

# Project - High Level Design

On

## AgriLearn (AI Platform )

Course Name- GENRATIVE AI

**Institution Name:** Medicaps University – Datagami Skill Based Course

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

Sr no	Student Name	Enrolment Number
1	Deepak Patro	EN22CS301316
2	Dev kumar Das	EN22CS301321
3	Charu Nariya	EN22CS301290
4	Bhoomika Patidar	EN22CS301270
5	Avichal Mishra	EN22CS301243

*Group Name : 07D3*

*Project Number : GAI-31*

*Industry Mentor Name: Mr Suraj Nayak*

*University Mentor Name: Prof Vineeta Rathore*

*Academic Year: 2026*

## Table of Contents :-

### **1. Introduction.**

- 1.1. Scope of the document.
- 1.2. Intended Audience
- 1.3. System overview.

### **2. System Design.**

- 2.1. Application Design
- 2.2. Process Flow.
- 2.3. Information Flow.
- 2.4. Components Design
- 2.5. Key Design Considerations
- 2.6. API Catalogue.

### **3. Data Design.**

- 3.1. Data Model
- 3.2. Data Access Mechanism
- 3.3. Data Retention Policies
- 3.4. Data Migration

### **4. Interfaces**

### **5. State and Session Management**

### **6. Caching**

### **7. Non-Functional Requirements**

- 7.1. Security Aspects
- 7.2. Performance Aspects

### **8. References**

## 1. Introduction -

### 1.1. Scope of the document.

This document outlines the architectural and structural design of the AgriLearn AI application. It covers the frontend application logic, integration with external Large Language Model (LLM) APIs, state management within the Streamlit framework, and data flow mechanisms. It does not cover underlying infrastructure provisioning (like server deployment) or the internal mechanics of the Google Generative AI models.

### 1.2. Intended Audience.

This document is intended for software engineers, product managers, and UI/UX designers involved in the development, maintenance, or future scaling of the AgriLearn AI platform.

### 1.3. System overview.

AgriLearn AI is a dynamic, web-based educational tool that tests users' knowledge on modern agricultural topics. Built on Python and Streamlit, it leverages Google's Gemini Generative AI models to dynamically generate multiple-choice assessments and provide deep, contextual, and personalized feedback based on the user's specific performance.

## 2. System Design -

### 2.1. Application Design.

The application follows a monolithic, serverless frontend architecture. Streamlit serves as both the presentation layer and the application logic controller.

The application relies entirely on external APIs for content generation, meaning it does not house an internal database of questions or pre-written feedback.

**Application Design - Agriculture Quiz/Tutor Bot**

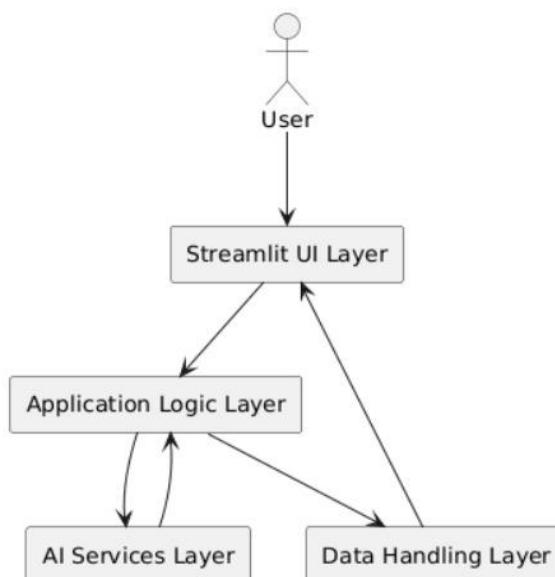


Figure – 2.1

## 2.2. Process Flow.

The user journey is divided into a three-stage sequential flow:

- Setup: The user selects a topic and difficulty level.
- Assessment: The application generates 5 questions. The user answers them one by one.
- Evaluation: The application grades the answers, fetches a detailed AI-generated analysis of the user's specific choices, and displays the final report.

**Process Flow Diagram**

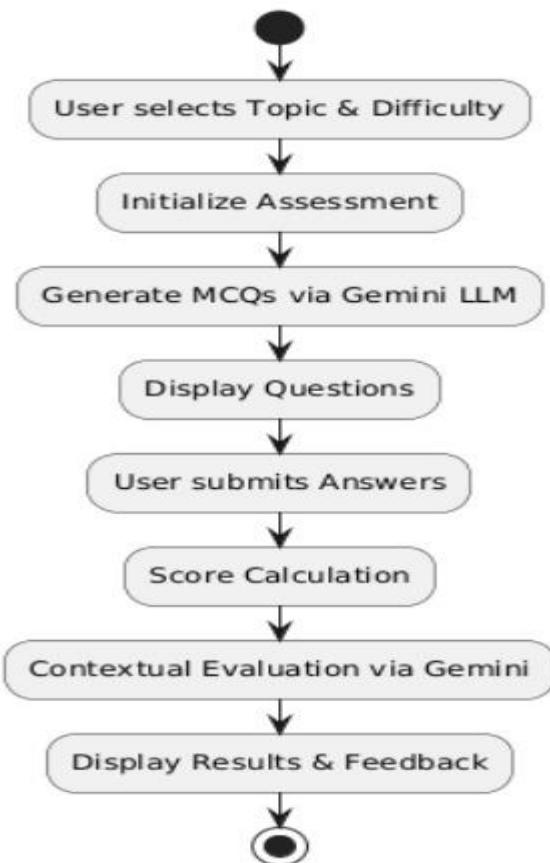


Figure – 2.2

## 2.3. Information Flow.

- Input: User selects parameters (Topic, Difficulty) in the UI.
- Prompt Generation: Application formats these parameters into a structured text prompt.
- API Call 1 (Generation): The prompt is sent to the Gemini API.
- Parsing: The AI returns a JSON array, which the app parses into Python dictionaries and stores in session memory.
- Interaction: User submits answers, which are appended to a list in memory.
- API Call 2 (Evaluation): Questions and user answers are compiled into a new prompt and sent to the Gemini API.
- Display: The resulting markdown evaluation is rendered on the screen.

### Information Flow Diagram

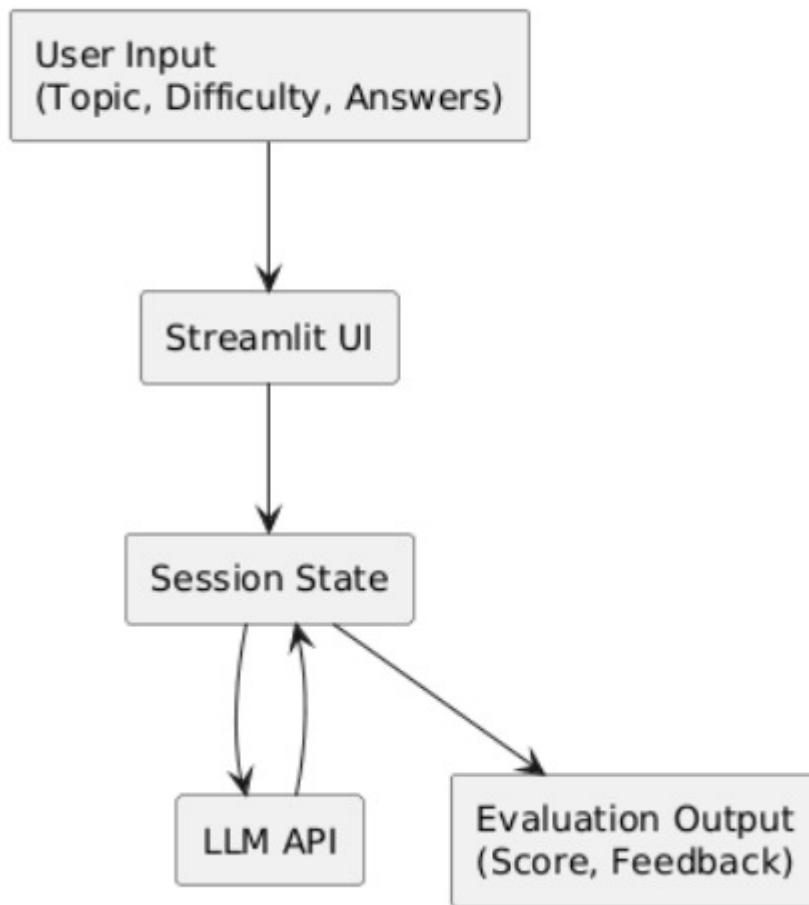


Figure – 2.3

#### 2.4. Components Design.

- Sidebar Configurator: Captures global parameters (topic, difficulty) and houses the system reset trigger.
- Quiz Engine: The core generative function (generate\_questions). It interfaces with a faster, lower-latency model to instantly build the curriculum.
- Assessment UI: The interactive loop that displays the progress bar, questions, options (via radio buttons), and hints.
- Evaluation Engine: The analysis function (get\_evaluation). It uses a reasoning-heavy model to analyze performance gaps and output formatted Markdown feedback.

### Components Design Diagram

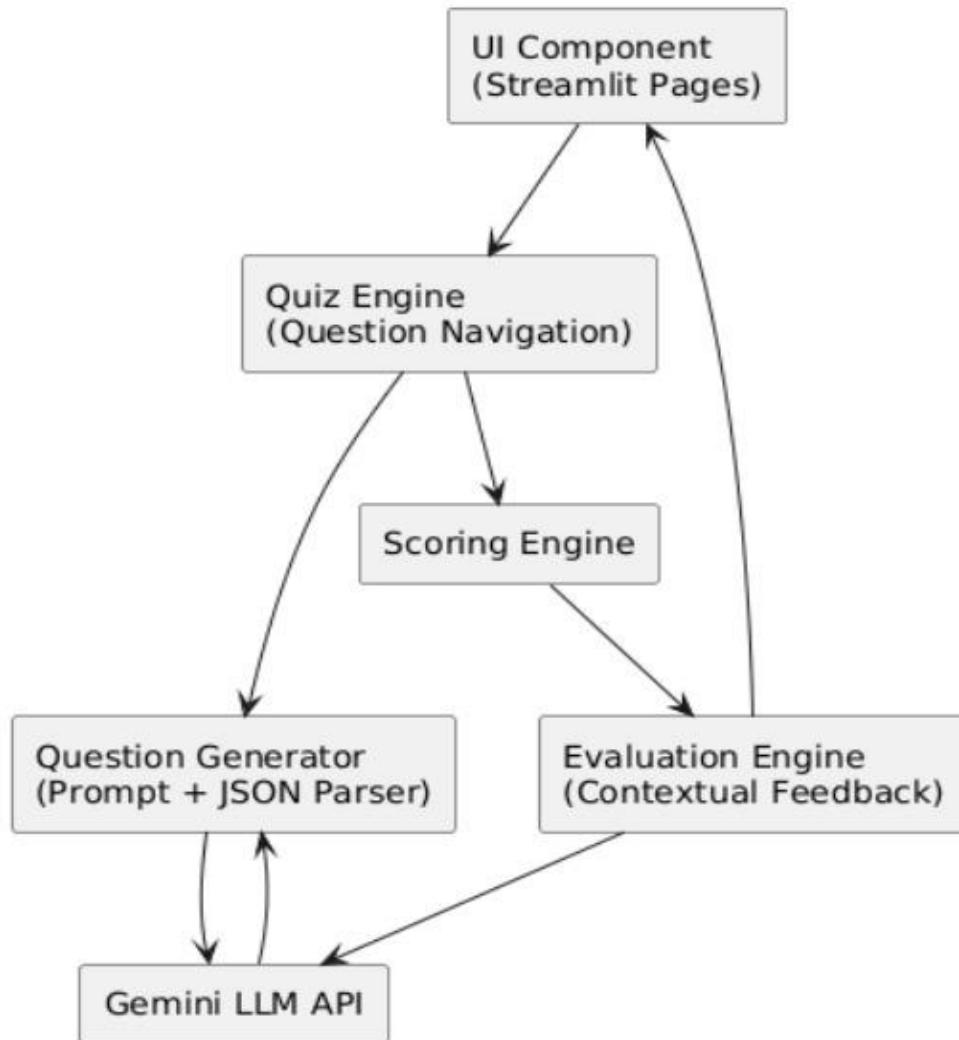


Figure – 2.4

### 2.5. Key Design Considerations.

- Synchronous Execution: Streamlit runs synchronously from top to bottom on every user interaction. The design must aggressively utilize state management to prevent infinite loops of API calls.
- Strict Output Formatting: Because the system relies on LLMs for programmatic data (the quiz questions), JSON schema enforcement is critical to prevent application crashes during the parsing phase.
- Security: API keys must be isolated from the source code via Streamlit's secrets management (.streamlit/secrets.toml).

## 2.6. API Catalogue.

The system acts as a client consuming the following external API endpoints (via the google-generativeai SDK):

- Content Generation (JSON):
  - *Payload*: Topic, Difficulty prompt.
  - *Expected Response*: Strict JSON array of objects (question, options, answer, hint).
  - *Purpose*: Generate quiz questions.
  
- Content Generation (Text/Markdown):
  - *Purpose*: Generate user feedback.
  - *Payload*: Formatted string of questions, correct answers, and user selections.
  - *Expected Response*: Formatted Markdown string.

### API Catalogue - LLM Integration

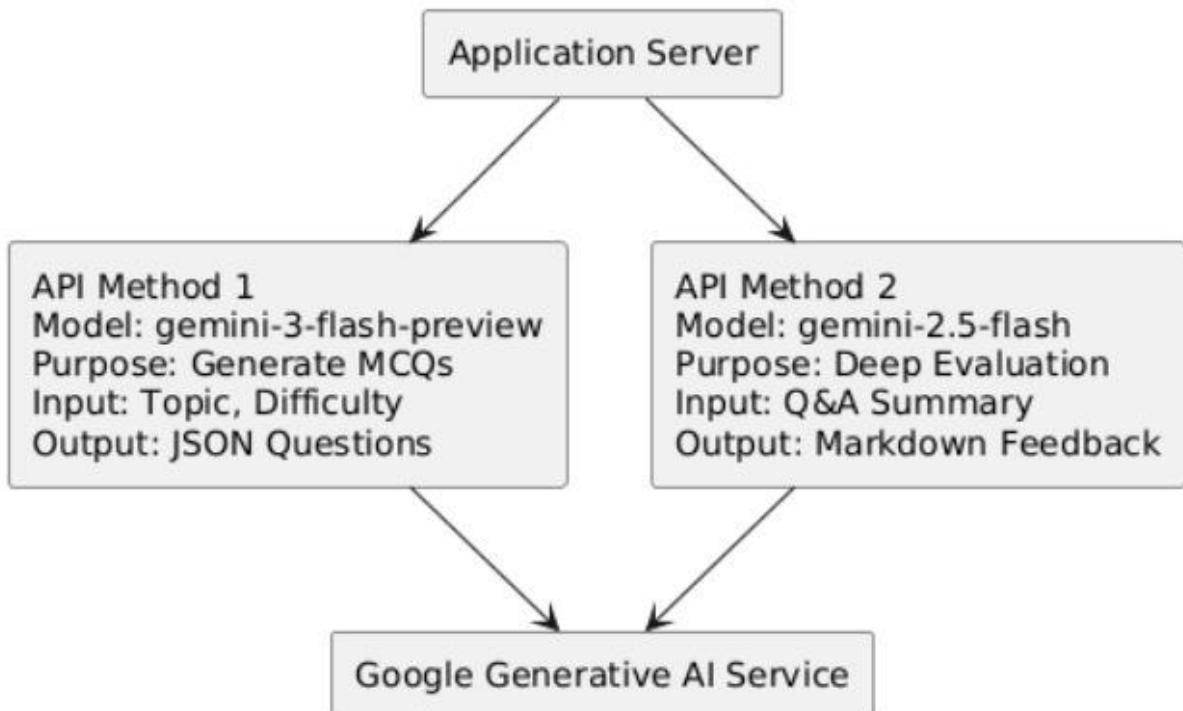


Figure – 2.5

### 3. Data Design -

#### 3.1. Data Model

The application does not use a traditional relational or NoSQL database. Data is modeled using ephemeral, in-memory Python objects:

- Question Object (Dictionary):
  - question (String)
  - options (Array of Strings, exactly 4)
  - answer (String, must match one option)
  - hint (String)
- Session State Variables:
  - phase (String: 'setup', 'quiz', 'results')
  - questions (List of Question Objects)
  - user\_answers (List of Strings)
  - current\_idx (Integer)
  - evaluation (String / Markdown text)

**Data Model Diagram**

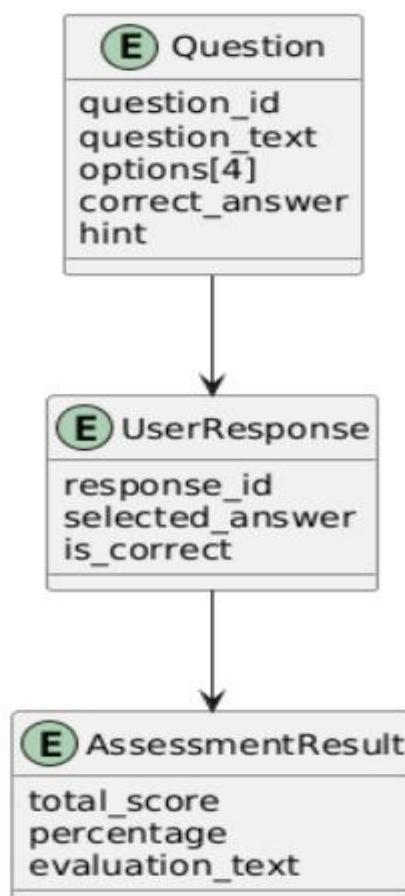


Figure – 3.1

### 3.2. Data Access Mechanism

All data is accessed and mutated directly via Streamlit's `st.session_state` dictionary object during the user's active browser session.

### 3.3. Data Retention Policies

Zero Retention. The application is entirely stateless between sessions. All generated questions, user answers, and AI evaluations are stored in volatile memory and are permanently destroyed when the user refreshes the page, closes the browser tab, or clicks "Restart Assessment."

### 3.4. Data Migration

Not Applicable (N/A). As there is no persistent storage, database schemas, or historical user data, data migration strategies are not required for this architecture.

## 4. Interfaces -

- User Interface (UI): A responsive web interface rendered by Streamlit, featuring sidebar navigation, progress indicators, interactive radio buttons, and expandable accordions (`st.expander`) for reviewing results.
- External AI Interface: HTTPS integration via the Google Generative AI Python SDK.

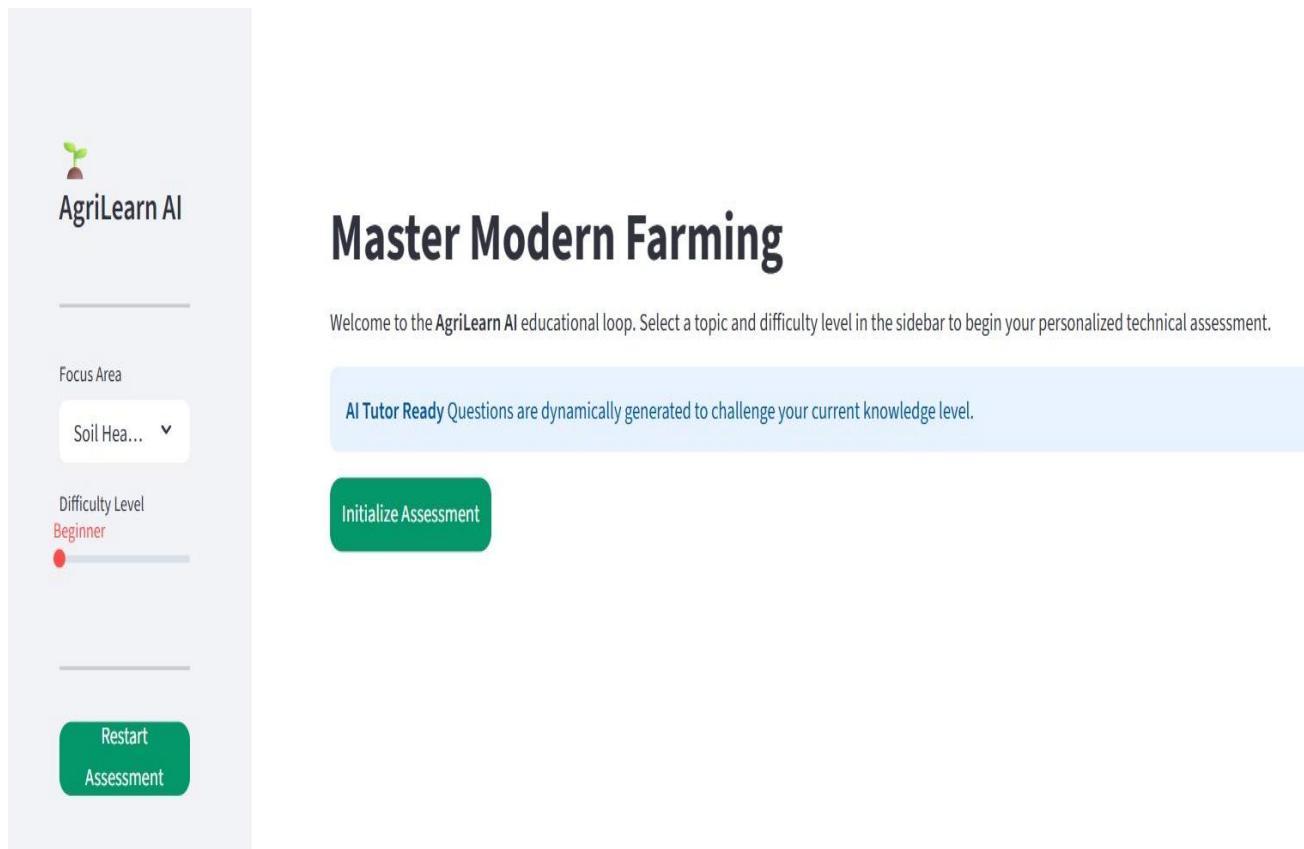


Figure – 4.1

Question 3 of 5 – Sustainable Irrigation

### Which of the following best describes the 'Crop Coefficient' (Kc) used in irrigation scheduling?

Select the best answer:

- The ratio of the actual crop evapotranspiration to the reference evapotranspiration
- The percentage of nitrogen required per unit of water applied
- The maximum depth to which a specific crop's roots can grow
- The rate at which water infiltrates a specific soil type

**Pro Tip:** It is a multiplier that adjusts the standard ET<sub>0</sub> value based on the specific crop type and its current growth stage.

**Submit Answer**

Figure – 4.2

### Assessment Analysis

Expertise Score  
20.0%  
1/5 Correct

#### Deep Contextual Feedback

As an expert agricultural scientist, I have reviewed your responses to the irrigation science questions. This evaluation provides a deep contextual analysis, elucidates the underlying scientific principles, and addresses potential misconceptions inherent in your selections.

#### Expert Evaluation of Irrigation Science Assessment

Your performance on this assessment demonstrates a foundational understanding in some areas of sustainable irrigation, alongside specific conceptual gaps that are critical in advanced agricultural water management. Let's delve into each response.

##### 1. Soil Water Tension Measurement for Precision Irrigation

Q: Which device is primarily used in sustainable irrigation to measure soil water tension, allowing for precise scheduling based on plant needs? User: Tensiometer **Correct:** Tensiometer

Evaluation: **Correct.** Your answer is spot-on and indicates a sound understanding of a fundamental tool in precision irrigation.

Scientific Explanation: A tensiometer is a cornerstone instrument for direct measurement of soil water potential, often expressed as soil water tension or suction. It consists of a porous ceramic cup connected via a water-filled tube to a vacuum gauge. As the soil dries, water is drawn out. The brilliance of the tensiometer lies in its direct correlation to plant water availability. Plants expend energy to extract water from the soil; the higher the soil water tension, the more energy required. By monitoring soil water tension, growers can determine precisely when irrigation is required.

##### 2. Goals of Regulated Deficit Irrigation (RDI)

Q: In the context of Regulated Deficit Irrigation (RDI), what is the primary goal during the non-critical growth stages of a crop? User: To maximize vegetative biomass production **Correct:** To reduce water application to induce beneficial stress and save resources

Evaluation: **Incorrect.** Your response reflects a common misconception regarding the core principles and objectives of RDI.

Figure – 4.3

## 5. State and Session Management

State management is the critical backbone of this application. Streamlit inherently reruns the entire script upon any user interaction (like clicking "Submit Answer"). To maintain continuity, the app uses `st.session_state` as a finite state machine:

- Initialization: Variables (phase, questions, etc.) are initialized only if they do not already exist in the session state.
- Phase Routing: Main conditional blocks (`if/elif phase == ...`) dictate which UI components render.
- Mutations: Variables are updated (e.g., `st.session_state.current_idx += 1`) right before an `st.rerun()` command, ensuring the next script execution loads the correct step in the flow.

### State and Session Management

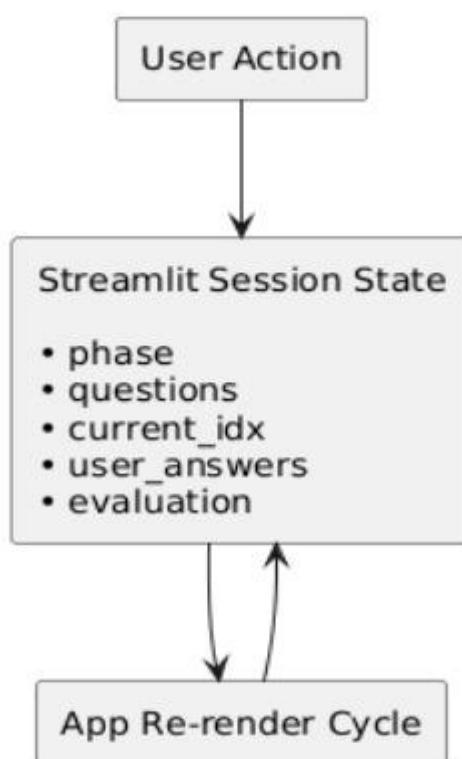


Figure – 5.1

## 6. Caching -

Currently, persistent caching (such as Streamlit's `@st.cache_data`) is not utilized.

- Why: The primary value of the app is *dynamic* content generation. Caching the `generate_questions` function would result in users getting the exact same quiz every time they selected "Advanced" and "Soil Health."
- Session Caching: The `st.session_state` acts as an ephemeral cache, ensuring that once a quiz is generated, or an evaluation is fetched, the API is not called again during that specific user's flow.

## 7. Non-Functional Requirements -

### 7.1. Security Aspects

- Credential Management: Hardcoded API keys are strictly prohibited in the source code. The application utilizes Streamlit's native secrets management. In local development, credentials are read from a `.streamlit/secrets.toml` file that is excluded from version control (`.gitignore`). In production deployments (e.g., Streamlit Community Cloud), secrets are injected securely via the platform's workspace settings.
- Data Privacy & Ephemerality: The system does not collect, log, or persist Personally Identifiable Information (PII) or user session data to any backend database. All user inputs, generated questions, and AI evaluations are strictly ephemeral and are permanently destroyed when the user terminates the session or refreshes the application.
- Prompt Injection Mitigation: By strictly typing the expected output from the LLM using JSON Schemas (`response_mime_type="application/json"`), the system heavily reduces the risk of anomalous, malformed, or malicious text breaking the application's parsing logic.
- Encryption in Transit: All interactions between the Streamlit frontend UI and the Google Generative AI API endpoints are encrypted in transit over HTTPS (TLS 1.2+).

### 7.2. Performance Aspects

- Latency Optimization via Model Tiering: The application intentionally splits computational workloads between two distinct AI models. The fast-tier model is utilized for question generation to ensure near-instantaneous curriculum creation, minimizing user wait time. The heavier, more capable model is reserved strictly for the final evaluation phase where deeper scientific reasoning takes precedence over immediate speed.
- Stateful Caching: To prevent redundant and expensive API calls, the system uses `st.session_state` as an active memory buffer. Once the final AI evaluation is generated, it is stored in state, allowing the user to seamlessly navigate the UI, open expanders, or review questions without triggering a re-generation.
- Stateless Scalability: Because the application does not rely on a persistent backend database and manages state purely on the client-session level, it can be horizontally scaled with ease on serverless container platforms like Google Cloud Run or Streamlit Community Cloud.

### Non-Functional Design Overview

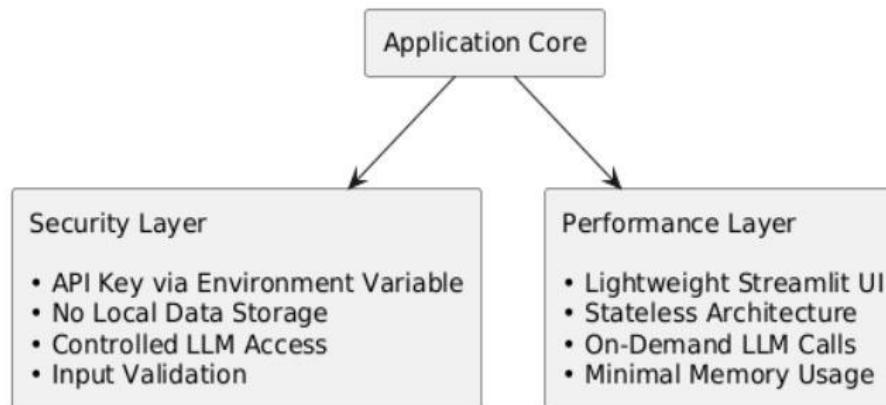


Figure – 7.1

## 8. References -

- [1] Google AI, “Gemini API Documentation (Gemini-2.5-Flash),” : <https://ai.google.dev>
- [2] Streamlit, “Streamlit Documentation,” Streamlit Inc.: <https://docs.streamlit.io>
- [3] Python Software Foundation, “Python 3 Language Reference,” : <https://www.python.org>
- [4] Food and Agriculture Organization, “FAO Knowledge Resources for Sustainable Agriculture,” FAO of the United Nations.: <https://www.fao.org>