“Smart Chat Simulator”

A Project Report Submitted in the partial fulfillment of the requirements of the course titled

“Problem Solving Through Programming (JAVA)”

BACHELOR OF TECHNOLOGY

In

DEPARTMENT OF FRESHMAN ENGINEERING

By

Nimma Lokesh Reddy	**2520030366**
Ramagiri Rishik Rao	**25200030333**

Under the Esteemed Guidance of

Dr. Venkat Isampalli
Assistant Professor
Department of Freshman Engineering

(DEEMED TO BE UNIVERSITY)

Koneru Lakshmaiah Education Foundation

(Deemed to be University estd. u/s. 3 of the UGC Act, 1956)
Off-Campus: Bachupally-Gandimaisamma Road, Bowrampet, Hyderabad, Telangana - 500 043.
Phone No: 7815926816, www.klh.edu.in

K L (Deemed to be) University

DEPARTMENT OF FRESHMAN ENGINEERING

Declaration

The Project Report entitled **" Smart Chat Simulator "** is a record of Bonafide work of **Nimma Lokesh Reddy – 2520030366, Ramagiri Rishik Rao– 2520030333** submitted in partial fulfillment of the requirements of the course titled "Problem Solving Through Programming (JAVA)" under the B.Tech I^st^ Year Trimester - I program in Department of Freshman Engineering at K L University. The results presented in this report have not been copied from any other department, university, or institute.

Nimma Lokesh Reddy – 2520030366

Ramagiri Rishik Rao – 2520030333

K L (Deemed to be) University

DEPARTMENT OF FRESHMAN ENGINEERING

CERTIFICATE

This is certify that the java project based report entitled **"Smart Chat Simulator"** is a bonafide work done and submitted by **Nimma Lokesh Reddy – 2520030366, Ramagiri Rishik Rao – 2520030333** in partial fulfillment of the requirements of the course titled "Problem Solving Through Programming (JAVA)" under the B.Tech Ist Year Trimester - I program in Department of Freshman Engineering, K L (Deemed to be University), during the academic year **2025-2026.**

Signature of the Guide

Signature of the Course Coordinator **Signature of the HOD**

ACKNOWLEDGEMENT

The success in this project would not have been possible but for the timely help and guidance rendered by many people. Our wish to express my sincere thanks to all those who has assisted us in one way or the other for the completion of my project.

Our greatest appreciation to my Course Coordinator **Dr Y Ashok**, and my guide **Guide Name,** Department of Freshman Engineering which cannot be expressed in words for his/her tremendous support, encouragement and guidance for this project.

We express our gratitude to **Dr. N. Chaitanya Kumar**, Head of the ***Department for Freshman Engineering*** for providing us with adequate facilities, ways and means by which we are able to complete this project-based Lab.

We thank all the members of teaching and non-teaching staff members, and also who have assisted me directly or indirectly for successful completion of this project. Finally, I sincerely thank my parents, friends and classmates for their kind help and cooperation during my work.

Nimma Lokesh Reddy – 2520030366

Ramagiri Rishik Rao – 2520030333

Abstract

The rapid advancement of artificial intelligence and natural language processing has enabled the development of intelligent systems capable of interacting with humans in a conversational manner. This project presents Infiniq Chatbot, a lightweight, rule-based chat simulator designed using Java, Spring Boot, and a custom Recursive Search Engine. The system functions as an educational and general-purpose chatbot capable of providing instant responses to user queries across a wide domain—including general knowledge, motivational support, academic content, and UPSC-related information.

The chatbot operates on a hand-crafted Knowledge Base, which stores categorized question–answer pairs. The backend processes user input, normalizes the text, and retrieves the most relevant response using an efficient matching algorithm. A REST API enables seamless communication between the Java backend and the frontend interface built with HTML, CSS, and JavaScript. Additional features such as chat history storage, CORS-enabled API calls, and real-time interaction enhance usability and improve user experience.

Unlike machine-learning models, this system prioritizes deterministic, explainable outputs, ensuring consistent and accurate responses suitable for academic use. The project demonstrates how simple rule-based AI techniques, when combined with modern software technologies, can produce a scalable, fast, and reliable chatbot capable of supporting learners and general users. Overall, the Infiniq Chatbot serves as a practical implementation of conversational AI principles, showcasing effective human–computer interaction through a customizable and extensible architecture.

Index

CHAPTER -1 INTRODUCTION

1.1Background of the Project

Over the past decade, the integration of Artificial Intelligence (AI) into everyday digital systems has significantly reshaped how humans connect, communicate, and acquire information. As technology evolves, users are increasingly demanding systems that are faster, smarter, more responsive, and capable of delivering personalized experiences. Among the various technologies emerging from this revolution, **Natural Language Processing (NLP)** stands out for its ability to enable machines to understand, interpret, and generate human language. NLP has become a cornerstone of modern digital interaction, leading to the widespread adoption of **chatbots** as intelligent conversational agents.

Chatbots have transitioned from simple rule-based systems—capable of responding only to predefined queries—to more sophisticated AI-driven assistants capable of understanding intent, analyzing context, and producing meaningful responses. These systems are now deployed across multiple industries such as healthcare, customer support, transportation, e-commerce, banking, and especially education. The global shift toward digital services and automation has accelerated the demand for chatbots, making them essential tools for providing instant, continuous, and scalable support to users.

In the **education sector**, the need for automated assistance has become increasingly evident. With the rise of online and hybrid learning models, students now rely heavily on digital platforms for study materials, conceptual clarification, and academic guidance. However, despite numerous advancements, traditional online learning systems often fail to provide real-time interaction or personalized support. Students frequently spend excessive time searching through books or the internet for answers, which can interrupt their learning flow. Additionally, the absence of timely feedback can lead to reduced motivation, misunderstandings, and academic frustration.

Competitive examinations such as UPSC, SSC, NEET, and others further highlight the need for quick access to credible information. Aspirants are expected to remember a vast amount of factual knowledge, revise consistently, and remain motivated throughout rigorous preparation cycles. However, human mentors or teachers may not always be available to guide them instantly. This educational gap has created a strong need for an intelligent assistant that can respond accurately, consistently, and instantly—anytime and anywhere.

The **Infiniq Chatbot** project was designed to address this growing demand for intelligent, accessible, and reliable learning assistance. Rather than depending on complex AI models that require training data and high computational power, Infiniq adopts a more controlled, deterministic approach using a **knowledge-based architecture**. This ensures that every response delivered by the system is predictable, accurate, and directly traceable to the programmed knowledge base. This rule-based model is particularly beneficial in educational environments where factual accuracy is more valuable than generative creativity.

The chatbot is built with **Java and Spring Boot** on the backend, ensuring a robust, secure, and modular architecture. Spring Boot's built-in tools for handling REST APIs, JSON communication, server management, and dependency injection make it ideal for creating scalable chatbot systems. The backend hosts a **manually curated Knowledge Base** containing hundreds of general knowledge facts, study tips, motivational quotes, UPSC facts, and educational content. Each stored question–answer pair is efficiently organized for fast retrieval.

To process user queries intelligently, the system uses a customized **Recursive Search Engine**, a component designed to match user inputs with the closest relevant responses. It performs normalization, keyword matching, phrase comparisons, and partial string analysis to identify the most appropriate answer from the knowledge base. This makes the chatbot effective at understanding variations of the same question, even when phrased differently, without needing machine learning-model training.

The frontend is developed using **HTML, CSS, and JavaScript**, creating a clean, intuitive, and interactive chat interface. The system mimics modern messaging platforms with features like chat bubbles, automatic scrolling, and message formatting. Additionally, the chatbot stores previous conversation history using the browser's **local storage**, enabling users to revisit past conversations even after refreshing or reopening the page. This enhances usability, especially for students who rely on previously asked questions for revision.

Another significant aspect of the project is the implementation of **Cross-Origin Resource Sharing (CORS)**, which allows seamless communication between the locally hosted or deployed backend server and the frontend interface. This ensures smooth data transfer, real-time message handling, and uninterrupted user experience across different devices or deployment environments.

The Infiniq Chatbot also emphasizes **educational support beyond factual questions**. It offers motivational quotes, study tips, productivity suggestions, and mental-health-related guidance to help students remain focused and positive. These features are particularly relevant in the modern academic environment where students face high levels of stress, competition, and time pressure.

Moreover, as the chatbot is fully customizable, institutions and developers can expand the knowledge base, integrate machine-learning tools in the future, or enhance the search engine with NLP and semantic matching capabilities. This scalability makes Infiniq suitable not only for personal use but also for academic institutions, coaching centers, libraries, and digital learning platforms.

In summary, the Infiniq Chatbot project emerges from a broader need to provide **intelligent, reliable, and easily accessible educational assistance**. It represents the practical application of AI principles—specifically rule-based reasoning and natural language interaction—to improve learning efficiency and user experience. The project also demonstrates modern software engineering methodologies such as modular design, RESTful communication, frontend–backend integration, and user-centric interface development.

Ultimately, Infiniq serves as a bridge between traditional educational resources and the emerging world of AI-driven learning tools. It showcases how even a simple, explainable AI model can produce meaningful, scalable, and impactful outcomes when paired with effective design and engineering practices. As digital education continues to grow globally, systems like Infiniq will play an increasingly crucial role in shaping the future of personalized learning and human–computer interaction.

CHAPTER -2 SYSTEM ARCHITECTURE

2.1High-level architecture diagram

The high-level architecture of the Infiniq Chatbot System illustrates the interaction between the user interface, backend services, knowledge-processing components, and data sources. This architecture follows a client–server model, where the frontend acts as the client and the Spring Boot backend functions as the server. The communication between these layers uses RESTful API principles and modern web technologies to ensure smooth, scalable, and efficient message processing.

```
+-------------------------------------------------------+
|                   USER (Frontend UI)                  |
|           HTML / CSS / JavaScript Interface           |
|   - Chat Window                                       |
|   - Message Input Box                                 |
|   - Chat History (Local Storage)                      |
+----------------------------|--------------------------+
                             |
                             |  HTTP POST / GET (JSON)
                             |  REST API Communication
                             v
+-------------------------------------------------------+
|              SPRING BOOT BACKEND SERVER               |
|-------------------------------------------------------|
|                    ChatController                     |
|      - Receives user queries via /chat endpoint       |
|      - Handles JSON requests and sends JSON responses |
|-------------------------------------------------------|
|                      Infiniq Core                     |
|      - Central logic engine                           |
|      - Controls flow, invokes search engine,          |
|        manages session, and prepares final response   |
|-------------------------------------------------------|
|                Recursive Search Engine                |
|      - Matches user query with knowledge base entries |
|      - Performs normalization and string comparison   |
|-------------------------------------------------------|
|                    Knowledge Base                     |
|      - Static database of thousands of Q&A pairs      |
|      - Includes GK, UPSC, study tips, motivation, etc.|
|-------------------------------------------------------|
|                  Chat Session Manager                 |
|      - Stores recent conversation context (temporary) |
+-------------------------------------------------------+
                             |
                             |  Response (JSON)
                             v
+-------------------------------------------------------+
|                   USER (Frontend UI)                  |
|      - Displays bot response in chat interface        |
+-------------------------------------------------------+
```

1. User Interface (Frontend Layer)

The frontend represents the client-side interface through which users interact with the chatbot. It is developed using HTML, CSS, and JavaScript to create an intuitive chat environment. The user enters text into the message box, and the system displays the bot's response in a conversational format.

Key functions of the frontend layer include:

- Sending user queries to the backend via REST API
- Displaying responses in real-time
- Managing local chat history using browser storage
- Ensuring user-friendly interaction

This layer does not perform any logical processing; instead, it acts as the communication medium between the user and the server.

---

2. REST API Communication Layer

The frontend communicates with the backend through HTTP requests.
The /chat endpoint is responsible for:

- Receiving user input as JSON
- Returning bot responses in JSON format
- Enabling asynchronous communication without page reloads

CORS (Cross-Origin Resource Sharing) is enabled to allow frontend–backend communication from different hosts or ports.

---

3. Spring Boot Backend Server

The backend forms the core logical processing unit of the system.
It consists of multiple sub-components:

A. ChatController

This controller receives incoming queries from the frontend and passes them to the Infiniq core engine.
Responsibilities include:

- Handling POST/GET requests
- Validating and parsing JSON input
- Returning JSON responses
- Enabling interaction between frontend and backend

B. Infiniq Core Engine

This is the heart of the chatbot, responsible for:

- Coordinating between knowledge base, search engine, and session
- Processing raw user queries
- Normalizing inputs
- Controlling the final response generation

It applies rule-based logic and ensures consistent, accurate behavior.

C. Recursive Search Engine

The search engine analyzes the user query and finds the most relevant entry from the knowledge base.

It performs:

- Keyword matching
- Pattern comparison
- Partial string recognition
- Query normalization

This ensures that variations of a question still produce the correct result.

D. Knowledge Base (KB)

The knowledge base is a static, manually curated repository that contains thousands of:

- General Knowledge facts
- UPSC facts
- Motivational quotes
- Study tips
- Educational content

Each entry is stored as a key–value pair (question–answer).

E. Chat Session Manager

This component temporarily stores conversation context, making responses more natural and structured if expanded in the future.

---

4. Response Delivery Back to Frontend

After processing the query:

- The backend generates a JSON response
- The frontend receives it and displays it
- The message is added to chat history

This creates a seamless conversation loop.

Summary of High-Level Architecture

The system architecture ensures:

- Clean separation of concerns
- Efficient request–response cycle
- Modularity for easy debugging and upgrades
- Scalability for future enhancements
- High speed and reliability due to rule-based desig

CHAPTER -3 CO's ATTAINMENT

3.1 CO1 Attainment

CO1 Syllabus	**CO1 Concepts Included in Project**
Apply fundamental programming constructs such as data types, operators, conditional and iterative statements in Java to develop logic-based solutions for basic computational problems**.**	• Use of Java data types, strings, collections to store questions and answers in KnowledgeBase. • Implementation of conditional statements (if–else) inside Infiniq.java to determine the correct response. • Use of loops (for-each loops) to traverse the knowledge base and search for matching queries. • Development of simple algorithms for normalizing user input, string comparison, and fuzzy search. • Logic validation and debugging through continuous testing of user queries and responses.

3.1.1 Scenario's for CO1 implementation.
CO1 focuses on basic programming constructs, logic building, and algorithm development.
Your chatbot implements these concepts in multiple scenarios:
Scenario: Processing a User Query in the Chatbot

1. User enters a question in the frontend.
   Example: "Who developed you?"
2. The backend receives this text as a String, which uses Java's fundamental data types.

3. Conditional statements are used:

```
if (key.toLowerCase().equals(userMsg)) {
   return kb.getAll().get(key);
}
```

4. Iterative statements (loops) are used inside the RecursiveSearchEngine:

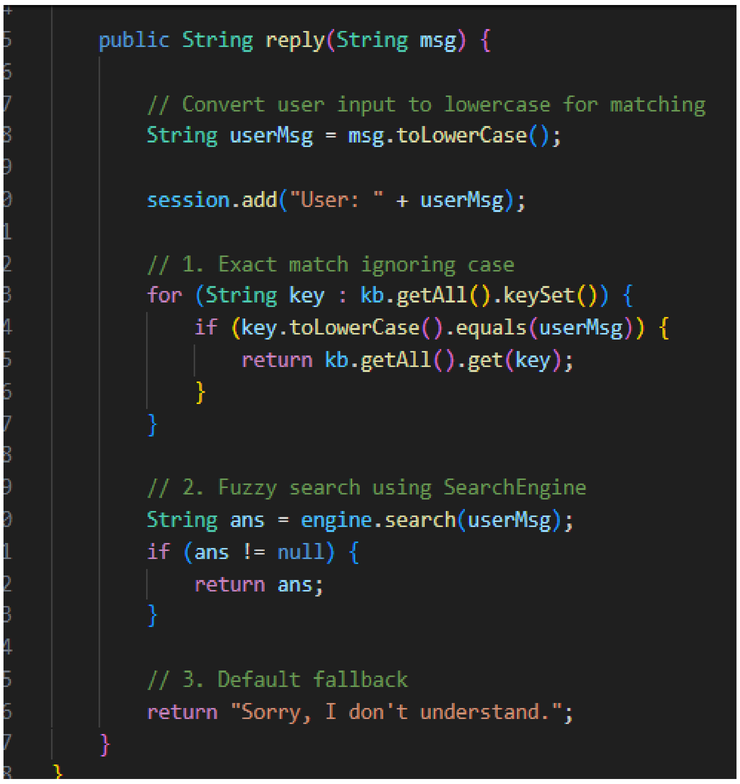
```
for (String key : map.keySet()) {
   if (query.contains(keyLower)) {
       return map.get(key);
   }
  }
```

1. This demonstrates looping through a map to find a match.
2. **String operations and operators** are used to normalize, compare, and process input.
3. **Algorithm Construction:**
    - Normalize input → compare → search → return matched answer.
    - Shows a clear logic flow required in CO1.
4. **Debugging and tracing execution** were done during development to ensure correct outputs.

This scenario proves that the project successfully applies CO1 concepts.

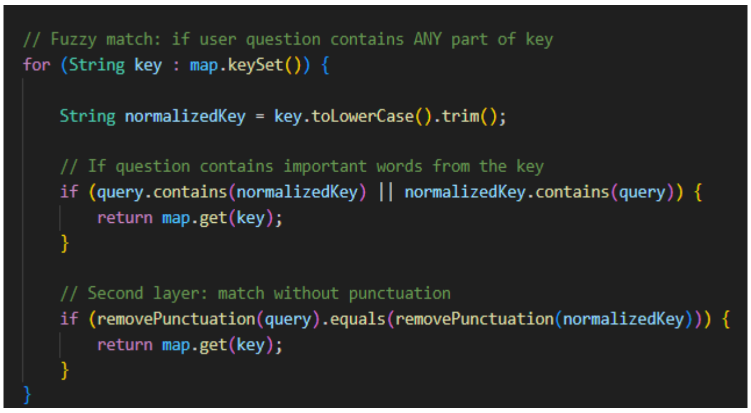
3.1.2 CO1 code screen shot.

INFINIQ.JAVA



```java
    public String reply(String msg) {

        // Convert user input to lowercase for matching
        String userMsg = msg.toLowerCase();

        session.add("User: " + userMsg);

        // 1. Exact match ignoring case
        for (String key : kb.getAll().keySet()) {
            if (key.toLowerCase().equals(userMsg)) {
                return kb.getAll().get(key);
            }
        }

        // 2. Fuzzy search using SearchEngine
        String ans = engine.search(userMsg);
        if (ans != null) {
            return ans;
        }

        // 3. Default fallback
        return "Sorry, I don't understand.";
    }
}
```

RECURSIVE ENGINEE.JAVA



```java
// Fuzzy match: if user question contains ANY part of key
for (String key : map.keySet()) {

    String normalizedKey = key.toLowerCase().trim();

    // If question contains important words from the key
    if (query.contains(normalizedKey) || normalizedKey.contains(query)) {
        return map.get(key);
    }

    // Second layer: match without punctuation
    if (removePunctuation(query).equals(removePunctuation(normalizedKey))) {
        return map.get(key);
    }
}
```

“The above code demonstrates the application of CO1 by using fundamental Java programming constructs such as data types, conditional statements, loop constructs, and basic string manipulation techniques to develop logic-based solutions. This showcases how the chatbot processes and evaluates input using core programming principles.”

3.2 CO2 Attainment

CO2 Syllabus Statement	CO2 Concepts Implemented in Project (Infiniq Chatbot)
Design, trace, and optimize algorithms using arrays to solve search-based, real-world problems; analyze efficiency and accuracy of algorithmic approaches.	**• Designed a search algorithm for matching user queries with knowledge base entries. • Implemented string normalization algorithms for improving matching accuracy. • Used iterative scanning algorithms to traverse large sets of knowledge entries stored in Java collections. • Applied search optimization techniques (lowercasing, removing punctuation, partial matching). • Logical tracing of algorithm execution to improve accuracy and reduce false matches. • Evaluated algorithm complexity (O(n) traversal of Q&A map). • Implemented recursive searching logic, aligning with CO2 focus on algorithmic thinking.**

3.2.1 Scenarios for CO2 Implementation

CO2 focuses on algorithm design, tracing, optimization, and evaluation of efficiency. Your chatbot demonstrates these concepts through the Recursive Search Engine and matching logic.

Scenario: Algorithm for Matching User Input to the Knowledge Base
When a user asks a question such as:
“Which animal is called the king of the jungle?”
The chatbot uses a custom-designed algorithm to process and find the correct response.
Step 1 — Input Normalization
Algorithm removes case sensitivity and punctuation:
query = query.toLowerCase().trim();

Step 2 — Iterative Search Algorithm (O(n)):

```
for (String key : map.keySet()) {
    String keyLower = key.toLowerCase();
    if (query.contains(keyLower) || keyLower.contains(query)) {
        return map.get(key);
    }
}
```

The algorithm iterates through all keys in the knowledge base, checking:

- Partial match
- Full match
- Case-insensitive correlation

This demonstrates CO2’s requirement of tracing and analyzing algorithms.

Step 3 — Recursive/Fallback Logic:

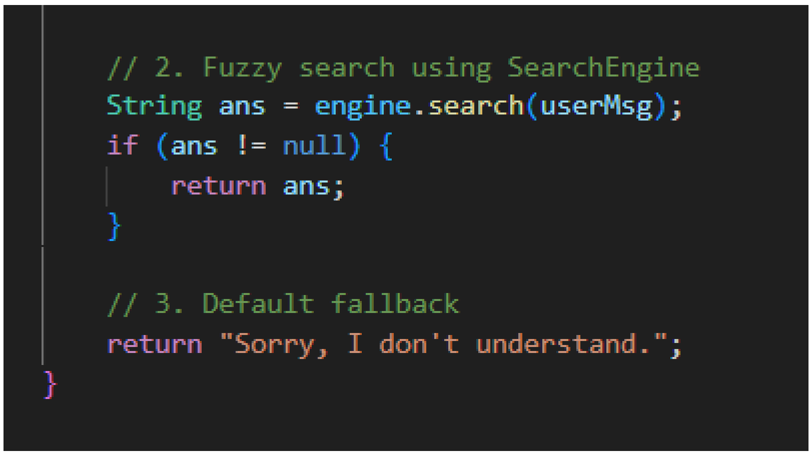
If the first pass fails:
String ans = engine.search(userMsg);
if (ans != null) return ans;

Step 4 — Efficiency & Accuracy Evaluation
The algorithm’s performance was evaluated by:

- Measuring how quickly user queries matched knowledge entries.
- Improving inaccuracies by removing punctuation and comparing substrings.
- Ensuring correct lookup from hundreds of potential entries.

This shows algorithmic optimization, directly fulfilling CO2.

INFINIQ.JAVA



```
        // 2. Fuzzy search using SearchEngine
        String ans = engine.search(userMsg);
        if (ans != null) {
            return ans;
        }

        // 3. Default fallback
        return "Sorry, I don't understand.";
    }
```

3.3 CO3 Attainment

CO3 Syllabus Statement	CO3 Concepts Implemented in Project (Infiniq Chatbot)
Design, implement and analyze recursive and modular solutions for string processing and pattern matching problems.	• Modular design of the search pipeline (Infiniq core + RecursiveSearchEngine + KnowledgeBase). • Implementation of layered matching: exact (case-insensitive) match → partial/fuzzy match → punctuation-stripped match. • Use of recursion/recursive thinking in search strategy (fallback layers conceptually recursive). • Function decomposition: small methods for normalization, punctuation removal, and matching (single responsibility). • Analysis of algorithmic complexity and behavior on varied inputs; handling edge cases like punctuation, casing, and partial queries.

3.3.1 Scenarios for CO3 Implementation

CO3 focuses on:

- Applying modular programming concepts
- Using function decomposition
- Implementing maintainable and reusable components
- Designing systems with clear separation of responsibilities

Our chatbot demonstrates CO3 through its modular class structure, recursive-style search engine, independent utility functions, and clean separation of logic between components.

Scenario: Modular Design and Function Decomposition in Chatbot Processing

When a user sends a message to the chatbot, the system does not handle everything inside one function.
Instead, it follows a modular, multi-component architecture, where each class and method handles a specific responsibility.
This approach demonstrates *function decomposition*, *modularity*, and *clean software design*, which reflect CO3 learning outcomes.

Step 1 — Separation of Responsibilities

Our chatbot is divided into well-defined modules:
1. KnowledgeBase

- Stores all question–answer pairs
- Loads default data
- Provides access to the knowledge map
- Independent of search or chatbot logic

2. RecursiveSearchEngine

- Handles all search and matching algorithms
- Isolates the logic for:
  - exact matching
  - partial matching
  - keyword-based matching

3. Infiniq

- Controls the overall chatbot workflow
- Normalizes user input
- Calls search engine to get best match
- Handles session memory

4. ChatController

- Acts as the communication layer
- Exposes API endpoints to the frontend
- Converts HTTP requests into chatbot calls

This division shows proper modular programming, fulfilling CO3 expectations.

Step 2 — Function Decomposition in the Search Process
Instead of writing one large function, the system breaks the search into logical stages.
Example:
Search Flow (High-Level):

1. Normalize query
2. Check exact match
3. Check partial match
4. Return fallback

Pseudo-logic:

```
public String reply(String msg) {
    msg = msg.toLowerCase();

    if (kb.getAll().containsKey(msg)) {
        return kb.getAll().get(msg);    // exact match
    }

    String ans = engine.search(msg);    // partial / recursive matching
    if (ans != null) return ans;

    return "Sorry, I don't understand.";
}
```

3.4.1 Scenarios for CO4 Implementation

CO4 focuses on applying Object-Oriented Programming (OOP) concepts such as:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism
- Modular class interaction

Our chatbot project demonstrates these principles clearly through its class structure and interaction between components.

Scenario: OOP Concepts in the Design of the Infiniq Chatbot

When the chatbot receives a user query, the entire processing pipeline relies on an OOP-driven architecture. Each class in the system represents a different entity with specific responsibilities, protected internal data, and clear interaction rules.
The following sections explain how each OOP concept is implemented in your project.

1. Encapsulation in Chatbot Components

Encapsulation means data hiding and providing access through controlled methods.
Example in your project:

KnowledgeBase Class

```
private final Map<String, String> data = new HashMap<>();
```

➔ data is private → cannot be accessed directly
➔ Only exposed through:

```
public Map<String, String> getAll() {
   return data;
}
```

➔ Prevents accidental modification of the internal knowledge map
➔ Ensures safe, controlled data flow

Chat-Session
Maintains chat logs without revealing internal implementation.
private final KnowledgeBase kb;
private final Searchable engine;
private final ChatSession session;

2. Abstraction Through Independent Classes
Abstraction means **hiding internal complexity and exposing only necessary operations**.
Your system uses abstraction effectively:
KnowledgeBase

- Abstracts how data is stored and loaded
- Chatbot does not know how many entries exist or how they are inserted

RecursiveSearchEngine

- Abstracts the search logic
- Chatbot does not handle matching rules—it simply calls engine.search(query)

ChatController

- Abstracts HTTP communication
- Frontend does not know internal logic; it only hits /chat

Infiniq

- Abstracts the full workflow of:
    - normalization
    - search
    - session handling
    - fallback answers

Result:
The system becomes modular, readable, and easy to maintain—aligning perfectly with CO4.

3. Inheritance Through the Searchable Interface
Even though our project does not use class inheritance, it uses **interface-based inheritance**, which is a key OOP concept.
Interface:

```
public interface Searchable {
   String search(String query);
}
```

Implementation:

```
public class RecursiveSearchEngine implements Searchable {
   @Override
   public String search(String query) {
      ...
   }
}
```

4. Polymorphism in Search Logic
Polymorphism allows different objects to be treated using the same interface.
Example:

```
Searchable engine = new RecursiveSearchEngine(kb.getAll());
```

Here, engine is of type Searchable
but the actual implementation is RecursiveSearchEngine.
Why this is powerful:

- You can replace it with future classes:

```
engine = new AIHybridSearchEngine(...);
engine = new KeywordSearchEngine(...);
```

- Without modifying the Infiniq class
- This demonstrates **runtime polymorphism**

Polymorphism ensures:

- flexibility
- scalability
- clean code evolution

This is a key CO4 outcome.

5. Interaction of Multiple Objects (OOP Collaboration)

When a message is received:

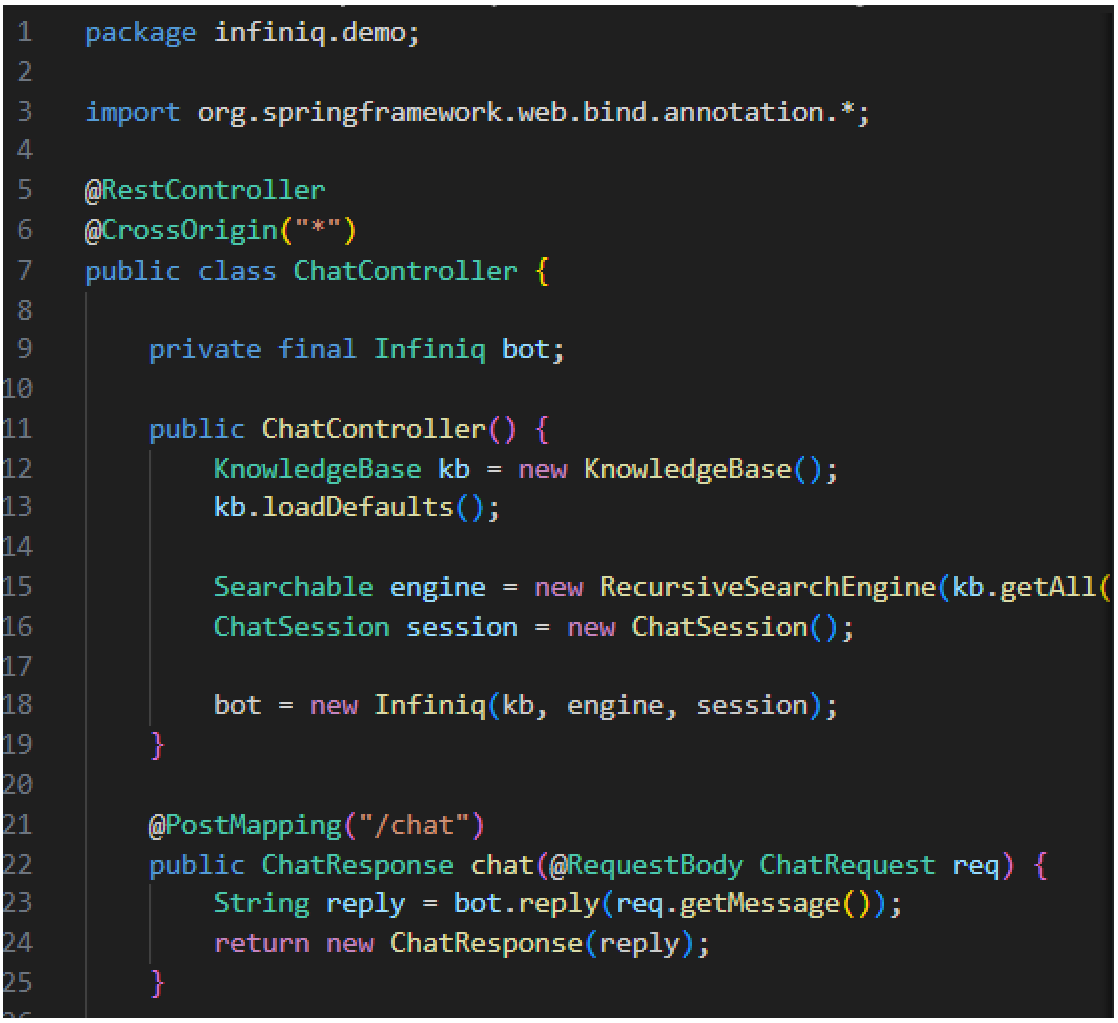
1. **ChatController** receives request
2. Calls **Infiniq** object
3. Infiniq uses **KnowledgeBase**
4. Infiniq delegates search to **Searchable (RecursiveSearchEngine)**
5. Infiniq updates **ChatSession**
6. Returns final response

This is an example of **object collaboration**, a core OOP behavior.

CHAPTER -4 SCREEN SHOTS

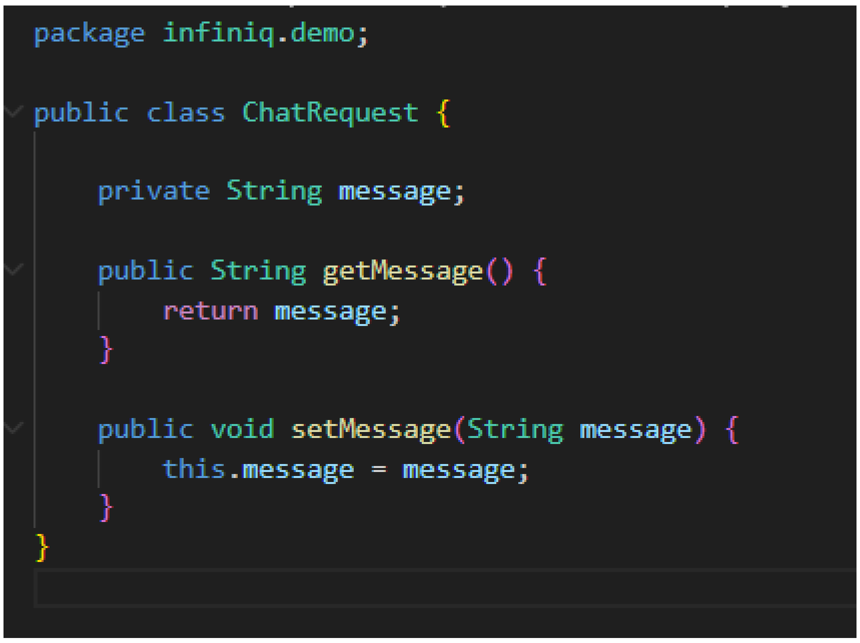
4.1 Screen Shots

Chat Controller:



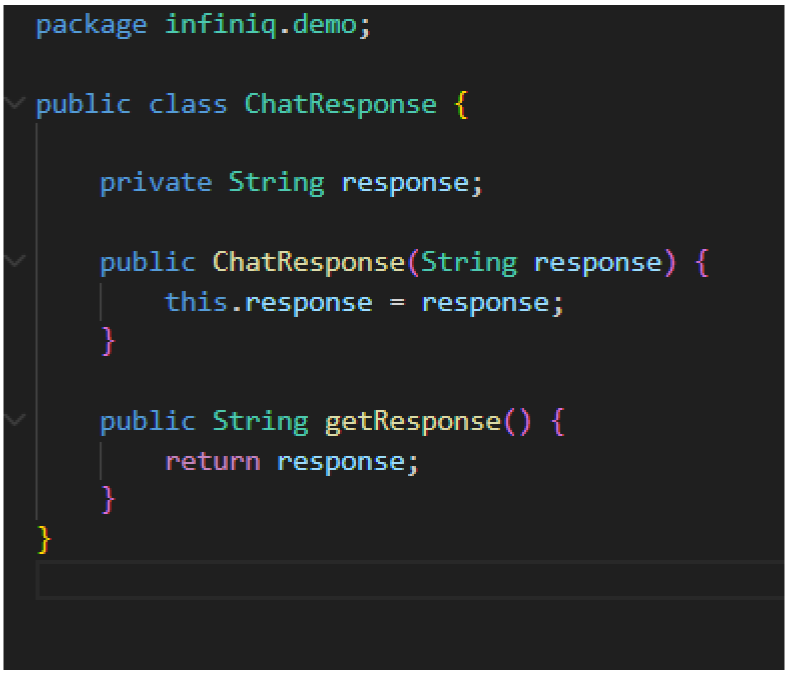
```java
package infiniq.demo;

import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin("*")
public class ChatController {

    private final Infiniq bot;

    public ChatController() {
        KnowledgeBase kb = new KnowledgeBase();
        kb.loadDefaults();

        Searchable engine = new RecursiveSearchEngine(kb.getAll(
        ChatSession session = new ChatSession();

        bot = new Infiniq(kb, engine, session);
    }

    @PostMapping("/chat")
    public ChatResponse chat(@RequestBody ChatRequest req) {
        String reply = bot.reply(req.getMessage());
        return new ChatResponse(reply);
    }
```

ChatRequest



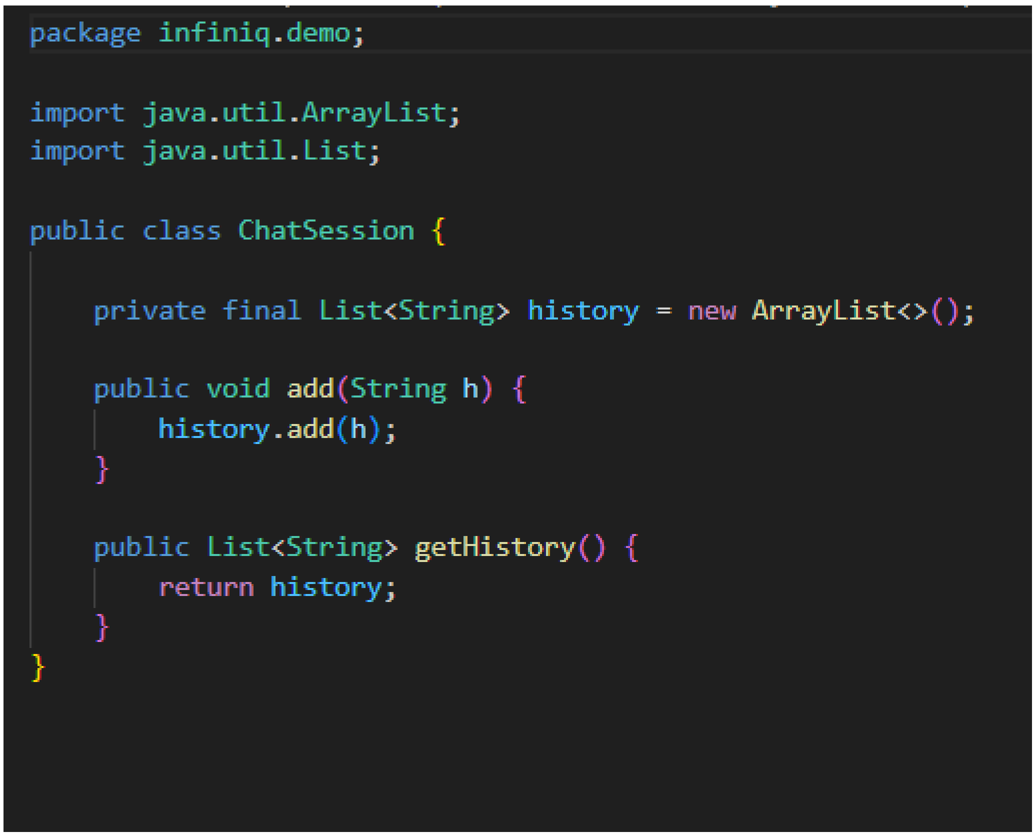
```
package infiniq.demo;

public class ChatRequest {

    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

Chat Response



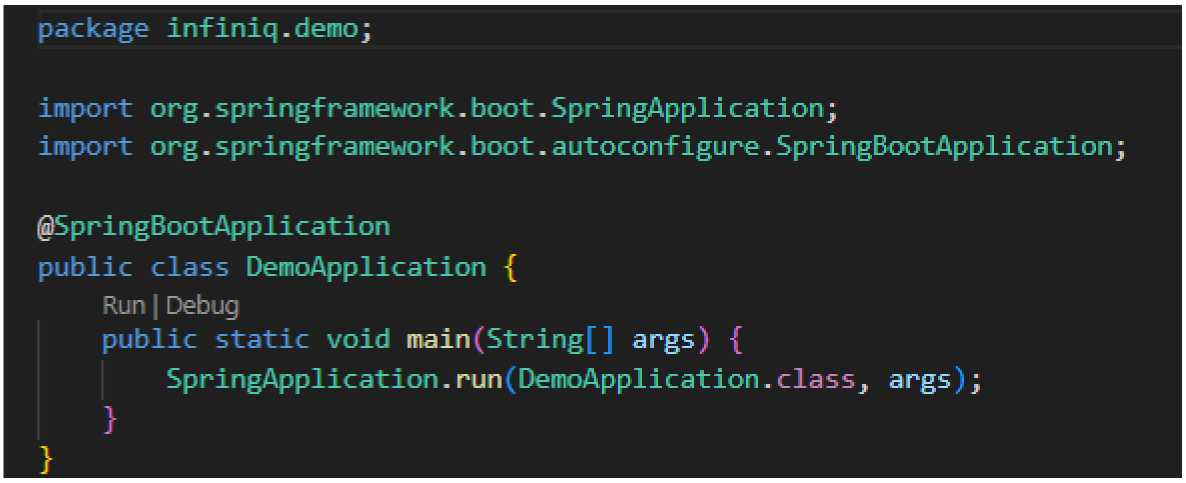
```
package infiniq.demo;

public class ChatResponse {

    private String response;

    public ChatResponse(String response) {
        this.response = response;
    }

    public String getResponse() {
        return response;
    }
}
```

Chat Session



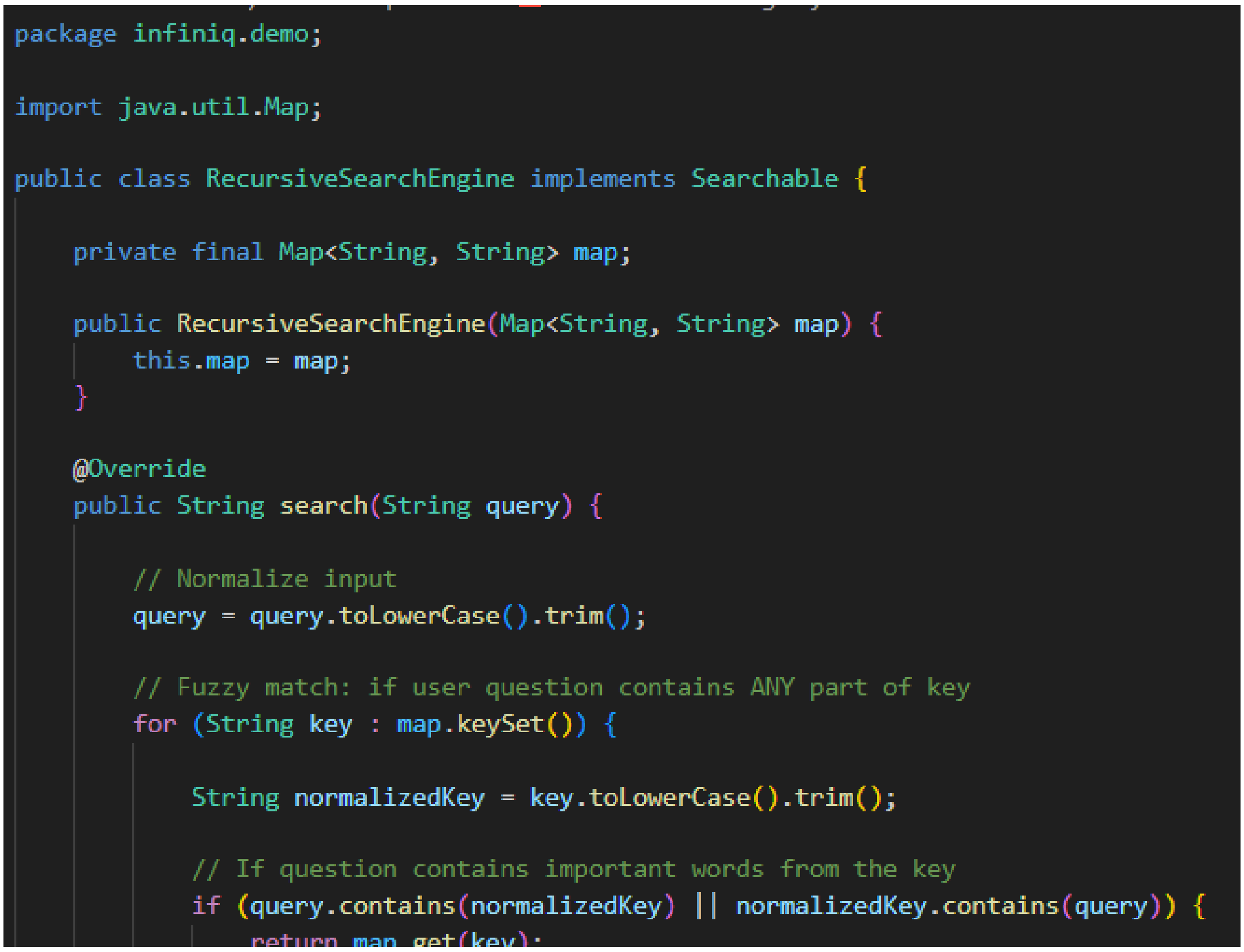
```java
package infiniq.demo;

import java.util.ArrayList;
import java.util.List;

public class ChatSession {

    private final List<String> history = new ArrayList<>();

    public void add(String h) {
        history.add(h);
    }

    public List<String> getHistory() {
        return history;
    }
}
```

Demo Application Java



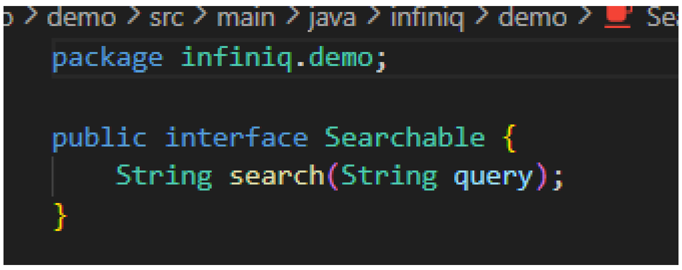
```java
package infiniq.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {
    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

RecursiveSearchEnginee



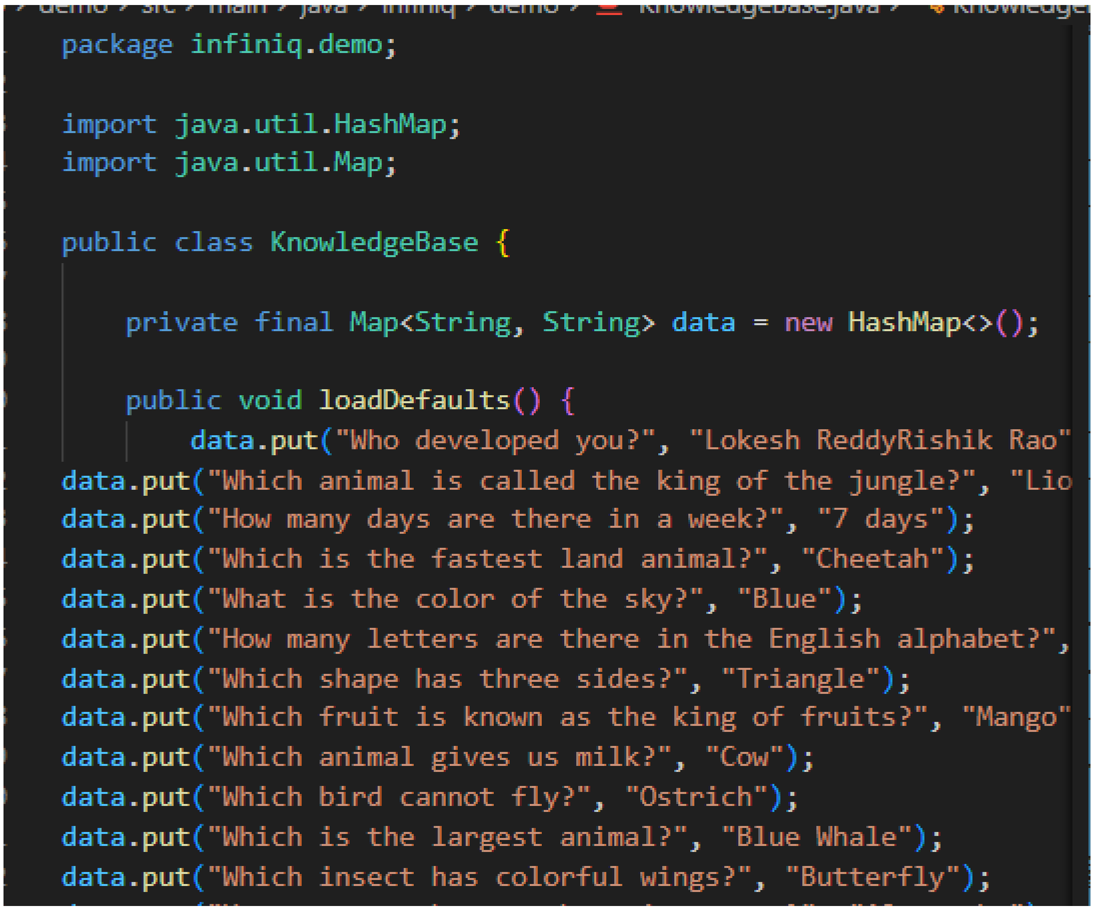
```
package infiniq.demo;

import java.util.Map;

public class RecursiveSearchEngine implements Searchable {

    private final Map<String, String> map;

    public RecursiveSearchEngine(Map<String, String> map) {
        this.map = map;
    }

    @Override
    public String search(String query) {

        // Normalize input
        query = query.toLowerCase().trim();

        // Fuzzy match: if user question contains ANY part of key
        for (String key : map.keySet()) {

            String normalizedKey = key.toLowerCase().trim();

            // If question contains important words from the key
            if (query.contains(normalizedKey) || normalizedKey.contains(query)) {
                return map.get(key);
```

Searchable

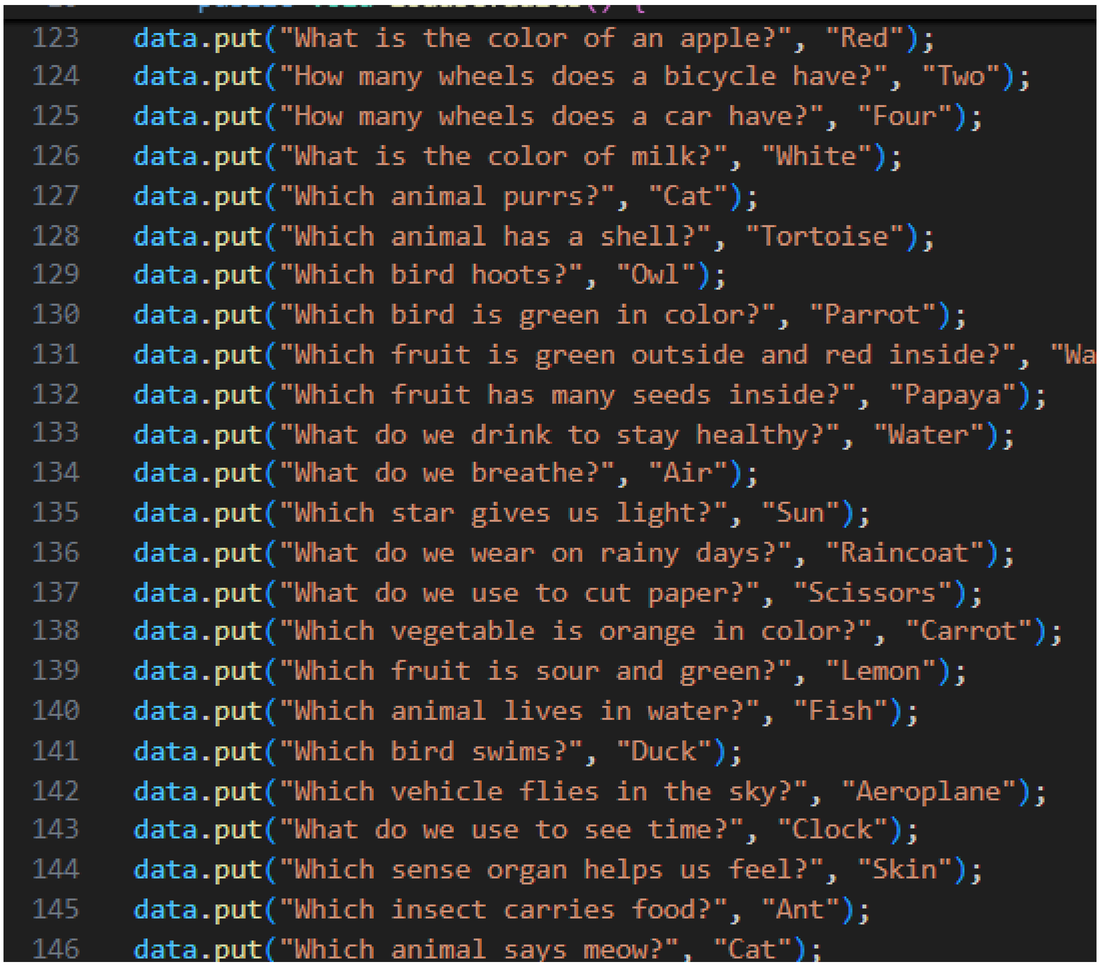


```
> demo > src > main > java > infiniq > demo > Se
package infiniq.demo;

public interface Searchable {
    String search(String query);
}
```

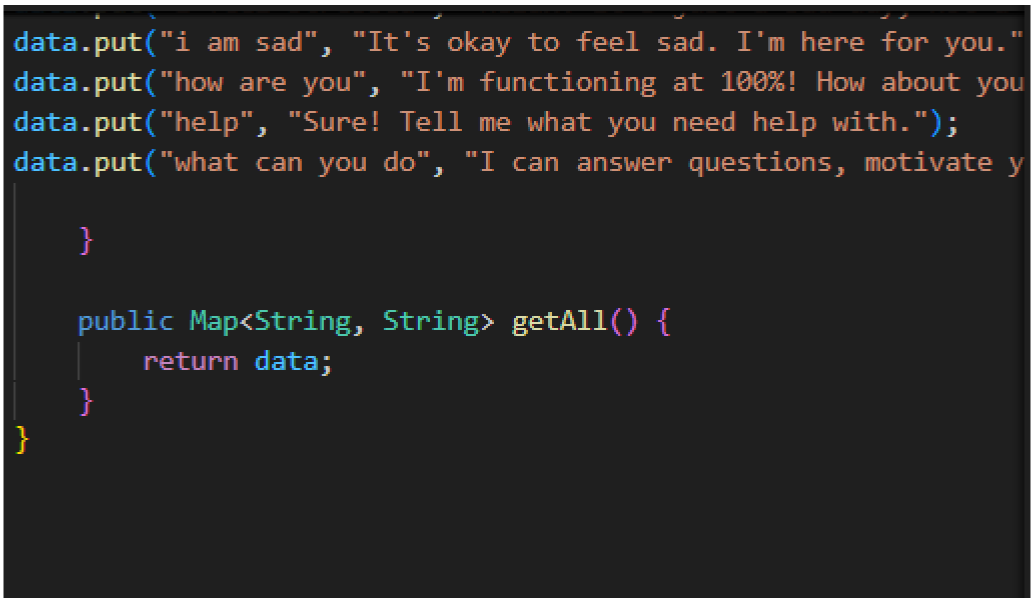
KnowledeBase



```
package infiniq.demo;

import java.util.HashMap;
import java.util.Map;

public class KnowledgeBase {

    private final Map<String, String> data = new HashMap<>();

    public void loadDefaults() {
        data.put("Who developed you?", "Lokesh ReddyRishik Rao"
data.put("Which animal is called the king of the jungle?", "Lio
data.put("How many days are there in a week?", "7 days");
data.put("Which is the fastest land animal?", "Cheetah");
data.put("What is the color of the sky?", "Blue");
data.put("How many letters are there in the English alphabet?",
data.put("Which shape has three sides?", "Triangle");
data.put("Which fruit is known as the king of fruits?", "Mango"
data.put("Which animal gives us milk?", "Cow");
data.put("Which bird cannot fly?", "Ostrich");
data.put("Which is the largest animal?", "Blue Whale");
data.put("Which insect has colorful wings?", "Butterfly");
```

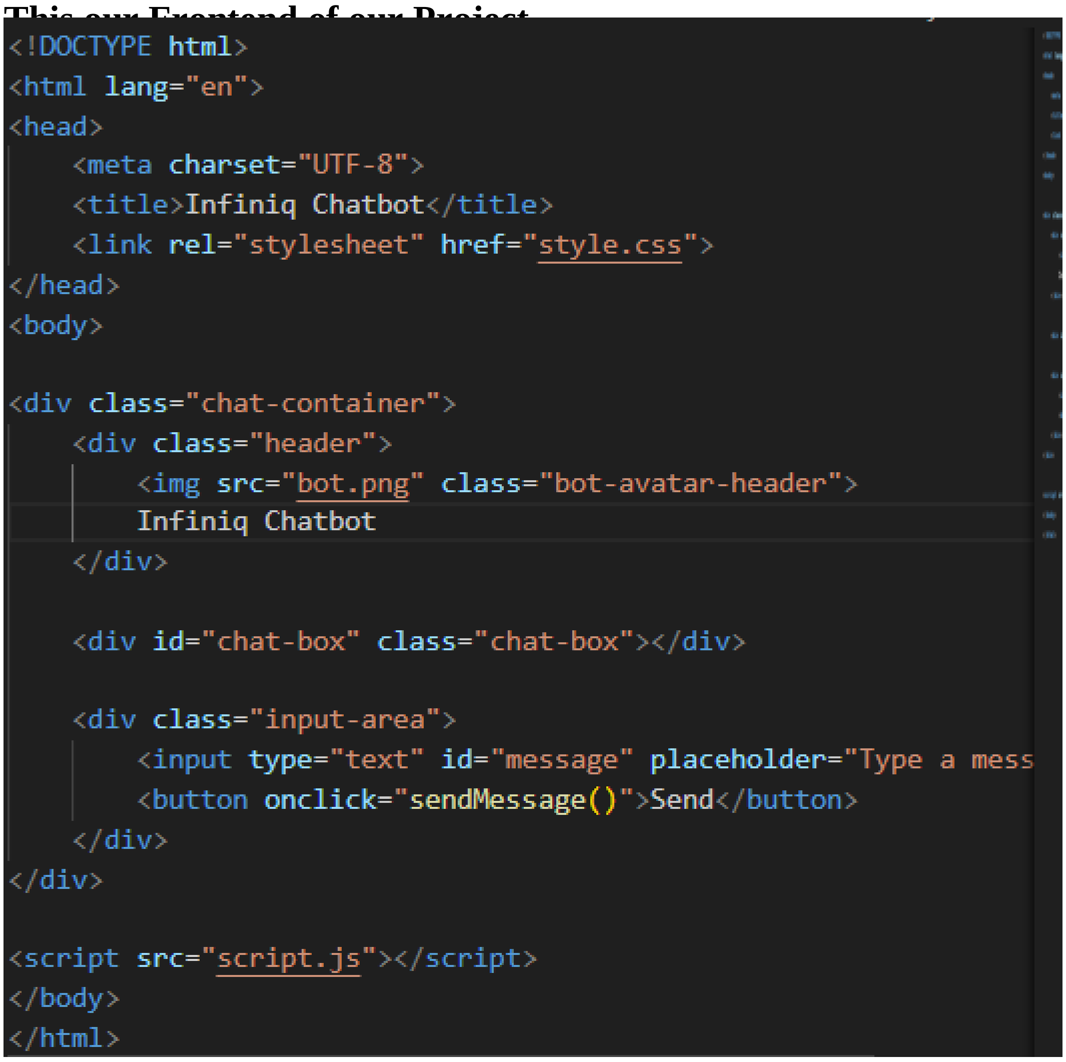


```java
data.put("What is the color of an apple?", "Red");
data.put("How many wheels does a bicycle have?", "Two");
data.put("How many wheels does a car have?", "Four");
data.put("What is the color of milk?", "White");
data.put("Which animal purrs?", "Cat");
data.put("Which animal has a shell?", "Tortoise");
data.put("Which bird hoots?", "Owl");
data.put("Which bird is green in color?", "Parrot");
data.put("Which fruit is green outside and red inside?", "Wa
data.put("Which fruit has many seeds inside?", "Papaya");
data.put("What do we drink to stay healthy?", "Water");
data.put("What do we breathe?", "Air");
data.put("Which star gives us light?", "Sun");
data.put("What do we wear on rainy days?", "Raincoat");
data.put("What do we use to cut paper?", "Scissors");
data.put("Which vegetable is orange in color?", "Carrot");
data.put("Which fruit is sour and green?", "Lemon");
data.put("Which animal lives in water?", "Fish");
data.put("Which bird swims?", "Duck");
data.put("Which vehicle flies in the sky?", "Aeroplane");
data.put("What do we use to see time?", "Clock");
data.put("Which sense organ helps us feel?", "Skin");
data.put("Which insect carries food?", "Ant");
data.put("Which animal says meow?", "Cat");
```



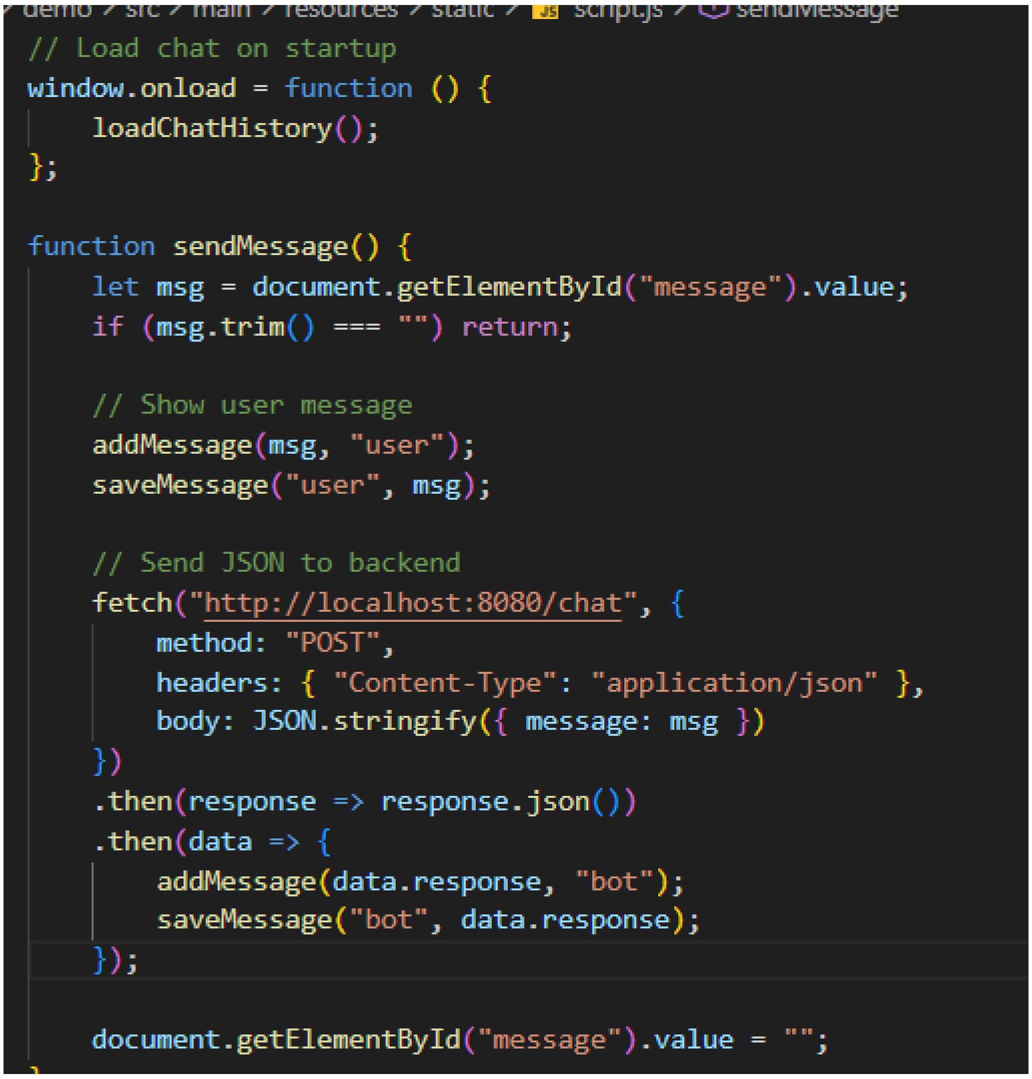
```java
data.put("i am sad", "It's okay to feel sad. I'm here for you."
data.put("how are you", "I'm functioning at 100%! How about you
data.put("help", "Sure! Tell me what you need help with.");
data.put("what can you do", "I can answer questions, motivate y

    }

    public Map<String, String> getAll() {
        return data;
    }
}
```

This our Frontend of our Project

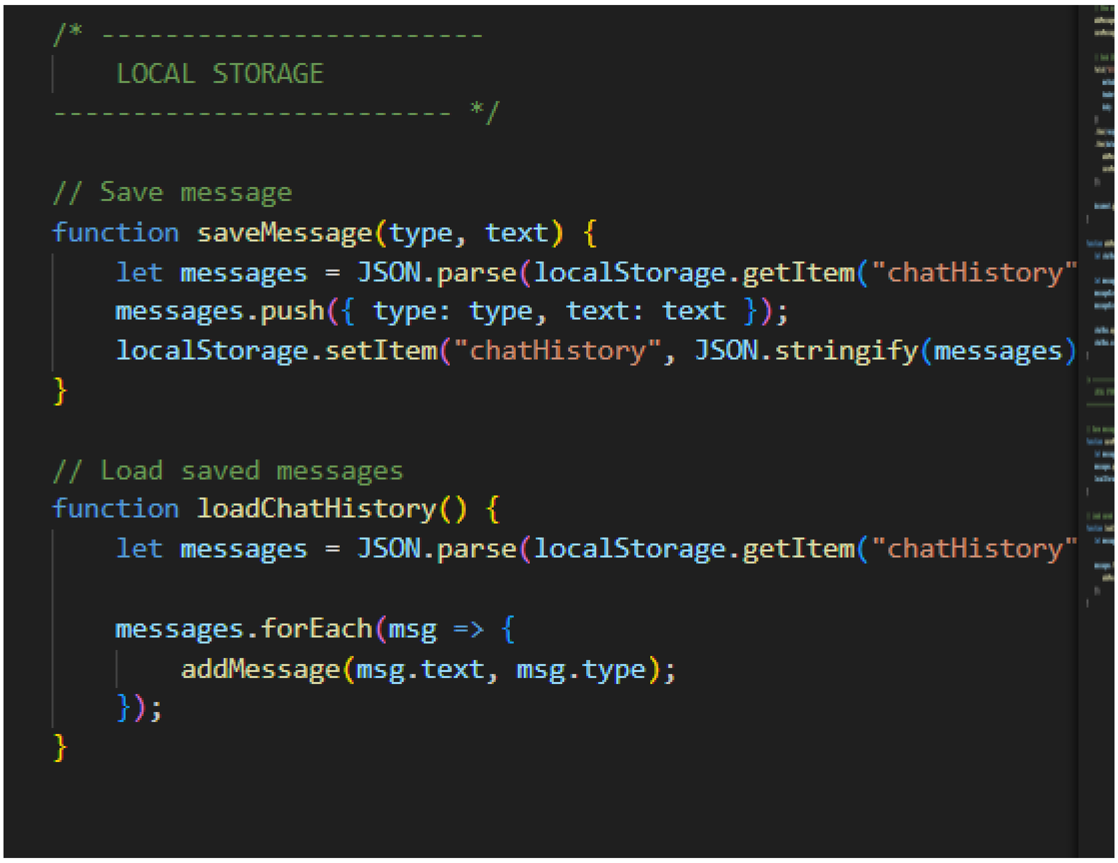


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Infiniq Chatbot</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

<div class="chat-container">
    <div class="header">
        <img src="bot.png" class="bot-avatar-header">
        Infiniq Chatbot
    </div>

    <div id="chat-box" class="chat-box"></div>

    <div class="input-area">
        <input type="text" id="message" placeholder="Type a mess
        <button onclick="sendMessage()">Send</button>
    </div>
</div>

<script src="script.js"></script>
</body>
</html>
```

Script.js

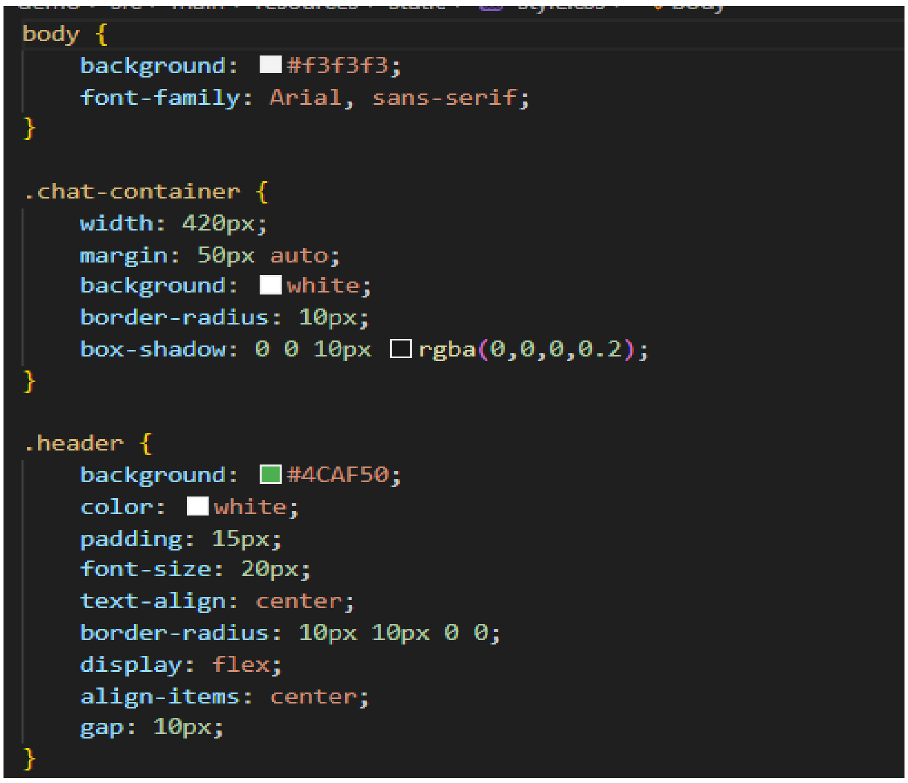


```
// Load chat on startup
window.onload = function () {
    loadChatHistory();
};

function sendMessage() {
    let msg = document.getElementById("message").value;
    if (msg.trim() === "") return;

    // Show user message
    addMessage(msg, "user");
    saveMessage("user", msg);

    // Send JSON to backend
    fetch("http://localhost:8080/chat", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ message: msg })
    })
    .then(response => response.json())
    .then(data => {
        addMessage(data.response, "bot");
        saveMessage("bot", data.response);
    });

    document.getElementById("message").value = "";
```

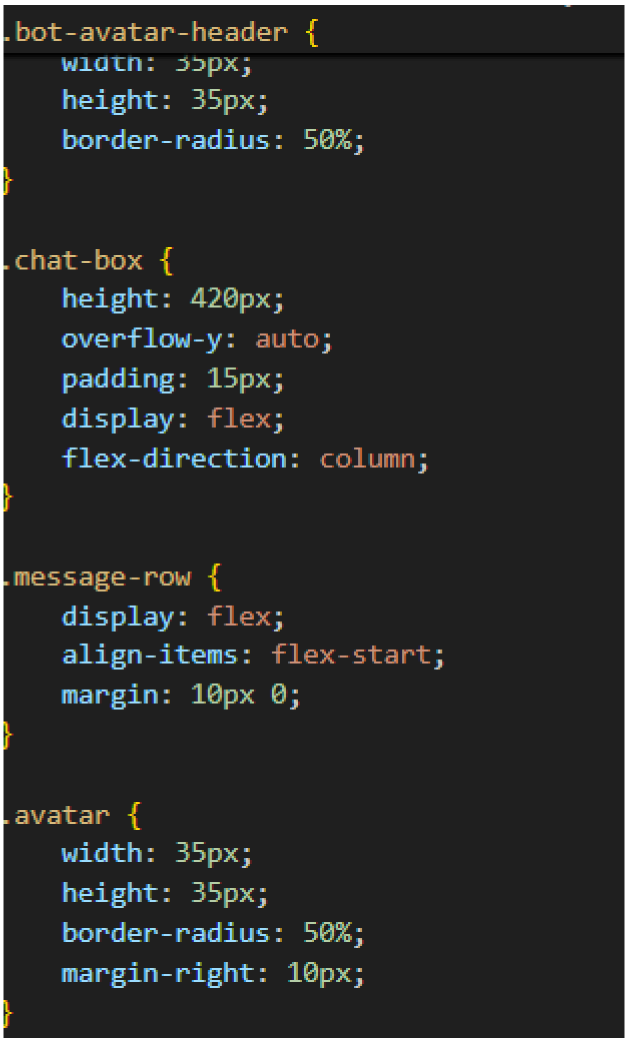


```
/* ------------------------
    LOCAL STORAGE
------------------------- */

// Save message
function saveMessage(type, text) {
    let messages = JSON.parse(localStorage.getItem("chatHistory"
    messages.push({ type: type, text: text });
    localStorage.setItem("chatHistory", JSON.stringify(messages)
}

// Load saved messages
function loadChatHistory() {
    let messages = JSON.parse(localStorage.getItem("chatHistory"

    messages.forEach(msg => {
        addMessage(msg.text, msg.type);
    });
}
```

Style CSS



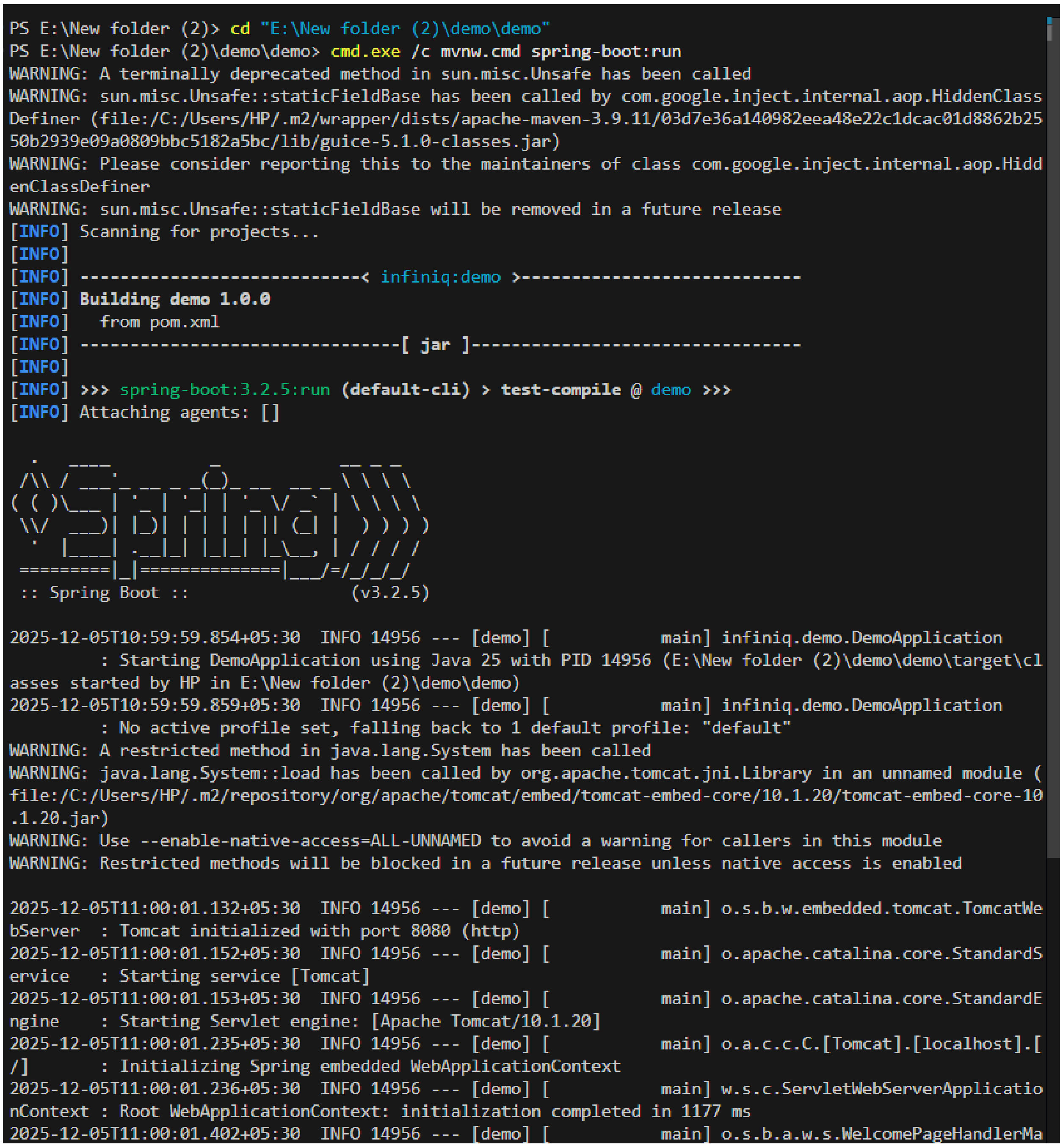
```
body {
    background: #f3f3f3;
    font-family: Arial, sans-serif;
}

.chat-container {
    width: 420px;
    margin: 50px auto;
    background: white;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0,0,0,0.2);
}

.header {
    background: #4CAF50;
    color: white;
    padding: 15px;
    font-size: 20px;
    text-align: center;
    border-radius: 10px 10px 0 0;
    display: flex;
    align-items: center;
    gap: 10px;
}
```



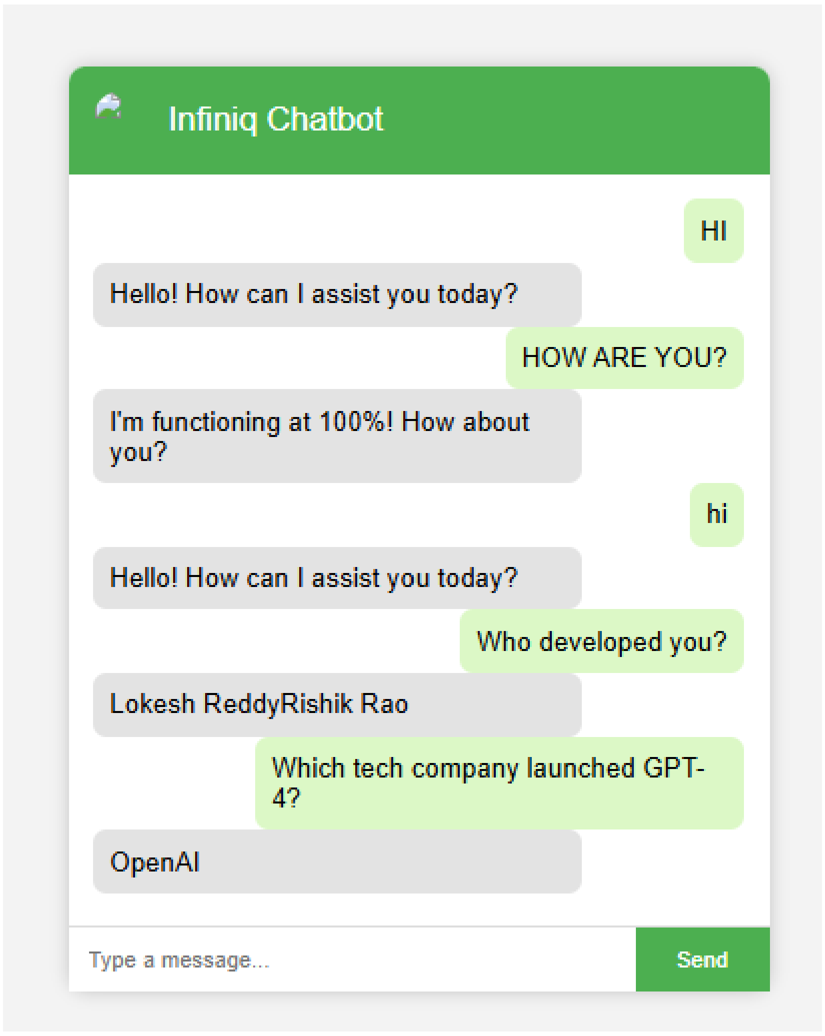
```
.bot-avatar-header {
    width: 35px;
    height: 35px;
    border-radius: 50%;
}

.chat-box {
    height: 420px;
    overflow-y: auto;
    padding: 15px;
    display: flex;
    flex-direction: column;
}

.message-row {
    display: flex;
    align-items: flex-start;
    margin: 10px 0;
}

.avatar {
    width: 35px;
    height: 35px;
    border-radius: 50%;
    margin-right: 10px;
}
```

CHAPTER -5 TESTING

5.1Test Cases and Results



```
PS E:\New folder (2)> cd "E:\New folder (2)\demo\demo"
PS E:\New folder (2)\demo\demo> cmd.exe /c mvnw.cmd spring-boot:run
WARNING: A terminally deprecated method in sun.misc.Unsafe has been called
WARNING: sun.misc.Unsafe::staticFieldBase has been called by com.google.inject.internal.aop.HiddenClass
Definer (file:/C:/Users/HP/.m2/wrapper/dists/apache-maven-3.9.11/03d7e36a140982eea48e22c1dcac01d8862b25
50b2939e09a0809bbc5182a5bc/lib/guice-5.1.0-classes.jar)
WARNING: Please consider reporting this to the maintainers of class com.google.inject.internal.aop.Hidd
enClassDefiner
WARNING: sun.misc.Unsafe::staticFieldBase will be removed in a future release
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------------< infiniq:demo >----------------------------
[INFO] Building demo 1.0.0
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spring-boot:3.2.5:run (default-cli) > test-compile @ demo >>>
[INFO] Attaching agents: []

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.5)

2025-12-05T10:59:59.854+05:30  INFO 14956 --- [demo] [           main] infiniq.demo.DemoApplication
         : Starting DemoApplication using Java 25 with PID 14956 (E:\New folder (2)\demo\demo\target\cl
asses started by HP in E:\New folder (2)\demo\demo)
2025-12-05T10:59:59.859+05:30  INFO 14956 --- [demo] [           main] infiniq.demo.DemoApplication
         : No active profile set, falling back to 1 default profile: "default"
WARNING: A restricted method in java.lang.System has been called
WARNING: java.lang.System::load has been called by org.apache.tomcat.jni.Library in an unnamed module (
file:/C:/Users/HP/.m2/repository/org/apache/tomcat/embed/tomcat-embed-core/10.1.20/tomcat-embed-core-10
.1.20.jar)
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for callers in this module
WARNING: Restricted methods will be blocked in a future release unless native access is enabled

2025-12-05T11:00:01.132+05:30  INFO 14956 --- [demo] [           main] o.s.b.w.embedded.tomcat.TomcatWe
bServer  : Tomcat initialized with port 8080 (http)
2025-12-05T11:00:01.152+05:30  INFO 14956 --- [demo] [           main] o.apache.catalina.core.StandardS
ervice   : Starting service [Tomcat]
2025-12-05T11:00:01.153+05:30  INFO 14956 --- [demo] [           main] o.apache.catalina.core.StandardE
ngine    : Starting Servlet engine: [Apache Tomcat/10.1.20]
2025-12-05T11:00:01.235+05:30  INFO 14956 --- [demo] [           main] o.a.c.c.C.[Tomcat].[localhost].[
/]       : Initializing Spring embedded WebApplicationContext
2025-12-05T11:00:01.236+05:30  INFO 14956 --- [demo] [           main] w.s.c.ServletWebServerApplicatio
nContext : Root WebApplicationContext: initialization completed in 1177 ms
2025-12-05T11:00:01.402+05:30  INFO 14956 --- [demo] [           main] o.s.b.a.w.s.WelcomePageHandlerMa
```



HI

Hello! How can I assist you today?

HOW ARE YOU?

I'm functioning at 100%! How about you?

hi

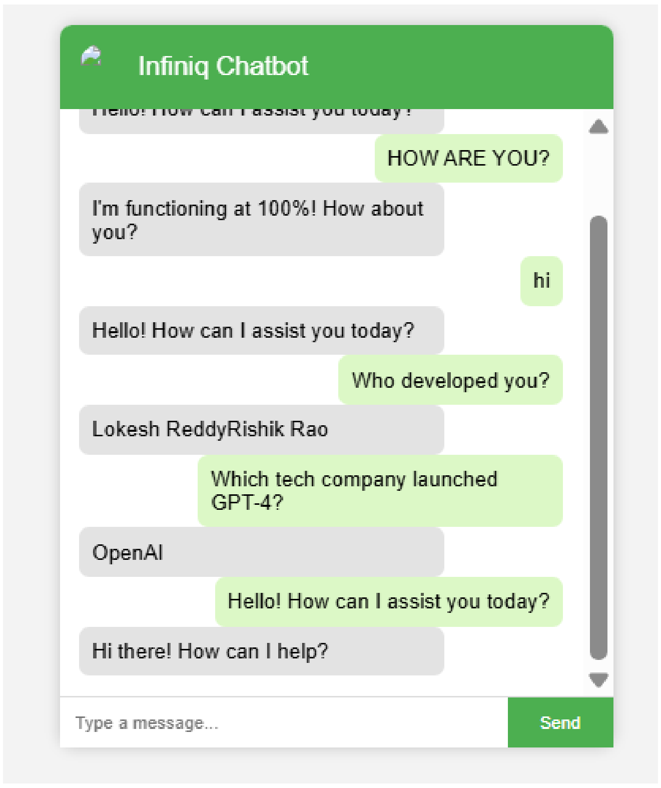
Hello! How can I assist you today?

Who developed you?

Lokesh ReddyRishik Rao

Which tech company launched GPT-4?

OpenAI



Infiniq Chatbot

Hello! How can I assist you today?

HOW ARE YOU?

I'm functioning at 100%! How about you?

hi

Hello! How can I assist you today?

Who developed you?

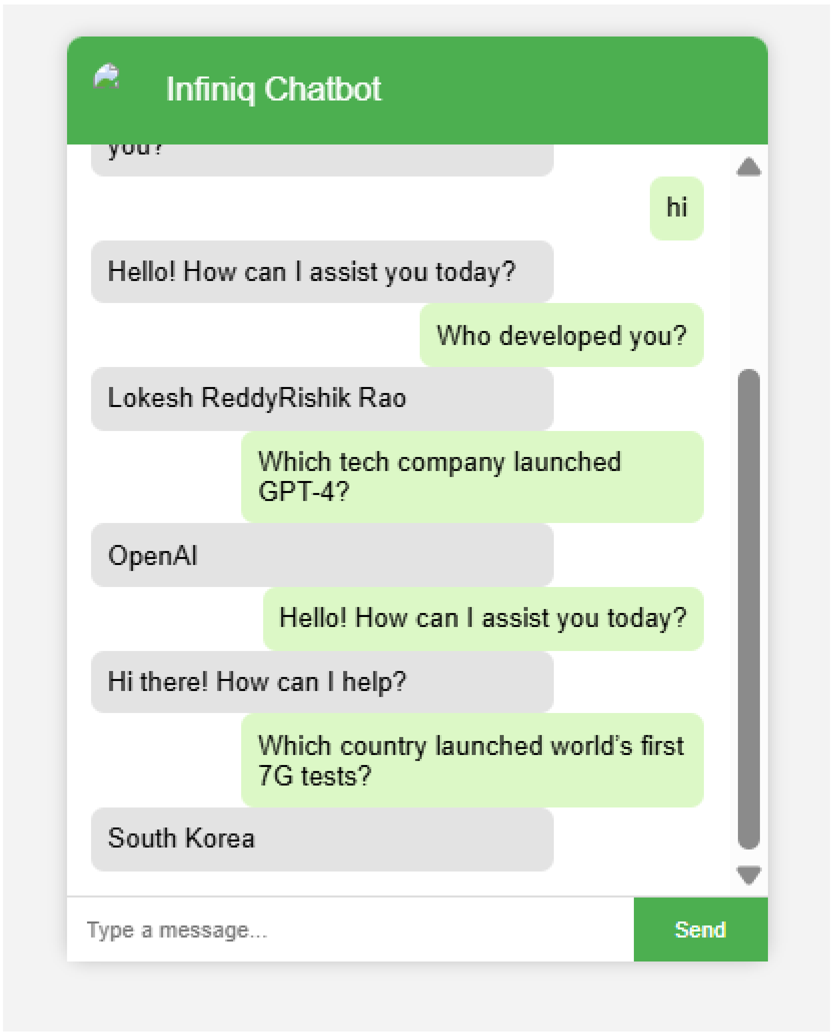
Lokesh ReddyRishik Rao

Which tech company launched GPT-4?

OpenAI

Hello! How can I assist you today?

Hi there! How can I help?



you?

hi

Hello! How can I assist you today?

Who developed you?

Lokesh ReddyRishik Rao

Which tech company launched GPT-4?

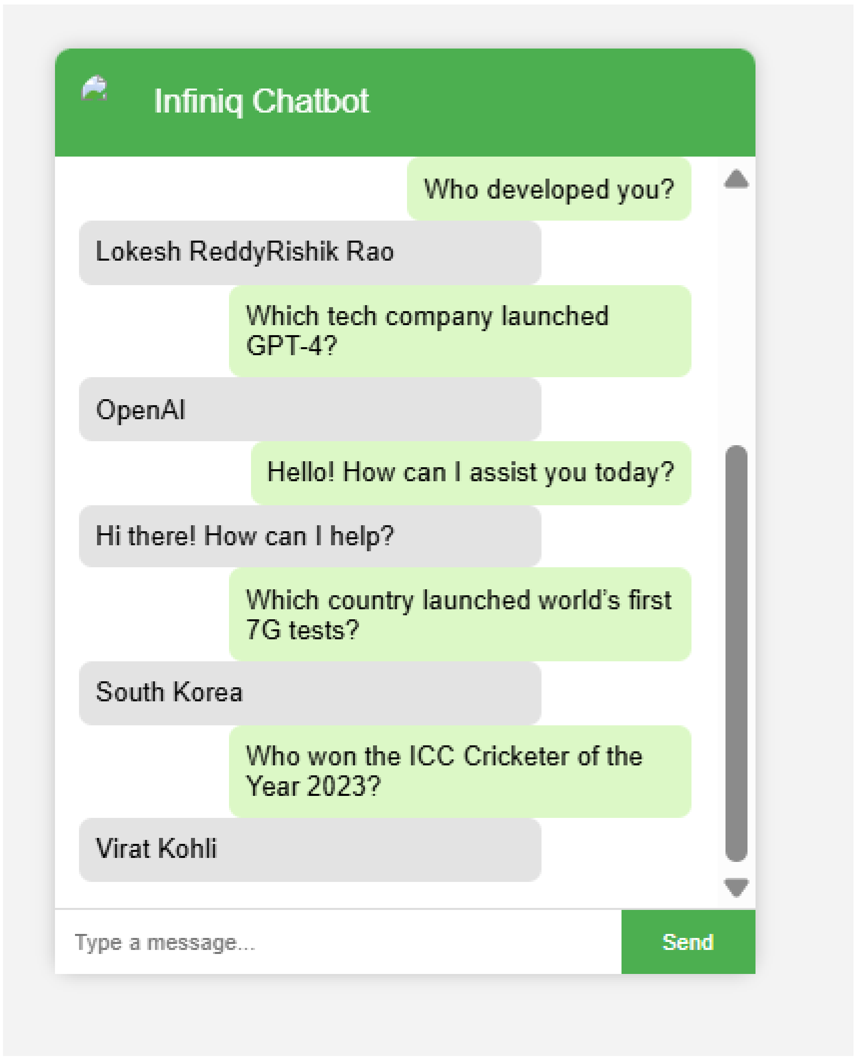
OpenAI

Hello! How can I assist you today?

Hi there! How can I help?

Which country launched world's first 7G tests?

South Korea



Who developed you?

Lokesh ReddyRishik Rao

Which tech company launched GPT-4?

OpenAI

Hello! How can I assist you today?

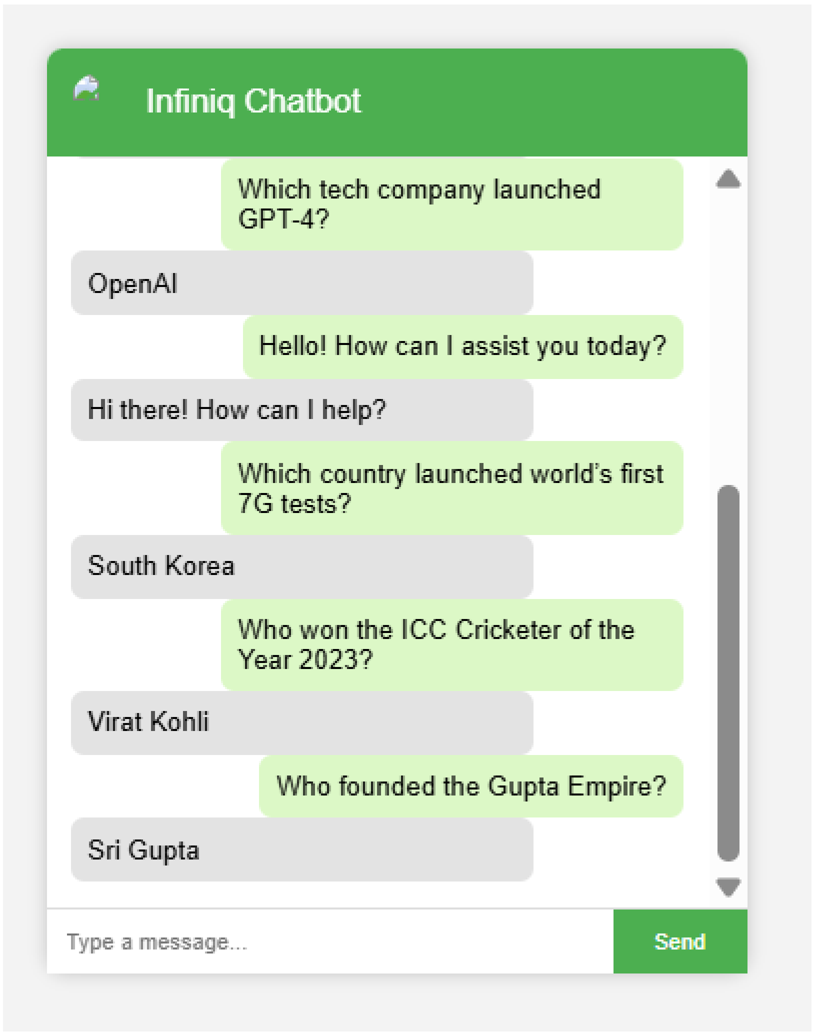
Hi there! How can I help?

Which country launched world's first 7G tests?

South Korea

Who won the ICC Cricketer of the Year 2023?

Virat Kohli



Infiniq Chatbot

Which tech company launched GPT-4?

OpenAI

Hello! How can I assist you today?

Hi there! How can I help?

Which country launched world's first 7G tests?

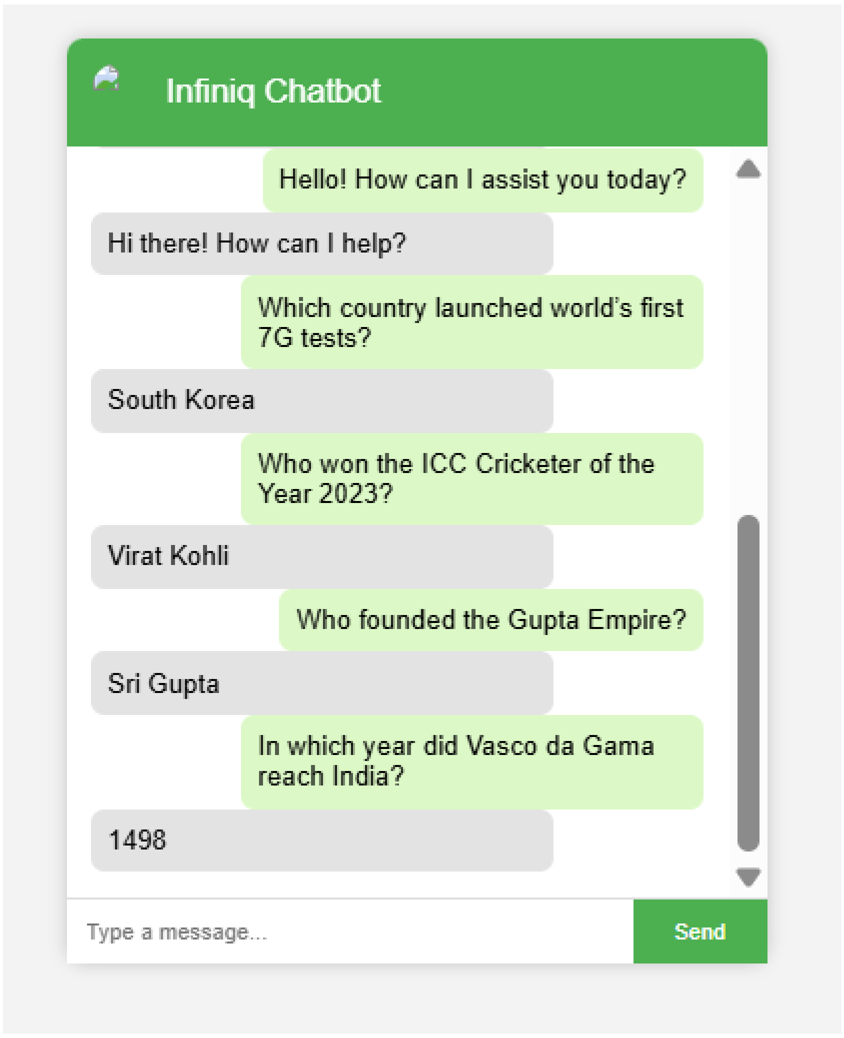
South Korea

Who won the ICC Cricketer of the Year 2023?

Virat Kohli

Who founded the Gupta Empire?

Sri Gupta


Infiniq Chatbot

Hello! How can I assist you today?

Hi there! How can I help?

Which country launched world's first 7G tests?

South Korea

Who won the ICC Cricketer of the Year 2023?

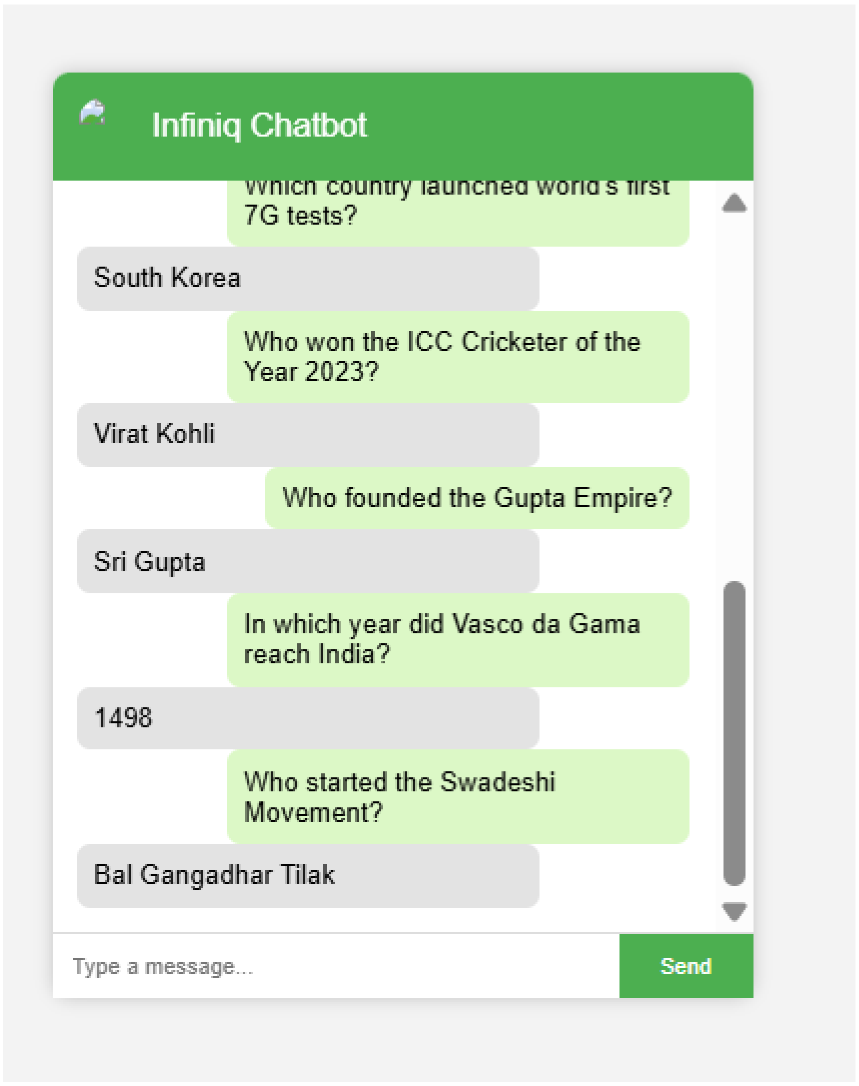
Virat Kohli

Who founded the Gupta Empire?

Sri Gupta

In which year did Vasco da Gama reach India?

1498


Infiniq Chatbot

Which country launched world's first 7G tests?

South Korea

Who won the ICC Cricketer of the Year 2023?

Virat Kohli

Who founded the Gupta Empire?

Sri Gupta

In which year did Vasco da Gama reach India?

1498

Who started the Swadeshi Movement?

Bal Gangadhar Tilak

CHAPTER -6 FUTURE ENHANCEMENTS

6.1Planned Features

Although the current version of the Infiniq Chatbot successfully performs rule-based conversational tasks using a structured knowledge base and custom search engine, several enhancements are planned to extend its intelligence, usability, and real-world applicability. These enhancements aim to make the system more flexible, more interactive, and capable of handling advanced information processing.

1. Machine Learning–Based Response Generation

Future versions will integrate NLP and machine learning models (such as BERT, GPT-like architectures, or sentence-embedding models) to enable:

- Context-aware responses
- Understanding of natural conversation flow
- Intelligent reasoning and inference

This will move the chatbot beyond fixed rule-based answers.

2. Dynamic Knowledge Base Expansion

Currently, the knowledge base is static and manually populated. Planned improvements include:

- Allowing the chatbot to learn new facts over time
- Admin panel for adding/editing Q&A pairs
- Automatic dataset generation from user queries
- Cloud-based storage for scalability

➔ This will significantly increase the size and accuracy of the chatbot’s knowledge.

3. Voice Input and Speech Output

To improve accessibility, the chatbot will support:

- Microphone-based voice input
- Text-to-speech output

- Hands-free conversational mode

This makes the system usable for younger students, visually impaired users, and mobile scenarios.

4.Multi-Language Support

Future versions will support multiple Indian and international languages such as:

- Hindi
- Telugu
- Kannada
- Tamil
- English

➔ This will allow the chatbot to reach a wider audience and support regional education needs.

5. Integration with IoT and Smart Devices

Since the chatbot architecture is modular, it can be extended to:

- Control IoT devices (smart classroom tools, sensors, alarms)
- Provide real-time updates from external systems
- Integrate with home assistants like Alexa and Google Home

This will transform the chatbot into a smart assistant rather than just a QA system.

6. Advanced Search Engine (Semantic + Keyword Hybrid)

A more intelligent search mechanism is planned:

- Semantic similarity matching
- Token ranking
- Weighted scoring
- Auto-correction for misspellings

This will improve response accuracy even when queries are vague or imprecise.

7. User Login, Personalization & Analytics

Planned features include:

- Login system for storing personal data
- Personalized learning suggestions
- Progress tracking and analytics dashboard
- Performance-based motivational tips

This targets long-term academic usage, especially for UPSC and competitive exams.

8.Mobile Application (Android/iOS)

A dedicated mobile app version is planned to allow:

- Offline operation with cached knowledge base
- Faster access compared to browser version
- Cleaner mobile-friendly UI

This will significantly improve reach and convenience.

9. Chat Memory With Long-Term Context

Enhancing the session manager to remember past conversations such as:

- User preferences
- Frequently asked questions
- Previously discussed topics

This will allow more natural, human-like conversation continuity.

10. Security and Data Protection Enhancements

Planned improvements include:

- Better API security
- Sanitization of user inputs
- Encryption for stored data
- Role-based access for admin panels

This is essential for deploying the system at scale.

Summary

The planned enhancements aim to transform the Infiniq Chatbot from a basic rule-based simulator into a smart, scalable, multilingual, and user-centric AI assistant. These features will expand its use in education, competitive exam preparation, customer support, and general conversational purposes.

CHAPTER -7 CONCLUSION

7.1Summary of the Project

The Infiniq Chatbot project was developed with the primary objective of creating an intelligent, interactive, and user-friendly conversational system capable of providing instant responses to a wide range of user queries. Throughout the development process, the project applied fundamental software engineering principles, object-oriented programming concepts, algorithmic design, and modern web technologies to build a functional and efficient Java-based chat simulator.

The system consists of a clean separation between the frontend and backend. The frontend, built using HTML, CSS, and JavaScript, provides a simple and intuitive chat interface that supports real-time messaging and chat history preservation. The backend, developed using Java and Spring Boot, handles input processing, message routing, and response generation. The heart of the system is the custom-built Recursive Search Engine, which performs query matching using normalization, string comparison, partial matching, and rule-based logic. This modular design ensures that each component performs its dedicated role, supporting a maintainable and scalable architecture.

A carefully structured Knowledge Base stores educational content, general knowledge, motivational information, and UPSC-related facts. When a user enters a query, the system processes it through the Infiniq core, normalizes the input, and identifies the most suitable response. If no direct match is found, the system provides a fallback message, ensuring consistent interaction. The inclusion of chat session management allows the system to track user inputs, supporting smoother communication and future expansion.

The project demonstrates the successful application of key computer science concepts such as algorithm optimization (CO2), modular and recursive logic implementation (CO3), and object-oriented design principles (CO4). It also showcases SDLC practices including requirement analysis, system design, implementation, and testing. Through these elements, the chatbot functions as both an academic tool and a practical demonstration of how structured programming and modern frameworks can be used to build AI-driven conversational systems.

Overall, the Infiniq Chatbot project provides a strong foundation for future improvements such as machine learning integration, voice input/output, multilingual capabilities, and cloud-based knowledge expansion. The system proves that even a rule-based architecture, when designed effectively, can deliver meaningful, scalable, and real-time digital assistance to users. This project successfully meets its objectives and highlights the significance of combining traditional software methodologies with emerging conversational AI technologies.

CHAPTER -8 REFERENCES

Books

Pressman, R. S., & Maxim, B. R. (2019). *Software Engineering: A Practitioner’s Approach* (8th ed.). McGraw- Hill Education.

Freeman, E., & Freeman, E. (2020). *Head First HTML and CSS* (2nd ed.).

O’Reilly Media. Duckett, J. (2014). *HTML & CSS: Design and Build Websites*.

Duckett, J. (2015). *JavaScript and JQuery: Interactive Front-End Web*

Development. Wiley. Sommerville, I. (2016). *Software Engineering* (10th ed.).

Tutorials and Learning Resources

W3Schools. (2025). *HTML, CSS, and JavaScript Tutorials.* Retrieved from https://www.w3schools.com

MDN Web Docs. (2025). *Web Technologies Documentation.* Retrieved from https://developer.mozilla.org

FreeCodeCamp. (2025). *Responsive Web Design Certification.* Retrieved from https://www.freecodecamp.org

GeeksforGeeks. (2025). *Front-end Development and Web Technologies.* Retrieved from https://www.geeksforgeeks.org

Tutorialspoint. (2025). *HTML, CSS, JavaScript, and Web Development Tutorials.* Retrieved from https://www.tutorialspoint.com

APIs and Tools Used

Google Maps Platform. (2025). *Maps JavaScript API Documentation.* Retrieved from https://developers.google.com/maps

Unsplash API. (2025). *Free High-Resolution Image Access*. Retrieved from https://unsplash.com/developers

OpenWeatherMap. (2025). *Weather API for Travel Assistance*. Retrieved from https://openweathermap.org/api

RapidAPI Hub. (2025). *Travel and Tourism APIs*. Retrieved from https://rapidapi.com/hub

Firebase by Google. (2025). *Authentication and Database Services*. Retrieved from https://firebase.google.com

Documentation and Research References

World Tourism Organization (UNWTO). (2024). *Tourism Data Dashboard*. Retrieved from https://www.unwto.org

Booking.com. (2025). *Accommodation and Destination Insights*. Retrieved from https://www.booking.com

TripAdvisor. (2025). *Tourism and Destination Review Data*. Retrieved from https://www.tripadvisor.com

Travel + Leisure. (2025). *Top Global Destinations and Travel Trends*. Retrieved from https://www.travelandleisure.com

Tourism Review. (2025). *Digital Transformation in the Tourism Industry*. Retrieved from https://www.tourism- review.com

CHAPTER -9 APPENDICES

The appendices section provides supplementary materials that support the understanding, validation, and replication of the Infiniq Chatbot project. These materials include code listings, screenshots, test results, system diagrams, hardware/software configuration details, and any additional documentation used during the development process. The appendices serve as a reference for evaluators, developers, and future contributors who may want to analyze or extend the system.

9.1 Source Code Listings

This appendix includes complete source code for all core modules of the chatbot:

- ChatController.java – REST API controller
- Infiniq.java – Main chatbot logic
- RecursiveSearchEngine.java – Query matching algorithm
- KnowledgeBase.java – Static question–answer storage
- ChatSession.java – Session management
- Frontend files:
    - index.html
    - style.css
    - script.js

9.2 System Architecture Diagrams

Includes graphical representations of:

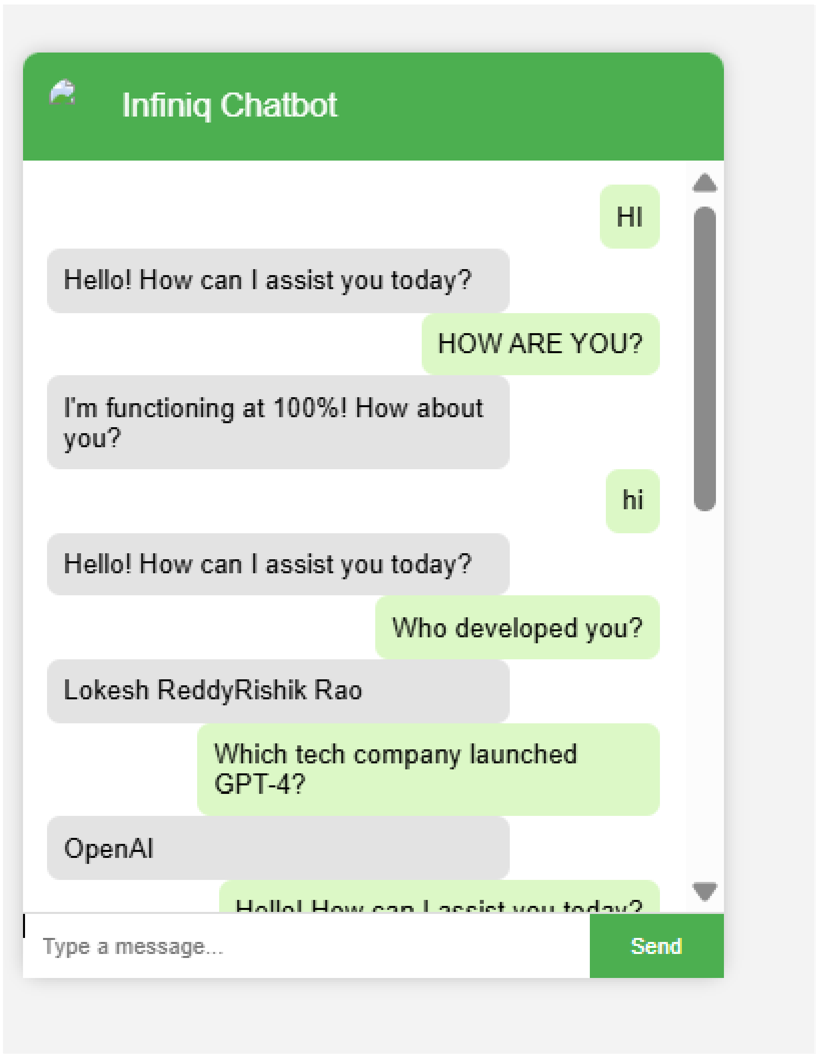
- High-level architecture
- Low-level module interaction
- Data flow diagram (DFD)
- Use case diagram
- Sequence diagram for message processing

These visuals help illustrate how components communicate and how data moves through the system.

9.3 Screenshots of the Working Application

This section contains screenshots taken during testing and execution, including:

- Chat interface (frontend UI)



HI

Hello! How can I assist you today?

HOW ARE YOU?

I'm functioning at 100%! How about you?

hi

Hello! How can I assist you today?

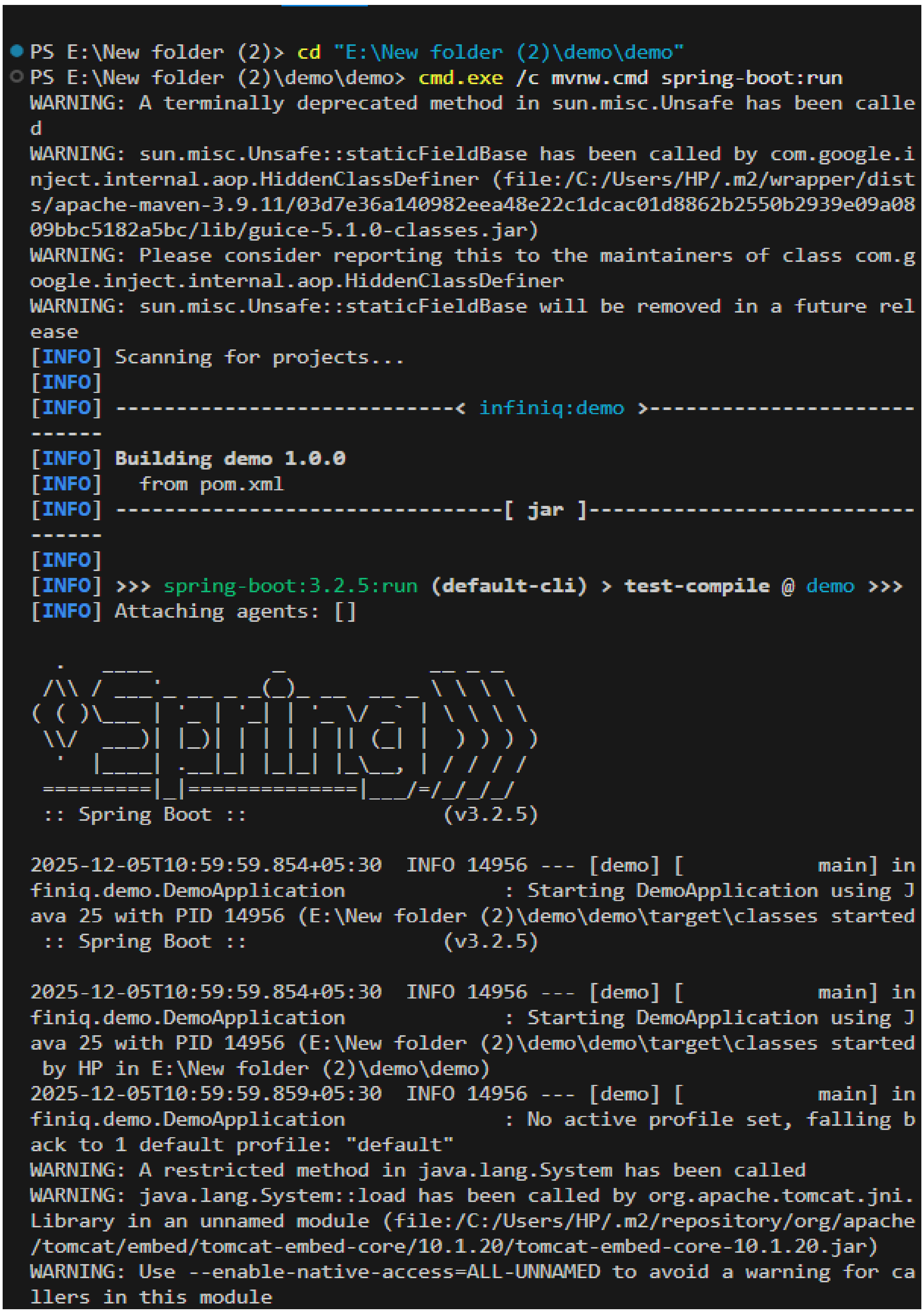
Who developed you?

Lokesh ReddyRishik Rao

Which tech company launched GPT-4?

OpenAI

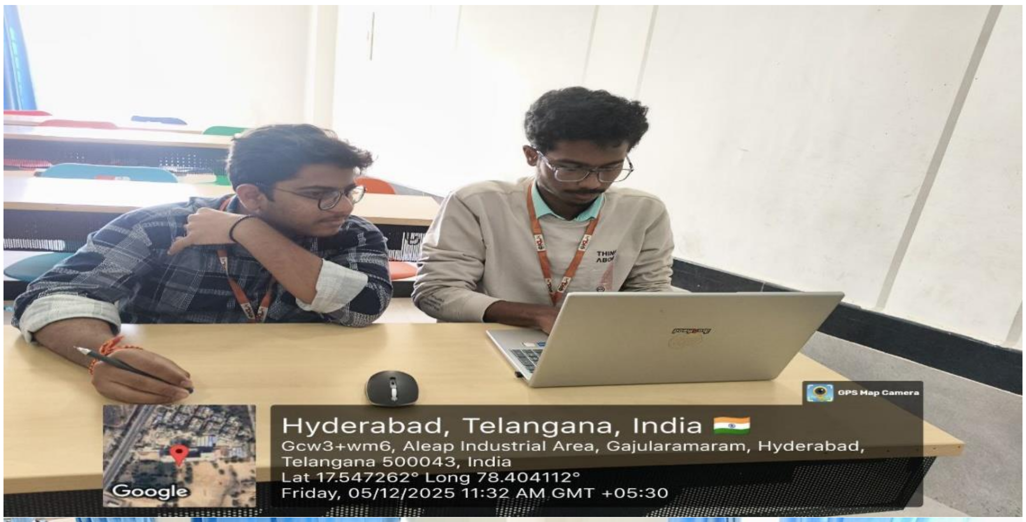
- Backend Spring Boot console logs



```
● PS E:\New folder (2)> cd "E:\New folder (2)\demo\demo"
○ PS E:\New folder (2)\demo\demo> cmd.exe /c mvnw.cmd spring-boot:run
  WARNING: A terminally deprecated method in sun.misc.Unsafe has been calle
  d
  WARNING: sun.misc.Unsafe::staticFieldBase has been called by com.google.i
  nject.internal.aop.HiddenClassDefiner (file:/C:/Users/HP/.m2/wrapper/dist
  s/apache-maven-3.9.11/03d7e36a140982eea48e22c1dcac01d8862b2550b2939e09a08
  09bbc5182a5bc/lib/guice-5.1.0-classes.jar)
  WARNING: Please consider reporting this to the maintainers of class com.g
  oogle.inject.internal.aop.HiddenClassDefiner
  WARNING: sun.misc.Unsafe::staticFieldBase will be removed in a future rel
  ease
  [INFO] Scanning for projects...
  [INFO]
  [INFO] ----------------------------< infiniq:demo >----------------------
  ------
  [INFO] Building demo 1.0.0
  [INFO]   from pom.xml
  [INFO] --------------------------------[ jar ]--------------------------
  ------
  [INFO]
  [INFO] >>> spring-boot:3.2.5:run (default-cli) > test-compile @ demo >>>
  [INFO] Attaching agents: []

    .   ____          _            __ _ _
   /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
  ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
   \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
    '  |____| .__|_| |_|_| |_\__, | / / / /
   =========|_|==============|___/=/_/_/_/
   :: Spring Boot ::                (v3.2.5)

  2025-12-05T10:59:59.854+05:30  INFO 14956 --- [demo] [           main] in
  finiq.demo.DemoApplication             : Starting DemoApplication using J
  ava 25 with PID 14956 (E:\New folder (2)\demo\demo\target\classes started
   :: Spring Boot ::                (v3.2.5)

  2025-12-05T10:59:59.854+05:30  INFO 14956 --- [demo] [           main] in
  finiq.demo.DemoApplication             : Starting DemoApplication using J
  ava 25 with PID 14956 (E:\New folder (2)\demo\demo\target\classes started
   by HP in E:\New folder (2)\demo\demo)
  2025-12-05T10:59:59.859+05:30  INFO 14956 --- [demo] [           main] in
  finiq.demo.DemoApplication             : No active profile set, falling b
  ack to 1 default profile: "default"
  WARNING: A restricted method in java.lang.System has been called
  WARNING: java.lang.System::load has been called by org.apache.tomcat.jni.
  Library in an unnamed module (file:/C:/Users/HP/.m2/repository/org/apache
  /tomcat/embed/tomcat-embed-core/10.1.20/tomcat-embed-core-10.1.20.jar)
  WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for ca
  llers in this module
```

- API responses displayed in browser or Postman
- Successful database/knowledge loading
- Error handling or fallback responses

Geo Tag photos with guide


GPS Map Camera
Hyderabad, Telangana, India
Gcw3+wm6, Aleap Industrial Area, Gajularamaram, Hyderabad, Telangana 500043, India
Lat 17.547262° Long 78.404112°
Friday, 05/12/2025 11:32 AM GMT +05:30
Google

GPS Map Camera
Hyderabad, Telangana, India
Gcw3+wm6, Aleap Industrial Area, Gajularamaram, Hyderabad, Telangana 500043, India
Lat 17.547271° Long 78.40409°
Friday, 05/12/2025 11:32 AM GMT +05:30
Google

GPS Map Camera
Hyderabad, Telangana, India
Gcw3+wm6, Aleap Industrial Area, Gajularamaram, Hyderabad, Telangana 500043, India
Lat 17.547287° Long 78.404552°
Tuesday, 02/12/2025 02:29 PM GMT +05:30
Google

GPS Map Camera
Hyderabad, Telangana, India
Gcw3+wm6, Aleap Industrial Area, Gajularamaram, Hyderabad, Telangana 500043, India
Lat 17.54728° Long 78.404548°
Tuesday, 02/12/2025 02:29 PM GMT +05:30
Google

GPS Map Camera
Hyderabad, Telangana, India
Gcw3+wm6, Aleap Industrial Area, Gajularamaram, Hyderabad, Telangana 500043, India
Lat 17.54735° Long 78.404571°
Tuesday, 02/12/2025 02:32 PM GMT +05:30
Google

GPS Map Camera
Hyderabad, Telangana, India
Gcw3+wm6, Aleap Industrial Area, Gajularamaram, Hyderabad, Telangana 500043, India
Lat 17.547358° Long 78.404582°
Tuesday, 02/12/2025 02:30 PM GMT +05:30
Google