Project - High Level Design

On

Agriculture

Quiz bot

Course Name: GEN-AI

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1	Aaditya Pandey	EN22CS301057
2	Aaditya Verma	EN22CS301009
3	Aakansha Rajani	EN22CS301012
4	Abhijeet patidar	EN22CS301029
5	Aditya Thakre	EN22CS301069

Group Name: *11D1*

Project Number:*GAI-11*

Industry Mentor Name: *Aashruti Shah*

University Mentor Name: *Prof. Ajaz Khan*

Academic Year:2025-2026

Table of Contents

1. Introduction

FarmWise is an AI-driven interactive educational web application designed for the agriculture domain. The system delivers targeted quiz assessments focused on farming tips, evaluates user responses using a Large Language Model (LLM) via Groq, and provides deep contextual explanations to reinforce knowledge. Session data and leaderboard scores are persisted in a Supabase PostgreSQL cloud database accessible in real time.

This document describes the complete system design of the FarmWise application, covering architectural decisions, data design, interface contracts, session management, caching strategy, and non-functional requirements.

1.1 Scope of the Document

This System Design Document (SDD) covers:

- The full-stack architecture of the FarmWise web application.
- Backend API design using Python/Flask, integrated with Groq and Supabase.
- Frontend single-page application structure (HTML/CSS/JS).
- Data model, persistence layer, and access mechanisms via Supabase.
- State and session management, caching strategy, and non-functional requirements.
- Security and performance considerations for production readiness.

1.2 Intended Audience

This document is for:

- Project evaluators
- Mentors
- Developers
- Anyone who wants to understand the system design

1.3 System Overview

The Agriculture Quiz Bot is a web-based application.

FarmWise is a three-tier web application:

Tier	Technology	Responsibility
Presentation	HTML5 / CSS3 / Vanilla JavaScript	Single-page quiz UI, routing, DOM management
Application	Python 3.x / Flask REST API	Business logic, Groq LLM orchestration, Supabase client
Data	Supabase (PostgreSQL)	Persistent session storage, leaderboard, real-time queries

External services consumed:

- Groq API — LLM inference endpoint (LLaMA 3 8B model) for answer evaluation, contextual explanation generation, and personalized summary creation.
- Supabase — managed PostgreSQL cloud database with REST and real-time APIs, used for persisting quiz sessions and serving the leaderboard.

2. System Design

2.1. Application Design

The application follows a RESTful client-server architecture. The Flask backend exposes a stateless JSON API consumed by the frontend SPA. No server-side rendering is performed; all view logic resides in the browser.

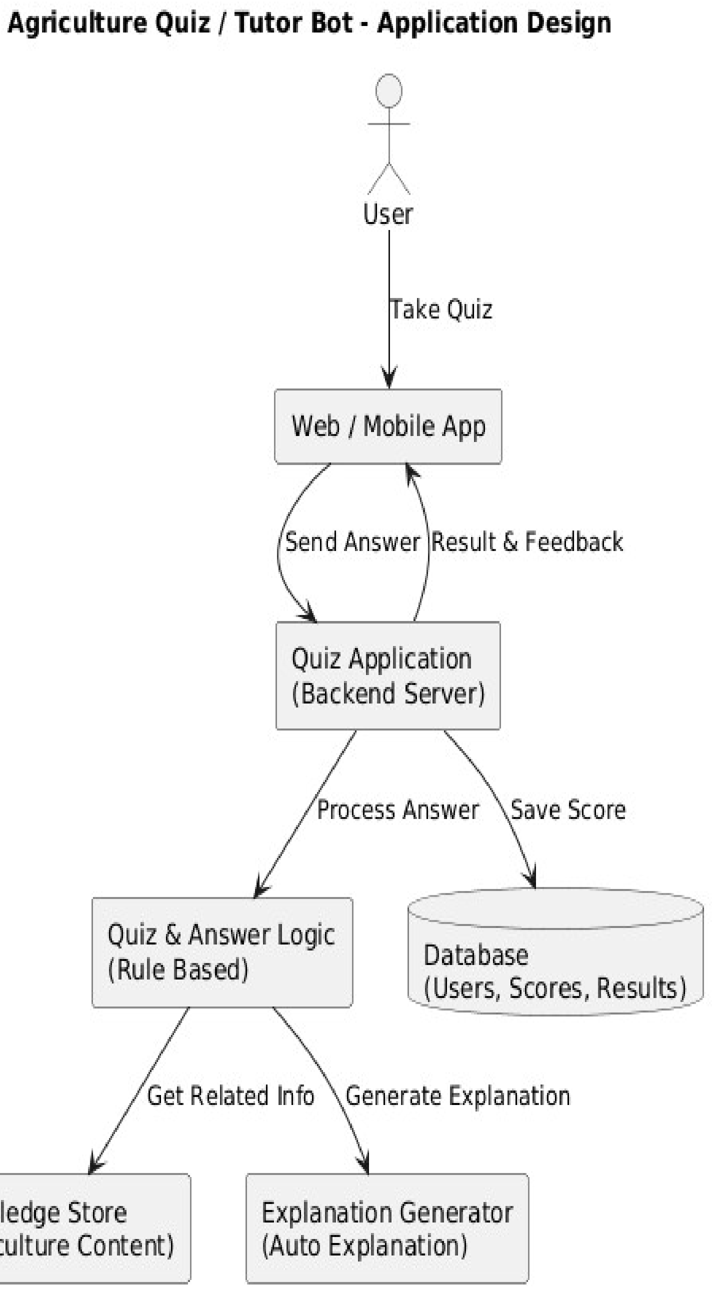
FarmWise is a three-tier web application:

Tier	Technology	Responsibility
Presentation	HTML5 / CSS3 / Vanilla JavaScript	Single-page quiz UI, routing, DOM management
Application	Python 3.x / Flask REST API	Business logic, Groq LLM orchestration, Supabase client

Data	Supabase (PostgreSQL)	Persistent session storage, leaderboard, real-time queries

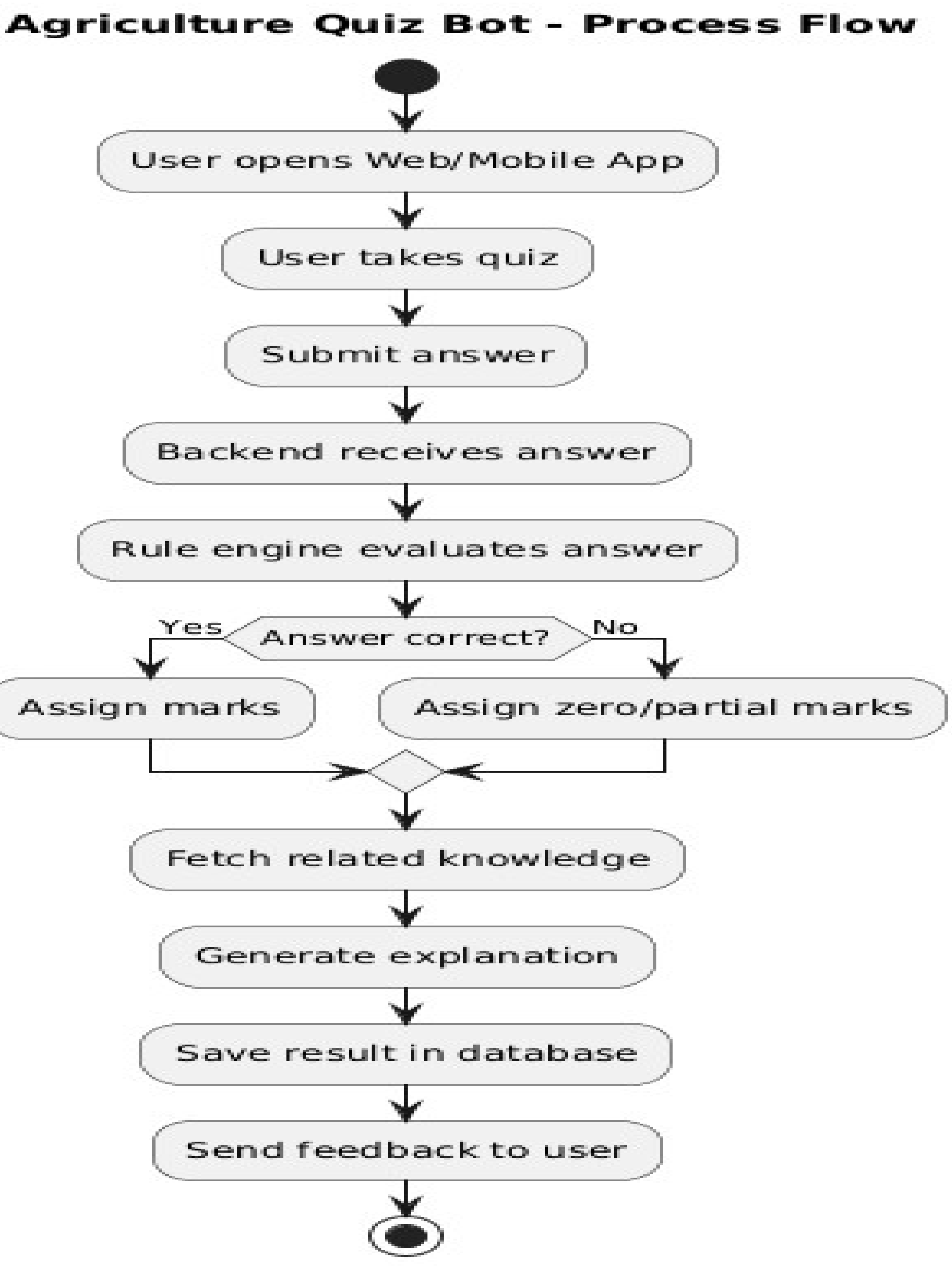
External services consumed:

1. Groq API — LLM inference endpoint (LLaMA 3 8B model) for answer evaluation, contextual explanation generation, and personalized summary creation.
2. Supabase — managed PostgreSQL cloud database with REST and real-time APIs, used for persisting quiz sessions and serving the leaderboard.

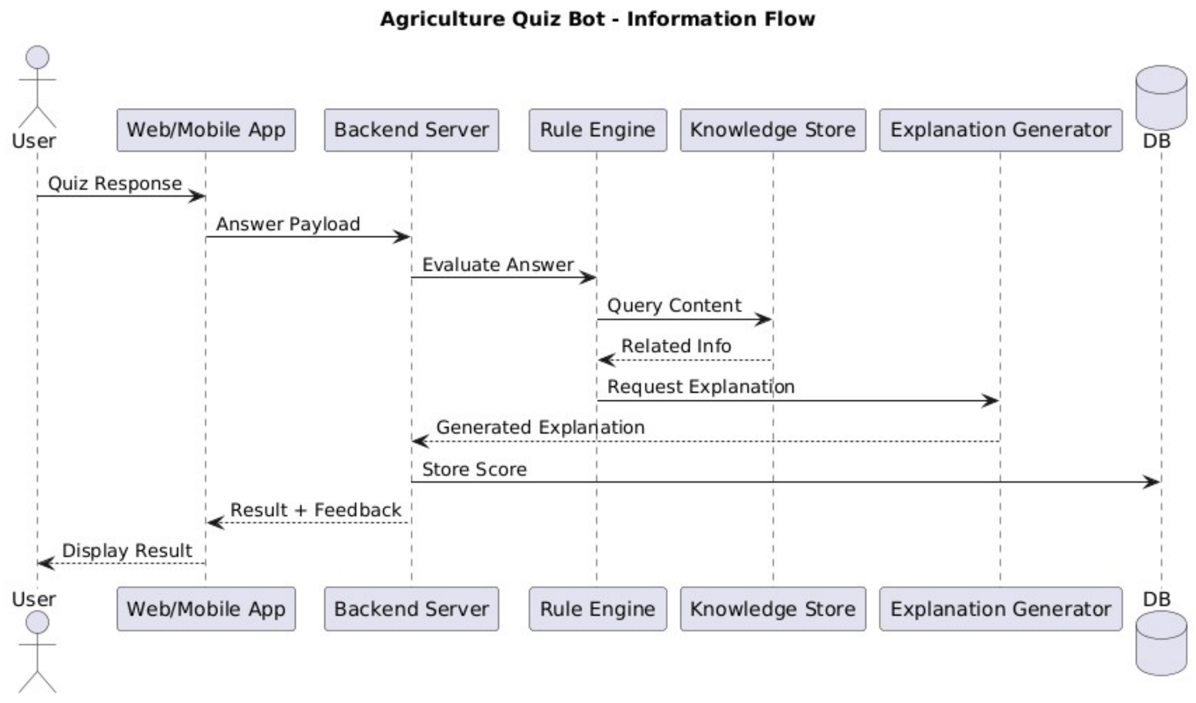

Agriculture Quiz / Tutor Bot - Application Design
User
Take Quiz
Web / Mobile App
Send Answer
Result & Feedback
Quiz Application (Backend Server)
Process Answer
Save Score
Quiz & Answer Logic (Rule Based)
Database (Users, Scores, Results)
Get Related Info
Generate Explanation
Knowledge Store (Agriculture Content)
Explanation Generator (Auto Explanation)

○

2.2 Process Flow



2.3 Information Flow



Input:

- User details
- Selected topic
- User answer

Processing:

- Question generation
- AI evaluation
- Score calculation

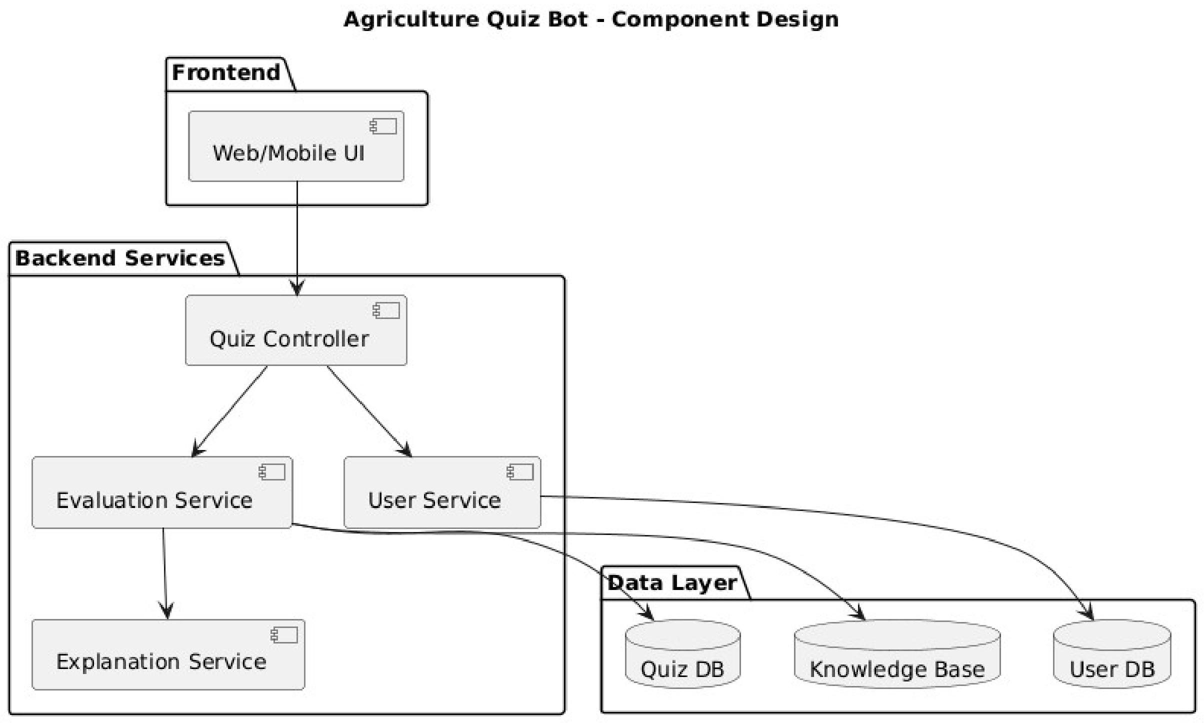
Output:

- Quiz score
- Correct explanation
- Feedback
- Stored result

Information flows like this:

User → Frontend → Backend → AI/Database → Backend → User

2.4 Components Design



Main components:

1. **Frontend**
   - User interface
   - Sends requests to backend
2. **Backend (Python)**
   - Handles logic
   - Connects to AI and database
3. **AI Model**
   - Generates questions
   - Evaluates answers
   - Example: OpenAI API
4. **Vector Database**
   - Stores agriculture knowledge
   - Helps in context-based answers
5. **Relational Database**
   - Stores users and quiz records

2.5 Key Design Considerations

Answer Security

The correct_answer field is deliberately stripped from the GET /api/questions response. Answers are only resolved server-side in the /api/evaluate endpoint, preventing client-side cheating via browser DevTools inspection.

LLM Prompt Engineering

Prompts are structured with explicit role assignment ("You are an expert agricultural educator"), question context, correct answer, user answer, result flag, and the relevant farming tip. This context grounding ensures factually accurate explanations anchored to the domain knowledge base rather than generic LLM outputs.

Groq Model Selection

LLaMA 3 8B (llama3-8b-8192) is selected on Groq for its combination of speed (low latency inference), free tier availability, and sufficient reasoning capability for educational explanation generation. Token limits are capped at 150-500 per call to control cost and response time.

No Build Step Frontend

The frontend is implemented as a single self-contained HTML file requiring no build toolchain (no webpack, npm, or transpilation). This reduces operational complexity, speeds development iteration, and allows the file to be served directly by Flask's static file handler.

2.6. API Catalogue

Method	Endpoint	Request Parameters	Response	Auth Required
GET	/api/questions	count (int), topic (str), difficulty (str) — all optional query params	{ questions: [...], total: int }	No
POST	/api/evaluate	{ question_id: str, answer: str }	{ is_correct, correct_answer, correct_answer_text, explanation, topic }	No
POST	/api/session/save	{ player_name, score, total, topic, difficulty, wrong_topics[] }	{ summary: str, saved: bool, id: uuid }	No

GET	/api/leaderboard	None	{ leaderboard: [{ player_name, score, total, percentage, topic, played_at }] }	No
GET	/api/topics	None	{ topics: [str] }	No
GET	/api/tips	None	{ tips: [{ id, topic, content }] }	No

3. Data Design

3.1. Data Model

The system persists a single entity to Supabase: quiz_sessions. The remaining data (farming tips, quiz questions) is held in application memory as Python lists and does not require database persistence.

quiz_sessions Table

Column	Type	Nullable	Default	Description
id	UUID	No	gen_random_uuid()	Primary key, auto-generated
player_name	TEXT	No	'Anonymous'	User-entered display name (max 30 chars)
score	INTEGER	No	—	Number of correct answers
total	INTEGER	No	—	Total questions attempted
percentage	INTEGER	No	—	Computed: (score/total) * 100
topic	TEXT	Yes	'Mixed'	Selected topic filter or 'Mixed'
difficulty	TEXT	Yes	'Mixed'	Selected difficulty or 'Mixed'

wrong_topics	TEXT[]	Yes	'{}'	Array of topic names answered incorrectly
summary	TEXT	Yes	NULL	AI-generated personalized learning summary
played_at	TIMESTAMPTZ	No	NOW()	Session completion timestamp (UTC)

In-Memory Data Structures

Structure	Type	Count	Key Fields
FARMING_TIPS	Python list of dicts	15	id, topic, content
QUIZ_QUESTIONS	Python list of dicts	15	id, topic, question, options{A-D}, correct_answer, tip_id, difficulty

3.2. Data Access Mechanism

All database operations use the supabase-py client library, which communicates with Supabase's PostgREST auto-generated REST API over HTTPS. No raw SQL is executed from application code (except in the migration script run once at setup).

Operation	Method	Code Pattern
INSERT session	supabase.table().insert()	supabase.table("quiz_sessions").insert(record).execute()
SELECT leaderboard	supabase.table().select().order().limit()	.select(...).order("percentage", desc=True).limit(10).execute()
Response parsing	result.data	result.data[0]["id"] for inserted record UUID

Row Level Security (RLS) is enabled on the quiz_sessions table with two policies: public SELECT (for leaderboard reads) and public INSERT (for saving sessions). No UPDATE or DELETE is permitted from application code.

3.3. Data Retention Policies

- Quiz sessions are retained indefinitely on the Supabase free tier (500 MB limit).
- No automatic expiry or archival mechanism is implemented in v1.0.
- Leaderboard queries are bounded to TOP 10 results — no full-table scans are performed by the application.
- Player names are stored as entered; no PII validation or anonymisation is applied in v1.0.
- Recommended: implement a rolling 90-day retention policy via a Supabase scheduled function (pg_cron) in future versions.

3.4. Data Migration

The database schema is initialised via a single SQL migration script (supabase_migration.sql). This script is idempotent (uses IF NOT EXISTS) and must be executed once in the Supabase SQL Editor before first application launch.

Migration Step	Script	Action
1 — Create table	supabase_migration.sql	CREATE TABLE IF NOT EXISTS quiz_sessions (...)
2 — Enable RLS	supabase_migration.sql	ALTER TABLE quiz_sessions ENABLE ROW LEVEL SECURITY
3 — Public read policy	supabase_migration.sql	CREATE POLICY Public read ... FOR SELECT USING (true)
4 — Public write policy	supabase_migration.sql	CREATE POLICY Public insert ... FOR INSERT WITH CHECK (true)

No ORM migration framework (e.g., Alembic) is used in v1.0. Schema changes require manual SQL execution in the Supabase Dashboard.

4. Interfaces

4.1. User Interface

The frontend is a single-page application (SPA) with five screens managed by JavaScript DOM show/hide logic. No page reloads occur during navigation. Screens:

Screen	**Route (Logical)**	**Primary Components**
Home	home	Hero banner, stats counters (topics, players ranked), CTA buttons
Quiz	quiz	Setup form (name, count, topic, difficulty) + active quiz (question card, options, progress bar, AI explanation)
Results	results	Score ring (CSS conic-gradient), grade badge, AI summary card, answer breakdown table
Leaderboard	leaderboard	Ranked list fetched from Supabase, medal icons for top 3
Ask AI	ask	Text input, submit button, AI answer card
Tips	tips	Card grid of 15 farming tips fetched from /api/tips

4.2. External Service Interfaces

Groq API

Property	**Value**
Endpoint	https://api.groq.com/openai/v1/chat/completions (via SDK)
SDK	groq-python (groq package)
Model	llama3-8b-8192
Authentication	Bearer token — GROQ_API_KEY environment variable
Request format	messages: [{role, content}], max_tokens, temperature

Response parsing	response.choices[0].message.content (string)

Supabase API

Property	**Value**
Endpoint	SUPABASE_URL (project-specific, e.g. https://xyz.supabase.co)
SDK	supabase-py (supabase package)
Authentication	anon/public key — SUPABASE_ANON_KEY environment variable
Transport	HTTPS / PostgREST auto-generated REST API
Operations used	INSERT (sessions), SELECT with ORDER + LIMIT (leaderboard)

4.3. Configuration Interface

All external credentials and configuration are managed via environment variables loaded through python-dotenv from a .env file at application startup:

Variable	**Required**	**Description**
GROQ_API_KEY	Yes	API key for Groq inference service
SUPABASE_URL	Yes	Full Supabase project URL (https://xxx.supabase.co)
SUPABASE_ANON_KEY	Yes	Supabase anonymous/public API key

5. State and Session Management

State and Session Management ensures that the system securely tracks users after login and manages quiz activities properly.

5.1 Session Management

The system uses token-based authentication (JWT).
After successful login:

- A secure token is generated.

- The token is sent to the user.
- The user includes the token in every request.
- The backend validates the token before processing.

This ensures secure and scalable session handling.

5.2 State Management

The backend follows a stateless architecture.

- User session data is not stored permanently in server memory.
- Quiz attempts, scores, and feedback are stored in the database.
- Each request contains necessary authentication information.

This approach improves security, scalability, and performance.

6. Caching

Caching is used in the Agriculture Quiz Bot to improve system performance and reduce response time.

The system caches:

- Frequently accessed topics
- Common quiz questions
- Knowledge retrieval results
- Temporary session data

An in-memory caching tool like Redis can be used.

Caching helps in:

- Faster responses
- Reduced database load
- Reduced AI API calls
- Better user experience

7. Non-Functional Requirements

Non-Functional Requirements define how the Agriculture Quiz/Tutor Bot should perform in terms of security, speed, reliability, and scalability.

The system must be secure, responsive, and capable of handling multiple users efficiently.

7.1 Security Aspects

Security is important to protect user data and system integrity.

The system ensures:

- Secure login using JWT-based authentication
- Encrypted passwords stored in the database
- HTTPS for secure communication
- Role-based access (User/Admin)
- Input validation to prevent attacks
- Secure storage of API keys

These measures ensure that user information and quiz data remain protected.

7.2 Performance Aspects

The system is designed to provide fast and smooth performance.

- Response time should be less than 3 seconds
- Efficient database queries using indexing
- Optimized AI prompts to reduce processing time
- Caching of frequently accessed data
- Support for multiple concurrent users

References

The following resources were referred to during the design and development of the Agriculture Quiz/Tutor Bot:

1. OpenAI – API Documentation and Integration Guides
2. Python Official Documentation – Backend Development Reference
3. FastAPI / Flask Documentation – API Development Framework
4. Vector Database Documentation (FAISS / Pinecone) – Knowledge Embedding and Retrieval
5. MySQL / PostgreSQL Documentation – Database Design and Management
6. Agriculture domain learning materials (Farming practices, Soil management, Irrigation techniques)