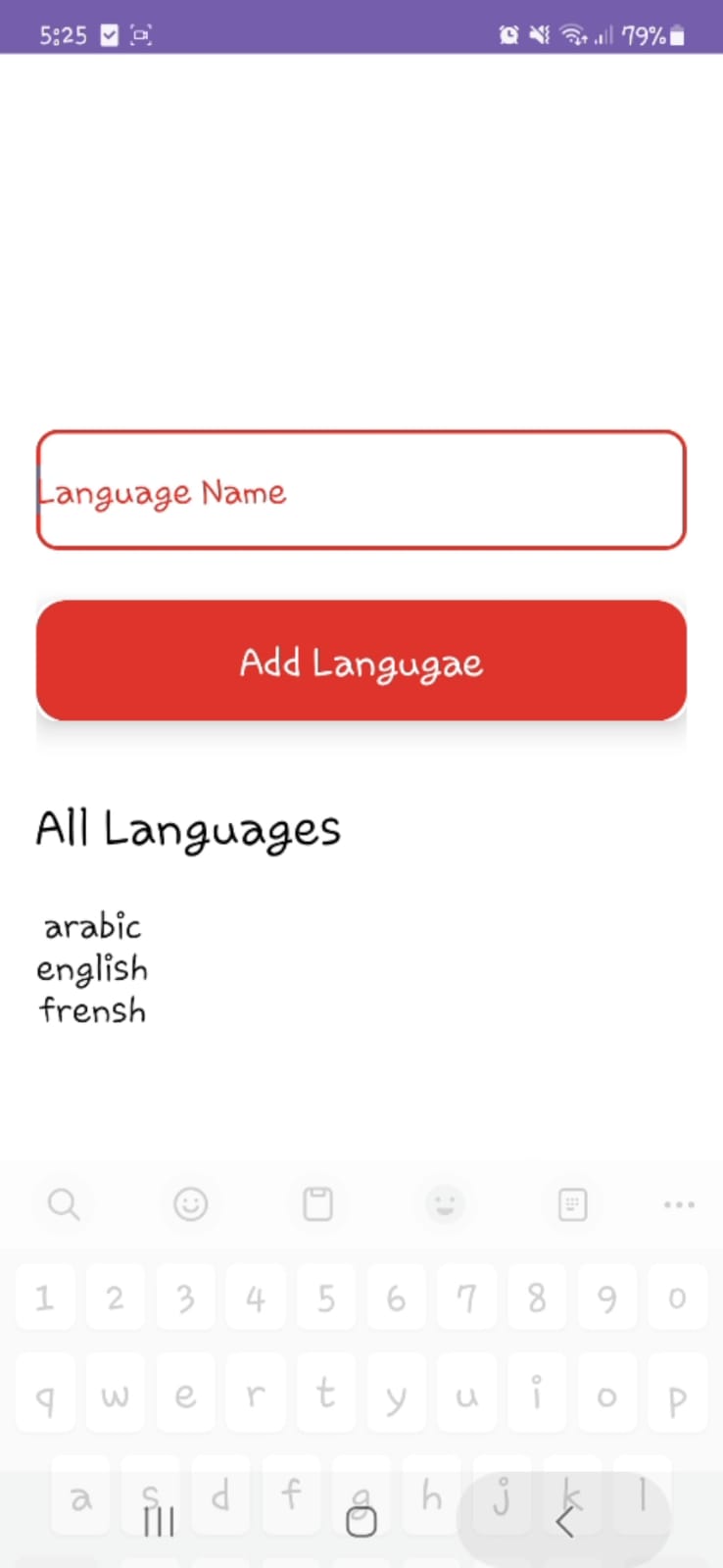
##### New academic Year

The admin navigates to the academic year management section of the application.In this section, there is an input field (EditText) provided for adding new academic year.The admin enters the name of the new academic year, “2023\_2024”, into this EditText field.Upon entering the academic year name, the admin proceeds to confirm the action by clicking on the ‘Add Academic Year’ button.The system then processes this input, it adds “2023\_2024” to the academic year list.

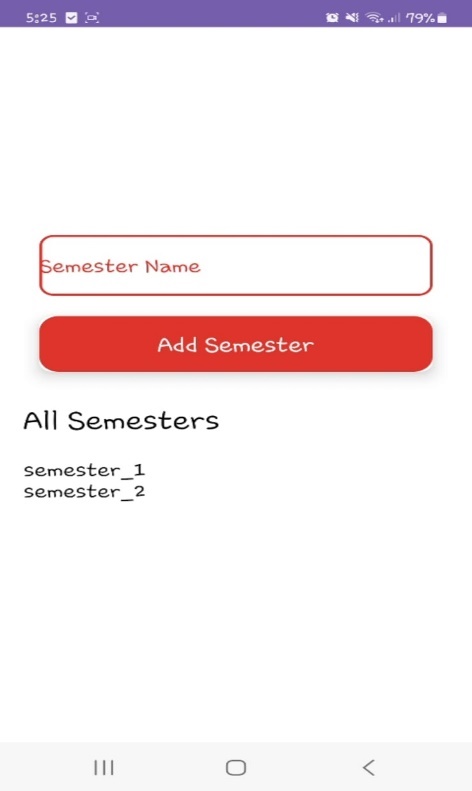
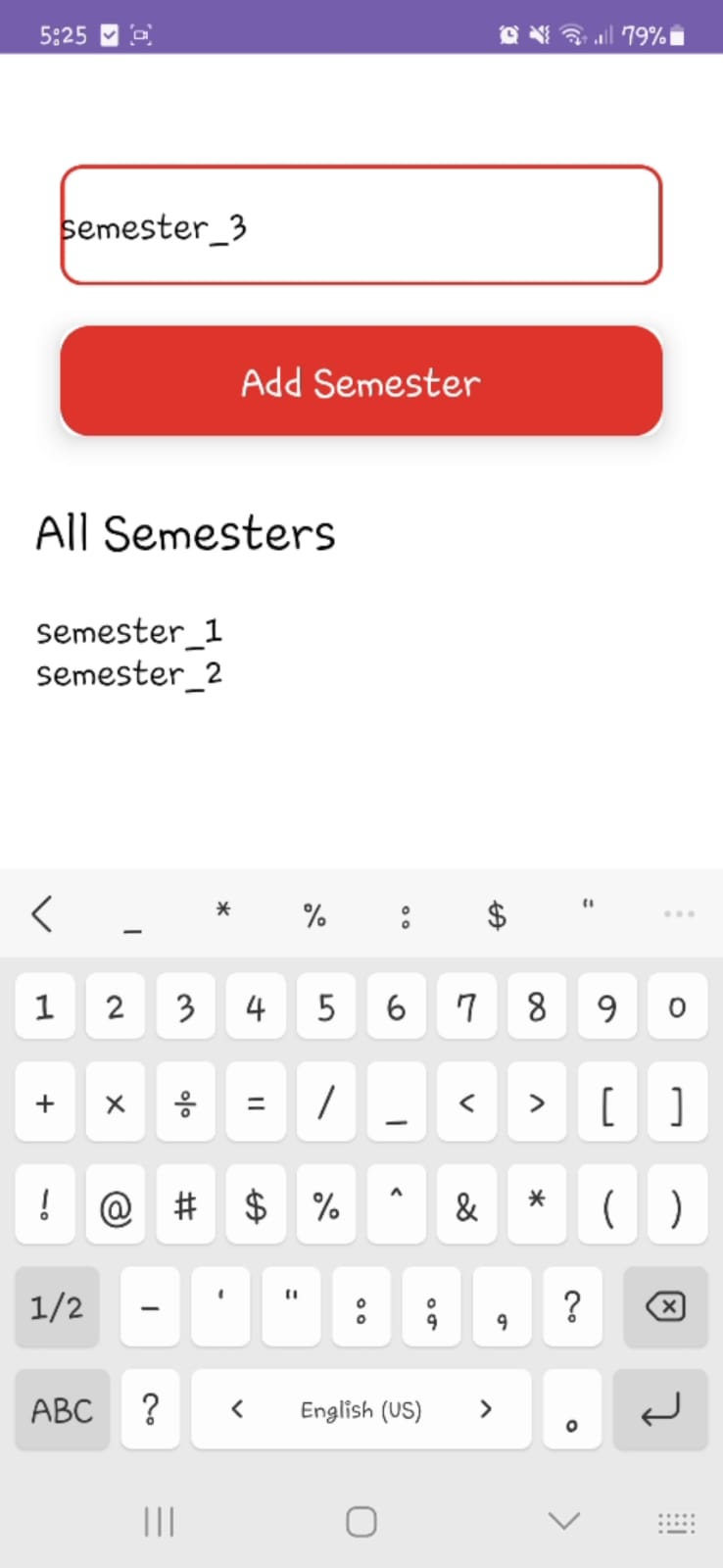
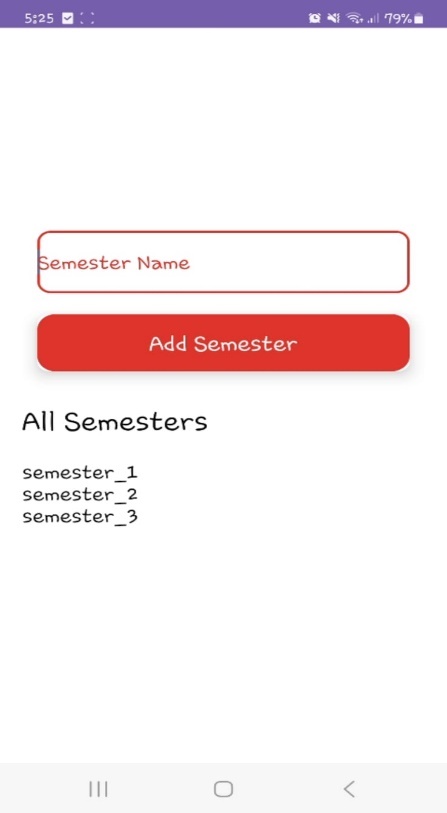
##### New Language

The admin navigates to the language management section of the application.In this section, there is an input field (EditText) provided for adding new languages.The admin enters the name of the new language, “arabic”, into this EditText field.Upon entering the department name, the admin proceeds to confirm the action by clicking on the ‘Add Language’ button.The system then processes this input, it adds “arabic” to the language list.

##### New Semester

The admin navigates to the semester management section of the application.In this section, there is an input field (EditText) provided for adding new semester.The admin enters the name of the new semester, “semester\_3”, into this EditText field.Upon entering the department name, the admin proceeds to confirm the action by clicking on the ‘Add Semester’ button.The system then processes this input, it adds “semester\_3” to the semester list.

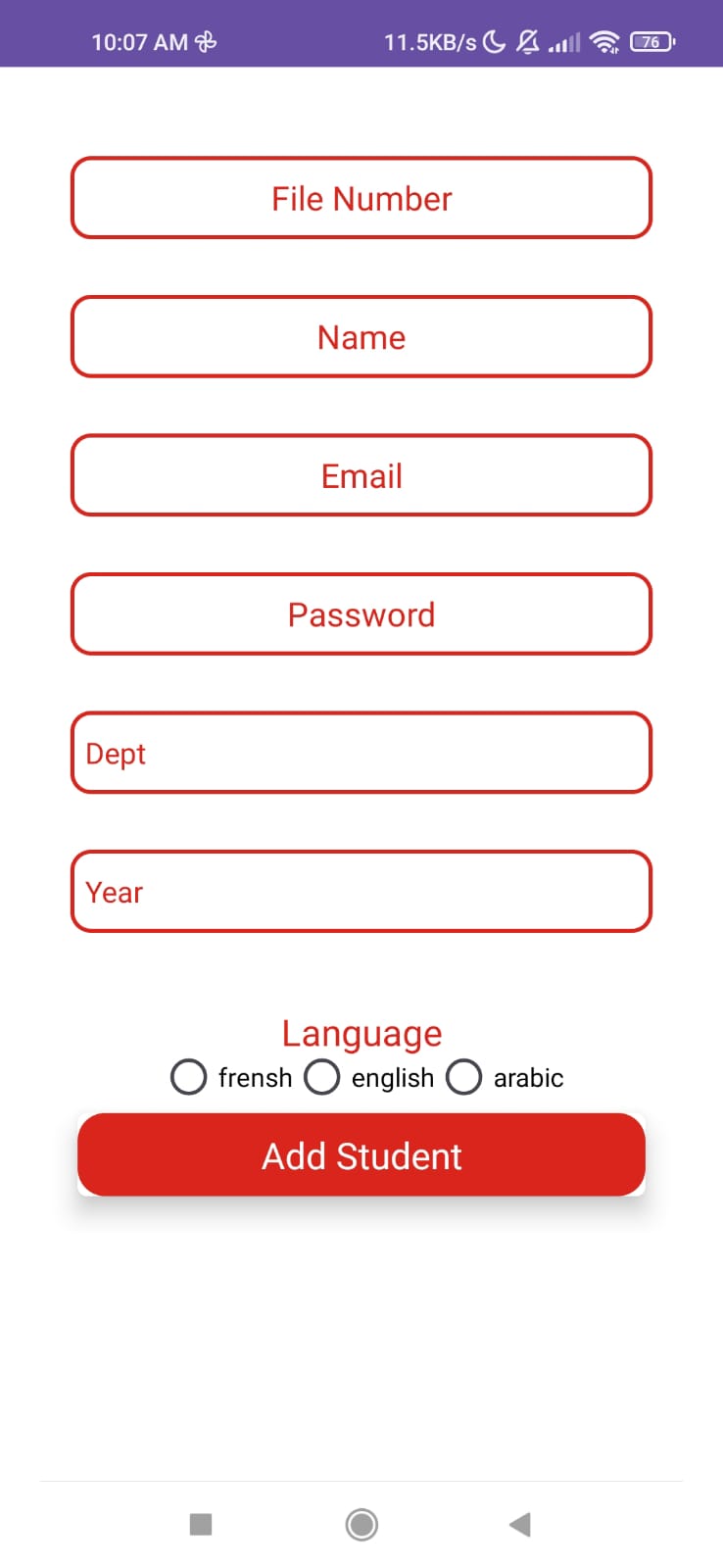
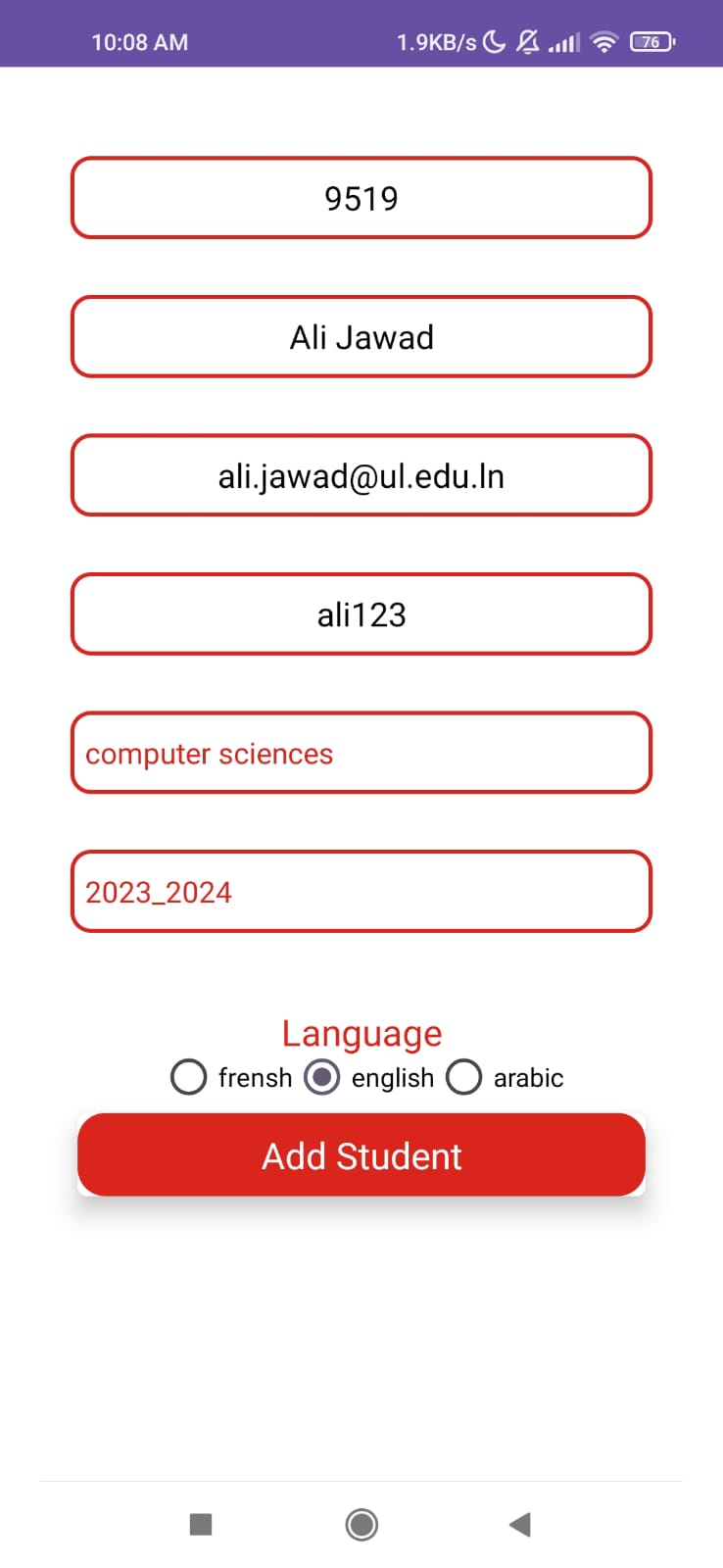
#### Add material

The administrator begins by adding the required data to add a new material.He write the material name and code ,then select the department,the language,the semester,then click the button to add the material.

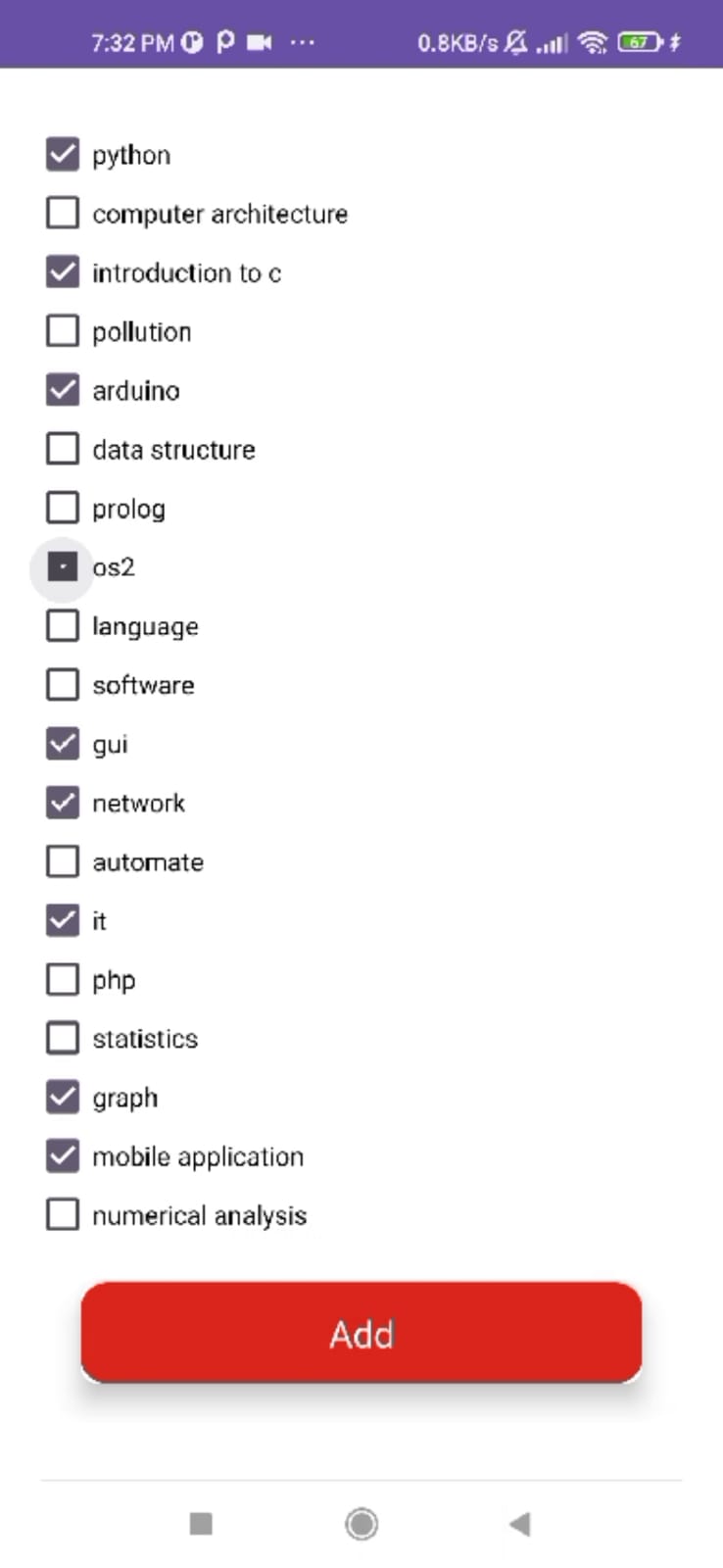
#### 

#### Add Student

The administrator begins by adding the required data to add a new student.He write the material file number,name,email and password ,then select the department,the language,the academic year,then click the button to add the student.

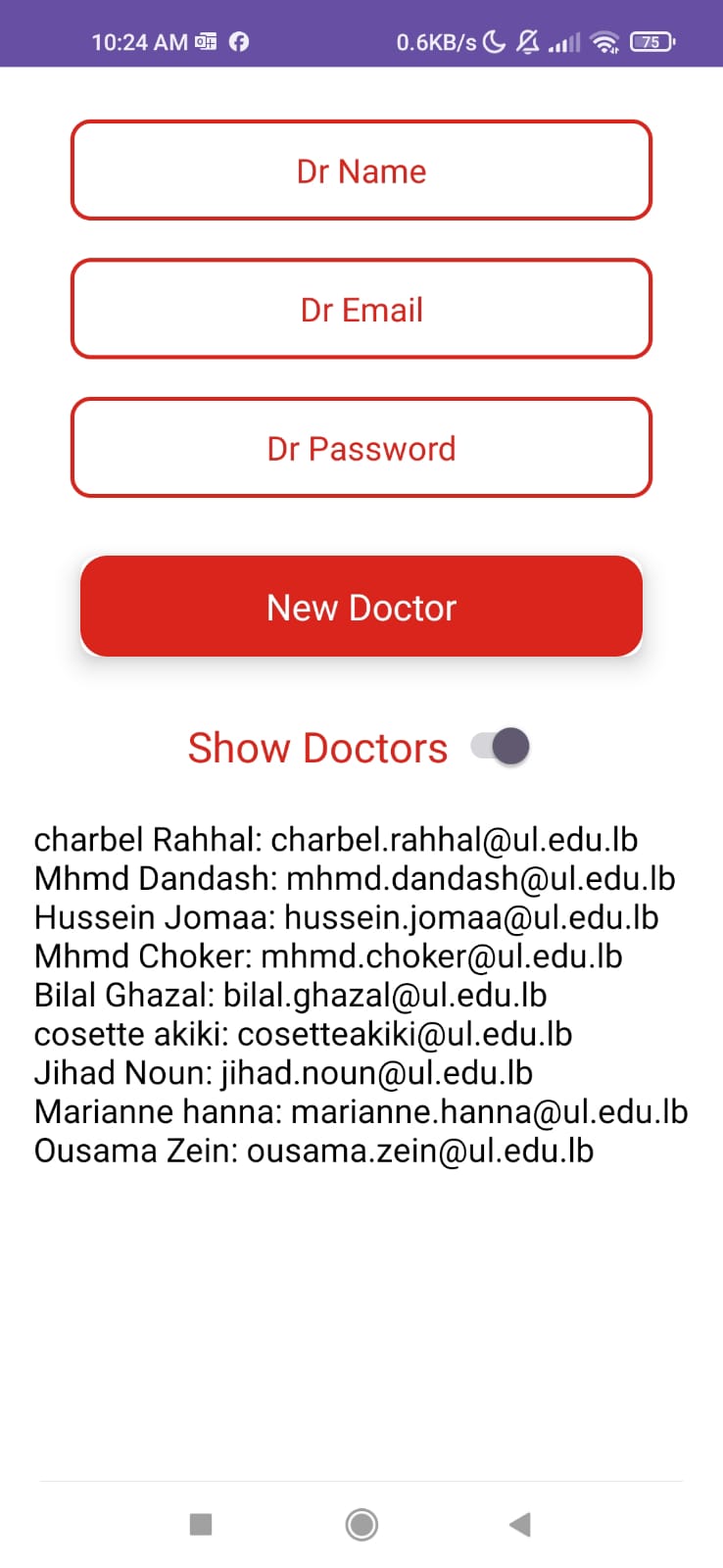
 

After adding the student ,the admin should add materials to this student by cheking the materials and then click “Add” button.

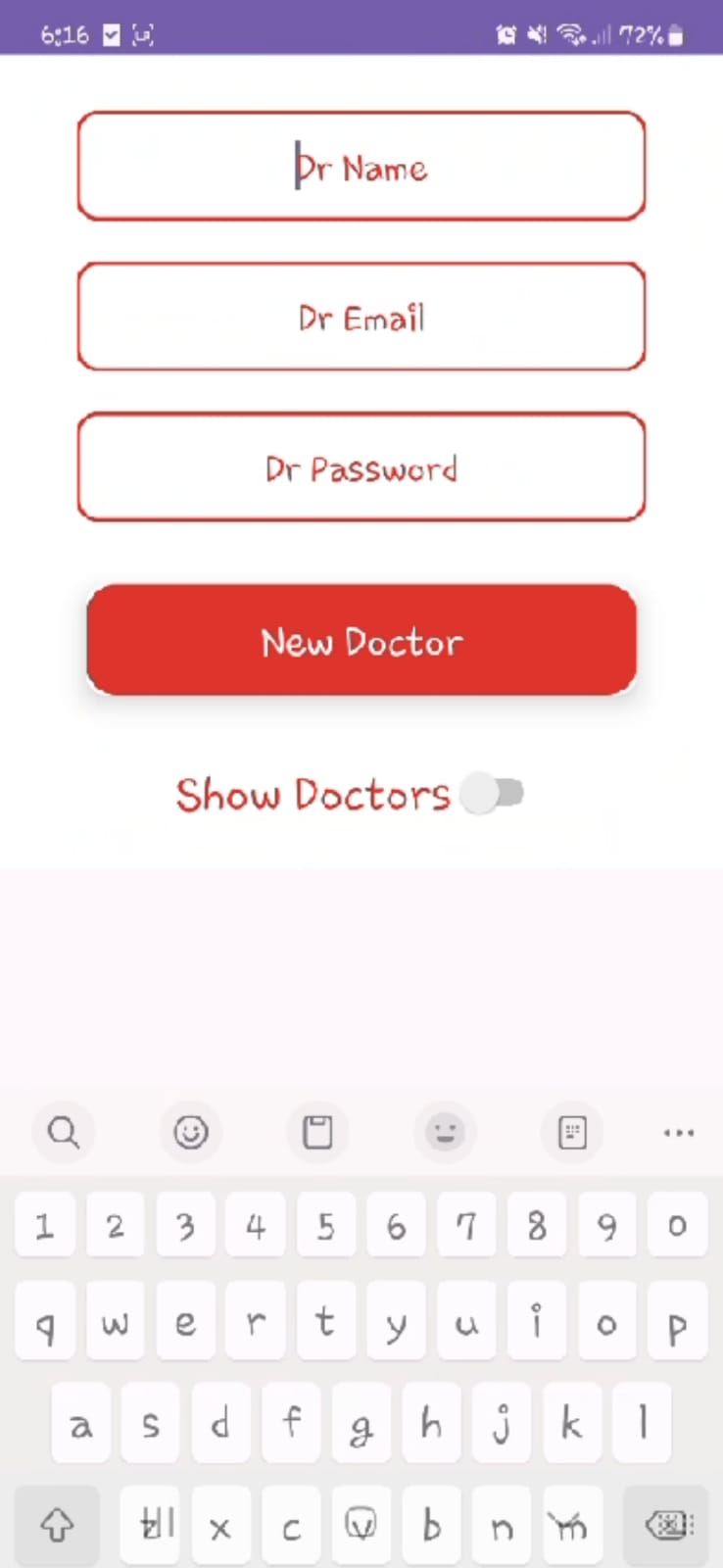
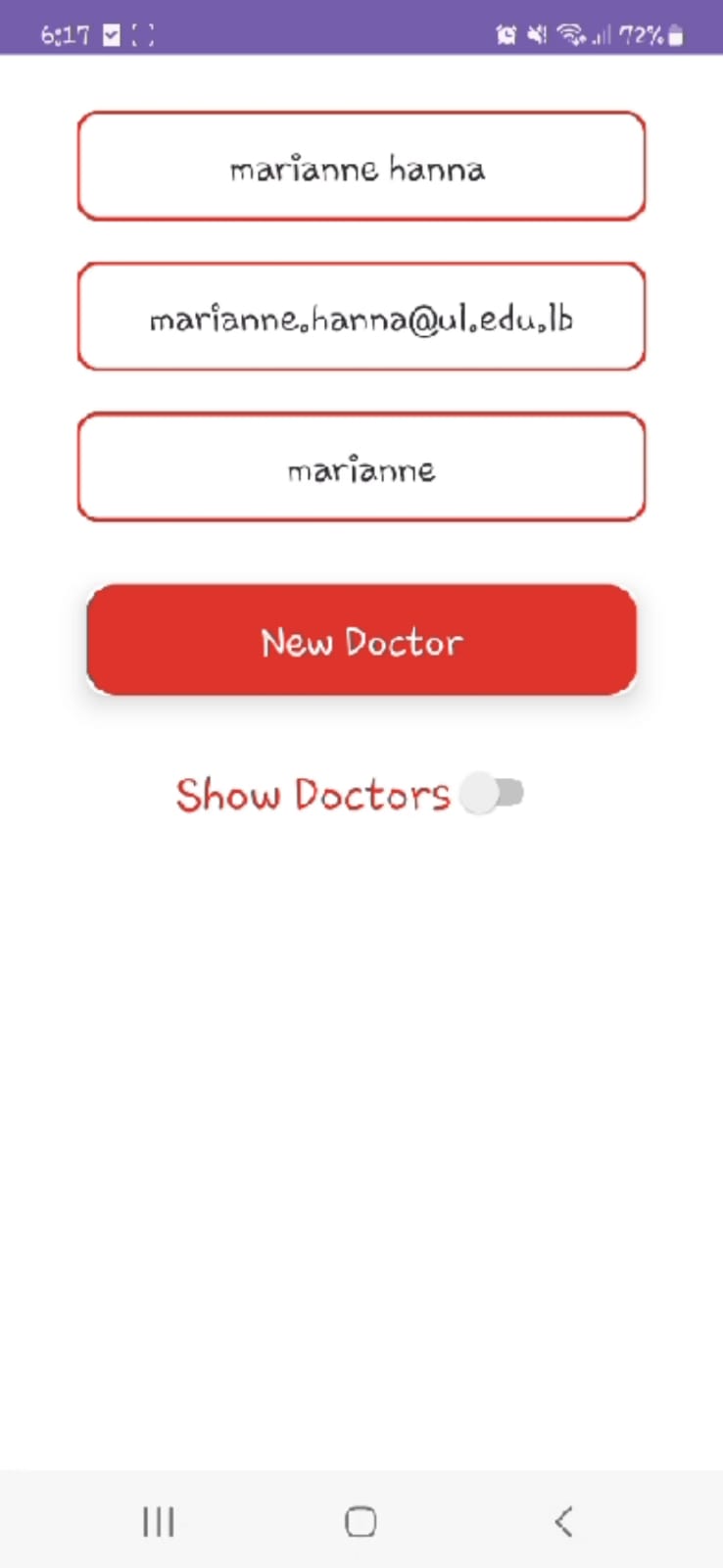
 

#### add doctor

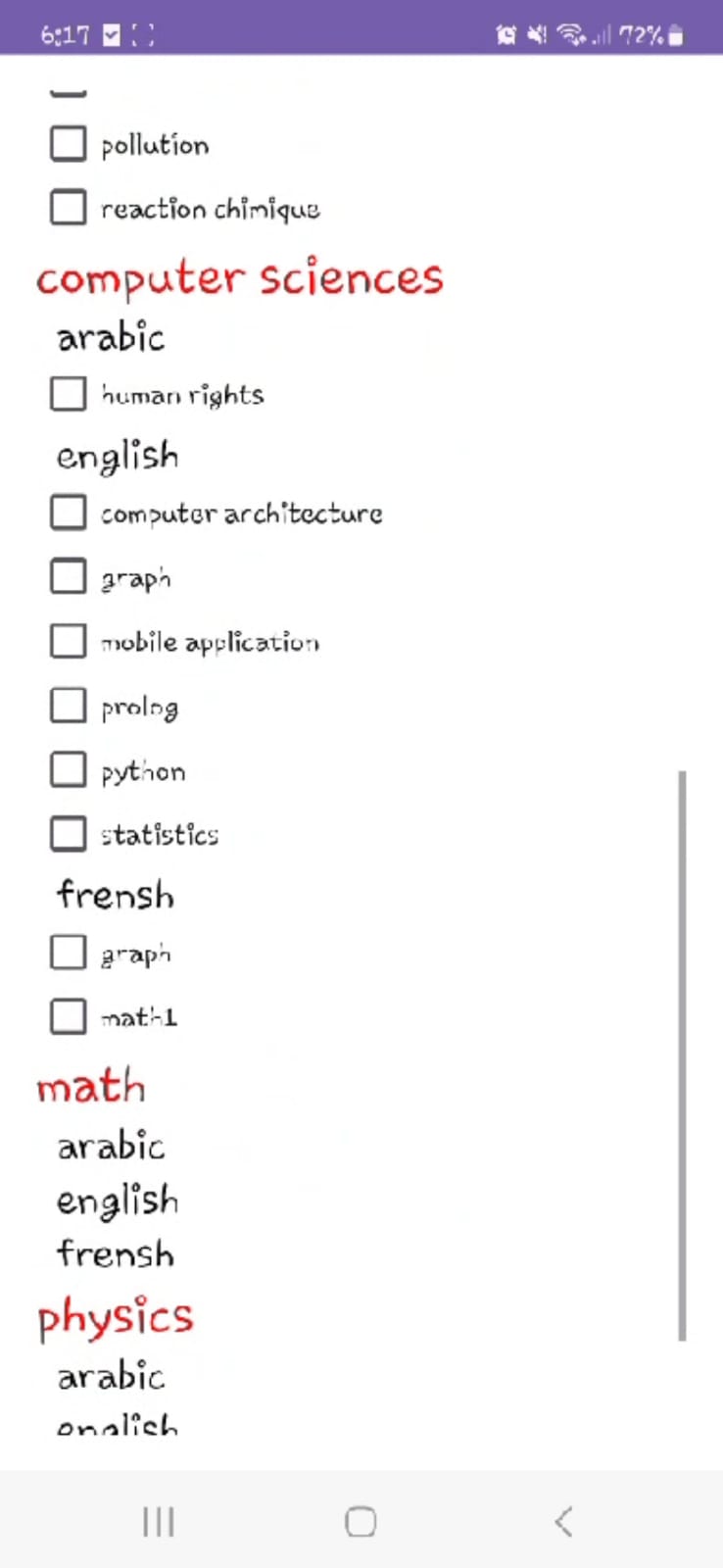
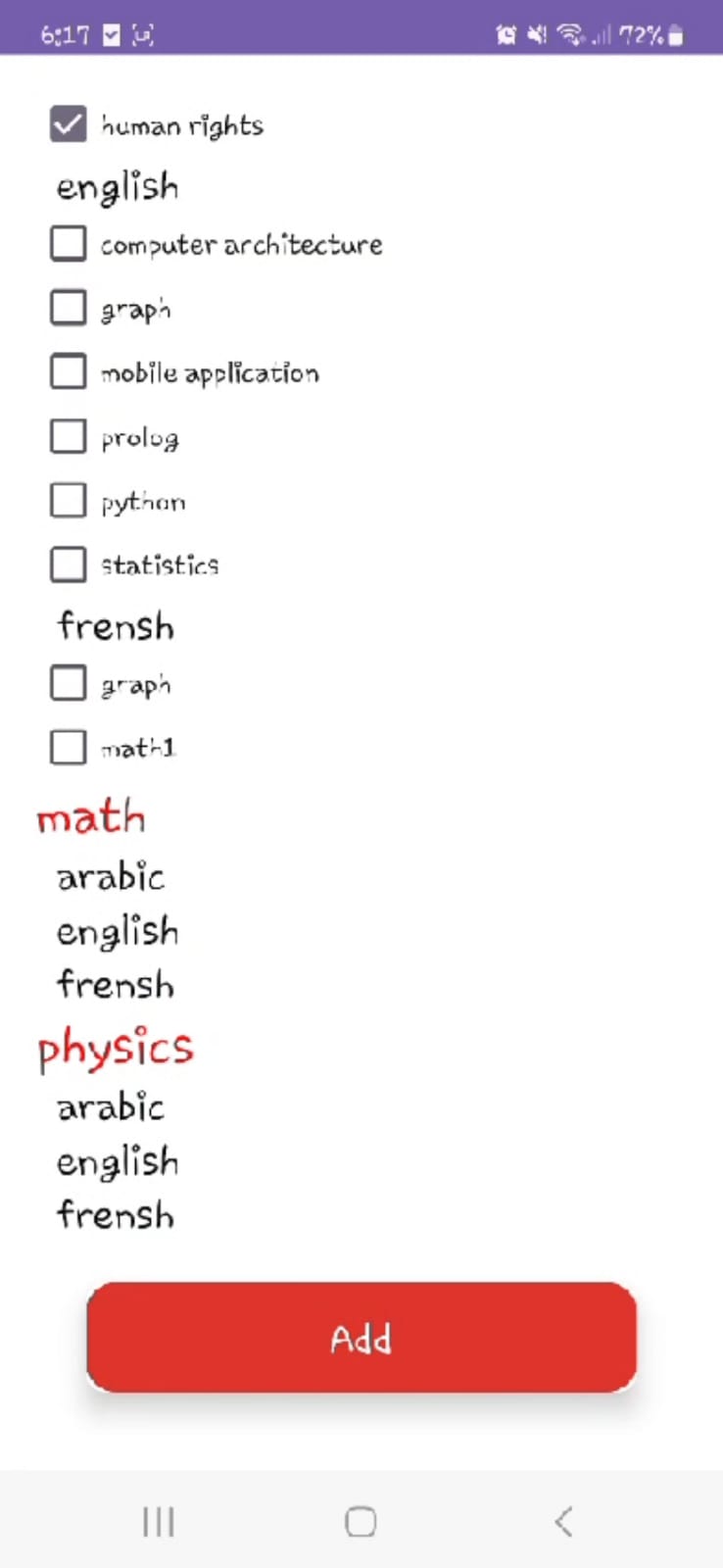
The admin can see all the doctor in the faculty



He begins to write name,email and password of the doctor,then click the button “New Doctor”.

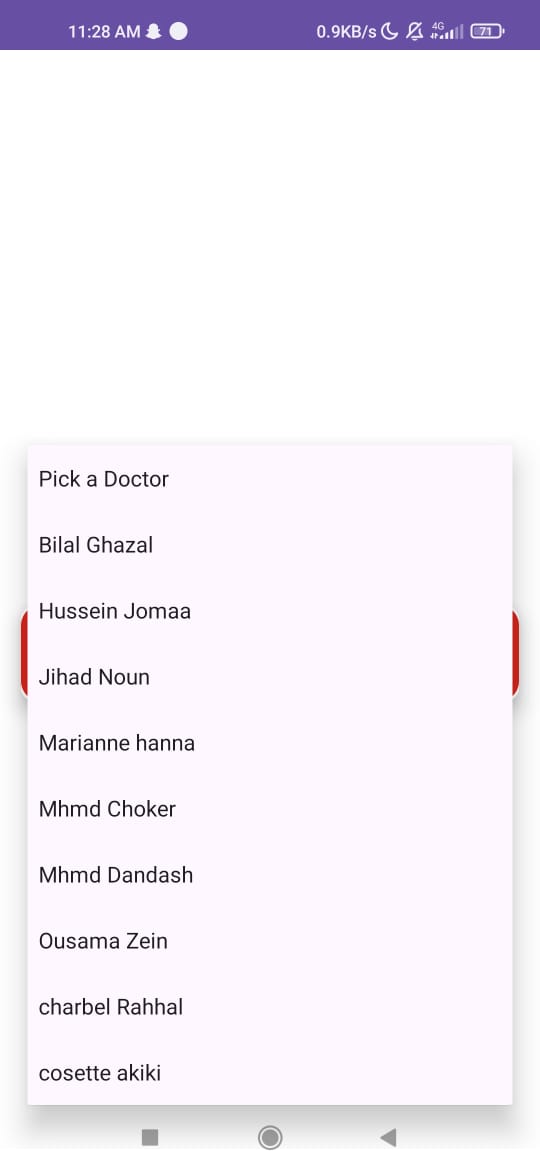
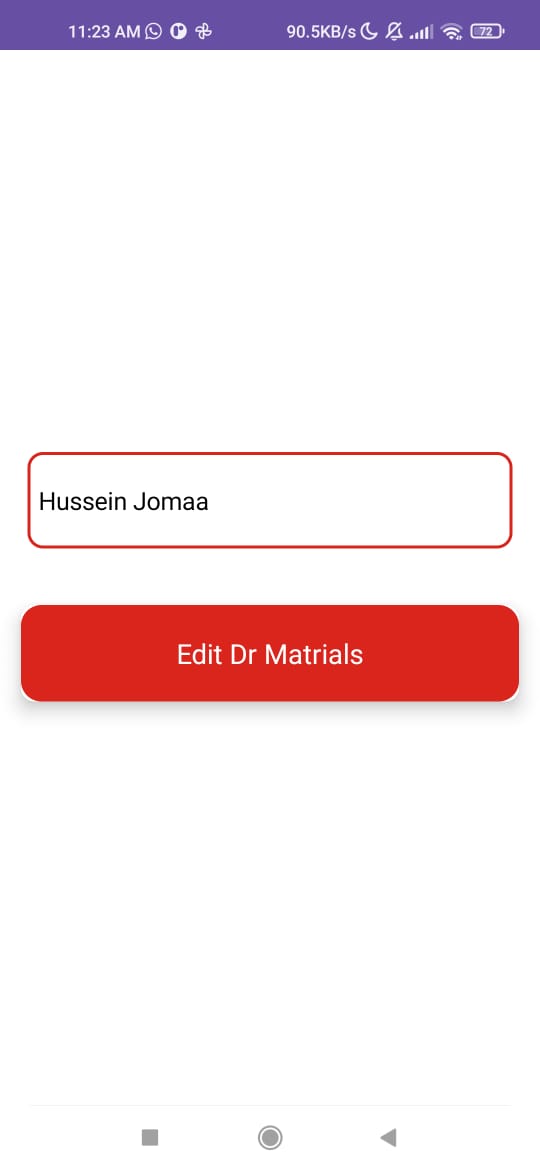
 

After adding the doctor,the admin should add materials to the doctor by checking each matrial that he want to register

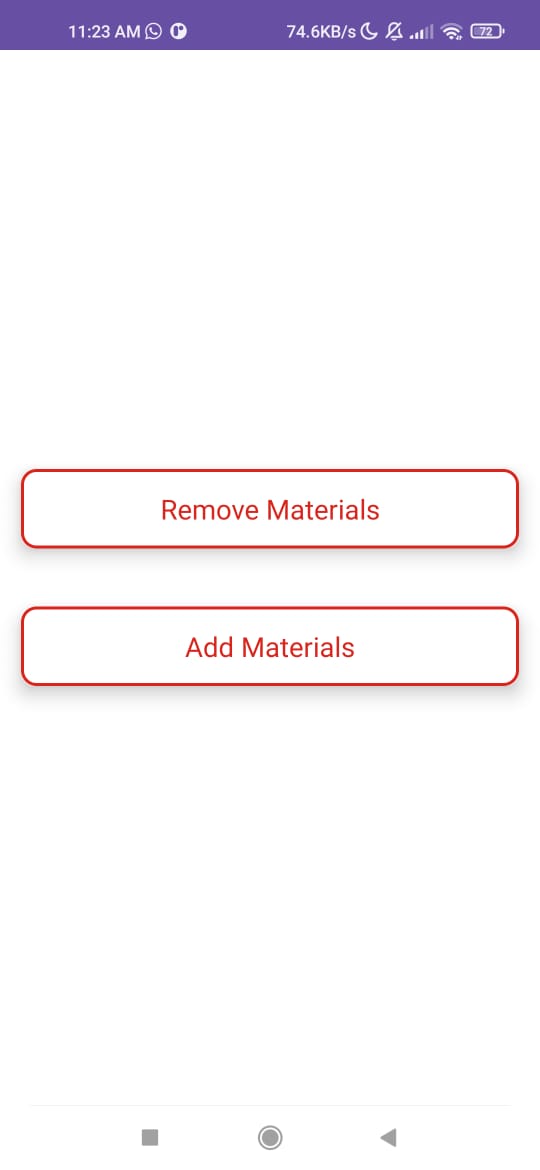
 

#### edit doctor materials

When the admin select the “edit doctor materials” option,he should select the doctor name from the spinner and then click the button “Edit Dr Materials” to start editing the materials of the selected doctor.

The admin now select what he want to do



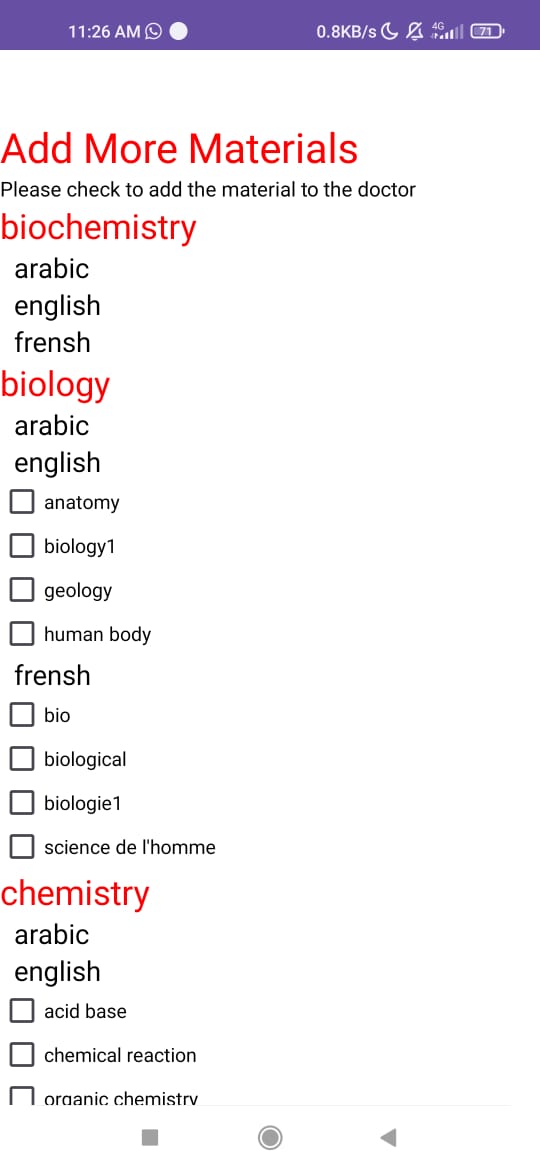
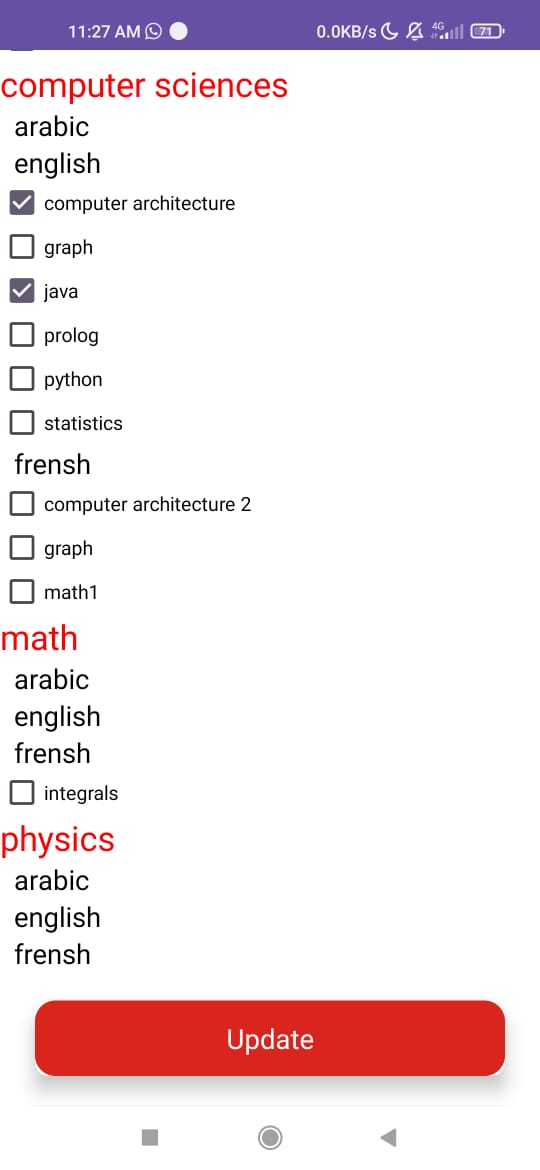
##### remove materials

The admin now uncheck the materials that he want to remove from the doctor

##### Add Materials

The admin now check the materials that he want to add to the doctor

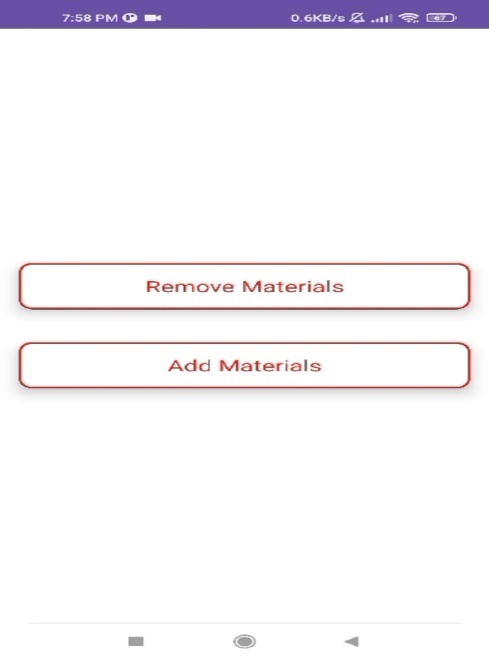
 

#### edit student materials

When the admin select the “edit doctor materials” option,he should select the departement,language,yearof the student ,that he want to edit his materials, from the spinners and then click the button “Show Students” to show the available students with the selected critiria.And them,check the radio button for the student whose materials he want to modify.

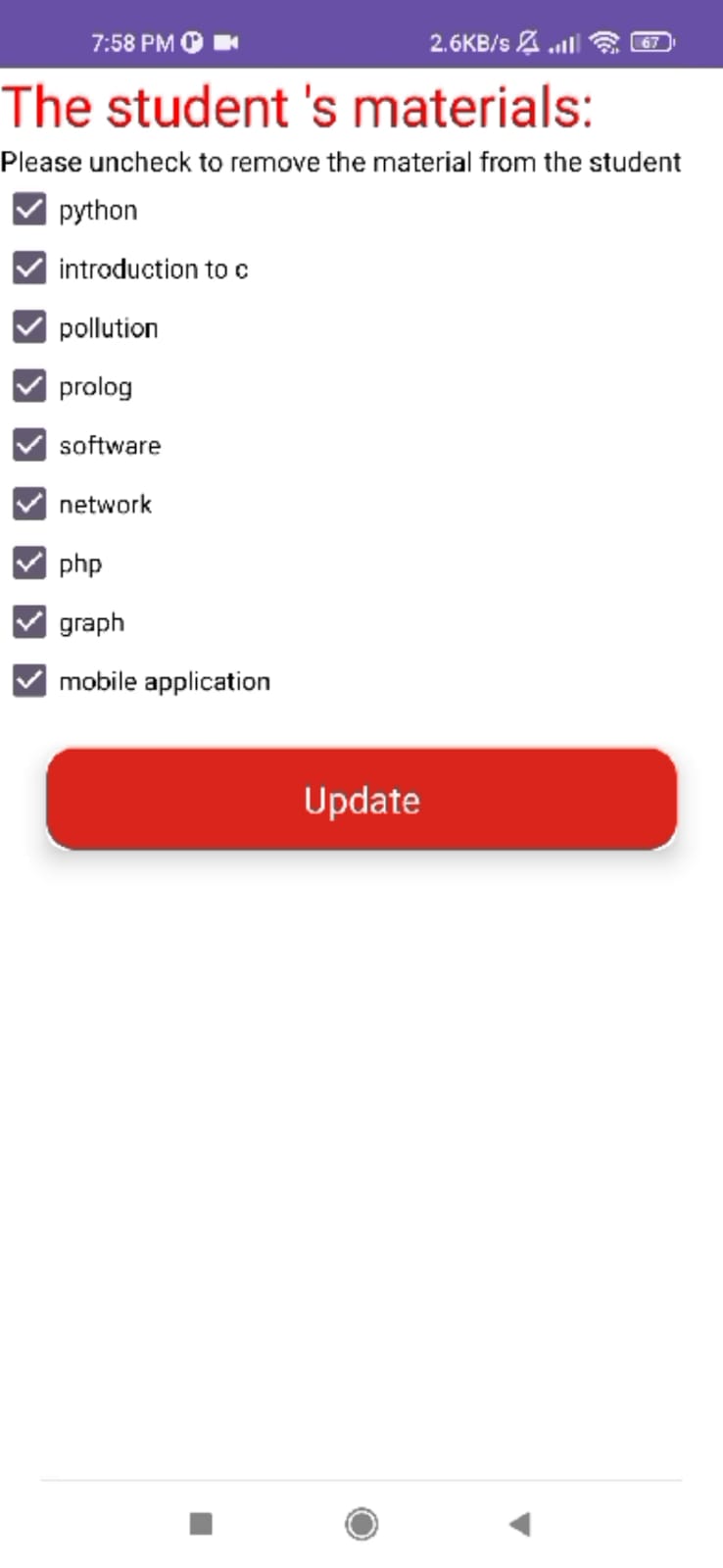
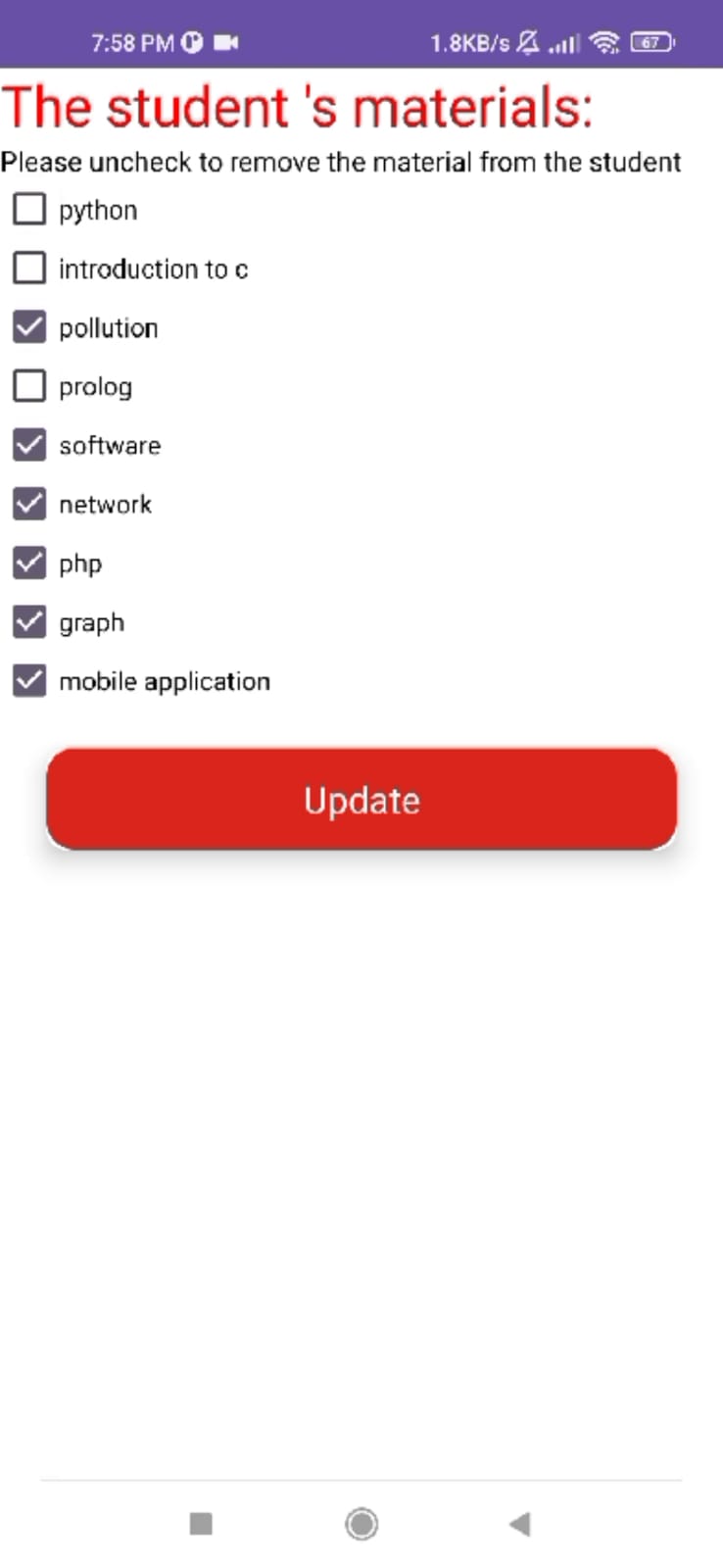
#### 

The admin now select what he want to do



##### remove materials

The admin now uncheck the materials that he want to remove from the student

##### Add Materials

The admin now check the materials that he want to add to the student

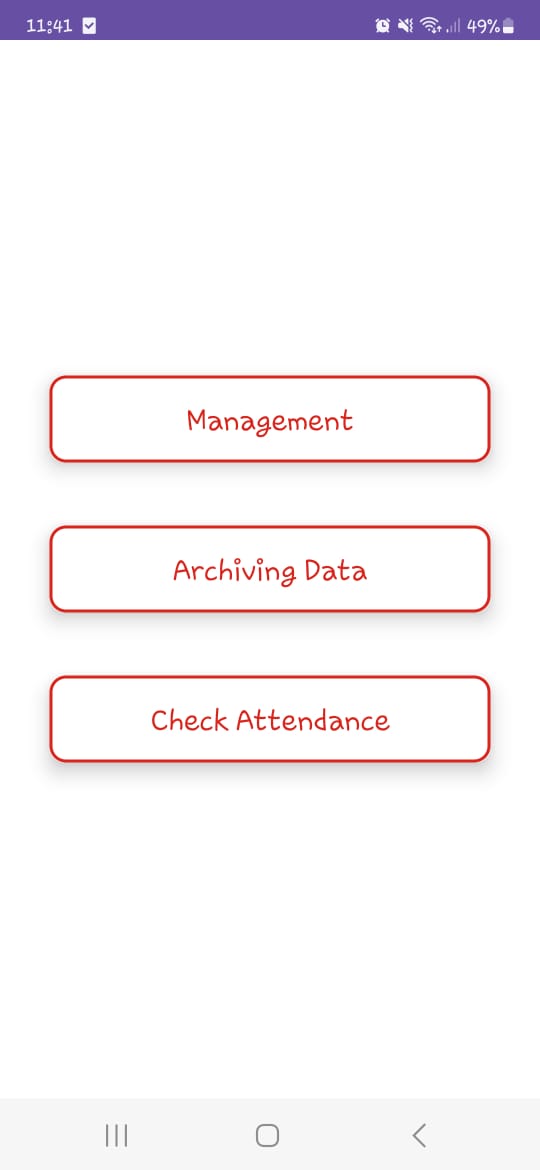
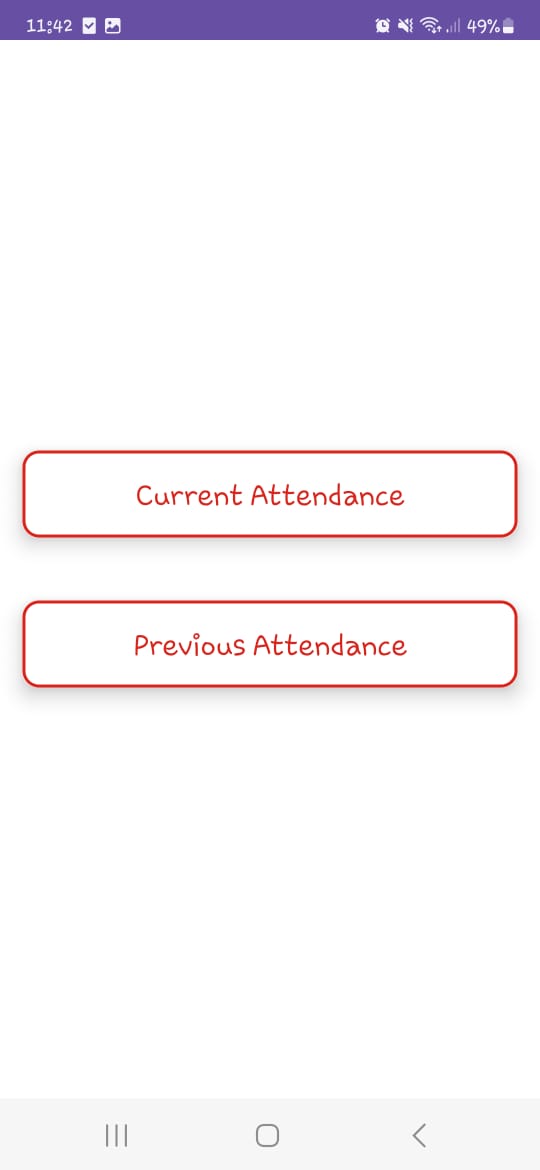
### Archiving Data

Upon selecting the Management option, the administrator will archive now the data of a selected year.

#### 

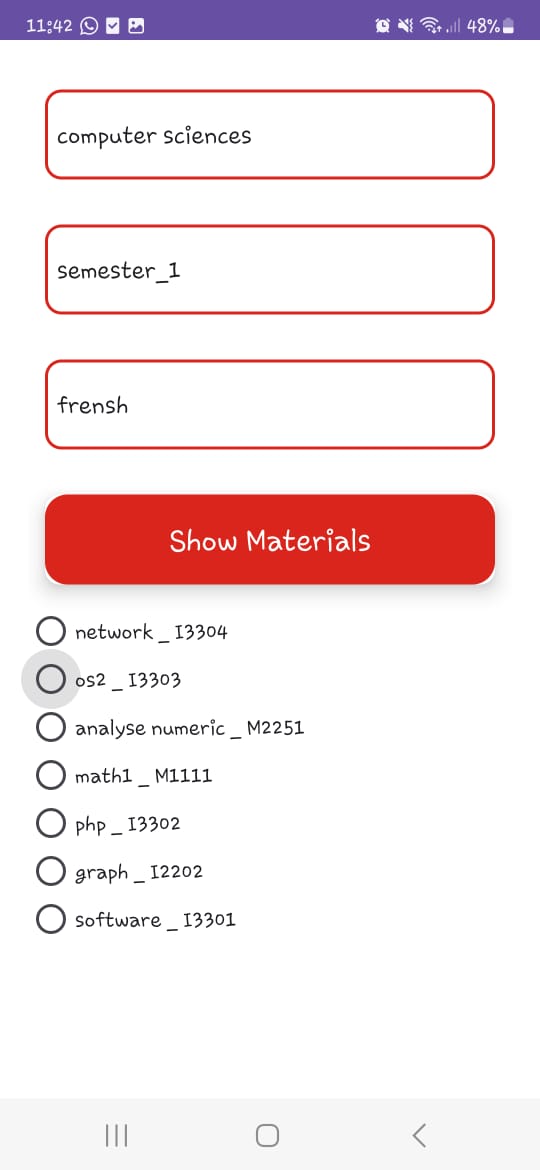
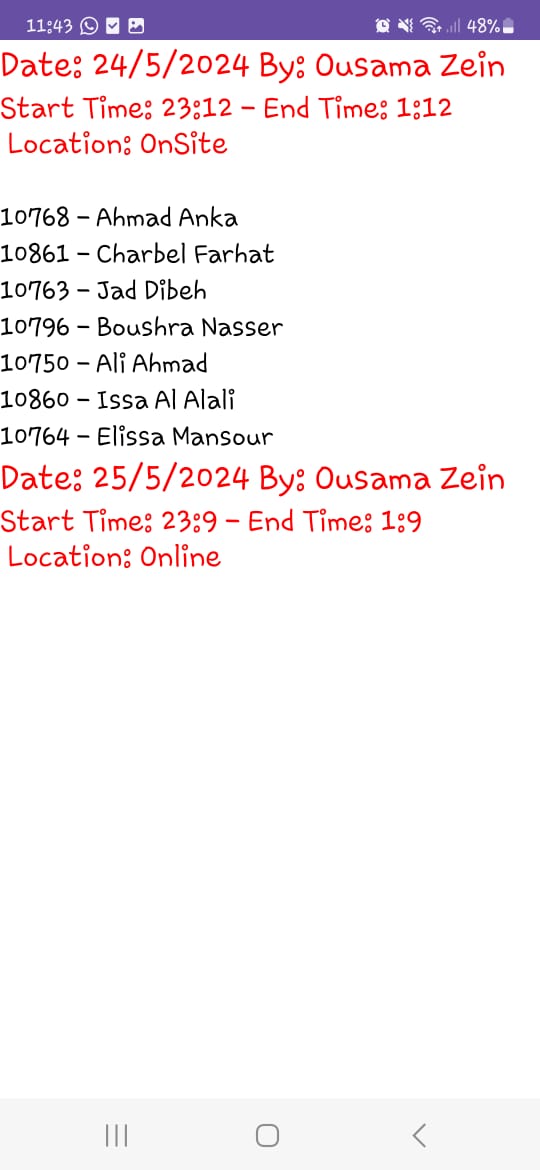
The admin select the year from the spinner ,then click on the button “Archive” to mode the data to archive

### check attendance

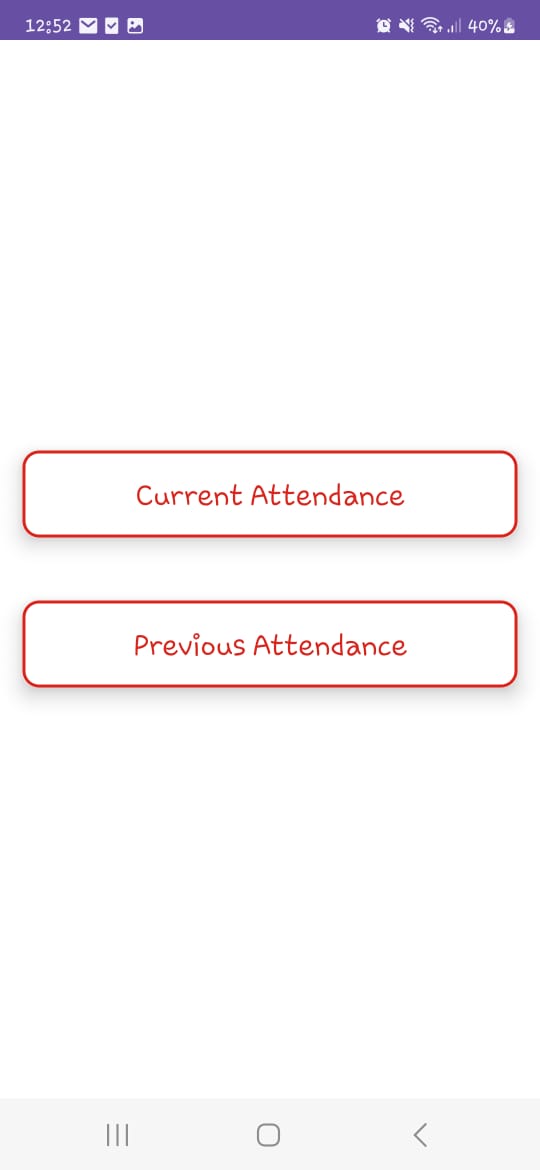
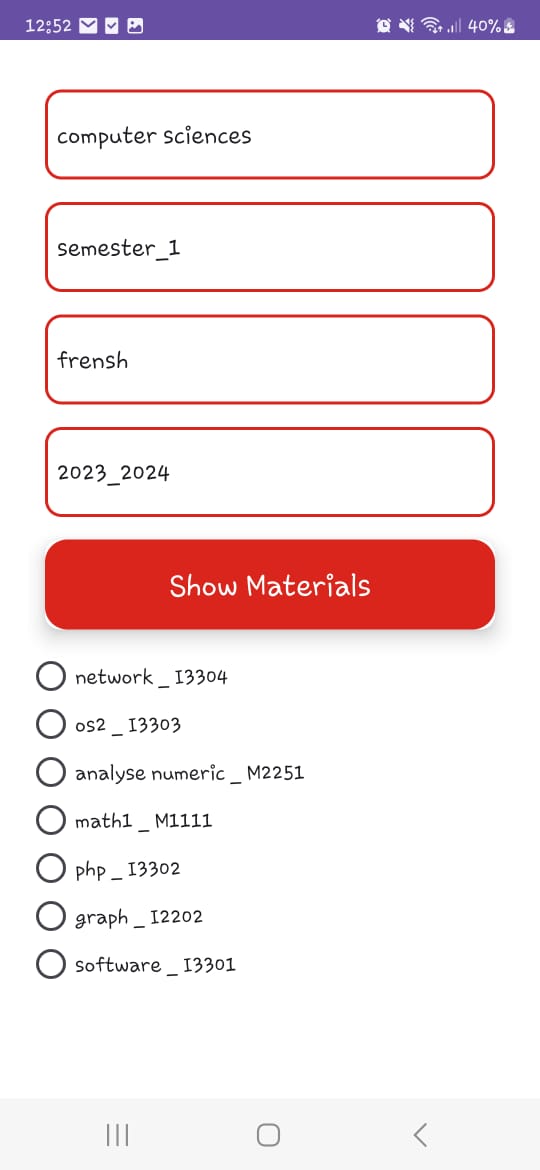
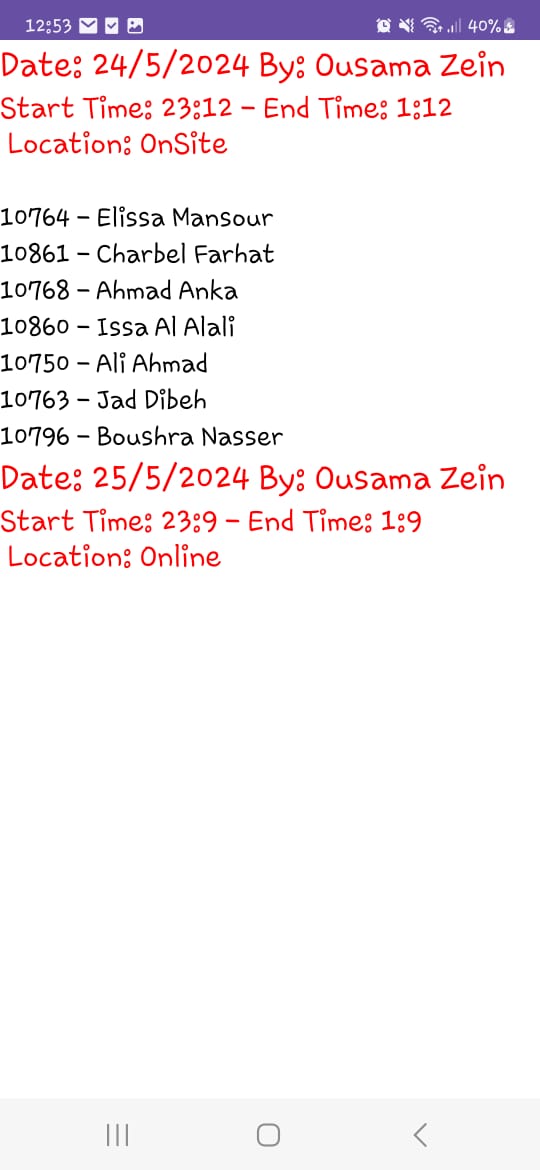
#### After selecting the Check Attendance option, the administrator will check either the Current Attendance or the Previous Attendance (archived).

#### Current Attendance

When the admin clicks on "Current Attendance", The administrator can filter and view attendance information for a specific course. They select a department, semester, and language to view a catalog of materials (courses). When he select a material, he will see information such as the number of lectures and enrolled students. Each student's attendance record and percentage are displayed. When you click on the detailed CardView, you'll get information about each session, such as the date, doctor, time, location, and IDs of students who are attending.

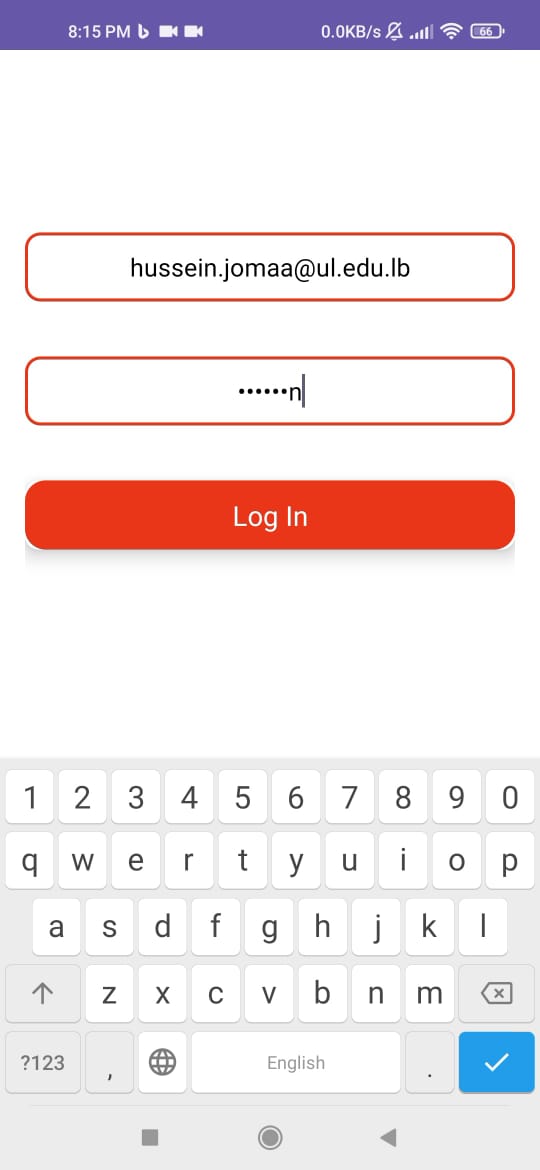
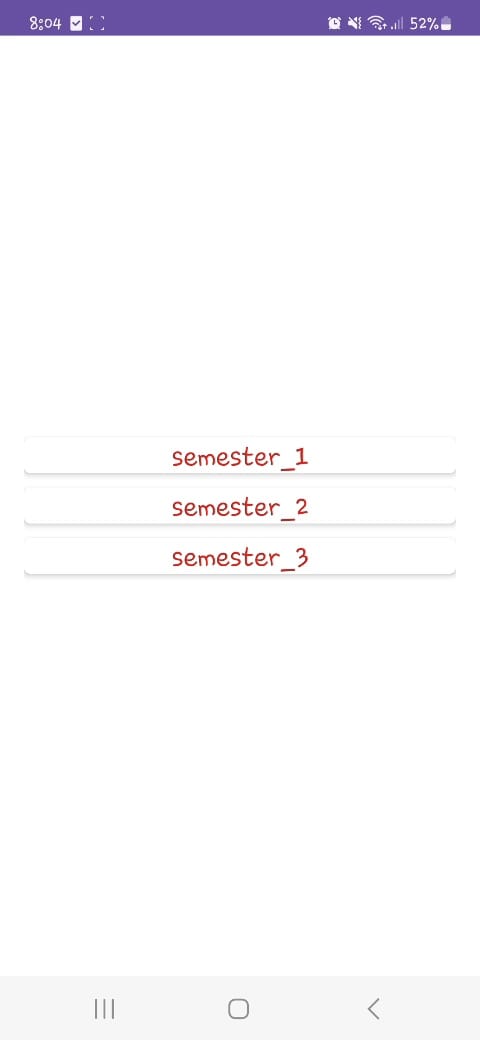
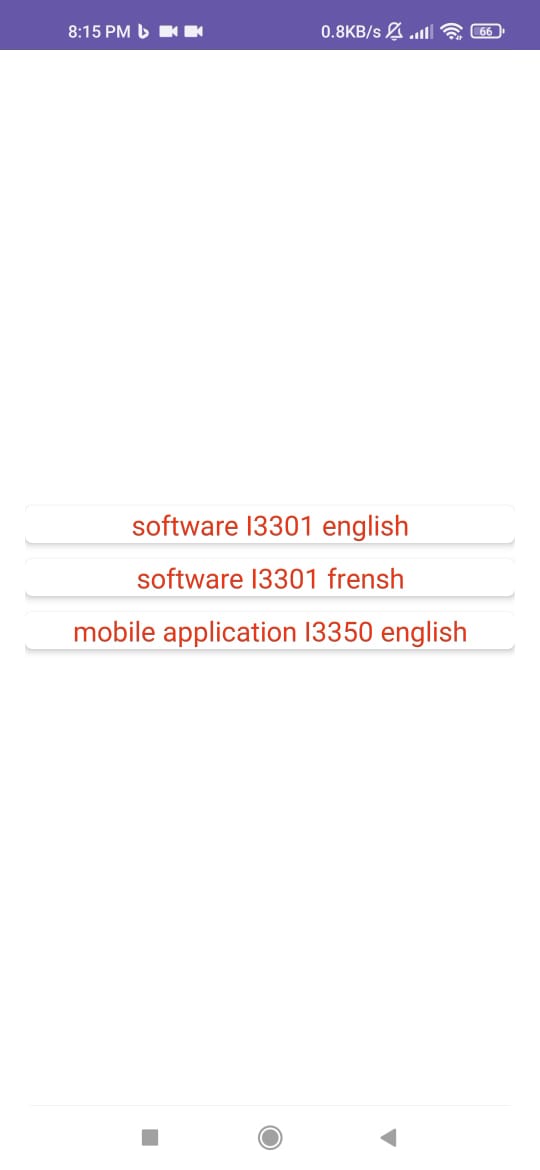
#### Previous Attendacnes

When the admin clicks on "Previous Attendance", The administrator can filter and view the archived attendance information for a specific course. They select a department, semester, language, and academic year to view a catalog of materials (courses). When you select a material, you will see information such as the number of lectures and enrolled students. Each student's attendance record and percentage are displayed. When you click on the detailed CardView, you'll get information about each session, such as the date, doctor, time, location, and IDs of students who are attending.

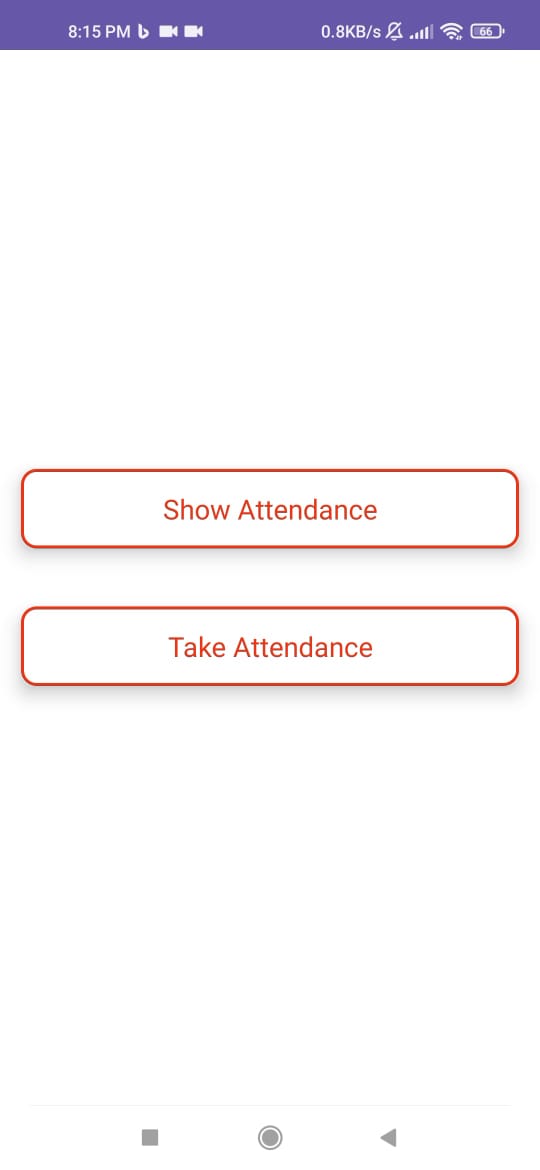
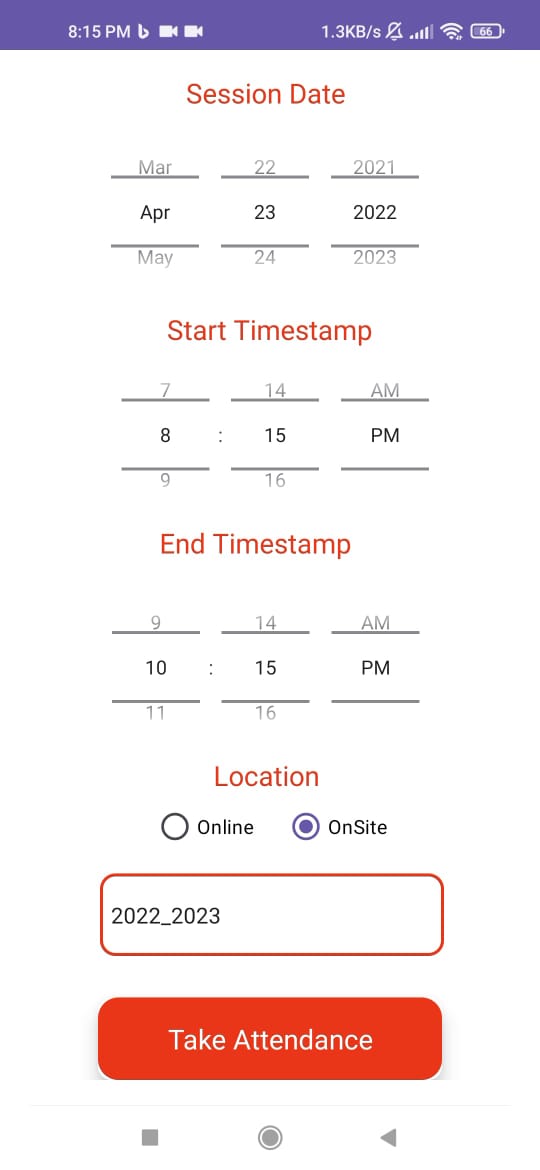
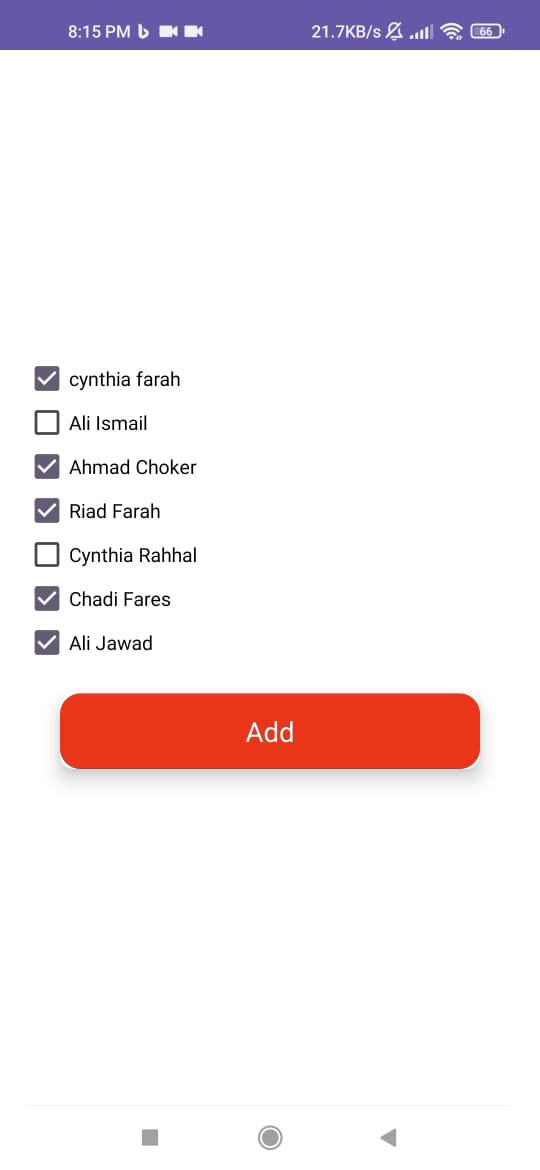
## Doctor

Doctors use the app to take attendances for each session, show attendances.

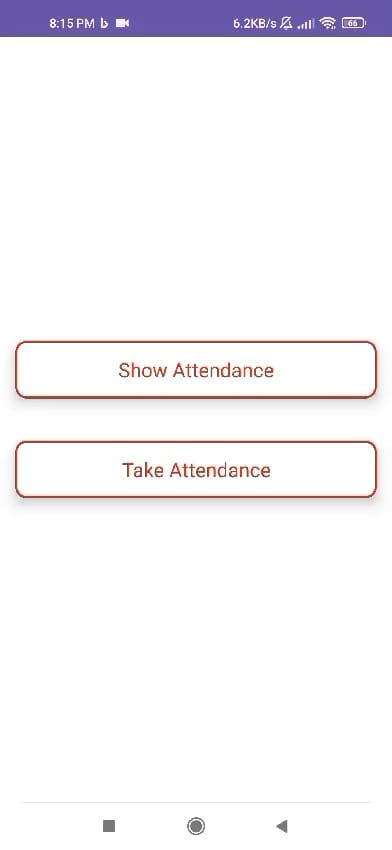
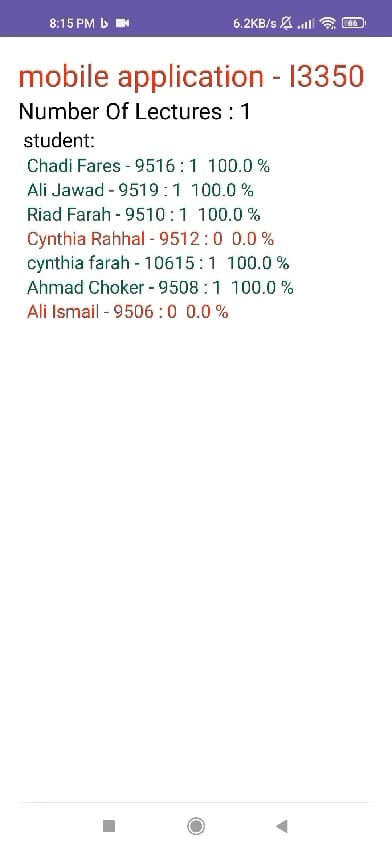
First, the doctor should log in using their email address and password, then choose a semester and the material that they want to show or take attendance on.

### take Attendance

Upon selecting the "Take Attendance" option, the doctor will first create a session by entering the date, start and end time, location, and academic year. They will then be directed to another screen to select the students who attended the session.

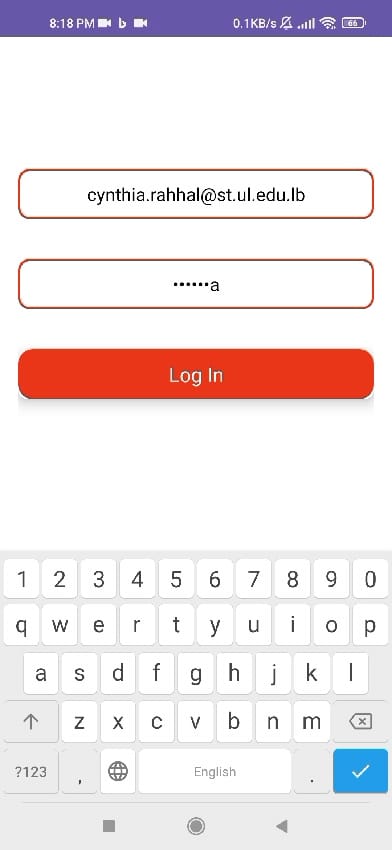
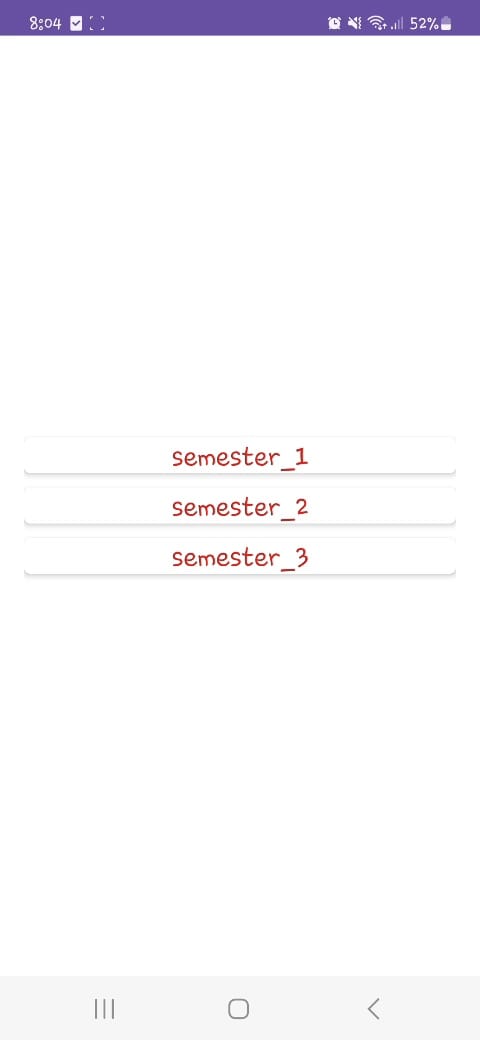
### Check attendance

Selecting "Show Attendance" displays the number of lectures delivered and enrolled students. Each student's attendance record and corresponding percentage are shown in green for those who exceed 50% attendance and red for those who fall below.

## Student

Students rely on the app to check their attendances.

Students begin by logging in with their email address and password. They can then select a semester to view a list of all materials they are registered for. Each material includes the student's attendance count as well as the overall number of sessions conducted. Attendance above 50% is displayed in green, while attendance below 50% is displayed in red.

# chapter 6: Final Thoughts and Future Scope

As we approach the end of this report, it is appropriate to reflect on "AttendEase"'s journey and envisage its future. This chapter summarizes our findings and suggests potential enhancements for the future of "AttendEase."

## Conclusion

The "AttendEase" mobile app offers a digital alternative to the manual attendance method, decreasing administrative hassles and eliminating human mistake. It enables administrators to check attendance records with a single click, making the procedure easier for faculty members. The application also keeps track of lecture numbers and student attendance, making it a dependable tool for monitoring student attendance. Students can track their attendance in real time, which improves academic planning and eliminates doubt about exam eligibility. The "AttendEase" program is an important step forward in using technology to improve academic administration efficiency and reliability. It provides useful answers and insights on the possibilities of mobile applications in academic administration. The research seeks to make additional contributions to this intriguing field.

## Future enhancment

As we look towards the future of “AttendEase”, it’s important to consider potential enhancements that could further improve its functionality and user experience. These enhancements are not only aimed at addressing the evolving needs of administrators, faculty members, and students, but also at leveraging the latest technological advancements. Let’s explore some of these potential future enhancements for the application.

**Integration with Academic Calendars**: The application could be integrated with the academic calendars of the institutions. This would allow automatic scheduling of classes and automatic tracking of attendance based on these schedules.

**Real-Time Notifications:** Implementing a system of real-time notifications for students and faculty members could be beneficial. For instance, students could receive notifications if their attendance falls below a certain threshold. Similarly, faculty members could be notified about unusual attendance patterns.

**Analytics Dashboard:** An analytics dashboard could be developed for administrators and faculty members. This dashboard could provide insights into attendance trends, patterns, and anomalies, aiding in decision-making processes.

**Offline Mode:** An offline mode could be added to allow attendance tracking even when internet connectivity is not available. The data could be synced when the connection is restored.

**Multi-Language Support:** To cater to a diverse user base, the application could offer multi-language support.

**Customizable Attendance Policies:** The application could allow administrators to set customizable attendance policies for different courses or departments.