QUIZ GENERATOR APPLICATION

CS19611 - MOBILE APPLICATION DEVELOPMENT LABORATORY

PROJECT REPORT

Submitted by

THARUN R L **(2116220701302)**

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY, CHENNAI

MAY 2025

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Project titled **"QUIZ GENERATOR APPLICATION"** is the bonafide work of **"THARUN R L (2116220701302)"** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. P. Kumar., M.E., Ph.D.,

HEAD OF THE DEPARTMENT

Professor

Department of Computer Science and Engineering,

Rajalakshmi Engineering College, Chennai - 602 105.

SIGNATURE

Dr. N. Duraimurugan., M.E., Ph.D.,

SUPERVISOR

Associate Professor

Department of Computer Science and Engineering,

Rajalakshmi Engineering College, Chennai-602 105.

Submitted to Mini Project Viva-Voce Examination held on _____

Internal Examiner **External Examiner**

ACKNOWLEDGMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and

our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN**, **Ph.D.,** for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. P. KUMAR, M.E., Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide **Dr. N. DURAIMURUGAN**, We are very glad to thank our Project Coordinator, **Dr. N. DURAIMURUGAN** Associate Professor Department of Computer Science and Engineering for his useful tips during our review to build our project.

THARUN R L **2116220701302**

ABSTRACT

The Quiz Generator with Gemini API is an Android application developed in Kotlin, designed to streamline the creation of subject-specific quizzes using generative AI. Leveraging the Gemini API, the app generates personalized multiple-choice questions based on user-provided topics, enabling efficient and dynamic learning experiences. Built with Android Studio, the application incorporates Retrofit for robust API communication and features a clean, user-friendly interface following modern Android design principles. The modular architecture ensures scalability for future enhancements such as topic categorization, performance analytics, and quiz history tracking. Key features include real-time quiz generation, automatic scoring, and interactive question navigation. This report explores the application's design, architecture, and functionality, demonstrating its value as an intelligent learning companion for students and self-learners alike.

TABLE OF CONTENTS

CHAPTER 1

1. INTRODUCTION

With the increasing demand for personalized and accessible learning tools, traditional methods of quiz creation—often manual and time-consuming—can hinder self-paced education and classroom engagement. The Quiz Generator with Gemini API is an Android application developed in Kotlin to streamline this process. By integrating the Gemini API, the app allows users to input a topic and instantly generate multiple-choice quizzes, eliminating the need for manual question creation or textbook referencing. Developed in Android Studio using Retrofit for API communication, the application delivers a seamless experience on mobile devices.The app features a clean and interactive user interface and is designed with a modular structure to support future enhancements like topic tracking, quiz analytics, or gamified learning. This chapter introduces the motivation behind the project, outlines its key objectives, and describes the modular structure that supports efficient quiz generation and management.

2. Objectives

The Quiz Generator project was designed with the following objectives:

- **Dynamic Quiz Creation**: Utilize the Gemini API to generate contextually accurate and diverse quiz questions based on user-input topics.

- **User-Friendly Experience**: Provide an intuitive interface for entering topics, viewing questions, and submitting answers with real-time scoring.

- **Modular Architecture**: Build the app in a modular format to support scalability, allowing future integration of features like subtopic selection or result tracking.

- **Performance Optimization**: Ensure efficient communication between the app and Gemini API using Retrofit, minimizing response time and handling errors gracefully.

- **Educational Value**: Promote personalized learning by enabling quick quiz generation for any topic, helping users study and test their knowledge interactively.

3. Modules

The application is organized into the following modules:

- **UI Module**: Provides a clean and responsive interface for users to input topics, answer questions, and view results.

- **API Integration Module**: Handles communication with the Gemini API using Retrofit to fetch quiz questions in real time.

- **Logic Module**: Manages quiz flow, response handling, and score calculation, ensuring smooth transitions and robust user interaction.

- **Topic Handling Module** *(optional/future)*: Supports dynamic subtopic generation and selection to enhance personalization.

CHAPTER 2

1. Software Description

The Quiz Generator is an Android application developed using Kotlin, leveraging the Gemini API to generate topic-based multiple-choice questions in real time. Built in Android Studio with Retrofit for API communication, the app offers a smooth and responsive user experience. It follows the MVVM (Model-View-ViewModel) architecture to ensure clean code separation and maintainability. While primarily focused on online functionality, its modular design allows easy integration of offline features such as quiz history or local storage. This combination of modern tools and architecture makes the app a scalable and efficient solution for personalized learning on mobile devices.

1.1 Kotlin

Kotlin is a modern, statically typed programming language fully interoperable with Java, widely adopted for Android development. In this project, Kotlin implements the app's logic, UI, API integration, and database operations. Its concise syntax and null-safety features enhance code reliability and developer productivity.

1.2 XML

Jetpack Compose is Android's declarative UI framework, used to build the app's interface. It enables the creation of dynamic, responsive layouts with minimal code, supporting features like text inputs and scrollable recipe displays. Compose's integration with Kotlin ensures a seamless development experience

1.3 Retrofit

Retrofit is a type-safe HTTP client for Android, used to communicate with the Gemini AI API. It simplifies API calls by converting JSON responses into Kotlin data classes, ensuring efficient and reliable data retrieval.

2. Requirements And Analysis

2.1 Requirements

Project Overview:

The Quiz Generator is an Android application that allows users to input a topic, which is processed by the Gemini API to generate a custom multiple-choice quiz. The app is built using Kotlin in Android Studio, with Retrofit for API communication and a clean, interactive user interface. Designed for accessibility and responsiveness, the application supports Android devices running API level 21 (Lollipop) or higher and follows the MVVM architecture for maintainability and scalability.

Functional Requirements:

- **Topic Input:** Users can enter a quiz topic via a text field on the main screen.
- **Quiz Generation:** The app sends the topic to the Gemini API and retrieves a set of quiz questions with multiple-choice options and correct answers.
- **Question Display:** The questions are presented one at a time, with options for the user to select and submit an answer.
- **Score Display:** At the end of the quiz, the user receives a score summary and the correct answers.

Non-functional Requirements:

- **Usability:** The UI must be clean and intuitive, with smooth navigation between questions and instant feedback on interactions.
- **Performance:** API responses and quiz rendering should complete within 2 seconds.
- **Compatibility:** The application must run on Android devices with API level 21 or higher.
- **Efficiency:** The app should be optimized for smooth performance on devices with at least 2 GB RAM, ensuring minimal resource usage.

Analysis:

User Workflow: Users initiate the quiz by entering a topic, answering generated questions, and reviewing the score.

Data Model: Although primarily API-driven, the architecture allows for a simple data model for storing questions and results if offline access or history tracking is implemented in the future.

2.2 Hardware and Software

Requirements Hardware Requirements:

Minimum: Android device with 2 GB RAM, 100 MB storage, 1 GHz processor.

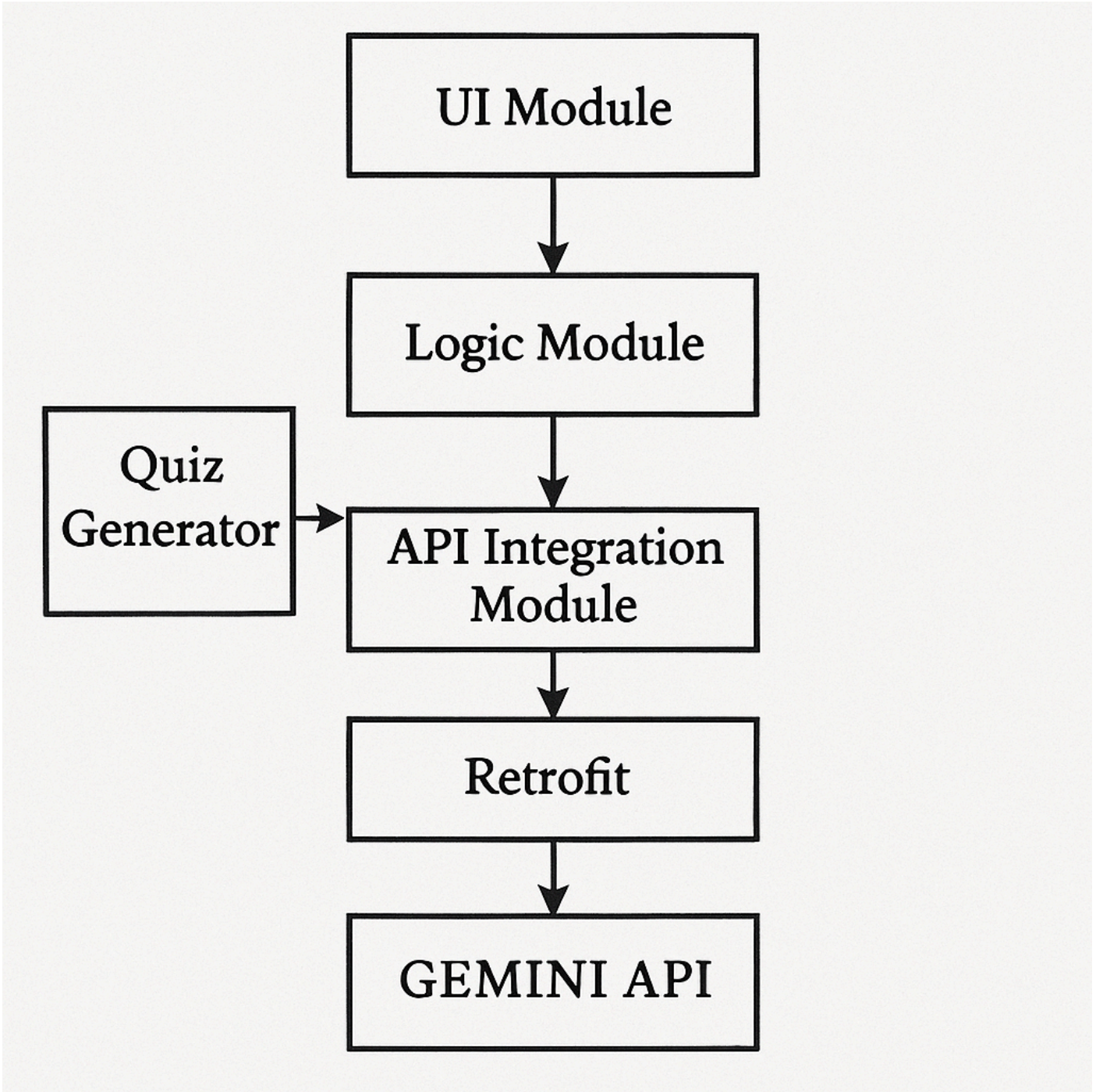
Recommended: Development machine with 8 GB RAM, 500 MB storage, 2 GHz processor.

Software Requirements:

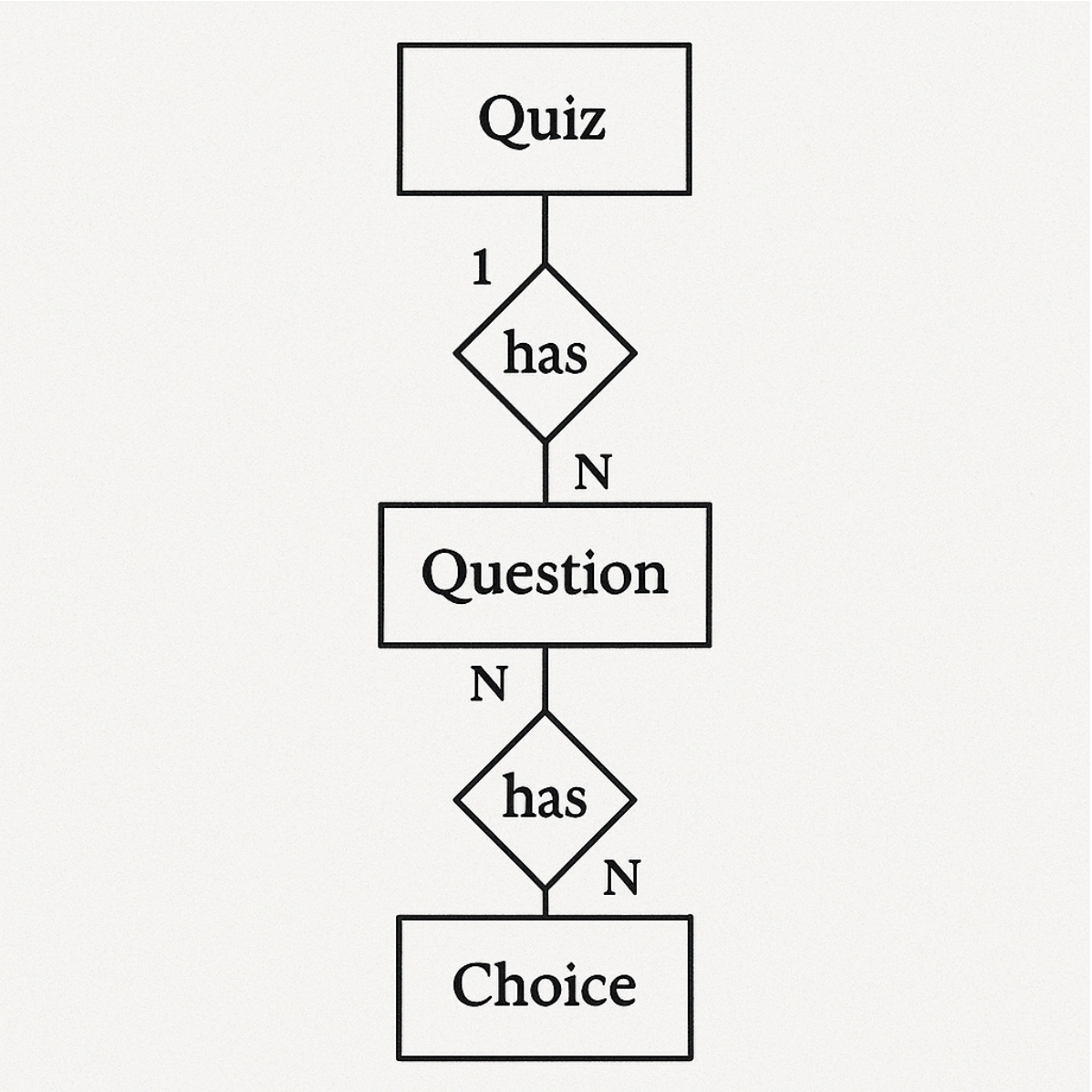
Android Studio 4.0 or higher. 8

- Kotlin 1.5 or higher.
- Android SDK API 21 or higher.
- Gemini AI API key and SDK.
- Gradle build system.

2.3 Architecture Diagram



2.4. ER Diagram



2.5 PROGRAM CODE

GeminiRequest.kt

```
package com.example.quizapp
data class GeminiRequest(
   val contents: List<GeminiContent>
)

data class GeminiContent(
   val parts: List<GeminiPart>,
   val role: String = "user"
)

data class GeminiPart(
   val text: String
)
```

GeminiResponse.kt

```
package com.example.quizapp
data class GeminiResponse(
   val candidates: List<GeminiCandidate>
)

data class GeminiCandidate(
   val content: GeminiContentResponse
)

data class GeminiContentResponse(
   val parts: List<GeminiPartResponse>
)

data class GeminiPartResponse(
   val text: String
)
```

GeminiService.kt

```
package com.example.quizapp
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.POST
import retrofit2.http.Query

interface GeminiService {
   @POST("v1beta/models/gemini-2.0-flash:generateContent")
   fun generateQuiz(
       @Body body: GeminiRequest,
       @Query("key") apiKey: String
```

```
    ): Call<GeminiResponse>
}
```

MainActivity.kt

```
package com.example.quizapp

import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.widget.*
import androidx.appcompat.app.AppCompatActivity
import org.json.JSONObject
import retrofit2.*
import retrofit2.converter.gson.GsonConverterFactory
import java.io.File
import org.json.JSONArray

class MainActivity : AppCompatActivity() {

    private lateinit var etTopic: EditText
    private lateinit var btnGenerate: Button
    private lateinit var savedQuizzesLayout: LinearLayout
    private val geminiApiKey = "AIzaSyBu_ggFYGfWooB7DFkyb6AF-CVKaahZALs" // 🔁
Replace with your key

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        etTopic = findViewById(R.id.etTopic)
        btnGenerate = findViewById(R.id.btnGenerate)
        savedQuizzesLayout = findViewById(R.id.savedQuizzesLayout)

        btnGenerate.setOnClickListener {
            val topic = etTopic.text.toString().trim()
            if (topic.isNotEmpty()) {
                generateQuizFromTopic(topic)
            } else {
                Toast.makeText(this, "Please enter a topic",
Toast.LENGTH_SHORT).show()
            }
        }

        loadSavedQuizzes()
    }

    private fun loadSavedQuizzes() {
        try {
            savedQuizzesLayout.removeAllViews()
            val headerText = TextView(this)
```

```
            headerText.text = "Saved Quizzes"
            headerText.textSize = 18f
            headerText.setPadding(0, 16, 0, 8)
            savedQuizzesLayout.addView(headerText)

            val indexFile = File(filesDir, "quiz_index.json")
            if (!indexFile.exists()) {
                val noQuizzesText = TextView(this)
                noQuizzesText.text = "No saved quizzes found."
                savedQuizzesLayout.addView(noQuizzesText)
                return
            }

            val indexJson = JSONObject(indexFile.readText())
            val quizzesArray = indexJson.optJSONArray("quizzes") ?: return

            if (quizzesArray.length() == 0) {
                val noQuizzesText = TextView(this)
                noQuizzesText.text = "No saved quizzes found."
                savedQuizzesLayout.addView(noQuizzesText)
            } else {
                for (i in 0 until quizzesArray.length()) {
                    val quizObj = quizzesArray.getJSONObject(i)
                    val topic = quizObj.getString("topic")
                    val filename = quizObj.getString("filename")

                    val quizButton = Button(this)
                    quizButton.text = topic
                    quizButton.setOnClickListener {
                        loadSavedQuiz(topic, filename)
                    }
                    savedQuizzesLayout.addView(quizButton)
                }
            }

        } catch (e: Exception) {
            e.printStackTrace()
            Log.e("MainActivity", "Error loading saved quizzes: ${e.message}")

            val errorText = TextView(this)
            errorText.text = "Error loading saved quizzes."
            savedQuizzesLayout.addView(errorText)
        }
    }
    private fun extractJsonFromMarkdown(text: String): String {
        val jsonPattern = Regex("```(?:json)?\\s*\\n?(\\[.*?\\])\\s*```",
RegexOption.DOT_MATCHES_ALL)
        val match = jsonPattern.find(text)

        return if (match != null) {
            match.groupValues[1].trim()
```

```
        } else {
            text.trim()
        }
    }
    private fun loadSavedQuiz(topic: String, filename: String) {
        try {
            val file = File(filesDir, "saved_quizzes/$filename")
            if (file.exists()) {
                val quizJson = file.readText()
                val jsonText = extractJsonFromMarkdown(quizJson)
                val jsonObject = JSONObject(jsonText)
                val quizArrayText = jsonObject.getJSONArray("questions").toString()

                val intent = Intent(this, QuizActivity::class.java)
                intent.putExtra("quiz_raw", quizArrayText)
                intent.putExtra("topic", topic)
                intent.putExtra("is_saved_quiz", true)
                startActivity(intent)
            } else {
                Toast.makeText(this, "Error: Quiz file not found",
Toast.LENGTH_SHORT).show()
            }
        } catch (e: Exception) {
            e.printStackTrace()
            Toast.makeText(this, "Error loading quiz: ${e.message}",
Toast.LENGTH_LONG).show()
        }
    }

    private fun generateQuizFromTopic(topic: String) {
        val prompt = """
            "Generate 10 questions about ["$topic"] in JSON format. The output
should include 'question', 'options', and 'answer'."
        """.trimIndent()

        val retrofit = Retrofit.Builder()
            .baseUrl("https://generativelanguage.googleapis.com/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        val service = retrofit.create(GeminiService::class.java)
        val request = GeminiRequest(
            contents = listOf(GeminiContent(parts = listOf(GeminiPart(prompt))))
        )

        service.generateQuiz(request, geminiApiKey).enqueue(object :
Callback<GeminiResponse> {
            override fun onResponse(call: Call<GeminiResponse>, response:
Response<GeminiResponse>) {
                if (response.isSuccessful) {
                    val quizText = response.body()?.candidates?.firstOrNull()
                        ?.content?.parts?.firstOrNull()?.text ?: "No quiz generated"
```

```
                    Log.d("MainActivity", "Quiz:\n$quizText")

                    val intent = Intent(this@MainActivity, QuizActivity::class.java)
                    intent.putExtra("quiz_raw", quizText)
                    intent.putExtra("topic", topic)
                    startActivity(intent)
                } else {
                    Log.d("Gemini", "Error: ${response.errorBody()?.string()}")
                    Toast.makeText(this@MainActivity, "Error:
${response.errorBody()?.string()}", Toast.LENGTH_LONG).show()
                }
            }

            override fun onFailure(call: Call<GeminiResponse>, t: Throwable) {
                Toast.makeText(this@MainActivity, "Failed: ${t.message}",
Toast.LENGTH_LONG).show()
            }
        })
    }

    override fun onResume() {
        super.onResume()
        loadSavedQuizzes()
    }
}
```

QuizActivity.kt

```
package com.example.quizapp

import android.content.Context
import android.os.Bundle
import android.util.Log
import android.widget.*
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import org.json.JSONArray
import org.json.JSONObject
import retrofit2.*
import retrofit2.converter.gson.GsonConverterFactory
import java.io.File

class QuizActivity : AppCompatActivity() {

    private lateinit var layout: LinearLayout
    private val userAnswers = mutableMapOf<Int, String>()
    private val questions = mutableListOf<ParsedQuestion>()
    private val existingQuestionTexts = mutableSetOf<String>()
    private val geminiApiKey = "AIzaSyBu_ggFYGfWooB7DFkyb6AF-CVKaahZALs"
    private var topic = "general knowledge"
```

```
    private var isSaved = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_quiz)

        layout = findViewById(R.id.quizLayout)

        val raw = intent.getStringExtra("quiz_raw") ?: ""
        topic = intent.getStringExtra("topic") ?: "general knowledge"

        val isSavedQuiz = intent.getBooleanExtra("is_saved_quiz", false)
        isSaved = isSavedQuiz

        val parsedQuestions = parseQuestions(raw)

        parsedQuestions.forEach { existingQuestionTexts.add(it.question) }
        questions.addAll(parsedQuestions)

        questions.forEachIndexed { index, question ->
            val questionText = TextView(this)
            questionText.text = "${index + 1}. ${question.question}"
            questionText.textSize = 16f
            questionText.setPadding(0, 16, 0, 8)
            layout.addView(questionText)

            val radioGroup = RadioGroup(this)
            question.options.forEach { opt ->
                val rb = RadioButton(this)
                rb.text = opt
                radioGroup.addView(rb)

                rb.setOnClickListener {
                    userAnswers[index] = opt
                }
            }
            layout.addView(radioGroup)
        }


        if (!isSaved) {
            val generateMoreButton = Button(this).apply {
                text = "Generate More Questions"
                setOnClickListener { generateMoreQuestions() }
            }
            layout.addView(generateMoreButton)
        }

        val saveButton = Button(this).apply {
            text = if (isSaved) "Quiz Already Saved" else "Save Quiz"
            isEnabled = !isSaved
```

```
            setOnClickListener { saveQuiz() }
        }
        layout.addView(saveButton)

        val submitButton = Button(this).apply {
            text = "Submit Quiz"
            setOnClickListener { showResult() }
        }
        layout.addView(submitButton)
    }

    private fun saveQuiz() {
        try {

            val quizData = JSONObject().apply {
                put("topic", topic)

                val questionsArray = JSONArray()
                questions.forEach { question ->
                    val questionObj = JSONObject().apply {
                        put("question", question.question)
                        put("options", JSONArray().apply {
                            question.options.forEach { option ->
                                put(option)
                            }
                        })
                        put("answer", question.correctAnswer)
                    }
                    questionsArray.put(questionObj)
                }
                put("questions", questionsArray)

                val answersObj = JSONObject()
                userAnswers.forEach { (index, answer) ->
                    answersObj.put(index.toString(), answer)
                }
                put("userAnswers", answersObj)
            }


            val directory = File(filesDir, "saved_quizzes")
            if (!directory.exists()) {
                directory.mkdirs()
            }

            val filename = sanitizeFilename(topic) + ".json"
            val file = File(directory, filename)


            file.writeText(quizData.toString(2))
```

```
            // Update saved quizzes index
            updateSavedQuizzesIndex(topic, filename)

            // Update UI
            isSaved = true
            val saveButton = layout.getChildAt(layout.childCount - 3) as Button
            saveButton.text = "Quiz Saved"
            saveButton.isEnabled = false

            Toast.makeText(this, "Quiz saved successfully",
Toast.LENGTH_SHORT).show()

        } catch (e: Exception) {
            e.printStackTrace()
            Toast.makeText(this, "Error saving quiz: ${e.message}",
Toast.LENGTH_LONG).show()
        }
    }

    private fun sanitizeFilename(input: String): String {
        // Replace invalid filename characters with underscores
        return input.replace(Regex("[^a-zA-Z0-9._-]"), "_")
    }

    private fun updateSavedQuizzesIndex(topic: String, filename: String) {
        try {
            // Create or load the index file
            val indexFile = File(filesDir, "quiz_index.json")
            val indexJson = if (indexFile.exists()) {
                JSONObject(indexFile.readText())
            } else {
                JSONObject()
            }

            val quizzesArray = if (indexJson.has("quizzes")) {
                indexJson.getJSONArray("quizzes")
            } else {
                JSONArray()
            }

            var exists = false
            for (i in 0 until quizzesArray.length()) {
                val quizObj = quizzesArray.getJSONObject(i)
                if (quizObj.getString("topic") == topic) {
                    exists = true
                    break
                }
            }

            if (!exists) {
                val quizObj = JSONObject().apply {
                    put("topic", topic)
```

```
                    put("filename", filename)
                    put("timestamp", System.currentTimeMillis())
                }
                quizzesArray.put(quizObj)
            }

            indexJson.put("quizzes", quizzesArray)
            indexFile.writeText(indexJson.toString(2))

        } catch (e: Exception) {
            e.printStackTrace()
            Log.e("QuizActivity", "Error updating quiz index: ${e.message}")
        }
    }
    private fun generateMoreQuestions() {
        val progressBar = ProgressBar(this)
        progressBar.isIndeterminate = true
        val loadingIndex = layout.childCount - 2
        layout.addView(progressBar, loadingIndex)

        val existingQuestions = existingQuestionTexts.joinToString(", ")

        val prompt = """
            Generate 10 more multiple choice questions about "$topic" in JSON
format.
            The output should include 'question', 'options', and 'answer'.
            Do NOT include any of these existing questions: [$existingQuestions].
            Only return the JSON array, nothing else.
        """.trimIndent()

        val retrofit = Retrofit.Builder()
            .baseUrl("https://generativelanguage.googleapis.com/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        val service = retrofit.create(GeminiService::class.java)
        val request = GeminiRequest(
            contents = listOf(GeminiContent(parts = listOf(GeminiPart(prompt))))
        )

        service.generateQuiz(request, geminiApiKey).enqueue(object :
Callback<GeminiResponse> {
            override fun onResponse(call: Call<GeminiResponse>, response:
Response<GeminiResponse>) {
                layout.removeView(progressBar)

                if (response.isSuccessful) {
                    val newQuizText = response.body()?.candidates?.firstOrNull()
                        ?.content?.parts?.firstOrNull()?.text ?: "No quiz generated"

                    Log.d("QuizActivity", "New Questions:\n$newQuizText")
```

```
                    val newQuestions = parseQuestions(newQuizText)

                    val uniqueNewQuestions = newQuestions.filter {
                        !existingQuestionTexts.contains(it.question)
                    }

                    if (uniqueNewQuestions.isNotEmpty()) {
                        uniqueNewQuestions.forEach {
existingQuestionTexts.add(it.question) }

                        val startIndex = questions.size
                        questions.addAll(uniqueNewQuestions)

                        uniqueNewQuestions.forEachIndexed { i, question ->
                            val index = startIndex + i
                            val questionText = TextView(this@QuizActivity)
                            questionText.text = "${index + 1}. ${question.question}"
                            questionText.textSize = 16f
                            questionText.setPadding(0, 16, 0, 8)

                            layout.addView(questionText, layout.childCount - 3)

                            val radioGroup = RadioGroup(this@QuizActivity)
                            question.options.forEach { opt ->
                                val rb = RadioButton(this@QuizActivity)
                                rb.text = opt
                                radioGroup.addView(rb)

                                rb.setOnClickListener {
                                    userAnswers[index] = opt
                                }
                            }
                            layout.addView(radioGroup, layout.childCount - 3)
                        }

                        val scrollView = layout.parent as? ScrollView
                        scrollView?.post {
scrollView.fullScroll(ScrollView.FOCUS_DOWN) }

                        Toast.makeText(
                            this@QuizActivity,
                            "Added ${uniqueNewQuestions.size} new questions",
                            Toast.LENGTH_SHORT
                        ).show()
                    } else {
                        Toast.makeText(
                            this@QuizActivity,
                            "No new unique questions generated",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                    if (isSaved) {
```

```
                            isSaved = false
                            val saveButton = layout.getChildAt(layout.childCount -
3) as Button
                            saveButton.text = "Save Quiz"
                            saveButton.isEnabled = true
                        }
                } else {
                    Log.d("Gemini", "Error: ${response.errorBody()?.string()}")
                    Toast.makeText(
                        this@QuizActivity,
                        "Error generating questions:
${response.errorBody()?.string()}",
                        Toast.LENGTH_LONG
                    ).show()
                }
            }

            override fun onFailure(call: Call<GeminiResponse>, t: Throwable) {

                layout.removeView(progressBar)

                Toast.makeText(
                    this@QuizActivity,
                    "Failed to generate questions: ${t.message}",
                    Toast.LENGTH_LONG
                ).show()
            }
        })
    }

    private fun parseQuestions(text: String): List<ParsedQuestion> {
        val list = mutableListOf<ParsedQuestion>()
        Log.d("Json",text)

        try {
            val jsonText = extractJsonFromMarkdown(text)
            val jsonArray = JSONArray(jsonText)

            for (i in 0 until jsonArray.length()) {
                val item = jsonArray.getJSONObject(i)
                val questionText = item.getString("question")
                val optionsArray = item.getJSONArray("options")
                val correctAnswer = item.getString("answer")

                val options = mutableListOf<String>()
                for (j in 0 until optionsArray.length()) {
                    options.add(optionsArray.getString(j))
                }

                list.add(ParsedQuestion(questionText, options, correctAnswer))
            }
        } catch (e: Exception) {
```

```
            e.printStackTrace()
            Log.d("Error parsing quiz data","${e.message}")
            Toast.makeText(this, "Error parsing quiz data: ${e.message}",
Toast.LENGTH_LONG).show()
        }

        return list
    }

    private fun extractJsonFromMarkdown(text: String): String {
        val jsonPattern = Regex("```(?:json)?\\s*\\n?(\\[.*?\\])\\s*```",
RegexOption.DOT_MATCHES_ALL)
        val match = jsonPattern.find(text)

        return if (match != null) {
            match.groupValues[1].trim()
        } else {
            text.trim()
        }
    }

    private fun showResult() {
        var correctCount = 0
        val resultSummary = StringBuilder()

        questions.forEachIndexed { index, q ->
            val selected = userAnswers[index]
            val correct = q.correctAnswer
            if (selected != null && selected == correct) {
                correctCount++
            }
            resultSummary.append("${index + 1}. ${q.question}\n")
            resultSummary.append("Your answer: ${selected ?: "Not Answered"}\n")
            resultSummary.append("Correct answer: $correct\n\n")
        }

        AlertDialog.Builder(this)
            .setTitle("Quiz Results")
            .setMessage("Score: $correctCount /
${questions.size}\n\n$resultSummary")
            .setPositiveButton("OK", null)
            .show()
    }

    data class ParsedQuestion(
        val question: String,
        val options: List<String>,
        val correctAnswer: String
    )
}
```

activity_main.xml

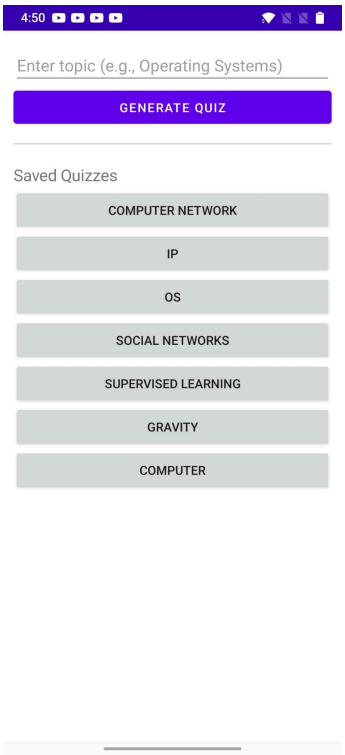
```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:padding="16dp"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <EditText
            android:id="@+id/etTopic"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Enter topic (e.g., Operating Systems)" />

        <Button
            android:id="@+id/btnGenerate"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Generate Quiz" />

        <View
            android:layout_width="match_parent"
            android:layout_height="1dp"
            android:background="#CCCCCC"
            android:layout_marginTop="16dp"
            android:layout_marginBottom="16dp" />

        <LinearLayout
            android:id="@+id/savedQuizzesLayout"
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

    </LinearLayout>
</ScrollView>
```

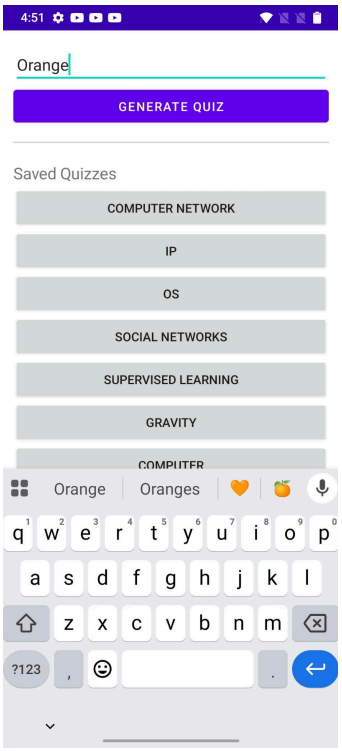
activity_quiz.xml

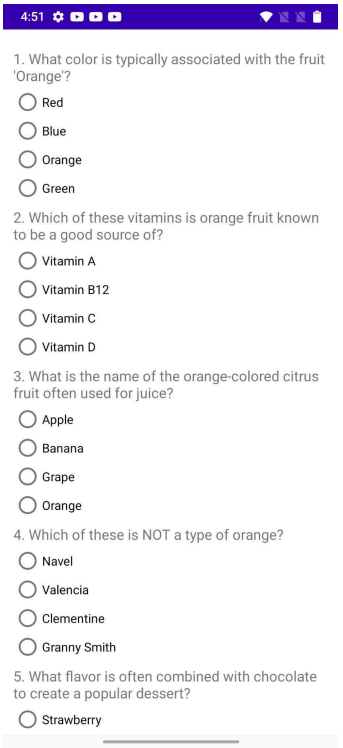
```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:id="@+id/quizLayout"
        android:orientation="vertical"
```

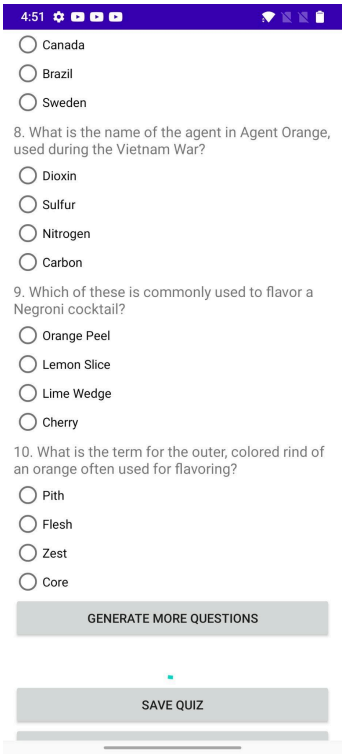
```
        android:padding="16dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</ScrollView>
```

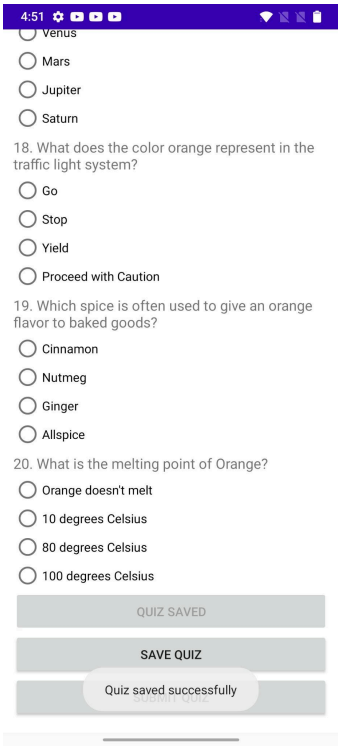
OUTPUT:

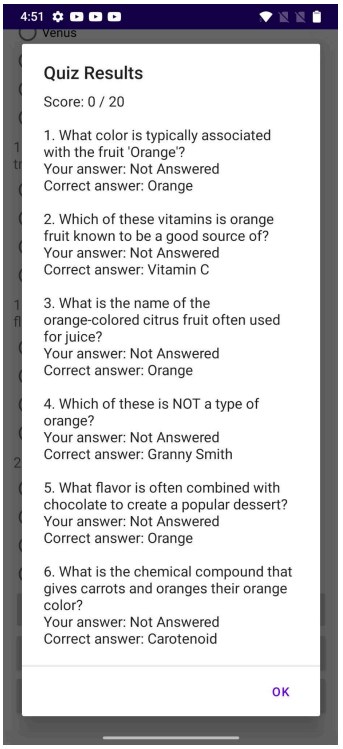

4:50
Enter topic (e.g., Operating Systems)
GENERATE QUIZ
Saved Quizzes
COMPUTER NETWORK
IP
OS
SOCIAL NETWORKS
SUPERVISED LEARNING
GRAVITY
COMPUTER


4:51
Orange
GENERATE QUIZ
Saved Quizzes
COMPUTER NETWORK
IP
OS
SOCIAL NETWORKS
SUPERVISED LEARNING
GRAVITY
COMPUTER
Orange
Oranges


4:51
1. What color is typically associated with the fruit 'Orange'?
Red
Blue
Orange
Green
2. Which of these vitamins is orange fruit known to be a good source of?
Vitamin A
Vitamin B12
Vitamin C
Vitamin D
3. What is the name of the orange-colored citrus fruit often used for juice?
Apple
Banana
Grape
Orange
4. Which of these is NOT a type of orange?
Navel
Valencia
Clementine
Granny Smith
5. What flavor is often combined with chocolate to create a popular dessert?
Strawberry


4:51
Canada
Brazil
Sweden
8. What is the name of the agent in Agent Orange, used during the Vietnam War?
Dioxin
Sulfur
Nitrogen
Carbon
9. Which of these is commonly used to flavor a Negroni cocktail?
Orange Peel
Lemon Slice
Lime Wedge
Cherry
10. What is the term for the outer, colored rind of an orange often used for flavoring?
Pith
Flesh
Zest
Core
GENERATE MORE QUESTIONS
SAVE QUIZ


4:51
Venus
Mars
Jupiter
Saturn
18. What does the color orange represent in the traffic light system?
Go
Stop
Yield
Proceed with Caution
19. Which spice is often used to give an orange flavor to baked goods?
Cinnamon
Nutmeg
Ginger
Allspice
20. What is the melting point of Orange?
Orange doesn't melt
10 degrees Celsius
80 degrees Celsius
100 degrees Celsius
QUIZ SAVED
SAVE QUIZ
Quiz saved successfully


4:51
Venus
Quiz Results
Score: 0 / 20
1. What color is typically associated with the fruit 'Orange'?
Your answer: Not Answered
Correct answer: Orange
2. Which of these vitamins is orange fruit known to be a good source of?
Your answer: Not Answered
Correct answer: Vitamin C
3. What is the name of the orange-colored citrus fruit often used for juice?
Your answer: Not Answered
Correct answer: Orange
4. Which of these is NOT a type of orange?
Your answer: Not Answered
Correct answer: Granny Smith
5. What flavor is often combined with chocolate to create a popular dessert?
Your answer: Not Answered
Correct answer: Orange
6. What is the chemical compound that gives carrots and oranges their orange color?
Your answer: Not Answered
Correct answer: Carotenoid
OK

6. RESULTS AND DISCUSSION

Upon deployment and testing, the Quiz Generator app met its core objectives efficiently. The user interface was responsive and intuitive, allowing users to input topics and receive AI-generated quizzes in real time. The integration with the Gemini API via Retrofit functioned reliably, with API responses processed swiftly even under varying network conditions. Functional testing on multiple Android devices confirmed consistent behavior across different screen sizes and Android versions.

Key Highlights:

- The real-time quiz generation was accurate and context-aware, showcasing the Gemini API's capability to produce relevant questions.
- The app's modular architecture supports easy maintenance and future feature integration, such as result tracking or topic history.
- Use of MVVM architecture ensured separation of concerns, reducing potential bugs and simplifying updates.

Potential Improvements Identified:

- Implementing offline mode using Room to store generated quizzes for later access.
- Adding a scoring dashboard to visualize user performance over time.
- Enhancing UI with animations and progress indicators for a more engaging user experience.

7. CONCLUSION

The Quiz Generator app effectively streamlines the process of quiz creation by using the Gemini API to deliver topic-based questions instantly. Developed using Kotlin and Retrofit within Android Studio, the application demonstrates excellent performance, responsiveness, and modularity. Its clean design and adherence to MVVM principles make it both scalable and maintainable. Future versions of the app can further enrich the user experience by incorporating offline functionality, result analytics, and a more personalized quiz generation system.

REFERENCES

1. Android Developer Documentation: https://developer.android.com/
2. Kotlin Official Documentation: https://kotlinlang.org/docs/
3. Retrofit Documentation: https://square.github.io/retrofit/
4. Gemini API Documentation: https://aistudio.google.com/
5. MVVM Architecture: https://developer.android.com/jetpack/guide
6. JSON.org: https://www.json.org/json-en.html