A REPORT OF SIX WEEKS TRAINING

at

DUCAT UNDER S O INFOTECH (P) LTD.

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science and Engineering)

JUNE – JULY,2026

SUBMITTED BY:

NAME : KARTIK KOUNDAL

UNIVERSITY ROLL NO. : 241370690028

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DRONACHARYA COLLEGE OF ENGINEERING
GURUGRAM, HARYANA

CERTIFICATE

S O INFOTECH (P) LTD.
PULSATING TECHNOLOGIES

Dated: 8/8/2026 Ref No. SO/PL26/D.V/0136

TO WHOM IT MAY CONCERN

This is to certify that **Mr. Kartik Koundal, S/O Mr. Sanjeev Kumar,** has successfully completed the project at our organization from **8th June 2026 to 20th July 2026.** Under training period, he has worked on the project entitled **"Student Organizer"** using **"Java Android Kotlin"** under the guidance of our Technical Head **Mr. Shyam Mishra.** During this project he was an active member of the team working on the project. The work carried by him was satisfactory and we wish him all the best for his future assignments.

Best Regards,

Auth. Signatory

Regd. Office
N-105, Greater Kailash - I, New Delhi - 110048
Ph.: 91-011-29236054

Corporate Office
A-52, Sector-16, Noida - 201 301 (UP) INDIA
Tel: 91-120-4646464, Mobile: 09871055180
E-mail: info@soinfotech.com

www.soinfotech.com
CIN No. U72900DL2001PTC109989

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
DRONACHARYA COLLEGE OF ENGINEERING, GURUGRAM

CANDIDATE'S DECLARATION

I, Kartik Koundal, hereby declare that I have undertaken six weeks of Software Training at Ducat under S O Infotech (P) LTD. during the period from 8th June to 20th July, in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering at the Department of Computer Science & Engineering, Dronacharya College of Engineering, Gurugram. The work presented in this training report, submitted to the Department of Computer Science & Engineering, Dronacharya College of Engineering, Gurugram, is an authentic record of my own training work.

Signature of the Student

The Software Training Viva–Voce Examination of Kartik Koundal has been held on and accepted.

Dr. Ashima Mehta

Signature of HOD

ABSTRACT

The rapid adoption of smartphones among the student community has created an opportunity to replace fragmented, paper-based methods of academic planning with a single, unified mobile application. This report documents the training undertaken in native Android application development and presents “Student Organizer”, an Android application built using Kotlin, that was developed as the outcome of this training.

Student Organizer allows a student to register and sign in securely, view course-wise academic progress on a home dashboard, maintain a list of enrolled courses, create and manage personal notes with full create-read-update-delete (CRUD) functionality, track assignments along with their due dates and completion status, view a monthly calendar to plan a day-wise schedule, and manage a personal profile. The application follows a layered architecture in which Activities and Fragments form the presentation layer, RecyclerView and ViewPager adapters bind data to the user interface, and the Room persistence library provides an abstraction over an on-device SQLite database for durable local storage. Kotlin Coroutines are used together with Jetpack lifecycle-aware scopes to perform database operations asynchronously without blocking the main thread.

During the course of the training, the trainee gained hands-on exposure to the Android Studio development environment, XML-based UI design with Material Design components, the Model-View pattern of structuring an Android codebase, local persistence using Room, fragment-based navigation using ViewPager and BottomNavigationView, and version control using Git. This report describes the organisation where the training was undergone, the tools and techniques learned, the practical exercises carried out during the training, the design and implementation of the Student Organizer application as the major project, the results obtained, and the scope for future enhancement of the application.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who supported and guided me during the course of my six weeks of software training and the preparation of this report.

I am thankful to Mr. Shyam Mishra, Technical Head, for providing me the opportunity to undergo training and for their valuable guidance, patience and encouragement throughout the training period.

I am equally grateful to Dr. Ashima Mehta, Head of the Department of Computer Science & Engineering, Dronacharya College of Engineering, Gurugram, and to all the faculty members for their continuous support and encouragement during the course of this training.

Finally, I would like to thank my family and friends for their unwavering support and motivation throughout this training.

Kartik Koundal

University Roll No.: 241370690028

LIST OF FIGURES

Figure No.	Title	Page No.
Figure 3.1	Navigation and screen flow of the Student Organizer application	25
Figure 4.1	Layered architecture of the Student Organizer application	30
Figure 4.2	Database schema used by the application	31
Figure 5.1	Onboarding and authentication screens – (a) Splash (b) Onboarding (c) Register (d) Login	34
Figure 5.2	Home dashboard and course tracking screens – (a) Home Dashboard (b) Course Tracking	35
Figure 5.3	Notes management screen	36
Figure 5.4	Assignment tracking screen	37
Figure 5.5	Calendar screen	38
Figure 5.6	Profile screen	39

LIST OF TABLES

Table No.	Title	Page No.
Table 2.1	Summary of Software Tools and Technologies Used During Training	19
Table 4.1	Hardware requirements	29
Table 4.2	Software requirements	29
Table 4.3	Schema of the users table (AppDatabase)	31
Table 4.4	Schema of the notes table (NotesDatabase)	32
Table 4.5	Module-wise description of the application	32
Table 5.1	Summary of functional testing carried out on the application	40

DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Abbreviations	Full Forms
API	Application Programming Interface
APK	Android Package Kit
CRUD	Create, Read, Update, Delete
DAO	Data Access Object
DB	Database
DBMS	Database Management System
GUI	Graphical User Interface
KSP	Kotlin Symbol Processing
ORM	Object Relational Mapping
SDK	Software Development Kit
SQL	Structured Query Language
UI	User Interface
UX	User Experience
IDE	Integrated Development Environment
XML	Extensible Markup Language

CONTENTS

Certificate by Company/Industry/Institute *2*

Candidate's Declaration *3*

Abstract *4*

Acknowledgement *5*

List of Figures *6*

List of Tables *7*

Definitions, Acronyms and Abbreviations *8*

CHAPTER 1 INTRODUCTION TO ORGANIZATION(S) 11-13

1.1 About the Organization 11

1.2 Vision and Mission 12

1.3 Products and Services Offered 12

1.4 Overview of the Training Programme 13

CHAPTER 2 SOFTWARE TRAINING WORK UNDERTAKEN 14-20

2.1 Introduction 14

2.2 Programming Language: Kotlin 14

2.3 Integrated Development Environment: Android Studio 15

2.4 Android Application Architecture Fundamentals 16

2.5 User Interface Design Tools 16

2.6 Data Persistence: Room Persistence Library and SQLite 17

2.7 Asynchronous Programming with Kotlin Coroutines 18

2.8 Summary of Tools and Technologies 19

2.9 Chapter Summary 20

CHAPTER 3 INDUSTRIAL TRAINING WORK UNDERTAKEN 21-26

3.1 Introduction 21

3.2 Practical Exercise 1 – Splash Screen and Onboarding Flow 21

3.3 Practical Exercise 2 – User Registration and Authentication Module 22

3.4 Practical Exercise 3 – Local Database Integration using Room 23

3.5 Practical Exercise 4 – List-Based UI Components using RecyclerView 24

3.6 Practical Exercise 5 – Fragment-based Navigation 25

3.7 Chapter Summary 26

CHAPTER 4 PROJECT WORK 27-33

4.1 Introduction / Problem Statement 27

4.2 Objectives of the Project 27

4.3 Scope of the Project 28
4.4 System Requirements 29
4.5 System Design 30
4.6 Methodology Adopted 33

CHAPTER 5 RESULTS AND DISCUSSION 34-41

5.1 Introduction 34
5.2 Onboarding and Authentication Module 34
5.3 Home Dashboard and Course Tracking Module 35
5.4 Notes Management Module 36
5.5 Assignment Tracking Module 37
5.6 Calendar Module 38
5.7 Profile Module 39
5.8 Summary of Functional Testing 40
5.9 Discussion 41

CHAPTER 6 CONCLUSION AND FUTURE SCOPE 42-43

6.1 Conclusion 42
6.2 Future Scope 43

REFERENCES 44

APPENDIX 45-46

CHAPTER 1: INTRODUCTION TO ORGANIZATION(S)

1.1 About the Organization

S O Infotech Private Limited (DUCAT) is an Indian organization operating in the field of information technology, computer-related activities and technical education. The company was incorporated on **13 March 2001** and is registered in Delhi. Public company records classify S O Infotech Private Limited as a private, non-government company engaged in computer and related activities. The organization has been associated with **DUCAT**, an established IT training and education brand providing industry-oriented technical training programs [1],[2].

DUCAT provides technical training through its centres in the Delhi-NCR region and other locations. Its Noida operations include the **Sector 16, Noida** centre, which is associated with S O Infotech (P) Ltd. and provides training in various information technology domains. The organization focuses on practical and industry-oriented learning, enabling students and professionals to develop technical skills relevant to current software and IT requirements. Public training records also identify S O Infotech (P) Ltd. at **A-52, Sector 16, Noida, Uttar Pradesh**, as a training organization [3].

The organization operates in the broader IT training and computer-related services domain. Through DUCAT, it offers structured training programs covering areas such as software development, programming, data science, cloud computing, cybersecurity, software testing, web development, UI/UX, digital marketing and other emerging technologies. The official DUCAT website describes its programs as industry-oriented and includes practical training, live projects and real-world case studies as part of its training approach [1].

1.2 Vision and Mission

The broad objective of **S O Infotech (P) Ltd. / DUCAT** is to provide industry-oriented technical education and practical exposure to students and professionals. The organization aims to bridge the gap between academic knowledge and the practical skills required in the IT industry by providing structured training in contemporary technologies and development practices [1].

DUCAT's training approach emphasizes practical learning, experienced trainers, updated technical curricula, live projects and real-world applications. Its programs are designed to help learners develop job-relevant technical capabilities and improve their readiness for professional roles in the information technology sector [1].

1.3 Products and Services Offered

S O Infotech (P) Ltd., through its DUCAT training operations, provides a wide range of **IT education and technical training services**. The organization offers courses and training programs in areas including **Java, Python, full-stack development, MERN and MEAN stack development, data analytics, data science, machine learning, artificial intelligence, cloud computing, AWS, Azure, cybersecurity, DevOps, software testing, UI/UX design and digital marketing**. The official DUCAT platform also highlights practical projects, case studies, interview preparation and placement assistance as components of its training ecosystem [1].

During the training period, the trainee was associated with the **Android Application Development** domain. The training focused on developing native Android applications using **Kotlin and XML**, along with Android Studio and commonly used Android development components. The practical work performed during the training culminated in the development of the **Student Organizer** Android application, which served as the major project of the training programme.

1.4 Overview of the Training Programme

The training was undertaken as a structured programme in **native Android application development**, with the objective of developing practical proficiency in designing, coding, testing and documenting a complete mobile application. The training was conducted during the period from **8 June 2026 to 20 July 2026**. It began with the fundamentals of Kotlin programming and Android development and gradually progressed towards the development of a complete Android application. The initial phase of the training focused on understanding the Android Studio development environment, Kotlin programming concepts, Android application structure, activities, fragments, layouts, XML-based user interface design and navigation between screens. The training also covered important Android development components such as **View Binding, SharedPreferences, RecyclerView, CardView and Room Database**, which were subsequently applied in practical exercises and application development.

The later phase involved integrating the concepts learned during the training into a single application, namely the **Student Organizer Android application**. The application was designed to provide students with a centralized platform for managing academic activities such as notes, courses, attendance, assignments and timetable information. Authentication and user-related functionality were implemented using Android development techniques learned during the programme.

The training provided practical exposure to the complete software development process, including user interface design, implementation of application functionality, database integration, testing and debugging. The experience helped in developing a better understanding of native Android application development and in applying theoretical programming concepts to a practical software project. The subsequent chapters of this report present the technologies and concepts learned, practical exercises undertaken, development of the Student Organizer application, testing results, and conclusions with possible future enhancements.

CHAPTER 2: SOFTWARE TRAINING WORK UNDERTAKEN

2.1 Introduction

This chapter describes the background of the training, the theoretical concepts covered, and the software tools and technologies that were learned before development of the major project began. The training focused on native Android application development using Kotlin, which is one of the officially recommended languages for Android development. The sub-sections that follow summarise, in the order in which they were studied, the major programming concepts, tools, libraries and techniques covered during the training.

2.2 Programming Language: Kotlin

Kotlin is a statically typed, general-purpose programming language that runs on the Java Virtual Machine and is fully interoperable with Java. It is officially supported by Google for Android development and is widely used for developing modern Android applications because of its concise syntax, null-safety features and interoperability with existing Java-based Android APIs [4].

During the training, the following language features were studied and applied: data classes for defining lightweight model objects; null-safety operators (?., ?:, !!) for reducing the possibility of null-pointer exceptions; lambda expressions and higher-order functions for concise event handling, such as click listeners; extension functions; and coroutines for structured asynchronous programming [4], [7].

Kotlin's data class construct was used to define model and entity classes of the application, such as the User, Notes, Course and Assignment classes. Data classes automatically provide

commonly used functions such as equals(), toString() and copy(), reducing the amount of boilerplate code required.

The use of these features helped in writing concise, readable and maintainable code during the development process.

2.3 Integrated Development Environment: Android Studio

Android Studio, the official IDE for Android development, was used throughout the training. It is built on the JetBrains IntelliJ platform and provides a Kotlin/Java code editor, a visual layout editor, an APK build system based on Gradle, tools for running applications on Android devices, and a Logcat window for run-time debugging [8].

The project uses the Gradle Kotlin DSL (build.gradle.kts) for build configuration, which allows dependencies, plugins and build settings to be declared using Kotlin-based configuration files. Android Studio was used for writing and organizing source code, designing XML layouts, managing dependencies, building the application and debugging runtime errors.

The application was tested on a physical Android device during development. Android Studio also provides support for virtual Android devices and emulators, although physical-device testing was used for the project.

The application is configured with a minimum SDK level of 24 and a target SDK level of 37. Kotlin Symbol Processing (KSP) is used for compile-time code generation associated with the Room database implementation.

2.4 Android Application Architecture Fundamentals

The training covered the fundamental building blocks that make up an Android application. An Activity represents a single, focused screen with which the user can interact and acts as an entry point for a particular part of an application. The Android application developed during the training uses activities such as SplashActivity, OnboardingActivity, LoginActivity, RegisterActivity and MainActivity.

A Fragment represents a reusable portion of a user interface that must be hosted inside an activity. The training covered the use of fragments for separating different sections of an Android application and managing their individual user interfaces.

The Activity and Fragment lifecycle methods, including onCreate(), onCreateView() and onDestroyView(), were studied during the training. Particular attention was given to the Fragment lifecycle and the appropriate handling of View Binding references so that references to destroyed views could be released in onDestroyView().

These concepts provided an understanding of how Android manages application components and their lifecycle during navigation and configuration changes.

2.5 User Interface Design Tools

User interfaces in the application are declared using XML layout resource files and are accessed using View Binding, which was enabled in the module-level build.gradle.kts file using buildFeatures { viewBinding = true }. View Binding generates a binding class for every XML layout, allowing views to be referenced in a type-safe manner without repeatedly using findViewById() [5].

The training covered ConstraintLayout for building flexible and responsive view hierarchies, particularly for screens such as login and registration. LinearLayout was also studied for arranging views horizontally or vertically.

RecyclerView was covered for rendering efficient and scrollable lists of data using a LayoutManager and a custom RecyclerView.Adapter. It was applied to areas of the application where lists of information needed to be displayed dynamically [9].

ViewPager2 was studied and used for implementing swipeable onboarding screens through a dedicated adapter [10]. BottomNavigationView, provided through the Material Components library, was used for persistent navigation between major sections of the application [11].

Card-based containers were also used for presenting information such as courses, notes and assignments in a structured manner.

Material Components for Android were studied to provide reusable interface components and maintain a consistent visual design across the application [11].

2.6 Data Persistence: Room Persistence Library and SQLite

Since Android applications often need to retain data even after the application process is closed, the training covered local data persistence using the Room Persistence Library. Room is an Android Jetpack library that provides an abstraction layer over SQLite and simplifies database operations while providing compile-time verification of SQL queries [6].

Room was chosen over direct SQLite APIs because it reduces database-related boilerplate, provides a structured programming interface and integrates well with modern Android development practices [6].

The training covered the three principal Room building blocks that are used in the project:

- The @Entity annotation, which maps a Kotlin data class to a database table.
- The @Dao annotation, which defines an interface containing data-access methods decorated with annotations such as @Insert, @Update, @Delete and @Query.
- The RoomDatabase class, which acts as the main access point to the database and can be instantiated using Room.databaseBuilder().

Room was used for persistent application data such as user and notes information. The use of Room provided a structured approach to storing, retrieving, updating and deleting local application data.

2.7 Asynchronous Programming with Kotlin Coroutines

Because database operations should not block the Android main thread, the training covered Kotlin Coroutines as a mechanism for asynchronous programming. Coroutines provide a structured way of performing long-running operations without unnecessarily blocking the user interface [7].

Data Access Object methods in the project that perform database operations were implemented using coroutine-compatible functions and invoked from lifecycle-aware coroutine scopes such as lifecycleScope.launch { ... }. These scopes are provided through AndroidX Lifecycle components and help associate asynchronous work with the lifecycle of the Activity or Fragment [7].

This pattern was applied to operations such as user registration, user authentication, and loading, inserting, updating or deleting note data. The use of coroutines helped keep database operations separate from the main user-interface thread and maintained application responsiveness.

2.8 Summary of Tools and Technologies

Table 2.1: Summary of Software Tools and Technologies Used During Training

Category	Tool / Technology	Purpose
Language	Kotlin	Primary programming language
IDE	Android Studio	Coding, building, debugging and testing
Build Tool	Gradle Kotlin DSL	Build configuration and dependency management
Build Processing	KSP	Compile-time code generation
UI Toolkit	XML Layouts, View Binding, ConstraintLayout	User-interface design and type-safe view access
UI Components	RecyclerView, ViewPager/ViewPager2, BottomNavigationView	Lists, onboarding and navigation
Design System	Material Components for Android	Consistent user-interface components
Persistence	Room over SQLite	Local relational data storage
Concurrency	Kotlin Coroutines, lifecycleScope	Asynchronous and lifecycle-aware operations
Session Storage	SharedPreferences	Lightweight session and application-state storage

2.9 Chapter Summary

This chapter presented the major programming language, development environment, Android technologies and software-development techniques covered during the training programme. Kotlin provided the primary programming foundation, while Android Studio and Gradle supported the development and build process. XML, View Binding and Material Components were used for user-interface development, while RecyclerView, ViewPager2 and BottomNavigationView provided commonly required interface and navigation capabilities.

Room and SQLite were studied for local data persistence, while Kotlin Coroutines provided a mechanism for performing database-related operations asynchronously. SharedPreferences was also used for lightweight application-state and session information.

These concepts and technologies formed the technical foundation for the practical exercises and major project development discussed in the subsequent chapters.

CHAPTER 3: INDUSTRIAL TRAINING WORK UNDERTAKEN

3.1 Introduction

This chapter presents an overview of the practical exercises that were carried out during the training, and the way in which techniques and tools introduced in Chapter 2 were applied hands-on before being integrated into the major project described in Chapter 4. Five practical exercises are described below, each of which corresponds to a functional module of the final application.

3.2 Practical Exercise 1 – Splash Screen and Onboarding Flow

The first exercise involved implementing an application launch flow consisting of a SplashActivity, built using the AndroidX Core SplashScreen API, followed by a three-slide OnboardingActivity built with ViewPager2 and a custom OnboardingAdapter. The AndroidX SplashScreen API provides a standardized mechanism for implementing application launch screens [13].

SplashActivity reads two Boolean flags, isFirstTime and isLoggedIn, from a SharedPreferences file named "Details" and routes the user to the onboarding flow, the login screen, or directly into the main application, as appropriate.

This exercise provided hands-on experience of SharedPreferences-based state persistence and of driving conditional navigation from an Activity's onCreate() method. It also provided practical experience in implementing a multi-page onboarding interface using ViewPager2.

3.3 Practical Exercise 2 – User Registration and Authentication Module

The second exercise involved building a registration and login flow backed by a local Room database. RegisterActivity performs client-side validation – checking that the name field is not empty, that the e-mail address matches android.util.Patterns.EMAIL_ADDRESS, that the password is at least eight characters long and matches the confirmation field, and that the terms-and-conditions checkbox has been ticked – before inserting a new User record through UserDao.insertUser(). LoginActivity subsequently authenticates the user with UserDao.loginUser(email, password) and, on success, stores the logged-in state in SharedPreferences before navigating to MainActivity. The following listing shows the core of the sign-up validation and persistence logic:

```
binding.btnSignUp.setOnClickListener {
val name = binding.etName.text.toString().trim()
val email = binding.etEmail.text.toString()
val password = binding.etPassword.text.toString()
...
if (name.isEmpty()) {
binding.etName.error = "Enter your name"
} else if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
binding.etEmail.error = "Enter valid email"
} else if (password.length < 8 || password != confirmPassword) {
binding.etPassword.error = "Enter valid password"
} else {
val user = User(name = name, email = email, password = password)
lifecycleScope.launch { db.userDao().insertUser(user) }
...
}
}
```

This exercise provided practical experience of form validation, of one-to-one mapping between a UI form and a Room entity, and of writing to a database from a coroutine launched on lifecycleScope.

3.4 Practical Exercise 3 – Local Database Integration using Room

The third exercise involved designing and wiring up two independent Room databases: AppDatabase, which stores the users table used for authentication, and NotesDatabase, which stores the notes table used by the Notes module.

Both databases are exposed as thread-safe singletons using the double-checked-locking pattern. The following listing illustrates the implementation of AppDatabase; NotesDatabase follows the same general pattern.

```
@Database(entities = [User::class], version = 1, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
    companion object {
        @Volatile private var INSTANCE: AppDatabase? = null
        fun getDatabase(context: Context): AppDatabase =
            INSTANCE ?: synchronized(this) {
                Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java, "user_database"
                ).build().also { INSTANCE = it }
            }
    }
}
```

This exercise gave hands-on experience of defining @Entity data classes, @Dao query interfaces, and of using Kotlin Symbol Processing to generate the Room implementation classes at compile time.

3.5 Practical Exercise 4 – List-Based UI Components using RecyclerView

The fourth exercise involved building three independent RecyclerView-based lists.

The **course list** displays the course code, course title and a linear progress indicator through CourseAdapter. The **assignment list** displays the assignment title, subject, due date and completion-status information through AssignmentAdapter. The **notes list** displays the title, subject, content preview and last-modified date through NoteAdapter.

The Notes module also supports user interactions such as tap-to-edit and long-press-to-delete actions. These actions are handled through lambda callbacks passed to the adapter.

This exercise provided practical experience of the RecyclerView.Adapter and RecyclerView.ViewHolder pattern, implementing callback-based interactions within adapters, and using View Binding inside RecyclerView ViewHolders. RecyclerView is designed for efficiently displaying collections of data by reusing item views as they move in and out of the visible area [9].

3.6 Practical Exercise 5 – Fragment-based Navigation with ViewPager and Bottom Navigation

The final exercise integrated the modules built in the earlier exercises into a single-activity, multi-fragment shell. MainActivity hosts a ViewPager, populated through a ViewPagerAdapter that extends FragmentPagerAdapter, with six fragments – Home, Course, Notes, Calendar, Assignment and Profile. A BottomNavigationView is kept in sync with the ViewPager in both directions: selecting a bottom navigation item sets tabViewpager.currentItem, while an

OnPageChangeListener on the ViewPager checks the corresponding bottom navigation item whenever the page is changed by swiping. The overall navigation flow of the application, from the splash screen through onboarding and authentication into the six-tab main shell, is summarised in Figure 3.1.

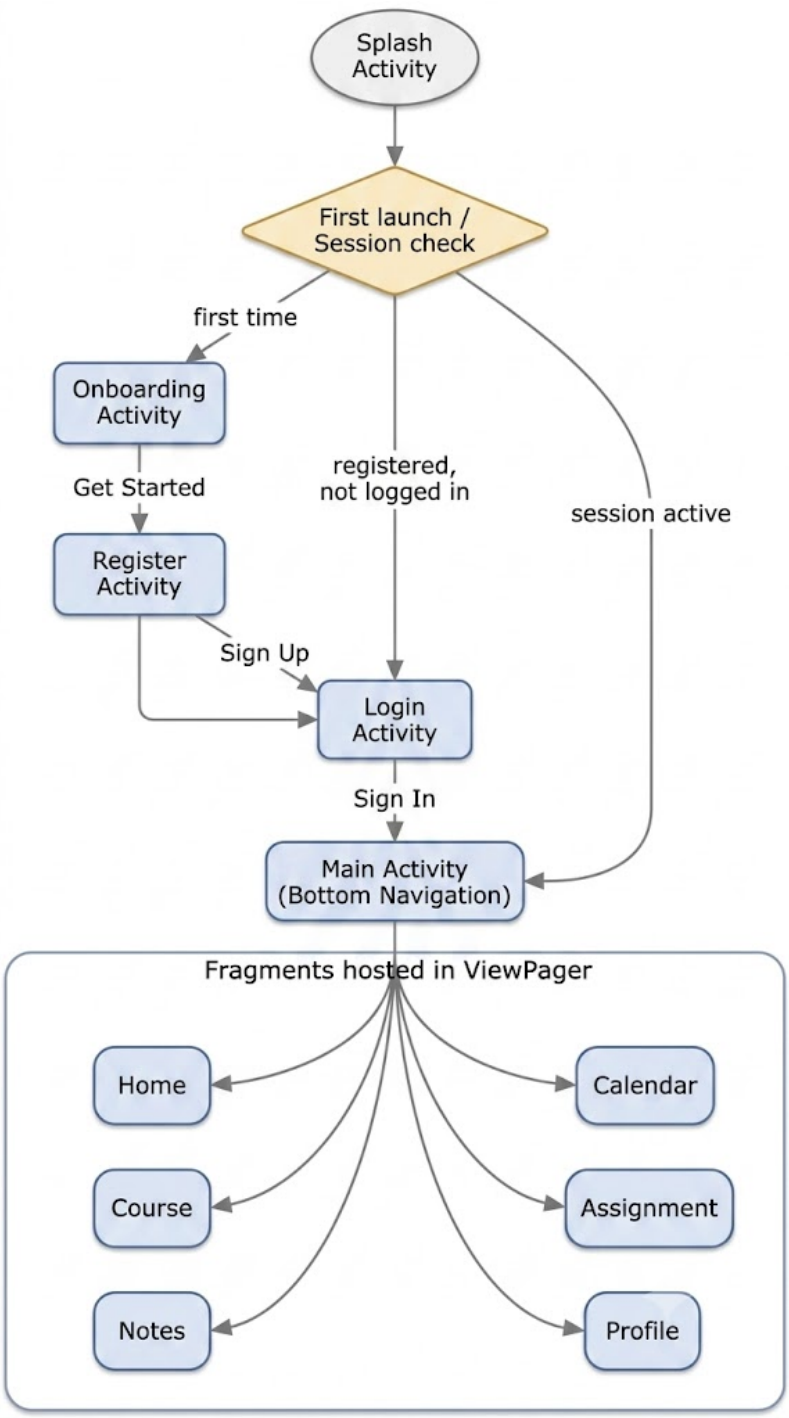


Figure 3.1: Navigation and screen flow of the Student Organizer application

3.7 Chapter Summary

The practical exercises undertaken during the training provided hands-on experience in applying the programming concepts and Android development technologies discussed in Chapter 2.

The exercises progressed from basic application launch and onboarding functionality to user authentication, local database integration, dynamic list-based interfaces and Fragment-based navigation. The individual exercises were subsequently integrated into the complete Student Organizer application, which is discussed in detail in Chapter 4.

The practical work helped in developing an understanding of the complete Android application development process, including user-interface implementation, input validation, database operations, asynchronous programming, navigation and integration of multiple functional modules.

CHAPTER 4: PROJECT WORK

4.1 Introduction / Problem Statement

A typical undergraduate student today has to keep track of several strands of academic information at once – the courses enrolled in and progress made in each, assignments and their due dates, lecture notes taken across subjects, and a day-to-day class and exam schedule. In practice, this information tends to be scattered across paper diaries, loose notes, and generic note-taking or reminder applications, none of which are designed specifically around a student's academic workflow. This fragmentation makes it easy to lose track of an upcoming deadline or to misplace notes taken during a lecture. The major project undertaken during this training, Student Organizer, addresses this problem by providing a single native Android application in which a student can manage courses, assignments, notes and a calendar-based schedule from one place, behind a simple account-based sign-in.

4.2 Objectives of the Project

The Student Organizer application was designed and developed with the following objectives:

- To provide a secure, account-based entry point into the application through a registration and login module backed by a local database.
- To present a consolidated home dashboard summarizing a student's enrolled courses and progress at a glance.
- To allow a student to maintain a list of enrolled courses along with a visual indicator of progress in each.

- To provide full create, read, update and delete (CRUD) functionality for personal notes, organised by subject.
- To allow assignments to be tracked along with their due date and completion status.
- To provide a calendar view through which a student can review a day-wise class or exam schedule.
- To allow a student to view and manage basic profile information from within the application.
- To keep all of the above data persisted locally on the device, so that it survives an application restart, using an offline-first architecture that does not require a network connection.

4.3 Scope of the Project

The scope of the present version of the application is limited to a single-user, offline, on-device experience: all data is stored locally using the Room persistence library, and no cloud synchronization, push notifications, or multi-device access are implemented at this stage. The calendar module supports viewing a monthly calendar and a day-wise schedule; recurring timetable generation is outside the current scope. Sections 6.2 (Future Scope) of this report discusses how these limitations could be addressed in a subsequent version of the application.

4.4 System Requirements

4.4.1 Hardware Requirements

Table 4.1: Hardware requirements

Component	Development Environment	Deployment / Target Device
Processor	Intel Core i5 (8th Gen) or higher / equivalent	ARM64 / x86_64 mobile SoC
RAM	8 GB minimum, 16 GB recommended	2 GB minimum
Storage	10 GB free space (SDK, emulator images)	100 MB free space for app and data
Display	1920 × 1080 or higher	Any standard Android smartphone display

4.4.2 Software Requirements

Table 4.2: Software requirements

Component	Specification
Operating System (development)	Windows 10/11, macOS or Linux
IDE	Android Studio (Gradle 9.x, Kotlin Gradle DSL)
Language / SDK	Kotlin; Android SDK, min SDK 24, target SDK 37
Key libraries	AndroidX Core-KTX, AppCompat, Material Components 1.14.0, Room 2.8.4, ViewPager2 1.1.0, Core-SplashScreen 1.2.0
Target OS (runtime)	Android 7.0 (API 24) and above
Version control	Git

4.5 System Design

4.5.1 Architecture of the Application

The application follows a simple, layered architecture rather than a full MVVM structure, chosen to keep the codebase approachable within the scope of a training project while still separating concerns cleanly. The presentation layer consists of the Activities and Fragments together with their XML layouts (bound through View Binding) and RecyclerView / ViewPager adapters. The data layer consists of the Room entities, DAOs and RoomDatabase subclasses, with SharedPreferences used alongside for lightweight session state. Communication between the two layers is mediated entirely through Kotlin Coroutines launched on lifecycleScope, so that no database call ever executes on the main thread. This structure is illustrated in Figure 4.1.

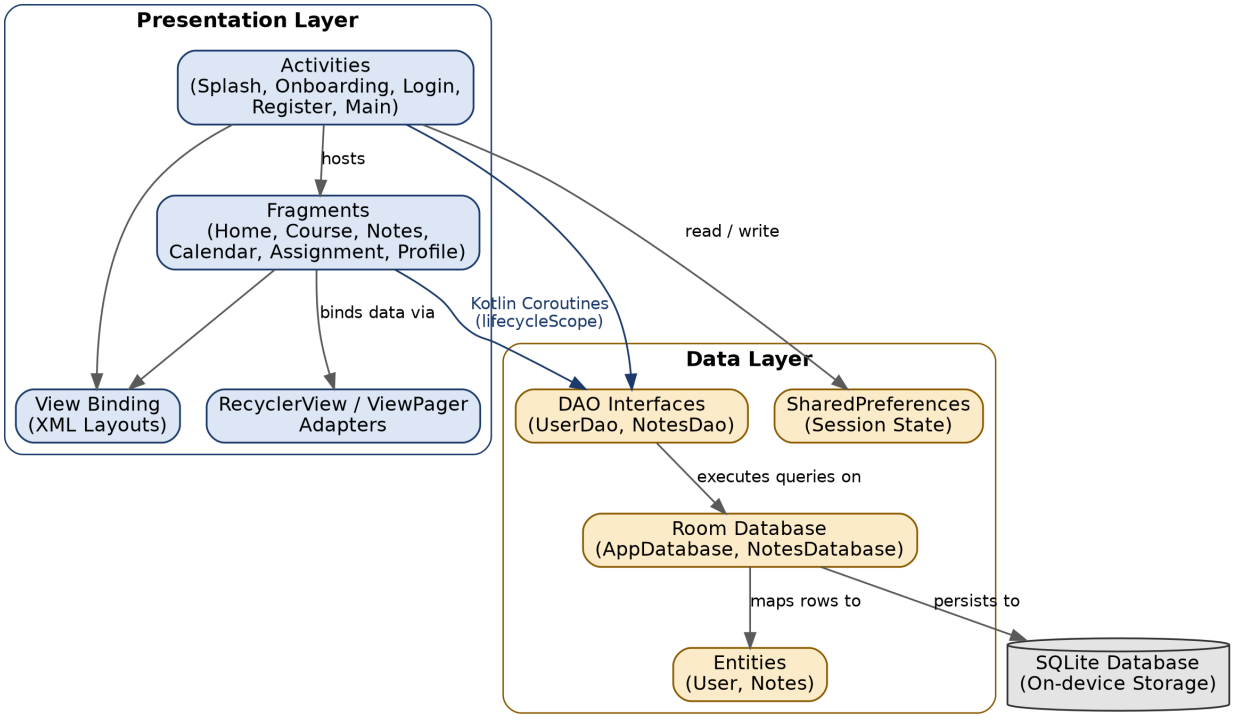


Figure 4.1: Layered architecture of the Student Organizer application

4.5.2 Database Design

The application uses two independent Room databases, each with a single entity, rather than one combined database. The users table (AppDatabase) stores the details required for authentication, and the notes table (NotesDatabase) stores the notes created by the user. The two schemas are shown in Figure 4.2 and detailed in Table 4.3 and Table 4.4. Keeping the two databases independent was a deliberate simplification for this stage of the project; Section 6.2 discusses consolidating them into a single database with foreign-key relationships as a future enhancement.

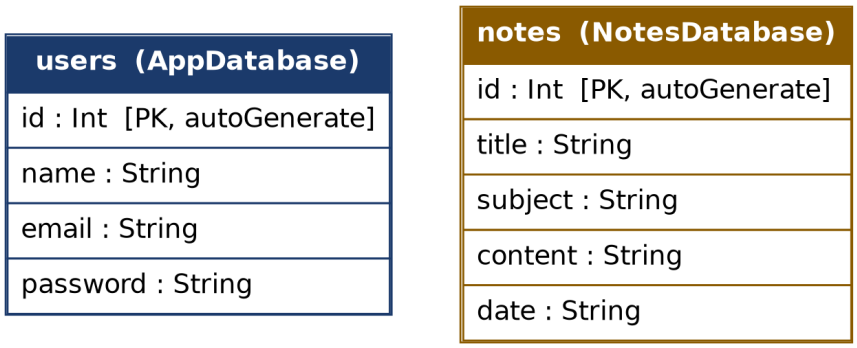


Figure 4.2: Database schema used by the application

Table 4.3: Schema of the users table (AppDatabase)

Column	Type	Constraints	Description
id	Int	Primary Key, auto-generate	Unique identifier for a user
name	String	Not null	Full name of the user
email	String	Not null	Registered e-mail address, used at login
password	String	Not null	Account password

Table 4.4: Schema of the notes table (NotesDatabase)

Column	Type	Constraints	Description
id	Int	Primary Key, auto-generate	Unique identifier for a note
title	String	Not null	Title of the note
subject	String	Not null	Subject / course the note relates to
content	String	Not null	Body text of the note
date	String	Not null	Date the note was created or last modified

4.5.3 Module-wise Description

Table 4.5: Module-wise description of the application

Module	Hosting Component	Responsibility
Splash	SplashActivity	Displays a launch screen and routes the user based on session state
Onboarding	OnboardingActivity	Introduces the application to a first-time user via three swipeable slides
Authentication	RegisterActivity, LoginActivity	Registers a new account and authenticates an existing one against AppDatabase
Home	HomeFragment	Shows a welcome banner and a summary of enrolled courses and progress
Course	CourseFragment	Lists all enrolled courses with a progress indicator for each
Notes	NotesFragment	Provides create, read, update and delete operations on personal notes
Calendar	CalendarFragment	Displays a monthly calendar and the schedule for the selected day
Assignment	AssignmentFragment	Lists assignments with due date and completion status
Profile	ProfileFragment	Displays account information and provides logout / reset options

4.6 Methodology Adopted

The project was developed using an incremental, module-wise methodology rather than attempting to build the complete application in one pass. Development proceeded in the following stages: (i) requirement identification and screen-flow planning, in which the modules described in Table 4.5 and the navigation flow shown in Figure 3.1 were finalised; (ii) UI design, in which each screen was first laid out in XML and reviewed in the Android Studio layout preview before any business logic was added; (iii) data-layer implementation, in which the Room entities, DAOs and database classes for a module were written and unit-tested informally by inserting and querying sample records; (iv) business-logic implementation, in which each screen was wired to its underlying data through View Binding, adapters and coroutine-based DAO calls; and (v) manual functional testing on an Android emulator and, where available, a physical device, the results of which are summarised in Chapter 5. Each module (authentication, notes, courses, assignments, calendar, profile) was carried through these five stages before development moved on to the next module, and changes were committed to the local Git repository at the end of each stage.

CHAPTER 5: RESULTS AND DISCUSSION

5.1 Introduction

This chapter presents the outcome of the project work described in Chapter 4, in the form of the functioning screens of the Student Organizer application, followed by a summary of the functional testing carried out and a discussion of the observations made. The screens shown in this chapter are representative interface mock-ups reconstructed directly from the XML layout resources and Kotlin logic implemented in the project, reproducing the exact screen structure, content and navigation of the running application.

5.2 Onboarding and Authentication Module

On first launch, the user is taken through a short onboarding flow (Figure 5.1(b)) before being asked to create an account (Figure 5.1(c)). On subsequent launches, a returning user is shown the login screen (Figure 5.1(d)) directly. Both the registration and login screens perform inline validation – for example, the registration screen rejects an invalid e-mail address or a password shorter than eight characters, and the login screen rejects credentials that do not match a record in the users table – before allowing the user into the main application.

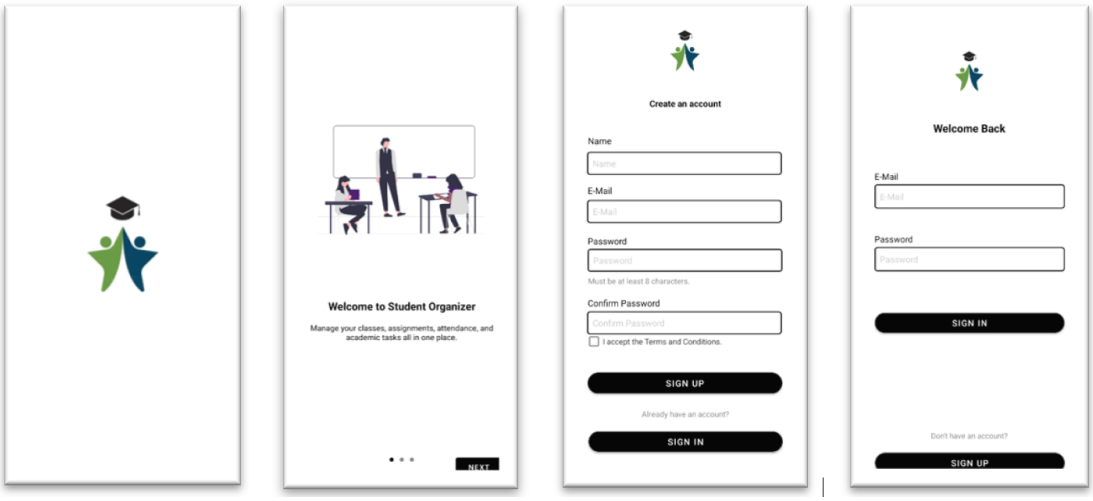


Figure 5.1: Onboarding and authentication screens – (a) Splash (b) Onboarding (c) Register (d) Login

5.3 Home Dashboard and Course Tracking Module

After signing in, the user lands on the Home tab (Figure 5.2(a)), which greets the user by name and lists enrolled courses with a linear progress indicator for each. The Course tab (Figure 5.2(b)) presents the complete list of enrolled courses in the same card-based style, allowing the user to scroll through every course rather than only the summary shown on the home dashboard.

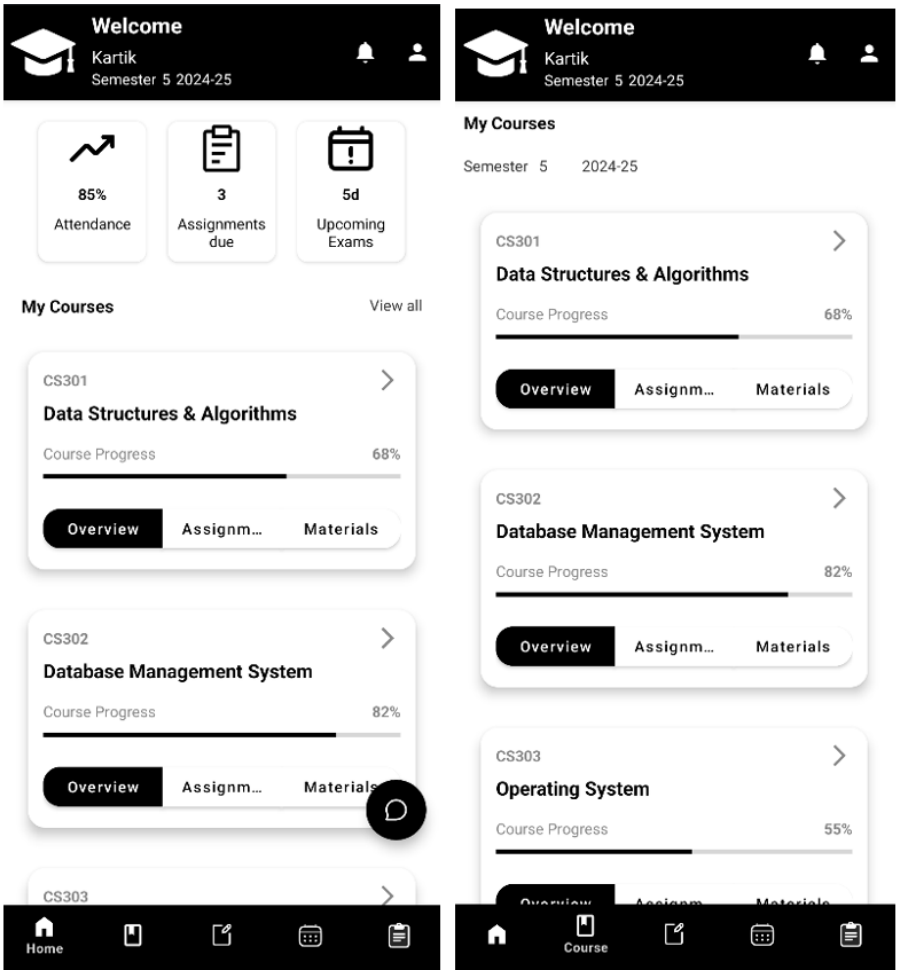


Figure 5.2: Home dashboard and course tracking screens – (a) Home Dashboard (b) Course Tracking

5.4 Notes Management Module

The Notes tab (Figure 5.3) lists every note created by the user, showing the title, subject and a short content preview, along with the date it was last modified. Tapping a note opens it for editing, a long press offers the option to delete it, and the floating action button in the bottom-right corner opens a dialog for creating a new note. All three operations write through to the notes table in NotesDatabase via NotesDao, and the list refreshes automatically once the underlying coroutine completes.

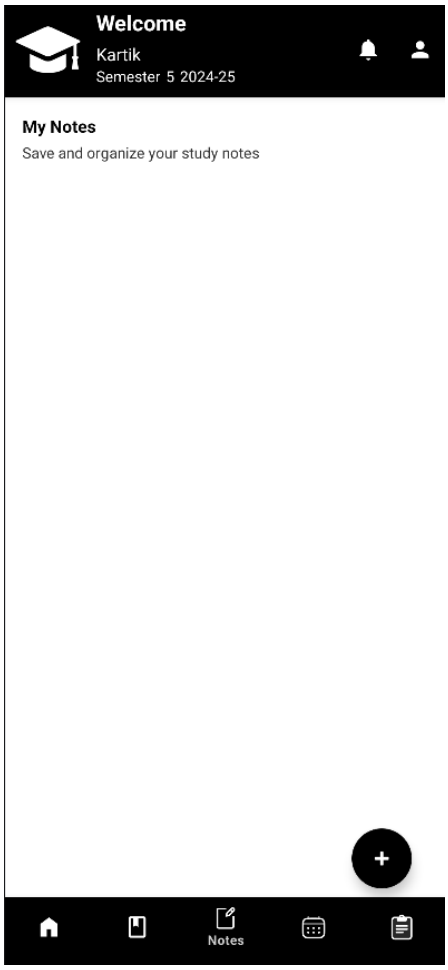


Figure 5.3: Notes management screen

5.5 Assignment Tracking Module

The Assignment tab (Figure 5.4) lists assignments together with the subject they belong to, their due date, and a coloured status pill indicating whether the assignment is Pending or Completed. This view allows a student to quickly identify which assignments are approaching their due date.

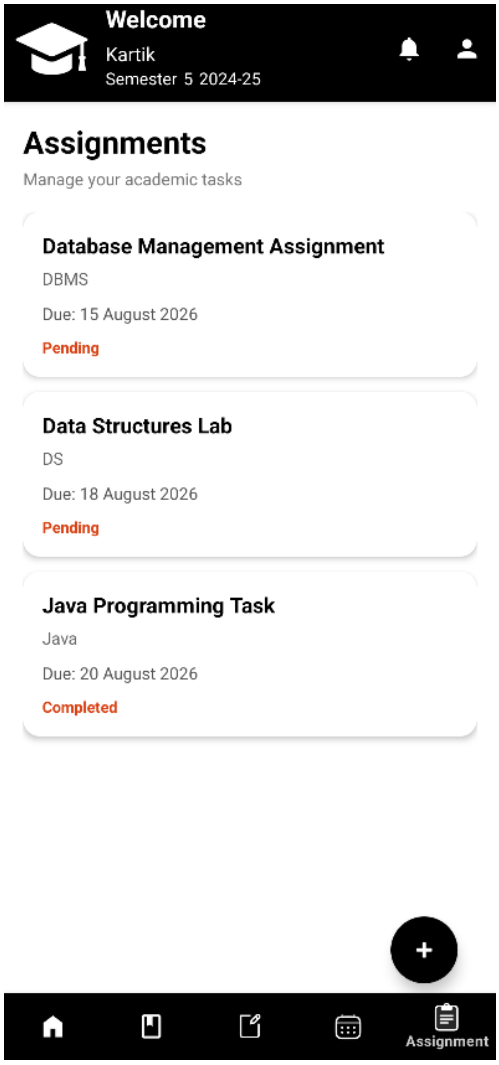


Figure 5.4: Assignment tracking screen

5.6 Calendar Module

The Calendar tab (Figure 5.5) presents a month view built around Android's CalendarView, with the current date highlighted, and a schedule list beneath it showing the events planned for the date currently selected.

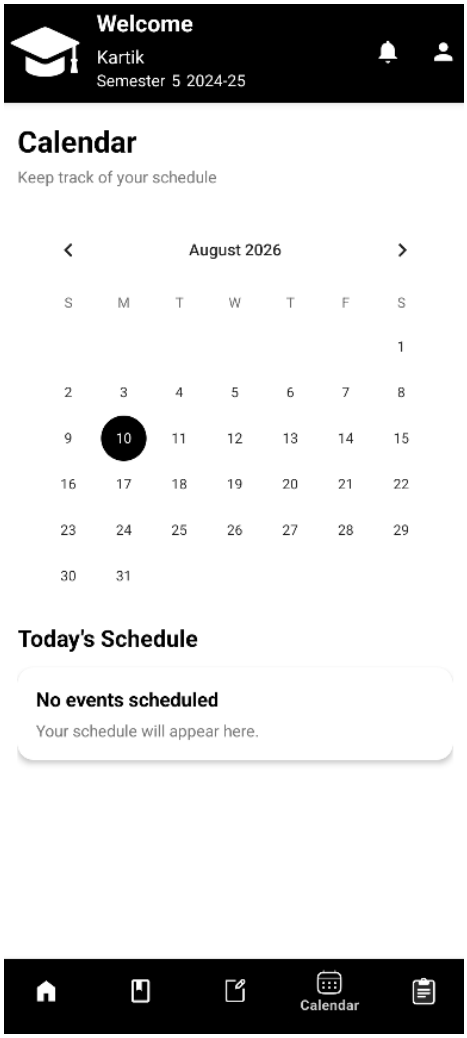


Figure 5.5: Calendar screen

5.7 Profile Module

The Profile tab (Figure 5.6) displays the signed-in user's name and e-mail address along with quick links to edit the profile, manage notification settings, view information about the application, and reset the session or log out, which clears the isLoggedIn flag held in SharedPreferences and returns the user to the login screen.

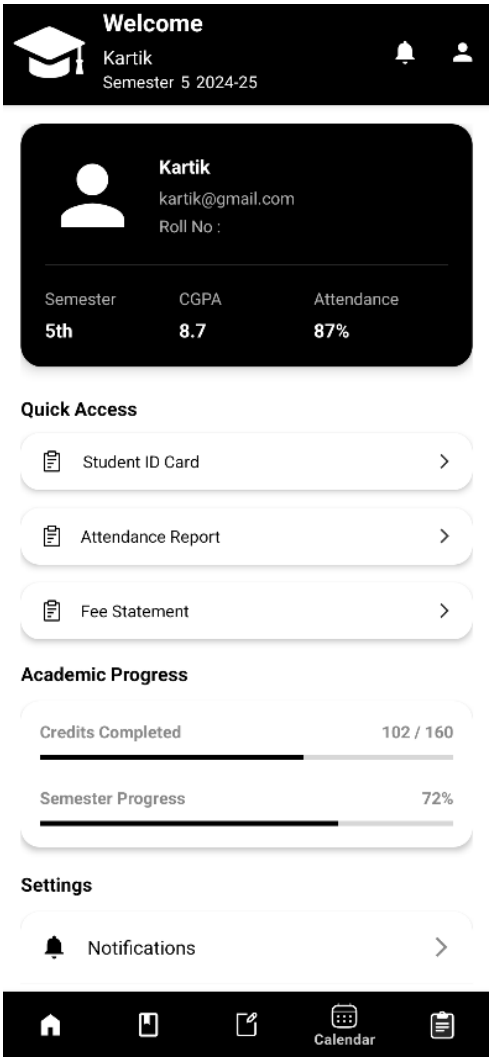


Figure 5.6: Profile screen

5.8 Summary of Functional Testing

The application was tested manually on an Android emulator running API level 33, covering the principal user flows of each module. A summary of the tests carried out is given in Table 5.1.

Table 5.1: Summary of functional testing carried out on the application

Module	Test Performed	Expected Result	Status
Splash	Launch app with no prior session	Routed to onboarding flow	Pass
Splash	Launch app with an active session	Routed directly to Home tab	Pass
Register	Submit form with an invalid e-mail address	Inline error shown; record not inserted	Pass
Register	Submit form with a valid, unique e-mail address	New row inserted into users table	Pass
Login	Sign in with correct credentials	isLoggedIn flag set; navigated to MainActivity	Pass
Login	Sign in with incorrect password	Error message shown; user remains on screen	Pass
Notes	Add a new note	New row inserted into notes table; list updates	Pass
Notes	Edit an existing note	Row updated in notes table; list reflects change	Pass
Notes	Delete a note (long press)	Row removed from notes table; list updates	Pass
Course	Scroll course list	All enrolled courses displayed with progress bars	Pass
Assignment	View assignment list	Assignments shown with correct status pill colour	Pass
Calendar	Select a date on the calendar	Schedule list below updates for the selected date	Pass
Profile	Tap Reset / Logout	Session flag cleared; user returned to Login screen	Pass

5.9 Discussion

The results obtained show that the core objectives set out in Section 4.2 were met: a student can register, sign in, view course progress, manage notes with full CRUD functionality, track assignments, consult a calendar, and manage a profile, entirely offline. The use of Room together with Kotlin Coroutines proved effective in keeping the user interface responsive, since every database operation observed during testing completed asynchronously without any visible delay on the emulator. Maintaining two separate Room databases (AppDatabase and NotesDatabase) simplified the initial implementation, as each module could be developed largely independently, but it also means that user-specific note ownership is not currently enforced at the database level – a limitation that is addressed as future scope in Section 6.2. Overall, the training and the resulting project provided practical, end-to-end experience of building, persisting data for, and testing a multi-module native Android application.

CHAPTER 6: CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

This training provided practical, hands-on experience of the complete life cycle of native Android application development – from learning the Kotlin language and the Android application framework, through hands-on practical exercises covering local persistence, list-based UI components and fragment-based navigation, to the design, implementation and testing of a complete, multi-module application. The resulting project, Student Organizer, successfully brings together account-based authentication, a course-progress dashboard, a full-featured notes module, assignment tracking, a calendar-based schedule view, and a profile module within a single, offline-first Android application built with Kotlin and the Room persistence library. Manual functional testing, summarised in Table 5.1, confirmed that each of these modules behaves correctly across the principal use cases exercised. Beyond the application itself, the training strengthened the trainee's understanding of layered application design, asynchronous programming with coroutines, and the use of industry-standard tooling such as Android Studio, Gradle and Git, all of which are directly transferable to future academic and professional software development work.

6.2 Future Scope

While the current version of Student Organizer meets its stated objectives, several enhancements could extend its usefulness and are identified as scope for future work:

- Cloud synchronisation – integrating a backend service (for example, Firebase Authentication and Cloud Firestore) so that a user's data is backed up and made available across multiple devices.
- Unified database – consolidating AppDatabase and NotesDatabase into a single Room database with a foreign-key relationship between users and notes, so that notes, courses and assignments are explicitly scoped to the signed-in user.
- Password security – replacing the current plain-text password storage with a salted-hash mechanism (for example, using the Android Keystore or a library such as BCrypt) before the application is considered for any real-world deployment.
- Reminders and notifications – using WorkManager and the Notification API to alert the user ahead of assignment due dates and scheduled events.
- Automatic timetable generation – allowing a recurring weekly timetable to be defined once and automatically populated into the calendar module.
- Attachments – allowing files or images to be attached to notes and assignments.
- Dark theme and accessibility improvements – extending the current Material theme with a dark mode and improved support for screen readers.
- Automated testing – introducing JUnit and Espresso test suites to complement the manual testing summarised in Chapter 5.

REFERENCES

[1] DUCAT, Official Website – S O Infotech Pvt. Ltd. (DUCAT). Available: https://ducatindia.com/

[2] ClearTax, S O Infotech Private Limited – Company Information. Available: https://cleartax.in/f/company/s-o-infotech-private-limited/U72120DL2001PTC109989/

[3] J.C. Bose University of Science and Technology, Training/Internship Records, S O Infotech (P) Ltd., A-52, Sector 16, Noida. Available: https://www.jcboseust.ac.in/assets/computer_application/uploads/media/MCA_Training_2021.pdf

[4]Android Developers, "Kotlin and Android," Google. Available: https://developer.android.com/kotlin

[5] Android Developers, "View Binding," Google. Available: https://developer.android.com/topic/libraries/view-binding

[6] Android Developers, "Save data in a local database using Room," Google. Available: https://developer.android.com/training/data-storage/room

[7] Android Developers, "Kotlin Coroutines on Android," Google. Available: https://developer.android.com/kotlin/coroutines

[8] Android Developers, "Meet Android Studio," Google. Available: https://developer.android.com/studio/intro

[9] Android Developers, "Create dynamic lists with RecyclerView," Google. Available: https://developer.android.com/develop/ui/views/layout/recyclerview

[10] Android Developers, "ViewPager2," Google. Available: https://developer.android.com/develop/ui/views/animations/vp2-migration

[11] Android Developers, "Material Components for Android," Google. Available: https://github.com/material-components/material-components-android

APPENDIX

This appendix contains representative source-code listings from the Student Organizer application, referred to in Chapters 3 and 4.

A.1 Entity Classes

```
@Entity(tableName = "users")
data class User(
@PrimaryKey(autoGenerate = true) val id: Int = 0,
val name: String,
val email: String,
val password: String
)

@Entity(tableName = "notes")
data class Notes(
@PrimaryKey(autoGenerate = true) val id: Int = 0,
val title: String,
val subject: String,
val content: String,
val date: String
)
```

A.2 Data Access Object (DAO) Interfaces

```
@Dao
interface UserDao {
    @Insert
    suspend fun insertUser(user: User)

    @Query("SELECT * FROM users WHERE email = :email AND password = :password")
    suspend fun loginUser(email: String, password: String): User?
}

@Dao
interface NotesDao {
    @Insert
    suspend fun insertNote(note: Notes)
    @Update
    suspend fun updateNote(note: Notes)
    @Delete
    suspend fun deleteNote(note: Notes)
    @Query("SELECT * FROM notes ORDER BY id DESC")
    suspend fun getAllNotes(): List<Notes>
}
```

A.3 RecyclerView Adapter (Course List) – Excerpt

```
class CourseAdapter(private val courses: List<Course>) :
    RecyclerView.Adapter<CourseAdapter.CourseViewHolder>() {

    inner class CourseViewHolder(val binding: ItemCourseBinding) :
        RecyclerView.ViewHolder(binding.root)

    override fun onBindViewHolder(holder: CourseViewHolder, position: Int) {
        val course = courses[position]
        holder.binding.tvCourseCode.text = course.code
        holder.binding.tvCourseName.text = course.name
        holder.binding.progressBar.progress = course.progress
    }
}
```