Project -High Level Design

on

Real Estate Tutor 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	Cherry Mehta	EN22CS301292
2	Bhumika Choyal	EN22CS301276
3	Ayush Jain	EN22CS301247
4	Ayush Patidar	EN22CS301253
5	Diya Goyal	EN22CS301351

Group Name: Group 03D3

Project Number: GAI-27

Industry Mentor Name: Suraj Nayak

University Mentor Name:Vineeta Rathore

Academic Year:2022-26

Table of Contents

1. Introduction

The Real Estate Quiz / Tutor Bot is an AI-powered educational chatbot designed to evaluate and enhance users' understanding of real estate property listings. The system leverages a Large Language Model (LLM) API to assess user responses, generate contextual explanations, and provide personalized feedback.

The objective of this project is to demonstrate practical implementation of LLM-based evaluation systems integrated with structured datasets and conversational interfaces.

The system combines:

- LLM-based evaluation
- Prompt engineering
- Interactive chat-based UI
- Optional Retrieval-Augmented Generation (RAG)

1.1. Scope of the Document

This document provides a High-Level Design (HLD) for the Real Estate Tutor Bot. It covers:

- System architecture
- Application design
- Component-level breakdown
- Data flow
- API interactions
- Security and performance considerations
- Non-functional requirements

This document serves as a reference for developers, evaluators, and academic reviewers.

1.2. Intended Audience

This document is intended for:

- Academic evaluators
- Industry mentors
- University mentors
- Developers involved in system enhancement
- Technical stakeholders

1.3. System Overview

The Real Estate Tutor Bot is an AI-driven conversational application that:

- Generates quiz-style interactions
- Evaluates open-ended answers
- Provides contextual explanations
- Identifies conceptual gaps
- Reinforces learning

Unlike rule-based systems, this system uses a Large Language Model (Gemini API) to interpret user responses semantically and provide intelligent evaluation.

The system follows a layered modular architecture ensuring scalability and extensibility.

2. System Design

The Real Estate Quiz / Tutor Bot follows a modular, layered system design that separates user interaction, business logic, AI processing, and data management. The design ensures scalability, maintainability, and extensibility while integrating Large Language Model (LLM) capabilities for intelligent evaluation.

The system is structured into the following layers:

- Presentation Layer
- Application Layer
- AI Service Layer
- Data Layer

This separation ensures clear responsibility boundaries and future scalability.

2.1 Application Design

The application is designed as an interactive AI-driven chatbot system that evaluates user responses related to property listings.

Architectural Style

- The system follows a Layered Architecture Pattern, consisting of:

- Presentation Layer (Frontend)
- Implemented using Gradio.
- Provides chat-based interaction.
- Displays evaluation results and explanations.
- Application Layer (Backend Controller)
- Implemented in Python.
- Handles session state and conversation management.
- Constructs structured prompts for evaluation.
- Coordinates data retrieval and LLM communication.
- AI Service Layer
- Integrates with Gemini LLM API.
- Performs semantic evaluation of user answers.
- Generates scoring and contextual explanations.
- Data Layer
- Kaggle Real Estate dataset (CSV format).
- Optional vector database (FAISS) for contextual retrieval.
- Design Principles Followed
- Separation of Concerns
- Modularization
- API-based Integration
- Scalability-Oriented Structure
- Extensibility for RAG implementation

2.2 Process Flow

The system operates through the following step-by-step process:

Step 1: User Interaction

The user opens the chatbot interface and submits an answer or query related to property listings.

Step 2: Input Handling

The frontend captures the input and forwards it to the backend controller function.

Step 3: Prompt Construction

The backend:

- Formats the user input.
- Adds evaluation instructions.
- Appends scoring guidelines.

- Adds contextual data from dataset retrieval. (Optional)

Step 4: Context Retrieval (If RAG Enabled)

- If vector database is implemented:
- The user query is converted into an embedding.
- The embedding is compared with stored dataset embeddings.
- Top relevant property descriptions are retrieved.
- Retrieved context is appended to the prompt.

Step 5: LLM Evaluation

- The structured prompt is sent to the Gemini LLM API.
- The LLM performs:
- Semantic understanding
- Concept validation
- Error detection
- Score generation
- Explanation and reinforcement suggestions

Step 6: Response Formatting

The backend processes the LLM output and formats it into a readable response.

Step 7: Output Display

The final evaluation is displayed to the user in the chat interface.

2.3 Information Flow

The logical flow of information in the system is as follows:

User Input

↓

Chat Interface

↓

Backend Controller

↓

Prompt Builder

↓

(Optional) Vector Database Retrieval

↓

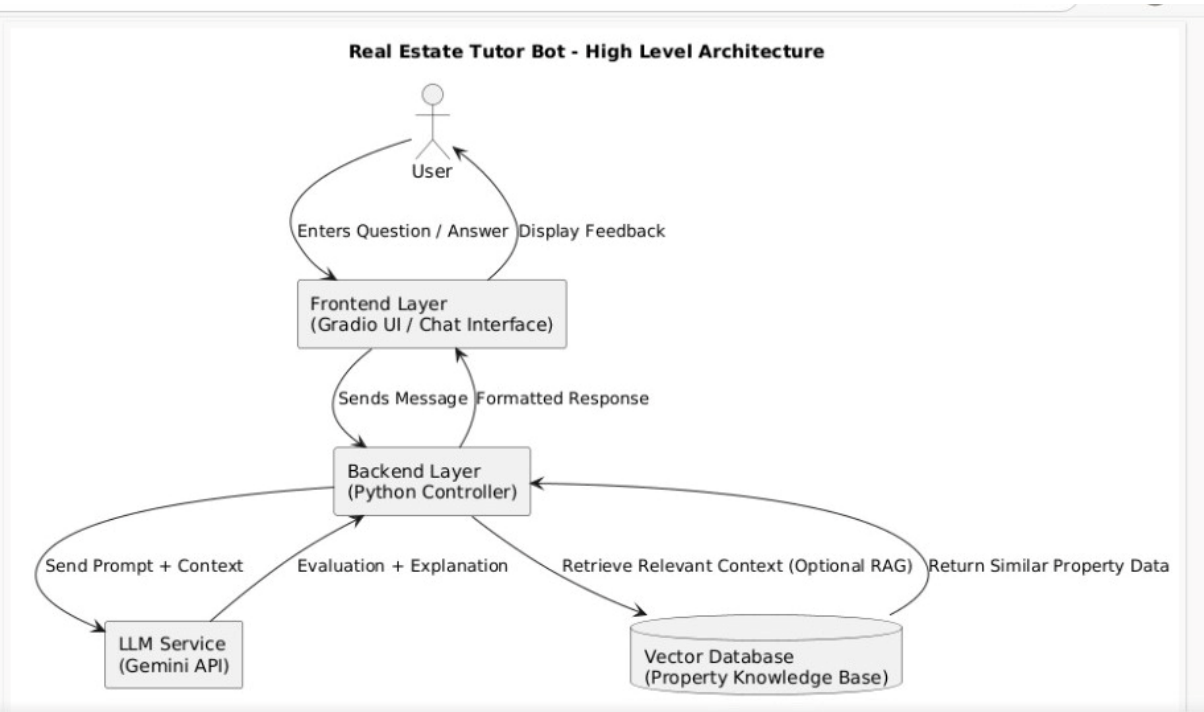
LLM API

↓

Response Processor

↓

Frontend Display



The information flow ensures minimal coupling between components and efficient communication between modules.

2.4 Components Design

The system is divided into the following components:

1. Chat Interface Module

Technology:

Gradio

Responsibilities:

- Capture user input
- Maintain conversation history
- Display AI responses
- Provide interactive chat experience

2. Chat Controller Module

Responsibilities:

- Receive input from frontend
- Manage conversation session
- Invoke prompt builder
- Call LLM service
- Return formatted output

This acts as the orchestration layer of the system.

3. Prompt Engineering Module

Purpose:

Ensures structured and consistent evaluation from the LLM.

Features:

- Defines system role (Real Estate Tutor Bot)
- Sets scoring instructions
- Requests explanation for incorrect answers
- Encourages reinforcement suggestions
- This module improves reliability and consistency of AI responses.

4. LLM Service Module

Responsibilities:

- Connect to Gemini API
- Send structured prompts
- Receive generated responses
- Handle API exceptions and errors
- This module encapsulates all external AI communication.

5. Data Management Module

Responsibilities:

- Load and preprocess Kaggle dataset
- Extract property listing data
- (Optional) Generate embeddings
- (Optional) Perform similarity search

If RAG is implemented:

- Embeddings are stored in FAISS
- Semantic search enhances contextual accuracy

2.5 Key Design Considerations

The system design was influenced by the following considerations:

1. Scalability

The modular design allows independent scaling of:

- Frontend
- Backend
- AI services
- Data storage

2. Maintainability

Clear separation between layers ensures easier debugging and upgrades.

3. Extensibility

- The system can be extended to:
- Web application
- Mobile application
- Cloud deployment
- Multi-user architecture

4. Accuracy Enhancement

Optional vector database allows Retrieval-Augmented Generation (RAG) to improve domain-specific evaluation.

5. Real-Time Performance

The system is optimized for near real-time responses (1–3 seconds depending on API latency).

6. Educational Focus

The design emphasizes explanation-based learning rather than simple scoring.

2.6 API Catalogue

1. Gemini LLM API

Purpose:

Evaluates user answers and generates contextual explanations.

Method Used:

generate_content()

Input Parameters:

- Structured prompt
- User response
- Context (if applicable)

Output:

- Evaluation score
- Detailed explanation
- Reinforcement suggestions

Protocol:

- HTTPS-based secure API call
- API key authentication

2. Embedding API (If RAG Implemented)

Purpose:

Convert property descriptions and user queries into vector representations.

Input:

Text data

Output:

High-dimensional embedding vector

3. Vector Database Interface (If Implemented)

Purpose:

- Store embeddings
- Perform similarity search
- Retrieve top-k relevant records
-

- Conclusion of System Design Section

The Real Estate Tutor Bot employs a layered and modular architecture integrating conversational AI with structured real estate data. The design ensures adaptability, scalability, and future extensibility toward full Retrieval-Augmented Generation and cloud deployment.

3. Data Design

Data Source:

Kaggle Real Estate Dataset

Data Format:

- CSV format
- Property descriptions
- Pricing information
- Location attributes

3.1. Data Model

Fields include:

- Property ID
- Location
- Price

- Area
- Bedrooms
- Amenities
- Description
- If Vector DB is used:
- Text embeddings stored as vectors
- Indexed for similarity search

3.2. Data Access Mechanism

- Dataset loaded into memory
- Embedding generation using LLM embedding API
- Vector search via FAISS (if implemented)

3.3. Data Retention Policies

- No personal user data stored

- Conversation data not persisted long-term
- Dataset stored locally
- API responses processed in real-time

3.4. Data Migration

If system scales:

- Dataset can migrate to cloud storage
- Vector DB can be moved to managed vector service
- LLM API configuration remains unchanged

4. Interfaces

User Interface:

- Chat-based UI
- Real-time interaction
- External Interface:
- Gemini LLM API

5. State and Session Management

- Conversation maintained using chat session object
- Session-based context retention
- No persistent database storage required

7. Non-Functional Requirements

- Scalability
- Reliability
- Usability
- Maintainability
- Extensibility

21. Security Aspects

- API key stored securely
- No exposure in frontend
- No sensitive user data stored

- Secure API communication via HTTPS

22. Performance Aspects

- Real-time API response (~1–3 seconds)

- Efficient memory usage
- Scalable architecture for future deployment
- Optional vector indexing improves retrieval efficiency

8. References

- Gemini LLM API Documentation
- Kaggle Real Estate Dataset
- Gradio Documentation
- FAISS Documentation (if implemented)