Table of Contents

[1. Project Synopsis 2](#_Toc164231655)

[1.1 Purpose: 2](#_Toc164231656)

[1.2 Goals: 2](#_Toc164231657)

[1.3 Intended Audience: 2](#_Toc164231658)

[2. Project Objectives 3](#_Toc164231659)

[2.1 [Objective 1] 3](#_Toc164231660)

[2.2 [Objective 2] 3](#_Toc164231661)

[2.3 [Objective 3] 3](#_Toc164231662)

[3. Project Outline 4](#_Toc164231663)

[3.1 [Module 1] 4](#_Toc164231664)

[3.2 [Module 2] 4](#_Toc164231665)

[3.3 [Module 3] 4](#_Toc164231666)

[3.4 Flow Diagram: 4](#_Toc164231667)

[4. Technologies Used 5](#_Toc164231668)

[5. Screenshots 6](#_Toc164231669)

[6. Project Features 7](#_Toc164231670)

[6.1 Feature 1: [Description] 7](#_Toc164231671)

[6. 2 Feature 2: [Description] 7](#_Toc164231672)

[6. 3 Feature 3: [Description] 7](#_Toc164231673)

[7. User Guide 8](#_Toc164231674)

[8. Conclusion 9](#_Toc164231675)

[9. References 10](#_Toc164231676)

# **1.PROJECT SYNOPSIS**

## The **College Companion Chatbot App** is an Android-based mobile application designed to assist students in navigating college-related information with ease and efficiency. This app utilizes **Google’s Gemini AI model** to provide intelligent and natural language responses to user queries regarding college facilities, departments, events, and more.

## The primary goal of the application is to **streamline access to college information** by eliminating the need to search through multiple websites or physical notice boards. By integrating modern AI capabilities with a user-friendly interface, the app aims to **enhance student experience and reduce the information gap** within the college environment.

## **1.1 Purpose:**

The main purpose of the **College Companion Chatbot App** is to provide students with a reliable and intelligent virtual assistant capable of answering college-related questions in real-time. Traditional methods of accessing information — such as bulletin boards, printed manuals, or navigating large websites — are often time-consuming and inefficient.

This mobile application bridges that gap by using a **smart chatbot powered by Google’s Gemini AI model**, which can understand and respond to natural language queries. Whether a student wants to know about the college's canteen options, gym timings, staffroom locations, or upcoming events, the chatbot delivers fast and relevant answers in a conversational way.

Additionally, the app supports:

* **Secure student authentication** via Firebase
* **User-friendly chat interface**
* **Contextual responses** based on student queries
* **Dynamic and scalable architecture** for future enhancements

By integrating cutting-edge AI with intuitive mobile design, the app ensures a **modern, accessible, and efficient** solution tailored to the needs of students and academic institutions

The main purpose of the **College Companion Chatbot App** is to provide students with a reliable and intelligent virtual assistant capable of answering college-related questions in real-time. Traditional methods of accessing information — such as bulletin boards, printed manuals, or navigating large websites — are often time-consuming and inefficient.

This mobile application bridges that gap by using a **smart chatbot powered by Google’s Gemini AI model**, which can understand and respond to natural language queries. Whether a student wants to know about the college's canteen options, gym timings, staffroom locations, or upcoming events, the chatbot delivers fast and relevant answers in a conversational way.

Additionally, the app supports:

* **Secure student authentication** via Firebase
* **User-friendly chat interface**
* **Contextual responses** based on student queries
* **Dynamic and scalable architecture** for future enhancements

By integrating cutting-edge AI with intuitive mobile design, the app ensures a **modern, accessible, and efficient** solution tailored to the needs of students and academic institutions

## **1.2 Goals:**

## **Goal 1:** To develop an AI-powered chatbot that can assist students with queries related to campus facilities, schedules, and general college information.

## **Goal 2:** To provide a simple and secure mobile application interface that supports student authentication and smooth chat experience.

## **Goal 3:** To reduce manual workload for faculty or help desk staff by automating frequently asked questions using the Gemini API integration.

## **1.3 Intended Audience:**

The primary target audience for this mobile application is **college students**, especially those who are new to the campus and require quick assistance regarding college facilities, departments, event schedules, or general inquiries.

# **2.PROJECT OBJECTIVES**

The project aims to achieve the following objectives:

* **Integrate Gemini API** to provide dynamic and context-aware responses to user queries.
* **Enable real-time chatbot interaction** to assist students with information about the college.
* **Allow secure login and authentication** using Firebase for personalized experiences.
* **Fetch and display college-related information** like departments, events, and facilities.
* **Provide a clean, intuitive UI/UX** using Android Studio and XML layouts for ease of navigation.
* **Enhance accessibility** to college resources for both students and staff through a centralized app

## **2.1** **Easy Access to College Information**

## The primary objective is to simplify the way students access college-related information, such as departments, events, canteens, and faculty details, by providing a chatbot-based interface within the app

## **2.2 Real-Time Interaction via Gemini API**

## The chatbot is powered by Google's Gemini API, which allows for real-time, intelligent responses to user queries. This ensures that students receive accurate and context-aware information without manual search.

**2.3 Secure and Personalized User Experience**

By integrating Firebase Authentication, the application ensures that users have a secure login process. This also allows for a more personalized experience based on user activity and roles within the college.

# **3.PROJECT OUTLINE**

**1. Core Components**

* Data Layer: Room Database, User Tracking, Chat History
* User Interface: Welcome, Auth, Chat, Profile, History screens
* Business Logic: Chat Processing, AI Integration, User Management
* External Services: Firebase, Gemini AI, Authentication

**2. Technical Implementation**

* Development: Android Studio, Kotlin, Gradle
* Architecture: Hilt, Room, ViewModel, LiveData, Coroutines
* UI: Activities, Material Design, RecyclerView
* Data: Local Database, Network Operations, Caching

**3. Features**

* Authentication: Registration, Login, Session Management
* Chat System: AI Processing, History Tracking, Quick Responses
* User Management: Profile, Settings, Privacy
* Analytics: Usage Tracking, Performance Monitoring

**4. Project Organization**

* Package Structure: data, di, ui, utils, models
* Resources: Layouts, Drawables, Strings, Styles
* Configuration: Build, Manifest, Gradle files

**5. Development & Maintenance**

* Version Control: Git, Branch Management
* Testing: Unit, Integration, UI Testing
* Deployment: Build, Release Management
* Documentation: Technical, User Guides
* Updates: Version Control, Security Patches
* Optimization: Performance, Security, Data Protection

**Modules:**

## **3.1 Data Module**

## **Purpose**: Manages all data-related operations and persistence.

## **Components**:

## **Room Database**: Handles local data storage.

## **Entities**: Defines data models like UserTrackingEntity.

## **DAOs**: Provides interfaces for database operations, e.g., UserTrackingDao.

## **Repositories**: Implements business logic and data handling, e.g., UserTrackingRepository.

## **3.2 UI Module**

## **Purpose**: Manages the user interface and user interactions.

## **Components**:

## **Activities**: Screens like WelcomeActivity, LoginActivity, ChatActivity.

## **Layouts**: XML files defining the UI structure.

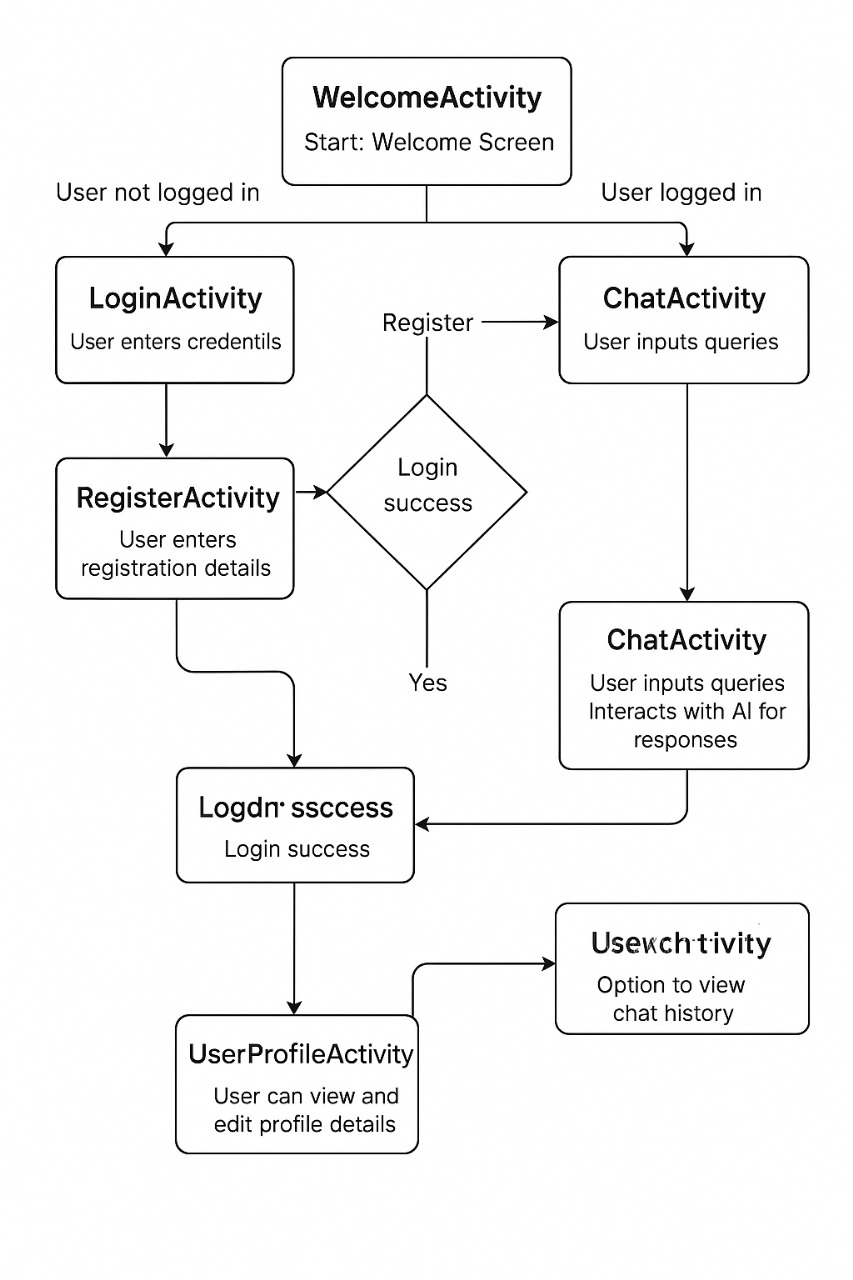
## **Adapters**: Manages data display in lists, e.g., ChatAdapter.

## **ViewModels**: Handles UI-related data and lifecycle awareness.

**3.3 Core Logic Module**

* **Purpose**: Implements the core business logic and application functionality.
* **Components**:
* **Chat Processing**: Manages AI interactions via GeminiChatHelper.
* **Authentication**: Handles user login and registration.
* **Dependency Injection**: Manages component lifecycles and dependencies using Hilt.
* **Network Operations**: Manages API calls and data fetching.

## 3.4 Flow Diagram:



# **4.TECHNOLOGIES USED**

**Programming Language**

* **Kotlin**: The primary programming language used for Android development, known for its concise syntax and interoperability with Java.

**Development Platform**

* **Android Studio**: The official integrated development environment (IDE) for Android app development, providing tools for coding, debugging, and testing.
* **Android SDK**: A collection of tools and libraries necessary for developing Android applications.

**Database**

* **Room Database**: A persistence library that provides an abstraction layer over SQLite, allowing for more robust database access while harnessing the full power of SQLite.
* **SQLite**: The underlying database engine used by Room for local data storage.

**Other Tools**

* **Firebase**: A platform developed by Google for creating mobile and web applications, used for authentication, real-time database, and cloud storage.
* **Hilt**: A dependency injection library for Android that reduces the boilerplate of doing manual dependency injection in your project.
* **Jetpack Components**: A suite of libraries to help developers follow best practices, reduce boilerplate code, and write code that works consistently across Android versions and devices.
* **Material Design**: A design language developed by Google, used for creating a consistent and intuitive user interface.
* **Gradle**: A build automation tool used to manage dependencies and build configurations in the project.
* **Git**: A version control system used for tracking changes in the project and collaborating with other developers.
* **Coroutines**: A Kotlin feature for asynchronous programming, used to simplify code that performs long-running tasks.
* **LiveData**: A lifecycle-aware data holder class that can be observed within a given lifecycle, used for UI updates based on data changes.

**5. SAMPLE CODES**

* **GeminiApiclient:**

package com.example.mobileapp.network

import android.util.Log

import com.example.mobileapp.BuildConfig

import okhttp3.\*

import okhttp3.MediaType.Companion.toMediaType

import okhttp3.RequestBody.Companion.toRequestBody

import org.json.JSONArray

import org.json.JSONObject

import java.io.IOException

import java.util.concurrent.TimeUnit

object GeminiApiClient {

private const val TAG = "GeminiApiClient"

private const val BASE\_URL = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent"

private val apiKey = BuildConfig.GEMINI\_API\_KEY

private val client: OkHttpClient by lazy {

OkHttpClient.Builder()

.connectTimeout(15, TimeUnit.SECONDS)

.readTimeout(15, TimeUnit.SECONDS)

.writeTimeout(15, TimeUnit.SECONDS)

.build()

}

fun getChatResponse(query: String, callback: (String) -> Unit) {

if (apiKey.isEmpty()) {

Log.e(TAG, "❌ API Key is missing")

callback("❌ API key not configured")

return

}

// Build JSON payload

val partsArray = JSONArray().apply {

put(JSONObject().put("text", query))

}

val messageObject = JSONObject().apply {

put("role", "user")

put("parts", partsArray)

}

val requestBodyJson = JSONObject().apply {

put("contents", JSONArray().put(messageObject))

}

val mediaType = "application/json".toMediaType()

val requestBody = requestBodyJson.toString().toRequestBody(mediaType)

val request = Request.Builder()

.url("$BASE\_URL?key=$apiKey")

.post(requestBody)

.addHeader("Content-Type", "application/json")

.build()

client.newCall(request).enqueue(object : Callback {

override fun onFailure(call: Call, e: IOException) {

Log.e(TAG, "❌ Request failed: ${e.message}")

callback("❌ Request error: ${e.message}")

}

override fun onResponse(call: Call, response: Response) {

val bodyString = response.body?.string().orEmpty()

if (!response.isSuccessful) {

Log.e(TAG, "❌ Error ${response.code}: $bodyString")

callback("❌ Error ${response.code}: $bodyString")

return

}

try {

val json = JSONObject(bodyString)

val reply = json.getJSONArray("candidates")

.getJSONObject(0)

.getJSONObject("content")

.getJSONArray("parts")

.getJSONObject(0)

.getString("text")

callback(reply)

} catch (e: Exception) {

Log.e(TAG, "❌ JSON parsing failed: ${e.message}")

callback("❌ Parsing error")

}

}

})

}

}

* **Registeractivity.kt:**

package com.example.mobileapp

import android.content.Intent

import android.os.Bundle

import android.os.Handler

import android.os.Looper

import android.widget.Button

import android.widget.TextView

import androidx.appcompat.app.AppCompatActivity

import com.example.mobileapp.utils.AnimationUtils

import com.google.android.material.textfield.TextInputLayout

class RegisterActivity : AppCompatActivity() {

private lateinit var nameLayout: TextInputLayout

private lateinit var emailLayout: TextInputLayout

private lateinit var passwordLayout: TextInputLayout

private lateinit var confirmPasswordLayout: TextInputLayout

private lateinit var registerButton: Button

private lateinit var loginText: TextView

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_register)

// Initialize views

nameLayout = findViewById(R.id.nameLayout)

emailLayout = findViewById(R.id.emailLayout)

passwordLayout = findViewById(R.id.passwordLayout)

confirmPasswordLayout = findViewById(R.id.confirmPasswordLayout)

registerButton = findViewById(R.id.registerButton)

loginText = findViewById(R.id.loginText)

// Play initial animations with delays

Handler(Looper.getMainLooper()).apply {

postDelayed({ AnimationUtils.playFadeInAnimation(nameLayout) }, 200)

postDelayed({ AnimationUtils.playFadeInAnimation(emailLayout) }, 400)

postDelayed({ AnimationUtils.playFadeInAnimation(passwordLayout) }, 600)

postDelayed({ AnimationUtils.playFadeInAnimation(confirmPasswordLayout) }, 800)

postDelayed({ AnimationUtils.playSlideUpAnimation(registerButton) }, 1000)

postDelayed({ AnimationUtils.playFadeInAnimation(loginText) }, 1200)

}

// Set click listeners

registerButton.setOnClickListener {

AnimationUtils.playButtonAnimation(it)

validateAndRegister()

}

loginText.setOnClickListener {

AnimationUtils.playButtonAnimation(it)

// Navigate back to login

finish()

overridePendingTransition(R.anim.fade\_in, R.anim.slide\_down)

}

}

private fun validateAndRegister() {

val name = nameLayout.editText?.text.toString()

val email = emailLayout.editText?.text.toString()

val password = passwordLayout.editText?.text.toString()

val confirmPassword = confirmPasswordLayout.editText?.text.toString()

// Reset errors

nameLayout.error = null

emailLayout.error = null

passwordLayout.error = null

confirmPasswordLayout.error = null

// Validate inputs

if (name.isEmpty()) {

AnimationUtils.playShakeAnimation(nameLayout)

nameLayout.error = "Name is required"

return

}

if (email.isEmpty()) {

AnimationUtils.playShakeAnimation(emailLayout)

emailLayout.error = "Email is required"

return

}

if (password.isEmpty()) {

AnimationUtils.playShakeAnimation(passwordLayout)

passwordLayout.error = "Password is required"

return

}

if (confirmPassword.isEmpty()) {

AnimationUtils.playShakeAnimation(confirmPasswordLayout)

confirmPasswordLayout.error = "Please confirm your password"

return

}

if (password != confirmPassword) {

AnimationUtils.playShakeAnimation(confirmPasswordLayout)

confirmPasswordLayout.error = "Passwords do not match"

return

}

// TODO: Implement actual registration logic

AnimationUtils.playSuccessAnimation(registerButton)

Handler(Looper.getMainLooper()).postDelayed({

startActivity(Intent(this, ChatActivity::class.java))

finish()

}, 1000)

}

}

* **Chatactivity.kt:**

package com.example.mobileapp

import android.content.Intent

import android.os.Bundle

import android.view.Menu

import android.view.MenuItem

import android.view.View

import android.widget.EditText

import android.widget.ProgressBar

import android.widget.Toast

import androidx.appcompat.app.AppCompatActivity

import androidx.lifecycle.lifecycleScope

import androidx.recyclerview.widget.LinearLayoutManager

import androidx.recyclerview.widget.RecyclerView

import com.google.android.material.floatingactionbutton.FloatingActionButton

import com.google.firebase.auth.FirebaseAuth

import dagger.hilt.android.AndroidEntryPoint

import kotlinx.coroutines.Dispatchers

import kotlinx.coroutines.launch

import kotlinx.coroutines.withContext

import javax.inject.Inject

import com.example.mobileapp.data.UserTrackingRepository

import java.util.Date

@AndroidEntryPoint

class ChatActivity : AppCompatActivity() {

private lateinit var messageInput: EditText

private lateinit var sendButton: FloatingActionButton

private lateinit var recyclerView: RecyclerView

private lateinit var progressBar: ProgressBar

private lateinit var chatAdapter: ChatAdapter

private val chatMessages = mutableListOf<ChatMessage>()

@Inject

lateinit var geminiChatHelper: GeminiChatHelper

@Inject

lateinit var userTrackingRepository: UserTrackingRepository

private val auth = FirebaseAuth.getInstance()

private val quickOptions = mapOf(

"1" to "Tell me about the college programs and courses",

"2" to "What are the admission requirements and fees?",

"3" to "What campus facilities are available?",

"4" to "Tell me about placements and achievements",

"5" to "Information about faculty and departments"

)

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_chat)

// Check if user is logged in

if (auth.currentUser == null) {

Toast.makeText(this, "Please login first", Toast.LENGTH\_LONG).show()

finish()

return

}

// Initialize views

messageInput = findViewById(R.id.messageInput)

sendButton = findViewById(R.id.sendButton)

recyclerView = findViewById(R.id.recyclerView)

progressBar = findViewById(R.id.progressBar)

// Setup RecyclerView

chatAdapter = ChatAdapter(chatMessages)

recyclerView.apply {

layoutManager = LinearLayoutManager(this@ChatActivity).apply {

stackFromEnd = true // Messages start from bottom

}

adapter = chatAdapter

}

// Add scroll listener to handle loading more messages

recyclerView.addOnScrollListener(object : RecyclerView.OnScrollListener() {

override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {

super.onScrolled(recyclerView, dx, dy)

if (dy < 0) { // Scrolling up

loadMoreMessages()

}

}

})

// Send button click listener

sendButton.setOnClickListener {

val message = messageInput.text.toString().trim()

if (message.isNotEmpty()) {

handleUserInput(message)

messageInput.text.clear()

} else {

Toast.makeText(this, "Please enter a message", Toast.LENGTH\_SHORT).show()

}

}

// Add test entry to verify database

lifecycleScope.launch {

try {

val userId = auth.currentUser?.uid ?: return@launch

userTrackingRepository.trackUserQuery(

userId = userId,

query = "Test query",

response = "Test response",

sessionDuration = 1000,

category = "test"

)

Toast.makeText(this@ChatActivity, "Test entry added to database", Toast.LENGTH\_SHORT).show()

} catch (e: Exception) {

Toast.makeText(this@ChatActivity, "Error: ${e.message}", Toast.LENGTH\_LONG).show()

}

}

// Welcome message

addMessage(ChatMessage("""

👋 Welcome to Kristu Jayanti College Assistant!

I'm here to help you with information. Choose a number or ask any question:

1️⃣ College programs and courses

2️⃣ Admissions and fees

3️⃣ Campus facilities

4️⃣ Placements and achievements

5️⃣ Faculty and departments

You can either:

• Type a number (1-5) for quick access

• Or ask your question in detail

How can I assist you today?

""".trimIndent(), isUser = false))

}

override fun onCreateOptionsMenu(menu: Menu): Boolean {

menuInflater.inflate(R.menu.chat\_menu, menu)

return true

}

override fun onOptionsItemSelected(item: MenuItem): Boolean {

return when (item.itemId) {

R.id.action\_history -> {

startActivity(Intent(this, UserHistoryActivity::class.java))

true

}

else -> super.onOptionsItemSelected(item)

}

}

private fun handleUserInput(message: String) {

// Add user message to chat

addMessage(ChatMessage(message, isUser = true))

// Convert numeric input to corresponding query

val actualQuery = if (message.matches(Regex("^[1-5]$"))) {

quickOptions[message] ?: message

} else {

message

}

// Show progress and disable send button

progressBar.visibility = View.VISIBLE

sendButton.isEnabled = false

// Get current user ID

val userId = auth.currentUser?.uid ?: run {

Toast.makeText(this, "Please login first", Toast.LENGTH\_LONG).show()

progressBar.visibility = View.GONE

sendButton.isEnabled = true

return

}

// Send message to Gemini

lifecycleScope.launch {

geminiChatHelper.sendMessage(actualQuery, userId, this@ChatActivity) { response ->

progressBar.visibility = View.GONE

sendButton.isEnabled = true

addMessage(response)

}

}

}

private fun addMessage(message: ChatMessage) {

chatMessages.add(message)

chatAdapter.notifyItemInserted(chatMessages.size - 1)

recyclerView.smoothScrollToPosition(chatMessages.size - 1)

}

private fun loadMoreMessages() {

// TODO: Implement loading more messages from database

}

}

* **LoginActivity.kt:**

package com.example.mobileapp

import android.content.Intent

import android.os.Bundle

import android.os.Handler

import android.os.Looper

import android.widget.Button

import android.widget.EditText

import android.widget.TextView

import androidx.appcompat.app.AppCompatActivity

import com.example.mobileapp.utils.AnimationUtils

import com.google.android.material.textfield.TextInputLayout

class LoginActivity : AppCompatActivity() {

private lateinit var emailLayout: TextInputLayout

private lateinit var passwordLayout: TextInputLayout

private lateinit var loginButton: Button

private lateinit var registerText: TextView

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_login)

// Initialize views

emailLayout = findViewById(R.id.emailLayout)

passwordLayout = findViewById(R.id.passwordLayout)

loginButton = findViewById(R.id.loginButton)

registerText = findViewById(R.id.registerText)

// Play initial animations

Handler(Looper.getMainLooper()).postDelayed({

AnimationUtils.playFadeInAnimation(emailLayout)

}, 300)

Handler(Looper.getMainLooper()).postDelayed({

AnimationUtils.playFadeInAnimation(passwordLayout)

}, 500)

Handler(Looper.getMainLooper()).postDelayed({

AnimationUtils.playSlideUpAnimation(loginButton)

}, 700)

Handler(Looper.getMainLooper()).postDelayed({

AnimationUtils.playFadeInAnimation(registerText)

}, 900)

// Set click listeners

loginButton.setOnClickListener {

AnimationUtils.playButtonAnimation(it)

validateAndLogin()

}

registerText.setOnClickListener {

AnimationUtils.playButtonAnimation(it)

startActivity(Intent(this, RegisterActivity::class.java))

overridePendingTransition(R.anim.slide\_up, R.anim.fade\_out)

}

}

private fun validateAndLogin() {

val email = emailLayout.editText?.text.toString()

val password = passwordLayout.editText?.text.toString()

if (email.isEmpty()) {

AnimationUtils.playShakeAnimation(emailLayout)

emailLayout.error = "Email is required"

return

}

if (password.isEmpty()) {

AnimationUtils.playShakeAnimation(passwordLayout)

passwordLayout.error = "Password is required"

return

}

// TODO: Implement actual login logic

AnimationUtils.playSuccessAnimation(loginButton)

Handler(Looper.getMainLooper()).postDelayed({

startActivity(Intent(this, ChatActivity::class.java))

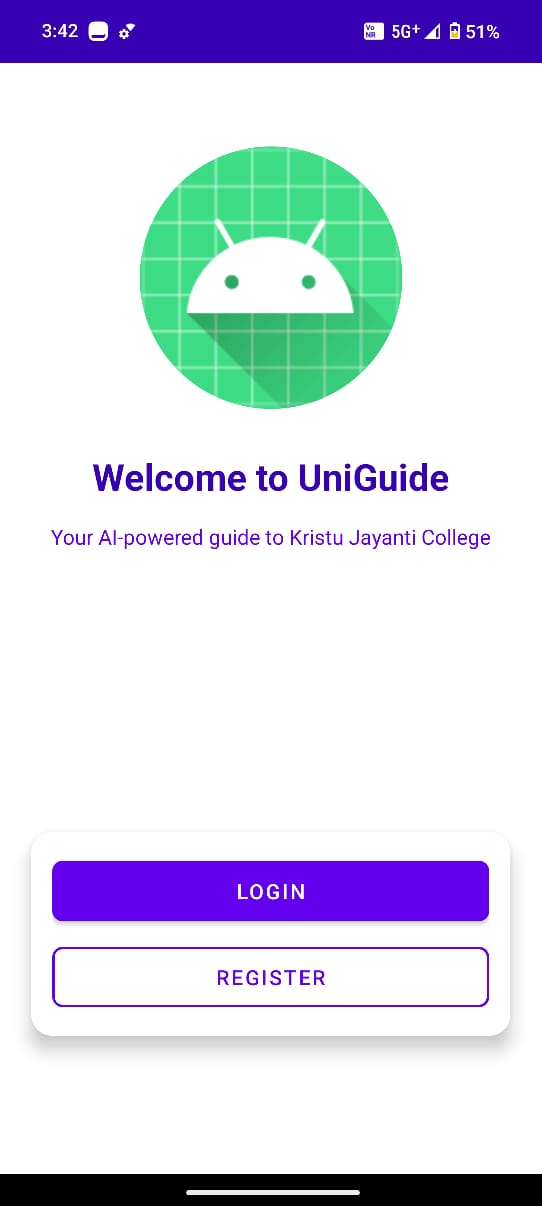
finish()

}, 1000)

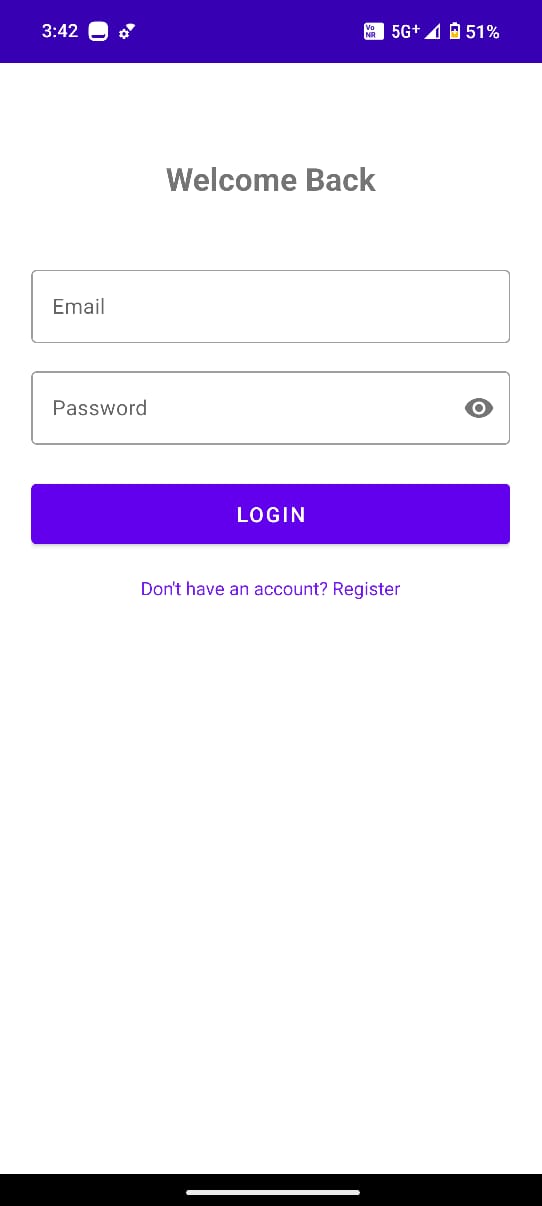
}

}

# **6. SCREENSHOTS**

 A screenshot of a login form

AI-generated content may be incorrect.

 A screenshot of a computer

AI-generated content may be incorrect.

# 

A screenshot of a chat

AI-generated content may be incorrect.

# **7.PROJECT FEATURES**

**7.1 Chat Interface with AI Integration**

* **Description**: The application provides a chat interface where users can interact with an AI assistant powered by Google's Gemini AI. Users can ask questions about Kristu Jayanti College, and the AI provides detailed, well-formatted responses.
* **Key Functionalities**:
* Real-time message processing
* Quick response options for common queries
* Formatted responses with highlights and bullet points
* Error handling and user feedback
* Session tracking and analytics

**7.2 User Authentication and Profile Management**

* **Description**: A secure authentication system that allows users to register, log in, and manage their profiles. The system uses Firebase Authentication for secure user management.
* **Key Functionalities**:
* User registration with email and password
* Secure login process
* Profile management and customization
* Session persistence
* Password recovery options
* User data privacy and security

**7.3 Chat History and Analytics**

* **Description**: The application maintains a comprehensive history of user interactions and provides analytics about chat usage. This feature helps users track their queries and responses over time.
* **Key Functionalities**:
* Local storage of chat history using Room Database
* View past interactions with timestamps
* Analytics on query categories and response times
* Export and backup options
* Search functionality within chat history
* Performance metrics and usage statistics

# **8.USER GUIDE**

**Installation Instructions**

**Step 1: Set Up Development Environment**

* **Description**: Prepare your development environment for building the application.
* **Instructions**:

1. Download and install Android Studio from the official website.
2. Install the necessary SDKs and tools through the SDK Manager in Android Studio.
3. Ensure you have the latest version of Kotlin and Gradle installed.

**Step 2: Clone the Project Repository**

* **Description**: Obtain the source code for the project.
* **Instructions**:

1. Use Git to clone the project repository to your local machine.
2. Open the project in Android Studio.
3. Sync the project with Gradle to download all dependencies.

**Step 3: Build and Run the Application**

* **Description**: Compile and run the application on an emulator or physical device.
* **Instructions**:

1. Connect an Android device or start an emulator.
2. Click the "Run" button in Android Studio to build and deploy the application.
3. Monitor the build process and address any errors that arise.

**User Interface Guide**

**Welcome Screen**

* **Description**: The initial screen users will see when they open the application.
* **Features**:
* Options to log in or register.
* Brief introduction to the application's features.
* Navigation to the main chat interface upon successful login.

**Login Screen**

* **Description**: Screen for user authentication.
* **Features**:
* Fields for entering email and password.
* Option to navigate to the registration screen.
* "Login" button to authenticate and proceed to the main interface.

**Registration Screen**

* **Description**: Screen for new users to create an account.
* **Features**:
* Fields for entering user details (e.g., name, email, password).
* Validation for input fields.
* "Register" button to create the account and proceed to login.

**Chat Interface**

* **Description**: The main screen where users will interact with the AI assistant.
* **Features**:
* Text input field for typing questions.
* Send button to submit queries.
* Display area for AI responses.
* Quick response options for common queries.
* Menu options for accessing profile and history.

**Profile Management**

* **Description**: Screen for managing user profile and settings.
* **Features**:
* View and edit profile information.
* Change password and other account settings.
* Logout option.

**History and Analytics**

* **Description**: Screen for viewing past interactions and analytics.
* **Features**:
* List of past chat interactions with timestamps.
* Analytics on query categories and response times.
* Options to export or backup chat history.

**Navigation and Interaction**

* **Description**: How users will navigate between different screens and interact with the application.
* **Instructions**:
* Use the menu options to navigate between screens.
* Tap on buttons and input fields to interact with the application.
* Follow on-screen prompts and instructions for specific actions.

# 

# **9.CONCLUSION**

**Overall Experience**

Developing the Kristu Jayanti College Assistant mobile application has been a rewarding and educational experience. The project provided an opportunity to work with modern Android development tools and practices, integrating various technologies to create a functional and user-friendly application. The use of Kotlin, Room Database, and Firebase services allowed for efficient development and robust functionality.

**Challenges Faced**

**1. Integration of Multiple Technologies**

* **Challenge**: Coordinating the integration of Firebase, Room Database, and Gemini AI services.
* **Solution**: Careful planning and modular design helped manage dependencies and ensure smooth integration.

**2. User Interface Design**

* **Challenge**: Creating an intuitive and responsive user interface that meets user expectations.
* **Solution**: Iterative design and user feedback were crucial in refining the UI to be both functional and aesthetically pleasing.

**3. Data Management**

* **Challenge**: Efficiently managing local and remote data storage and ensuring data consistency.
* **Solution**: Implementing a robust data layer with Room Database and Firebase helped maintain data integrity and performance.

**4. Performance Optimization**

* **Challenge**: Ensuring the application performs well under various conditions and on different devices.
* **Solution**: Profiling and optimizing code, along with using coroutines for asynchronous operations, improved performance.

**Lessons Learned**

**1. Importance of Planning**

* **Lesson**: Thorough planning and design before coding can prevent many issues and streamline development.
* **Application**: Future projects will benefit from more detailed initial planning and requirement gathering.

**2. Value of Testing**

* **Lesson**: Regular testing and feedback are essential for identifying and addressing issues early.
* **Application**: Implementing a more comprehensive testing strategy will be a priority in future projects.

**3. Adaptability**

* **Lesson**: Being flexible and ready to adapt to new technologies and changes is crucial in software development.
* **Application**: Staying updated with the latest tools and practices will continue to be a focus.

**4. Collaboration and Communication**

* **Lesson**: Effective communication and collaboration among team members are vital for project success.
* **Application**: Enhancing team communication and collaboration tools will be a key area for improvement.

**Future Improvements**

* **Enhanced User Experience**: Further refinements to the UI and user interactions.
* **Advanced Features**: Integration of additional features and services to enhance functionality.
* **Scalability**: Ensuring the application can handle increased user load and data volume.
* **Security**: Continuous updates and improvements to security measures to protect user data.

# **10.REFERENCES**

**Official Documentation**

* **Android Developer Documentation**: Comprehensive guides and API references for Android development.
* **Kotlin Documentation**: Official documentation for the Kotlin programming language
* **Firebase Documentation**: Guides and references for Firebase services.
* **Room Database Documentation**: Documentation for the Room persistence library.
* **Hilt Documentation**: Documentation for the Hilt dependency injection library.

**Online Resources**

* **Stack Overflow**: Community-driven Q&A platform for programming questions.
* **GitHub**: Source code hosting and collaboration platform.
* **Medium**: Articles and tutorials on various programming topics.

**Books and Courses**

* **"Android Programming: The Big Nerd Ranch Guide"**: A comprehensive guide to Android development.
* **"Kotlin in Action"**: A book on Kotlin programming.
* **Online Courses**: Platforms like Udacity, Coursera, and Pluralsight offer courses on Android development and Kotlin.

**Tools and Software**

* **Android Studio**: The official IDE for Android development.
* **Git**: Version control system for tracking changes in the project.
* **Gradle**: Build automation tool for managing dependencies and build configurations.