NAYANA the CHATBOT

Virtual Assistance for Computer-X Computer Hardware Shop

Prototype AI Project on NLP with GUI and EI

By

Ravindu Gayal (Student ID : KAN00257474)

Contents

1. Introduction

Artificial intelligence is the science of making machines and computers do tasks that normally require human intelligence. Natural Language Processing (NLP) is a major part of AI. It is one of the most challenging areas of AI because it involves human-machine communication in natural language. It is considered strong AI and is a key part of the original Turing test.

Chatbots are a sub-domain of NLP. A chatbot is a software program that communicates with a user in natural language, either through text or voice.

Today, chatbots are very common in online shops, where they act as virtual assistants to customers. According to Mero (2018), the real-time help from chat services in online shops has changed customer service into a two-way conversation, which has a big effect on customer trust and happiness.

This report describes a prototype chatbot named Nayana, created as a virtual assistant for an online store called Computer X. It answers user questions about different hardware products through an interactive graphical user interface (GUI).

The project is developed using the Java 21 programming language and the Spring Boot framework. The framework helps manage the application's structure and different parts, making the code clean and organized.

2. Problem Definition

This project is about creating an AI-based chatbot for the Computer X online hardware store. The chatbot, Nayana, will act as a virtual assistant, giving quick answers to customer questions about the store's products.

The chatbot is meant to be a key part of the customer service experience. It will help human support staff by answering common and repetitive questions. Customers can ask questions about product categories (like GPUs, motherboards), specific items, prices, and if items are in stock. The chatbot will give both short answers and, when needed, longer explanations.

The dynamic content, like product details and prices, is taken from a text-file database (`products.csv`). This makes sure the information can be easily updated. The chatbot will be available to users as a standalone desktop application, offering help 24/7.

Two important features of this chatbot are its ability to learn from user interaction (a form of machine learning) and its use of basic Emotional Intelligence (EI). The learning feature lets users teach the bot new information, which is saved for future conversations. The EI feature allows the bot to react to repeated questions, which makes the conversation feel more human.

3. Research

3.1 Introduction to Chatbots

Chatbots are almost as old as AI itself. One of the first successful chatbots was ELIZA, created between 1964 and 1966 by Joseph Weizenbaum. It used simple pattern matching to act like it was having a conversation and was an early test for the Turing Test. Modern assistants like Amazon Alexa and Google Assistant are the evolution of these ideas, using advanced technology to understand voice and text.

3.2 Linguistics in NLP

For a computer to process natural language, it needs both AI and linguistics (the study of language). The key parts relevant to our text-based chatbot are:

- **Morphology:** Understanding that different words can have the same root meaning (e.g., "laptop" and "laptops" refer to the same type of product). Our bot handles this with simple text matching.
- **Syntax:** Analyzing the words in a sentence to understand its grammar. Our bot uses a simple keyword-based method to find the main topic of a question.
- **Semantics:** Figuring out the meaning of a sentence. Our bot understands the user's goal by looking for keywords like "price," "stock," or "show me."
- **Pragmatics:** Understanding the hidden meaning in a conversation based on context. Our bot's Emotional Intelligence is a simple form of this, where the context (a repeated question) changes the bot's response.

3.3 Entities and Intent

A common way to understand a sentence is to find its intent and entities.

- **Intent:** The user's goal. For "show me all MSI gpus," the intent is to *find products*.
- **Entities:** The specific details that describe the intent. In the same example, "MSI" and "gpus" are entities. Our bot uses a keyword-matching system to find these entities and decide what action to take.

3.4 Randomness

When people talk, they don't always give the same answer to the same question. For example, "How are you?" can have many different replies. A good chatbot should act similarly by giving random but appropriate answers. Our bot does this for common phrases like "hello" and "thank you" to sound less robotic.

3.5 Dynamic Answers

Some questions need answers that can change over time. For example, the answer to "what laptops are in stock?" changes as products are sold. So, answers to these questions must come

from a database. Our bot uses a text file (`products.csv`) as a simple database, which allows product information to be updated without changing the program's code.

3.6 Emotional Intelligence (EI)

A key goal in human-computer interaction is to make computer agents more believable. EI in chatbots means recognizing the user's feelings and showing a personality. Our chatbot does this in a simple way by tracking the conversation. If a user asks the exact same question many times in a row, the bot's internal state changes from `NORMAL` to `ANNOYED`, which changes its answer and its picture (avatar).

3.7 User Profiling (Session Memory)

An advanced chatbot can remember users and their preferences. Our bot has a simple form of this. It can ask for the user's name and remember it for the rest of the conversation. It then sometimes uses the name in its replies to make the chat feel more personal (e.g., "You're welcome, Sarah."). This information is forgotten when the application closes.

3.8 Training the Chatbot (Machine Learning)

In real life, people learn from talking with others. It is important for a chatbot to learn from its users. When our bot doesn't know the answer to a question, it asks the user to provide the correct answer. This is a simple form of machine learning. The bot then saves this new information in an external text file (`data/learned.txt`) so it can remember it for future conversations.

3.9 Challenges

One of the hardest parts of NLP is dealing with ambiguity, where a word or sentence can have multiple meanings. For example, if a user asks, "Do you have a Dell?", they could mean a Dell laptop or a Dell monitor. Our simple keyword-based bot would likely show both, as it can't understand the user's specific context.

Another challenge is coreference resolution, which is understanding what pronouns like "it" or "they" refer to. For example, in the sentences "What is the price of the RTX 4090? Is it in stock?", our bot would not know that "it" refers to the RTX 4090. It treats every question as a separate, new query.

4. Design

4.1 P.E.A.S. Analysis

Intelligent agents can be described by their Performance, Environment, Actuators, and Sensors.

Performance	Give correct and fast answers to product questions. Have friendly conversations. Show believable emotional changes. Learn new information from the user and remember it correctly. Remember the user's name during a conversation.
Environment	A user's desktop computer. The bot runs inside a Java Swing GUI window. It can read and write files on the local computer to access its knowledge base.
Actuators	The bot acts on its environment by: • Displaying text in the `JTextArea`. • Changing the image in the `JLabel` (the avatar). • Showing pop-up dialogs (`JOptionPane`) to learn new answers.
Sensors	The bot senses its environment through: • Text input from the `JTextField`. • Clicks on the "Submit" button. • Input from the `JOptionPane` learning dialogs.

4.2 Architecture

The system uses a three-part architecture inside the Spring Boot framework.

1. **The Interface (GUI):** The `ChatFrame` class, built with Java Swing. This is the visual part where the user interacts with the bot.
2. **The Inference Engine (The "Brain"):** The `ChatService` class. This part processes the user's input, manages the bot's emotional state, gets information from the knowledge base, and creates a response.
3. **The Knowledge Base (Data Layer):** The `KnowledgeBaseService` class. This part is responsible for loading all data from the text files (`products.csv`, `smalltalk.properties`, `learned.txt`) and making it available to the Inference Engine.

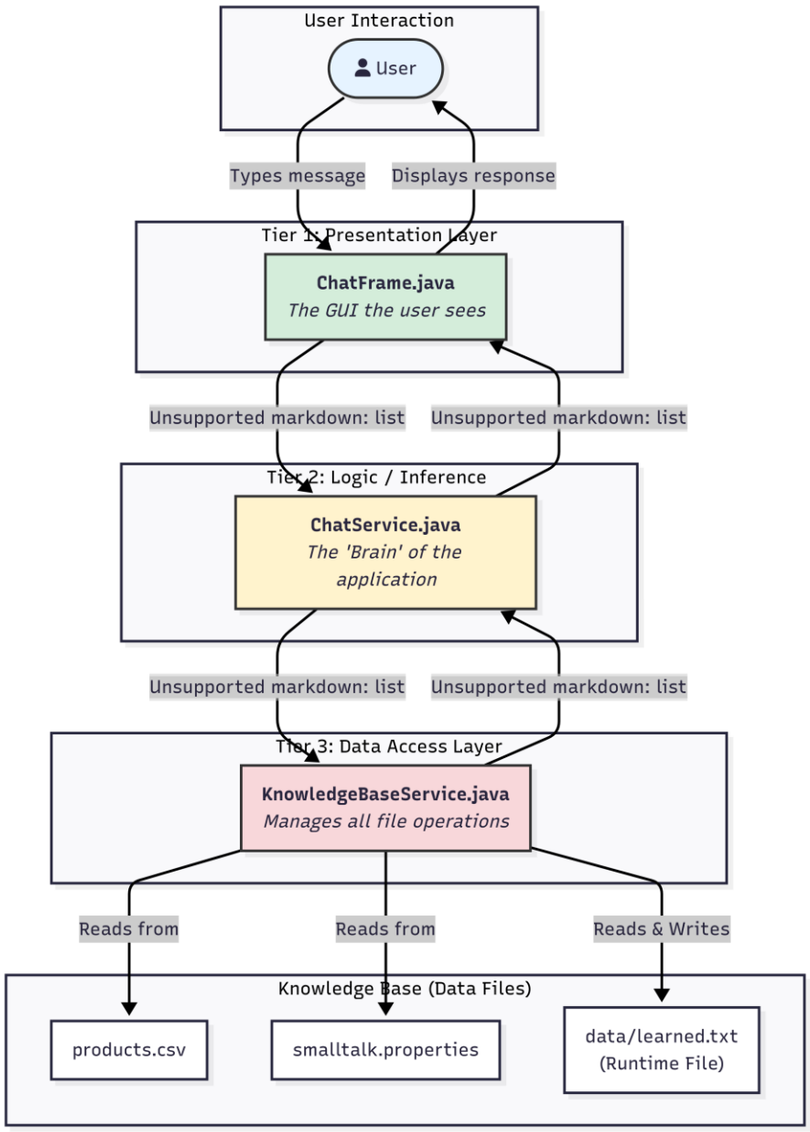


Figure 4.1 - Main components of the Nayana chatbot

4.3 GUI and Event Handling

The user interface is a single window with standard GUI parts. When the user types a message and presses Enter or clicks "Submit", an event is triggered. This event calls a method that sends the user's text to the `ChatService` for processing.

`// In ChatFrame.java`

```java
private void processUserInput() {  1 usage
    String userInput = inputField.getText().trim();
    if (userInput.isEmpty()) {
        return;
    }

    appendToChat( user: "You", userInput);
    inputField.setText("");

    ChatResponse response = chatService.getResponse(userInput);

    if (response.botState() == BotState.LEARNING) {
        String teachingAnswer = JOptionPane.showInputDialog(
                parentComponent: this,
                response.message(),
                title: "Teach Nayana",
                JOptionPane.QUESTION_MESSAGE
        );
        if (teachingAnswer != null && !teachingAnswer.isBlank()) {
            chatService.learn(userInput, teachingAnswer);
            appendToChat( user: "Nayana",  message: "Thank you! I've learned that.");
        } else {
            appendToChat( user: "Nayana",  message: "Okay, I won't learn that for now.");
        }
    } else {
        appendToChat( user: "Nayana", response.message());
    }

    setAvatar(response.botState());
```

Figure 4.2 - GUI event handling logic

4.4 Extracting Entities and Intent

The bot understands the user's intent by searching for simple keywords in the input. The `ChatService` checks the user's message for words like "price," "stock," or "show."

`// In ChatService.java`

```java
if (normalizedInput.contains("show") || normalizedInput.contains("list") || normalizedInput.contains("see")) {
    String keyword = extractKeyword(normalizedInput,  ...prefixes: "show me", "list all", "see all", "show", "list", "see");
    return findAndListProducts(keyword);
}
if (normalizedInput.startsWith("price of") || normalizedInput.startsWith("what is the price of")) {
    String keyword = extractKeyword(normalizedInput,  ...prefixes: "price of", "what is the price of");
    return findProductPrice(keyword);
```

Figure 4.3 - Keyword-based intent recognition

4.5 Smalltalk, Randomness, and Time-Awareness

Simple conversations are handled in two ways. Very simple, fixed answers are stored in a `smalltalk.properties` file. For more natural conversations, the bot gives random answers to common questions. This is handled directly in the `ChatService` code.

The bot is also aware of the current time of day for greetings.

`// In ChatService.java`

```java
private ChatResponse handleTimeBasedGreeting(String normalizedInput) {  1 usage
    int currentHour = LocalTime.now().getHour();
    String actualTimeOfDay;
    if (currentHour >= 5 && currentHour < 12) actualTimeOfDay = "morning";
    else if (currentHour >= 12 && currentHour < 17) actualTimeOfDay = "afternoon";
    else if (currentHour >= 17 && currentHour < 22) actualTimeOfDay = "evening";
    else actualTimeOfDay = "night";

    String userGreeting = null;
    if (normalizedInput.contains("morning")) userGreeting = "morning";
    else if (normalizedInput.contains("afternoon")) userGreeting = "afternoon";
    else if (normalizedInput.contains("evening")) userGreeting = "evening";
    else if (normalizedInput.contains("night")) userGreeting = "night";

    if (userGreeting != null) {
        if (userGreeting.equals(actualTimeOfDay)) {
            return new ChatResponse( message: "Good " + actualTimeOfDay + " to you too!",  endConversation: false, BotState.NORMAL,  isPersonaliz
        } else {
            return new ChatResponse( message: "Actually, it's " + actualTimeOfDay + " here, but good " + userGreeting + " to you anyway!"
        }
    }
    return null;
}
```

Figure 4.4 - Time-aware greeting logic

4.6 Emotional Intelligence

To feel more human, the bot tracks if the user is repeating the same question. If a question is asked more than twice, it adds a warning. If it's asked more than three times, it gets annoyed and changes its avatar.

`// In ChatService.java`

```java
if (repetitionCount > 3 && !baseResponse.endConversation()) {
    finalMessage = getRandomResponse( ...responses: "I've already answered that. Please ask a different question.
    finalState = BotState.ANNOYED;
} else if (repetitionCount == 3 && !baseResponse.endConversation()) {
    finalMessage += getRandomResponse( ...responses: "\n\n(By the way, you've asked me that a few times now.)", "
}

return new ChatResponse(finalMessage, baseResponse.endConversation(), finalState,  isPersonalizable: false);
```

Figure 4.5 - Emotional Intelligence logic

4.7 Dynamic Answers from Knowledge Base

Product information is loaded from `products.csv` when the application starts. The `ChatService` then searches this list of products to answer user questions.

```java
public List<Product> findProductsByKeyword(String keyword) {  3 usages
    String lowerKeyword = keyword.toLowerCase();
    return products.stream()
            .filter( Product p -> {
                String category = p.category().toLowerCase();
                boolean categoryMatch = category.contains(lowerKeyword) || lowerKeyword.contains(category);
                return p.name().toLowerCase().contains(lowerKeyword) ||
                        categoryMatch ||
                        p.brand().toLowerCase().contains(lowerKeyword) ||
                        p.description().toLowerCase().contains(lowerKeyword) ||
                        p.attributes().values().stream().anyMatch( String v -> v.toLowerCase().contains(lowerKeyword));
            })
            .collect(Collectors.toList());
}
```

Figure 4.6 - Searching the in-memory product list

4.8 Bot Training (Self-Learning)

If the `ChatService` cannot find an answer, it enters a `LEARNING` state. The `ChatFrame` sees this state and shows a pop-up box asking the user for the correct answer. This new information is then saved to an external `data/learned.txt` file.

```
public void saveLearnedResponse(String question, String answer) {  1 usage
    String cleanQuestion = question.toLowerCase().trim();
    learnedResponses.put(cleanQuestion, answer);
    Path path = Paths.get(LEARNED_FILE);
    try {
        Path parentDir = path.getParent();
        if (parentDir != null && !Files.exists(parentDir)) {
            Files.createDirectories(parentDir);
            log.info("Created directory for learned data at: {}", parentDir.toAbsolutePath());
        }

        String lineToAppend = cleanQuestion + LEARNED_DELIMITER + answer + System.lineSeparator();
        Files.write(path, lineToAppend.getBytes(), StandardOpenOption.APPEND, StandardOpenOption.CREATE);
        log.info("Saved new learned response to external file: {}", path.toAbsolutePath());
    } catch (IOException e) {
        log.error("Failed to save learned response to file. Check write permissions for the application directory.", e);
    }
}
```

Figure 4.7 - Saving learned knowledge to an external file

4.9 High-Level Software Design (Pseudo Code)

```
START Application
    Initialize Services (KnowledgeBaseService, ChatService)
    Create and show the GUI (ChatFrame)
    Display a welcome message

    WAIT for a user action (like a button click or pressing Enter)

    ON user action:
        Get the text from the input field
        IF the input text is not empty:
            Show the user's text in the main chat area
            Clear the input field
            CALL the ChatService to get a response for the user's text

        IF the response is a LEARNING response:
            SHOW a pop-up box to ask the user for the correct answer
            IF the user gives an answer:
                CALL the ChatService to learn the new question-answer pair
                Display a "Thank you" message
        ELSE:
            Display the bot's response message in the main chat area

        UPDATE the bot's picture based on the response's emotional state

        IF the response signals to end the conversation:
            CLOSE the application
        END IF
    REPEAT and wait for the next user action
END
```

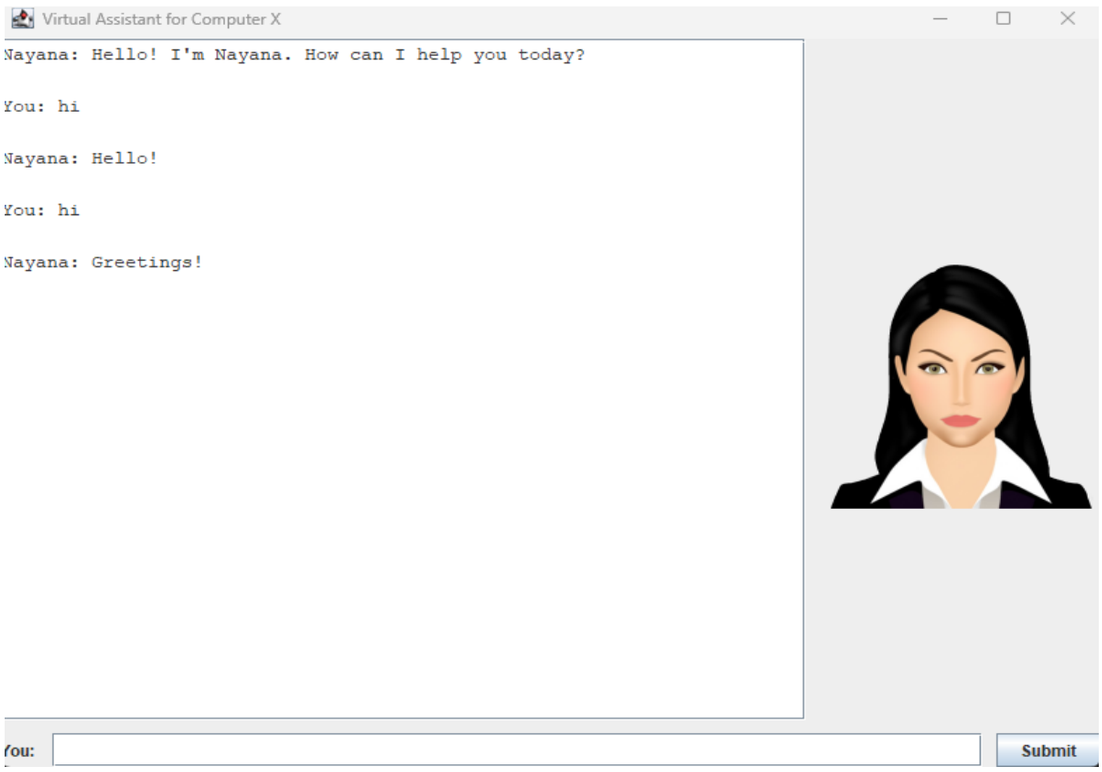
5. Testing

5.1 Test Cases

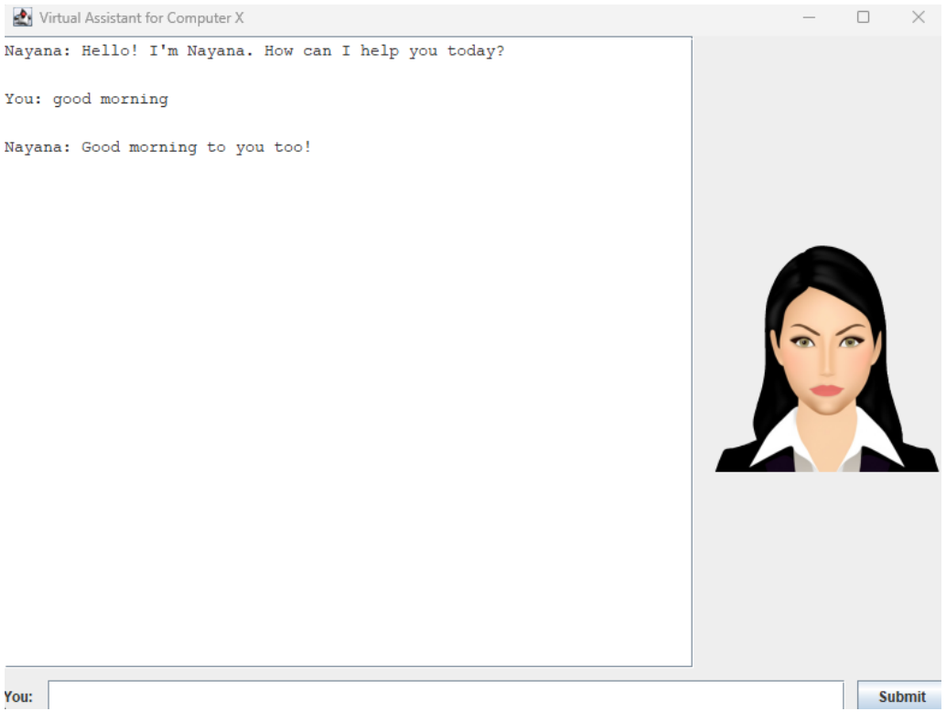
Test No	Test Case	Expected Output	Comment
1	`Hi (ask two times)`	Bot replies with different answers (can be save one sometimes due to randomness)	Random answers
2	`good morning` (when it is morning)	Bot replies "Good morning to you too!".	Time-Aware Greeting
3	`good morning` (when it is evening)	Bot replies "Actually, it's evening here, but good morning to you anyway!".	Time-Aware Correction
4	`what is your name`	Bot replies "I am Nayana... What's your name?".	User Profiling Start
5	User enters their name (e.g., "Chris")	Bot replies "It's a pleasure to meet you, Chris!".	User Profiling Memory
6	`thank you` (after name is known)	Bot gives a random reply, sometimes personalized (e.g., "You're welcome, Chris.").	Personalization
7	`show me all categories`	Bot lists all unique product categories (GPU, RAM, etc.).	Summarized List
8	`show me laptops`	Bot lists all products in the 'laptop' category.	Dynamic Answer
9	`hello` (asked 3 times in a row)	Bot gives the normal answer but adds a warning like "(By the way...)".	EI - Warning
10	`hello` (asked 4 times in a row)	Bot gives an annoyed response like "Why do you keep asking...". Avatar changes.	EI - Annoyed
11	`sorry` (after bot is annoyed)	Bot replies "It's okay. No problem." and avatar returns to normal.	EI - Reset
12	Ask an unknown question ("what is ai?")	Bot asks for the answer: "I'm not sure... Could you please tell me...".	Training Start
13	Ask the same unknown question again	Bot gives the answer that was just taught to it.	Training Recall

5.2 Test Results

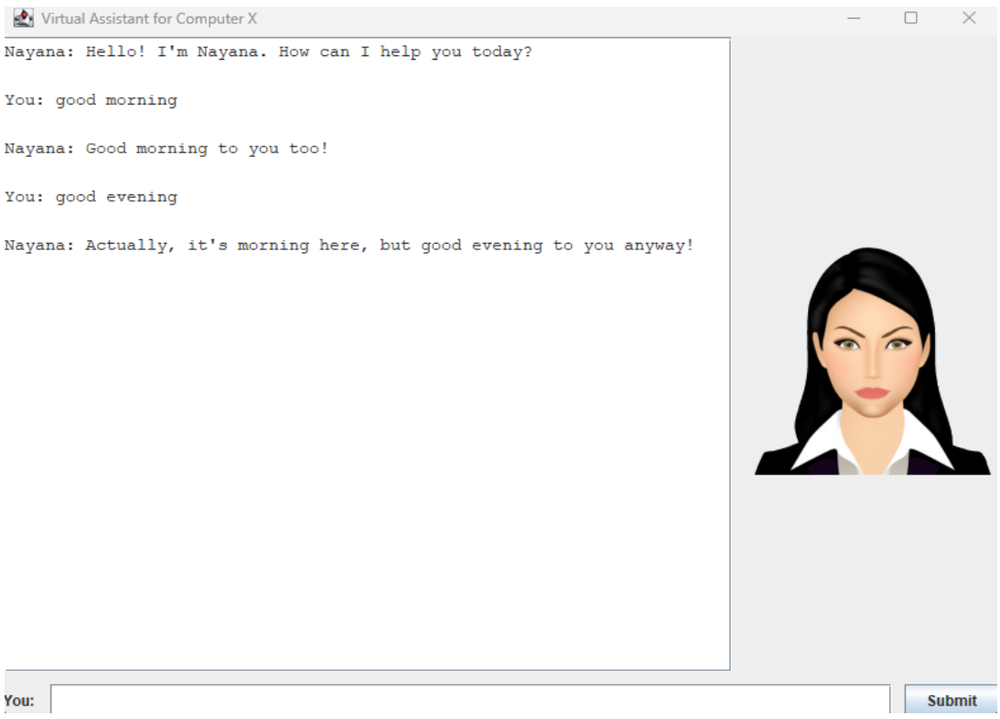
Test No 1 (Random Answers):



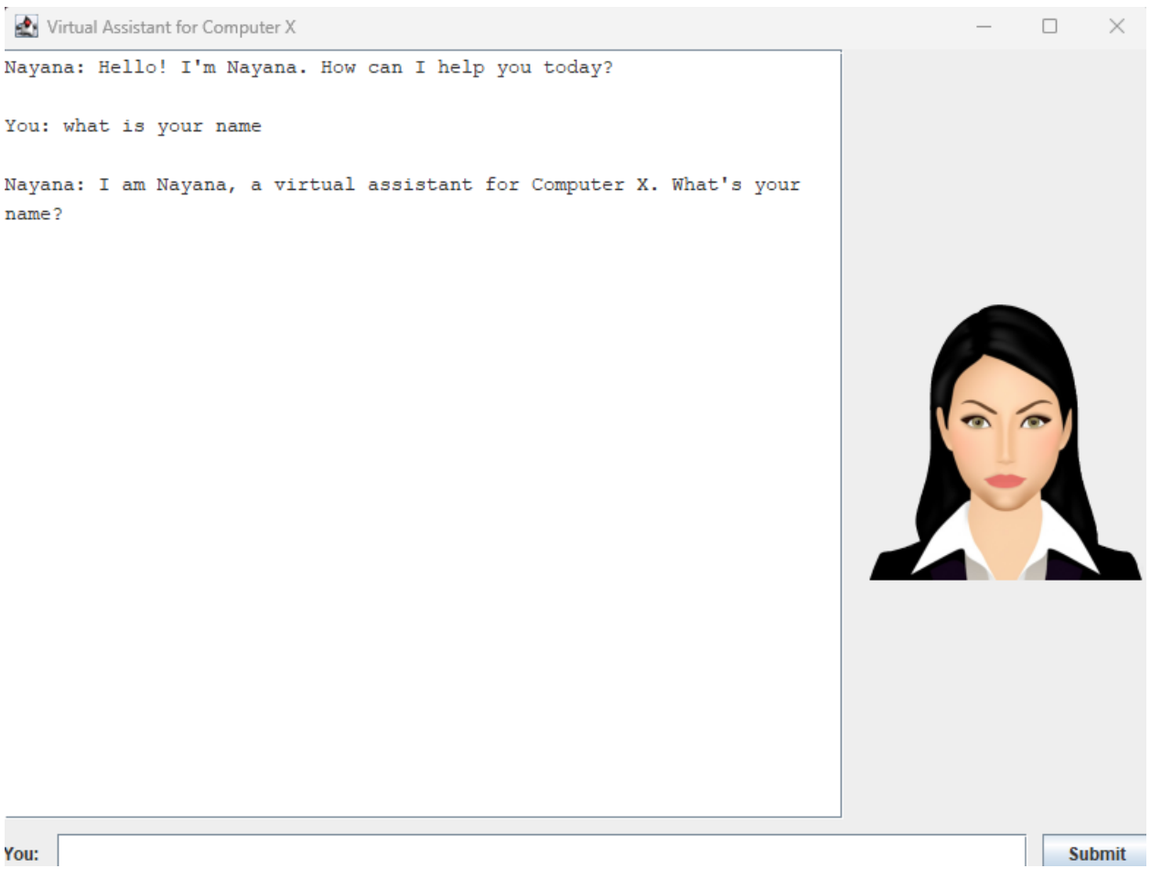
Test No 2 (Time-Awareness – Same time):



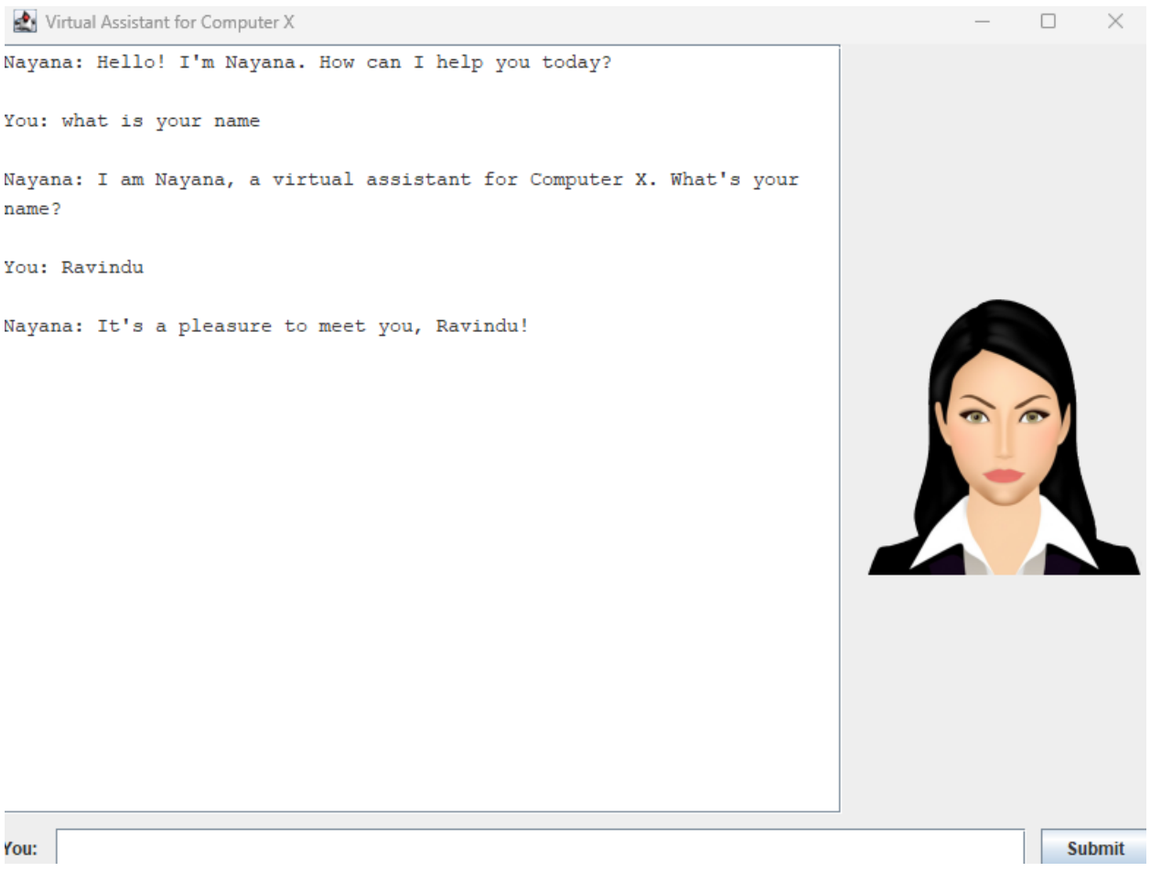
Test No 3 (Time-Awareness – different time):



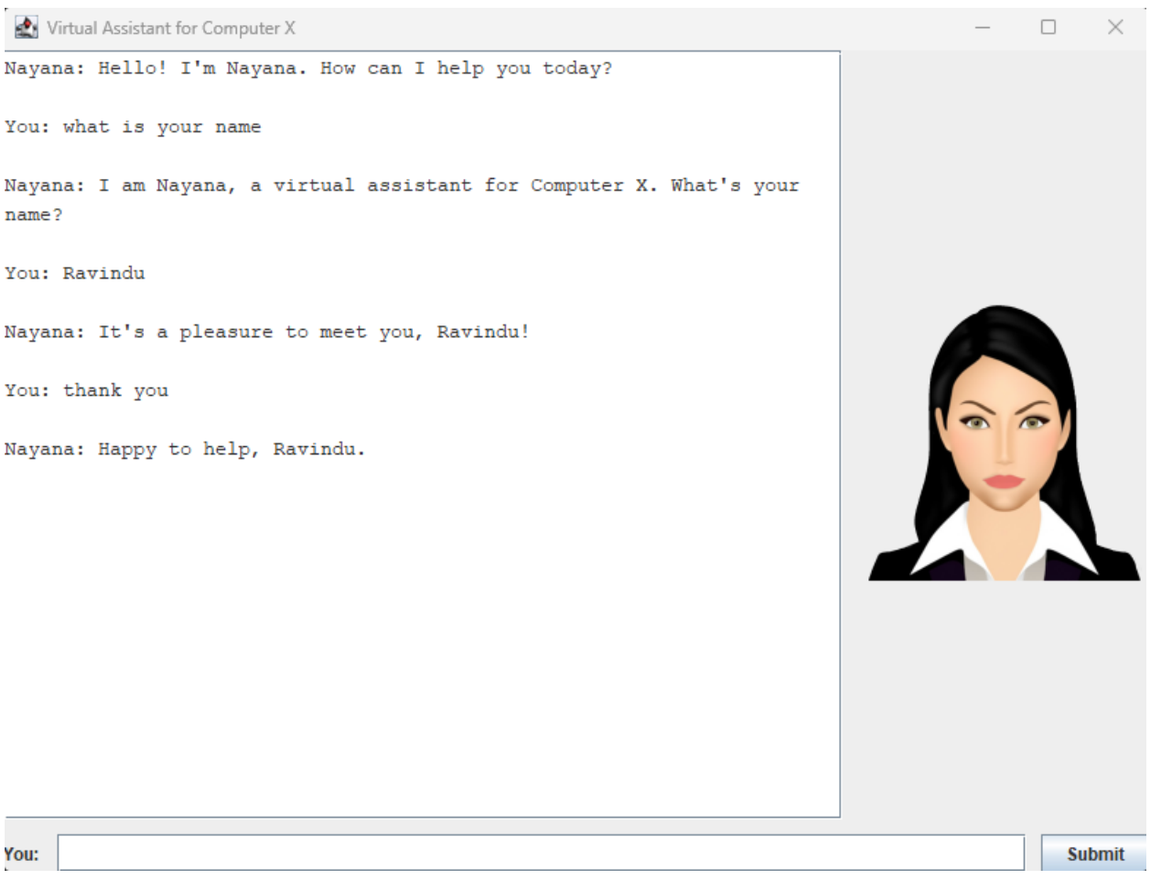
Test No 4 (User Profiling Start):



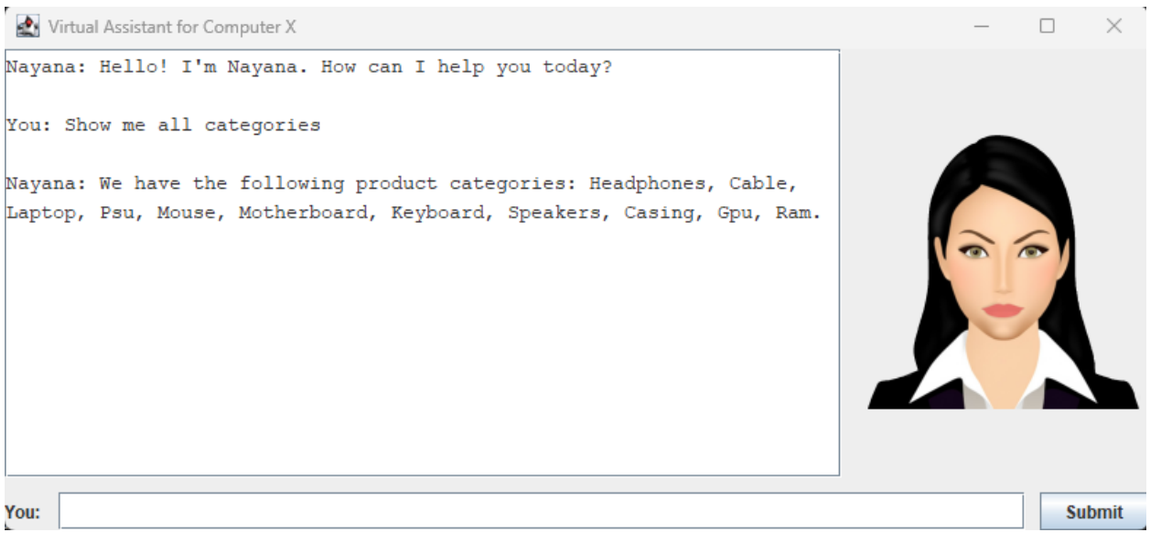
Test No 5 (User Profiling Memory:



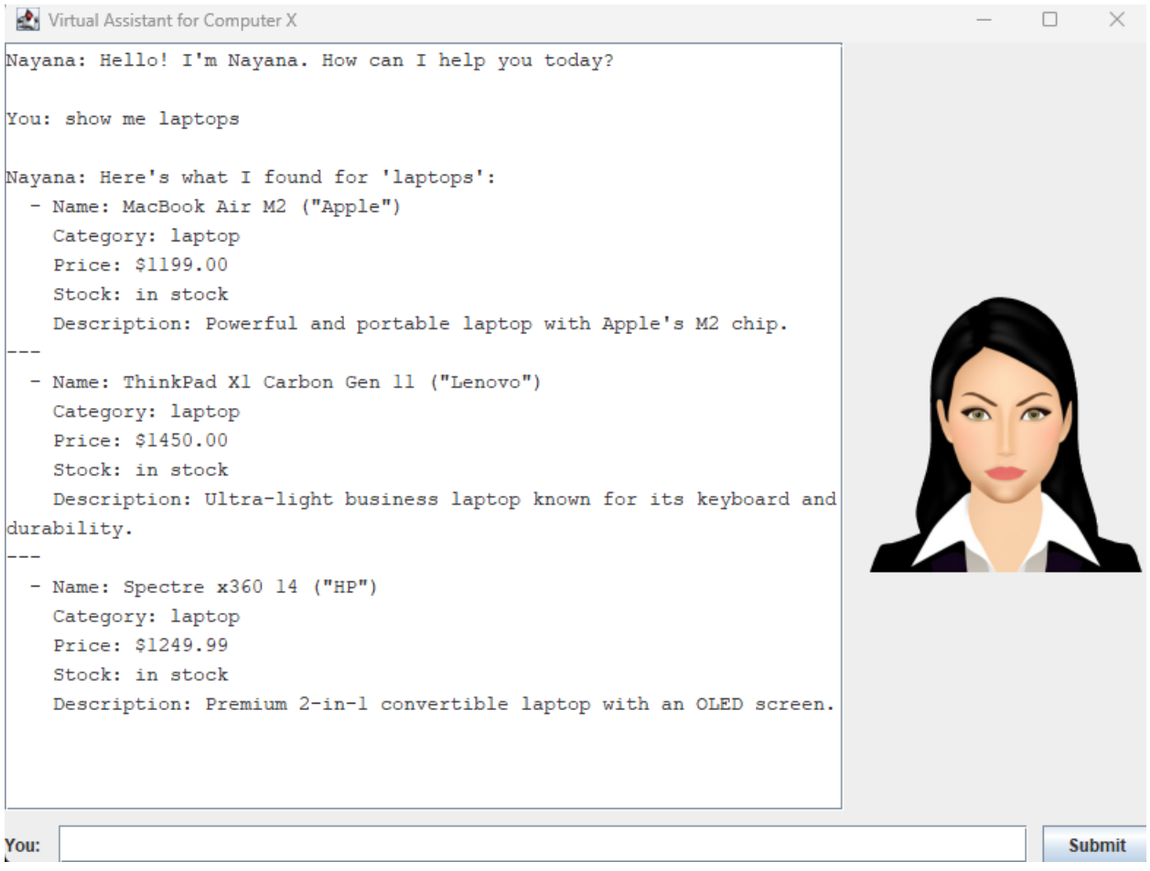
Test No 6 (Context Aware Responses):



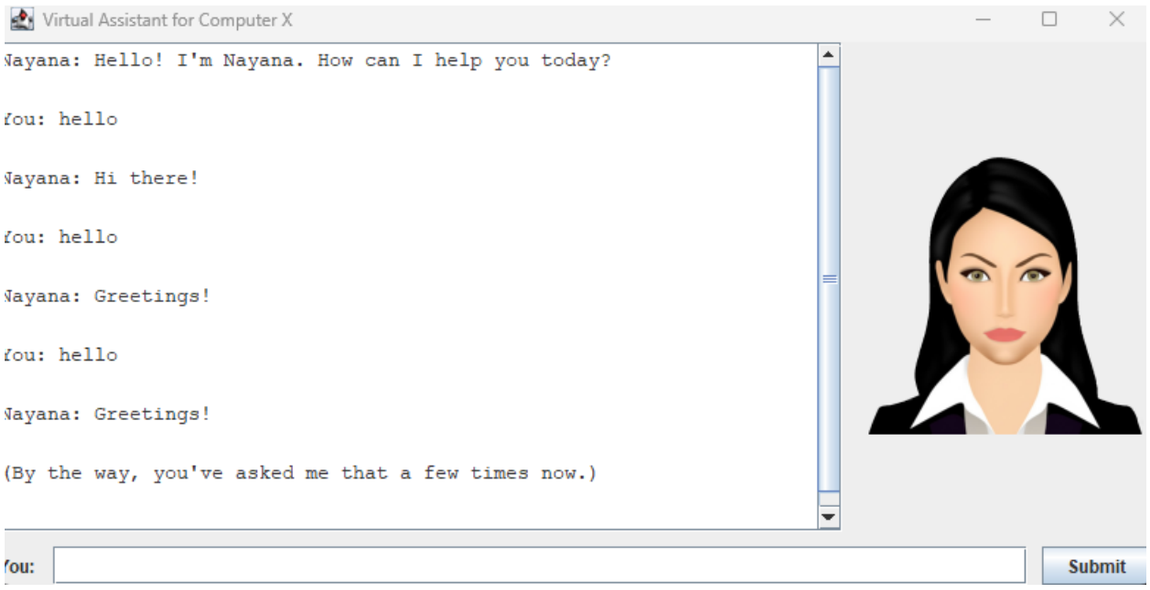
Test No 7 (Summarized List):



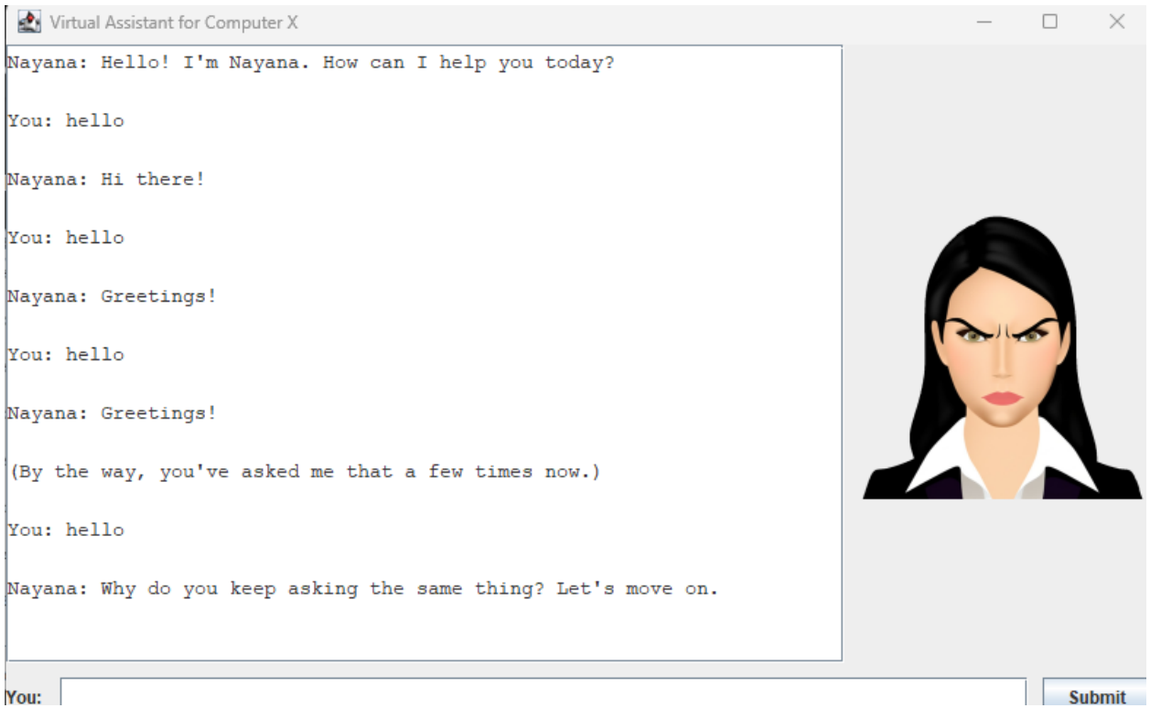
Test No 8 (Dynamic Answers):



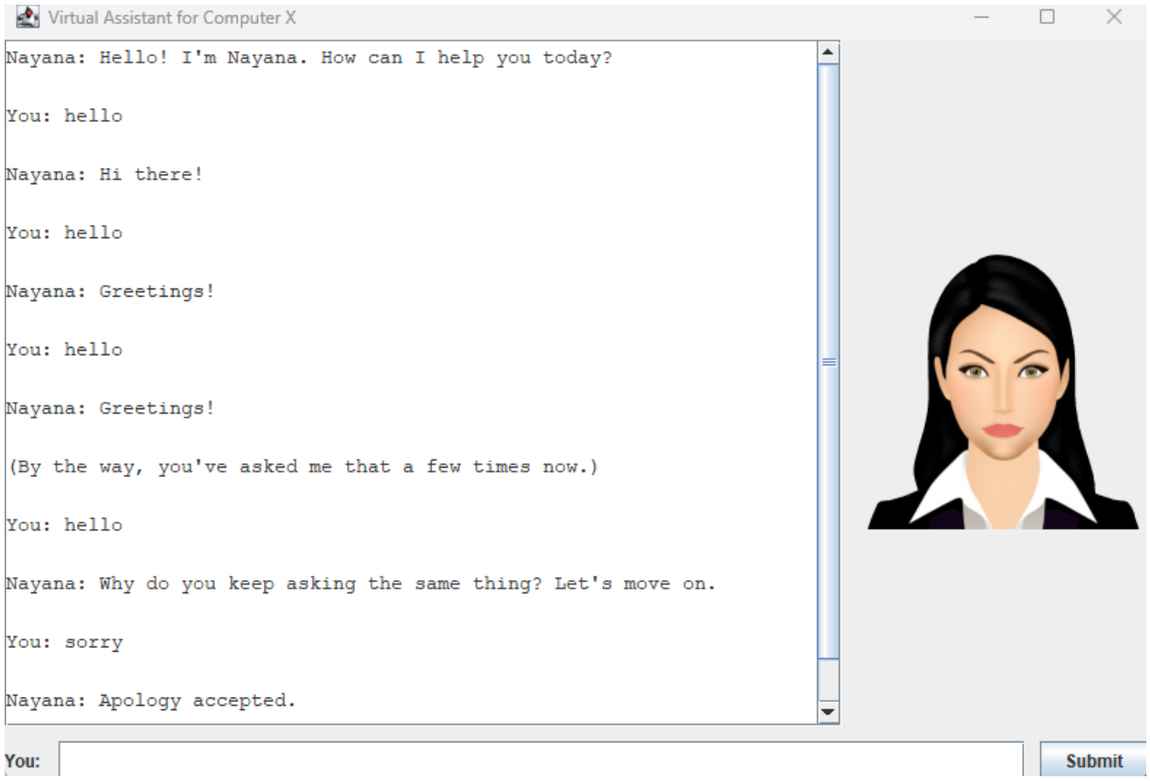
Test No 9 (Emotional Intelligence - Warning):



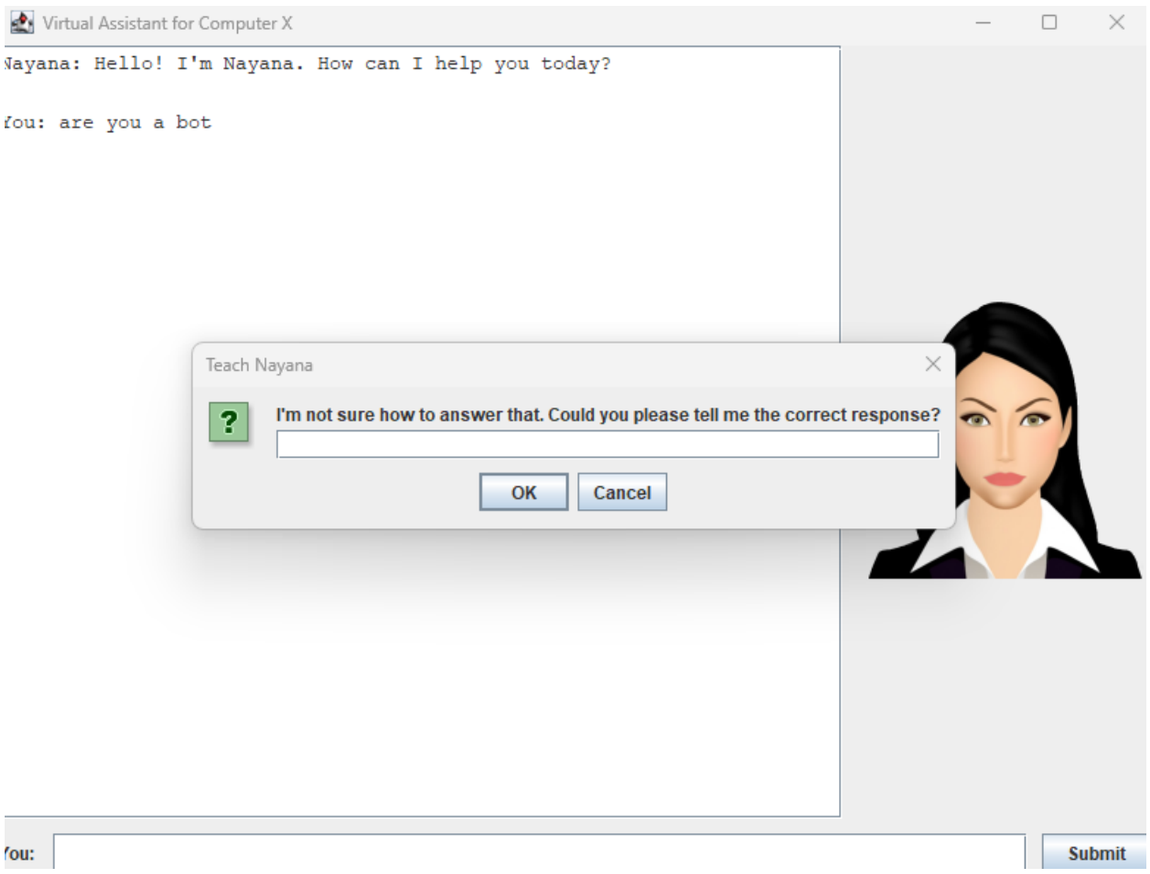
Test No 10 (Emotional Intelligence - Annoyed):



Test No 11 (Emotional Intelligence - Reset):



Test No 12 (Learning – Training start):



Test No 13 (Learning – Training Recall):

Virtual Assistant for Computer X

Nayana: Hello! I'm Nayana. How can I help you today?

You: are you a bot

Nayana: Thank you! I've learned that.

You: are you a bot

Nayana: yes

You:

Submit

6. Critical Review

The prototype chatbot, Nayana, has successfully met the project's goals. It was tested in many situations and was found to be useful in its role as a virtual assistant for Computer X.

Strengths:

- **Advanced User Experience:** The GUI with a visual avatar, combined with the Emotional Intelligence (EI) and user name recall, makes the conversation much more engaging and personal than a simple text-based bot.
- **Persistent Learning:** The bot's ability to learn new answers from the user and save them to an external file is a major strength. This was a weakness in the original "Robby" document, but it has been fully solved here. This allows the bot's knowledge to grow over time without needing a developer.
- **Robust and Well-Structured Code:** Using the Spring Boot framework allowed for a clean separation of concerns (GUI, logic, data). This makes the code easier to understand, maintain, and expand in the future. The file I/O for learning is robust and handles cases where directories don't exist.

Weaknesses:

- **Basic NLP:** The bot's understanding of language is based on simple keyword matching. It cannot handle spelling mistakes, complex sentences, or synonyms (e.g., it would not understand "show me graphics cards" because its data uses the word "gpu"). A more advanced version would use a real NLP library for better language understanding.
- **No Long-Term Memory:** The bot remembers the user's name for a single session but forgets it when the application is closed. A true user profiling system would save user information to a file or database to provide a personalized experience across multiple sessions.
- **Limited Context:** The bot cannot handle conversations that continue over multiple lines. For example, it cannot understand what "it" refers to in a follow-up question. Each input is treated as a new, separate query.

Overall, the project is a successful prototype that demonstrates several key AI concepts in a practical application. The next steps for improvement would be to integrate a more powerful NLP library and to implement a system for long-term user profiling.

7. References

Cassell, J. & Bickmore, T. (2003). *Negotiated collusion: Modeling social language and its relationship effects in intelligent agents*. User Modeling and User-Adapted Interaction, 13(1–2), 89–132.Mero, J. (2018). *The effects of two-way communication and chat service usage on consumer attitudes in the e-commerce retailing sector*. Electronic Markets, 28(2), 205–217.

Ramesh, A. (2019) *Using Natural Language Processing to Power Chatbots*. [Online] Available at: https://discover.bot/bot-talk/behind-the-scenes-using-nlp-to-power-chatbot

Xu, J. (2020) *How To Create A Chatbot with Python & Deep Learning In Less Than An Hour*. [Online] Available at: https://towardsdatascience.com/how-to-create-a-chatbot-with-python-deep-learning-in-less-than-an-hour-56a063bdfc44

Appendix: Source Code

1. pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.5</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.computerx</groupId>
    <artifactId>chatbot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>computer-x-chatbot-gui</name>
    <description>A GUI chatbot for a computer hardware shop with EI</description>
    <properties>
        <java.version>21</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

2. Src/main/java/com/computerx/chatbot/ChatbotApplication.java

```
package com.computerx.chatbot;

import com.computerx.chatbot.gui.ChatFrame;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.context.ConfigurableApplicationContext;

import java.awt.EventQueue;

@SpringBootApplication
public class ChatbotApplication {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = new
SpringApplicationBuilder(ChatbotApplication.class)
                .headless(false)
                .run(args);

        EventQueue.invokeLater(() -> {
            ChatFrame chatFrame = context.getBean(ChatFrame.class);
            chatFrame.setVisible(true);
        });
    }
}
```

3. src/main/java/com/computerx/chatbot/model/BotState.java

```
package com.computerx.chatbot.model;

public enum BotState {
    NORMAL,
    ANNOYED,
    LEARNING
}
```

4. src/main/java/com/computerx/chatbot/model/ChatResponse.java

```
package com.computerx.chatbot.model;

/**

    • Represents a response from the chatbot.
    • @param message The text content of the response.
    • @param endConversation If true, the application should close.
    • @param botState The emotional state of the bot for this response.
    • @param isPersonalizable If true, the user's name can be appended to this message.

*/ public record ChatResponse( String message, boolean endConversation, BotState botState,
boolean isPersonalizable ) {}
```

5. Src/main/java/com/computerx/chatbot/model/Product.java

```
package com.computerx.chatbot.model;

import java.util.Map;

public record Product(
    String category,
    String name,
    String brand,
    double price,
    String stock,
    String description,
    Map<String, String> attributes
) {
    @Override
    public String toString() {
        return String.format(
            "  - Name: %s (%s)\n" +
            "    Category: %s\n" +
            "    Price: $%.2f\n" +
            "    Stock: %s\n" +
            "    Description: %s",
            name, brand, category, price, stock, description
        );
    }
}
```

6. src/main/java/com/computerx/chatbot/service/KnowledgeBaseService.java

```
package com.computerx.chatbot.service;

import com.computerx.chatbot.model.Product; import jakarta.annotation.PostConstruct;
import lombok.Getter; import lombok.extern.slf4j.Slf4j; import
org.springframework.core.io.ClassPathResource; import
org.springframework.core.io.Resource; import org.springframework.stereotype.Service;

import java.io.BufferedReader; import java.io.IOException; import
java.io.InputStreamReader; import java.nio.file.Files; import java.nio.file.Path; import
java.nio.file.Paths; import java.nio.file.StandardOpenOption; import java.util.*; import
java.util.stream.Collectors;

@Service
@Slf4j
@Getter
public class KnowledgeBaseService {

private final List<Product> products = new ArrayList<>();
private final Map<String, String> learnedResponses = new ConcurrentHashMap<>();
private final Properties smallTalk = new Properties();

private static final String PRODUCTS_FILE = "data/products.csv";
private static final String LEARNED_FILE = "data/learned.txt";
private static final String SMALLTALK_FILE = "data/smalltalk.properties";
private static final String LEARNED_DELIMITER = ":::";

@PostConstruct
public void initialize() {
    loadProducts();
    loadSmallTalk();
    loadLearnedResponses();
}
```

```
public Set<String> getUniqueCategories() {
    return products.stream()
            .map(Product::category)
            .map(this::capitalize)
            .collect(Collectors.toSet());
}

public List<String> getAllProductNames() {
    return products.stream()
            .map(Product::name)
            .collect(Collectors.toList());
}

private void loadProducts() {
    try (BufferedReader reader = new BufferedReader(new InputStreamReader(new
ClassPathResource(PRODUCTS_FILE).getInputStream()))) {
        reader.readLine(); // Skip header
        String line;
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)");
            if (parts.length >= 6) {
                String category = parts[0].trim();
                String name = parts[1].trim().replace("\"", "");
                String brand = parts[2].trim();
                double price = Double.parseDouble(parts[3].trim());
                String stock = parts[4].trim();
                String description = parts[5].trim().replace("\"", "");
                Map<String, String> attributes = new HashMap<>();
                if (parts.length > 6 && !parts[6].trim().isEmpty()) {
                    String[] attrs = parts[6].trim().replace("\"", "").split(";");
                    for (String attr : attrs) {
                        String[] kv = attr.split(":");
                        if (kv.length == 2) {
                            attributes.put(kv[0].trim(), kv[1].trim());
                        }
                    }
                }
                products.add(new Product(category, name, brand, price, stock, description,
attributes));
            }
        }
        log.info("Loaded {} products from {}", products.size(), PRODUCTS_FILE);
    } catch (IOException | NumberFormatException e) {
        log.error("Failed to load or parse products file: {}. Make sure it's in
src/main/resources/data/", PRODUCTS_FILE, e);
    }
}

private void loadSmallTalk() {
    try {
        smallTalk.load(new ClassPathResource(SMALLTALK_FILE).getInputStream());
        log.info("Loaded {} small talk entries.", smallTalk.size());
    } catch (IOException e) {
        log.error("Failed to load small talk properties: {}", SMALLTALK_FILE, e);
    }
}

private void loadLearnedResponses() {
    Path externalPath = Paths.get(LEARNED_FILE);
    try {
        if (Files.exists(externalPath)) {
            Files.lines(externalPath).forEach(this::parseAndStoreLearnedLine);
            log.info("Loaded {} learned responses from external file: {}", learnedResponses.size(),
```

```
externalPath.toAbsolutePath());
        } else {
            Resource resource = new ClassPathResource(LEARNED_FILE);
            if (resource.exists()) {
                Files.lines(Paths.get(resource.getURI())).forEach(this::parseAndStoreLearnedLine);
                log.info("Loaded {} default learned responses from classpath.",
learnedResponses.size());
            } else {
                log.warn("No default or external learned.txt found. Will be created on first learn.");
            }
        }
    } catch (IOException e) {
        log.error("Failed to load learned responses: {}", e.getMessage());
    }
}

private void parseAndStoreLearnedLine(String line) {
    String[] parts = line.split(LEARNED_DELIMITER, 2);
    if (parts.length == 2) {
        learnedResponses.put(parts[0].trim(), parts[1].trim());
    }
}

public void saveLearnedResponse(String question, String answer) {
    String cleanQuestion = question.toLowerCase().trim();
    learnedResponses.put(cleanQuestion, answer);
    Path path = Paths.get(LEARNED_FILE);
    try {
        Path parentDir = path.getParent();
        if (parentDir != null && !Files.exists(parentDir)) {
            Files.createDirectories(parentDir);
            log.info("Created directory for learned data at: {}", parentDir.toAbsolutePath());
        }

        String lineToAppend = cleanQuestion + LEARNED_DELIMITER + answer +
System.lineSeparator();
        Files.write(path, lineToAppend.getBytes(), StandardOpenOption.APPEND,
StandardOpenOption.CREATE);
        log.info("Saved new learned response to external file: {}", path.toAbsolutePath());
    } catch (IOException e) {
        log.error("Failed to save learned response to file. Check write permissions for the
application directory.", e);
    }
}

public List<Product> findProductsByKeyword(String keyword) {
    String lowerKeyword = keyword.toLowerCase();
    return products.stream()
            .filter(p -> {
                String category = p.category().toLowerCase();
                boolean categoryMatch = category.contains(lowerKeyword) ||
lowerKeyword.contains(category);
                return p.name().toLowerCase().contains(lowerKeyword) ||
                        categoryMatch ||
                        p.brand().toLowerCase().contains(lowerKeyword) ||
                        p.description().toLowerCase().contains(lowerKeyword) ||
                        p.attributes().values().stream().anyMatch(v ->
v.toLowerCase().contains(lowerKeyword));
            })
            .collect(Collectors.toList());
}

private String capitalize(String str) {
    if (str == null || str.isEmpty()) {
```

```
        return str;
    }
    return str.substring(0, 1).toUpperCase() + str.substring(1).toLowerCase();
}


}
```

7. src/main/java/com/computerx/chatbot/service/ChatService.java

```
package com.computerx.chatbot.service;

import com.computerx.chatbot.model.BotState;
import com.computerx.chatbot.model.ChatResponse;
import com.computerx.chatbot.model.Product;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.time.LocalTime;
import java.util.List;
import java.util.Random;
import java.util.Set;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class ChatService {

    private final KnowledgeBaseService knowledgeBase;
    private final Random random = new Random();
    private static final Set<String> ALL_KEYWORDS = Set.of("all", "all items", "all
products", "everything");

    private String lastQuestion = "";
    private int repetitionCount = 0;
    private String userName = null;
    private boolean isWaitingForUserName = false;

    public ChatResponse getResponse(String userInput) {
        String normalizedInput = userInput.toLowerCase().trim();

        if (isWaitingForUserName) {
            this.userName = capitalize(userInput.trim());
            isWaitingForUserName = false;
            return new ChatResponse("It's a pleasure to meet you, " + this.userName + "!", false,
BotState.NORMAL, false);
        }

        if (normalizedInput.equals(lastQuestion) && !normalizedInput.isEmpty()) {
            repetitionCount++;
        } else {
            lastQuestion = normalizedInput;
            repetitionCount = 1;
        }

        if (normalizedInput.equalsIgnoreCase("sorry")) {
            repetitionCount = 0;
            lastQuestion = "";
            return new ChatResponse(getRandomResponse("It's okay. No problem.", "Apology
accepted."), false, BotState.NORMAL, true);
        }
```

```
        ChatResponse baseResponse = processQuery(normalizedInput);
        String finalMessage = baseResponse.message();
        BotState finalState = baseResponse.botState();

        if (baseResponse.isPersonalizable() && this.userName != null && random.nextInt(10) <
7) { // 70% chance
            if (finalMessage.endsWith(".") || finalMessage.endsWith("!") ||
finalMessage.endsWith("?")) {
                finalMessage = finalMessage.substring(0, finalMessage.length() - 1);
            }
            finalMessage = finalMessage + ", " + this.userName + ".";
        }

        if (repetitionCount > 3 && !baseResponse.endConversation()) {
            finalMessage = getRandomResponse("I've already answered that. Please ask a different
question.", "Why do you keep asking the same thing? Let's move on.");
            finalState = BotState.ANNOYED;
        } else if (repetitionCount == 3 && !baseResponse.endConversation()) {
            finalMessage += getRandomResponse("\n\n(By the way, you've asked me that a few
times now.)", "\n\n(Just letting you know, I believe I've answered this already.)");
        }

        return new ChatResponse(finalMessage, baseResponse.endConversation(), finalState,
false);
    }

    private ChatResponse processQuery(String normalizedInput) {
        ChatResponse timeResponse = handleTimeBasedGreeting(normalizedInput);
        if (timeResponse != null) {
            return timeResponse;
        }

        if (normalizedInput.contains("who are you") || normalizedInput.contains("what is your
name")) {
            if (this.userName != null) {
                return new ChatResponse("My name is Nayana. It's nice chatting with you, " +
this.userName + "!", false, BotState.NORMAL, false);
            } else {
                isWaitingForUserName = true;
                return new ChatResponse("I am Nayana, a virtual assistant for Computer X. What's
your name?", false, BotState.NORMAL, false);
            }
        }

        if (normalizedInput.equals("hi") || normalizedInput.equals("hello")) {
            return new ChatResponse(getRandomResponse("Hello!", "Hi there!", "Greetings!"),
false, BotState.NORMAL, true);
        }
        if (normalizedInput.equals("thanks") || normalizedInput.equals("thank you")) {
            return new ChatResponse(getRandomResponse("You're welcome!", "No problem!",
"Happy to help!"), false, BotState.NORMAL, true);
        }
        if (normalizedInput.equals("bye") || normalizedInput.equals("exit") ||
normalizedInput.equals("quit")) {
            return new ChatResponse(getRandomResponse("Goodbye!", "See you later!", "Have a
great day!"), true, BotState.NORMAL, true);
        }
        if (normalizedInput.contains("how are you")) {
             return new ChatResponse(getRandomResponse("I'm doing great, thanks for asking!",
"I'm a bot, so I'm always running at 100%!", "I'm fine, ready to help!"), false,
BotState.NORMAL, true);
        }

        if (normalizedInput.contains("categor")) {
```

```
            Set<String> categories = knowledgeBase.getUniqueCategories();
            String categoryList = String.join(", ", categories);
            return new ChatResponse("We have the following product categories: " + categoryList
+ ".", false, BotState.NORMAL, false);
        }
        if (normalizedInput.contains("product names") || normalizedInput.contains("item
names")) {
            List<String> names = knowledgeBase.getAllProductNames();
            String nameList = String.join(", ", names);
            return new ChatResponse("Here are all the product names we have: " + nameList + ".",
false, BotState.NORMAL, false);
        }

        String smallTalkResponse =
knowledgeBase.getSmallTalk().getProperty(normalizedInput.replace(" ", "."));
        if (smallTalkResponse != null) {
            return new ChatResponse(smallTalkResponse, false, BotState.NORMAL, false);
        }
        String learnedResponse = knowledgeBase.getLearnedResponses().get(normalizedInput);
        if (learnedResponse != null) {
            return new ChatResponse(learnedResponse, false, BotState.NORMAL, false);
        }

        if (normalizedInput.contains("show") || normalizedInput.contains("list") ||
normalizedInput.contains("see")) {
            String keyword = extractKeyword(normalizedInput, "show me", "list all", "see all",
"show", "list", "see");
            return findAndListProducts(keyword);
        }
        if (normalizedInput.startsWith("price of") || normalizedInput.startsWith("what is the price
of")) {
            String keyword = extractKeyword(normalizedInput, "price of", "what is the price of");
            return findProductPrice(keyword);
        }
        if (normalizedInput.contains("in stock") || normalizedInput.contains("available")) {
            String keyword = extractKeyword(normalizedInput, "is", "are", "in stock",
"available");
            return checkStock(keyword);
        }
        if (normalizedInput.contains("do you have") || normalizedInput.contains("any")) {
            String keyword = extractKeyword(normalizedInput, "do you have", "any");
            return findAndListProducts(keyword);
        }

        return new ChatResponse("I'm not sure how to answer that. Could you please tell me the
correct response?", false, BotState.LEARNING, false);
    }

    private ChatResponse handleTimeBasedGreeting(String normalizedInput) {
        int currentHour = LocalTime.now().getHour();
        String actualTimeOfDay;
        if (currentHour >= 5 && currentHour < 12) actualTimeOfDay = "morning";
        else if (currentHour >= 12 && currentHour < 17) actualTimeOfDay = "afternoon";
        else if (currentHour >= 17 && currentHour < 22) actualTimeOfDay = "evening";
        else actualTimeOfDay = "night";

        String userGreeting = null;
        if (normalizedInput.contains("morning")) userGreeting = "morning";
        else if (normalizedInput.contains("afternoon")) userGreeting = "afternoon";
        else if (normalizedInput.contains("evening")) userGreeting = "evening";
        else if (normalizedInput.contains("night")) userGreeting = "night";

        if (userGreeting != null) {
            if (userGreeting.equals(actualTimeOfDay)) {
```

```
            return new ChatResponse("Good " + actualTimeOfDay + " to you too!", false,
BotState.NORMAL, true);
          } else {
            return new ChatResponse("Actually, it's " + actualTimeOfDay + " here, but good " +
userGreeting + " to you anyway!", false, BotState.NORMAL, true);
          }
       }
       return null;
    }

    public void learn(String question, String answer) {
       knowledgeBase.saveLearnedResponse(question, answer);
    }

    private String getRandomResponse(String... responses) {
       return responses[random.nextInt(responses.length)];
    }

    private String capitalize(String str) {
       if (str == null || str.isEmpty()) return str;
       return str.substring(0, 1).toUpperCase() + str.substring(1).toLowerCase();
    }

    private ChatResponse findAndListProducts(String keyword) {
       if (keyword.isBlank()) {
          return new ChatResponse("What kind of products are you looking for? For example:
'show me laptops'.", false, BotState.NORMAL, false);
       }
       List<Product> foundProducts;
       if (ALL_KEYWORDS.contains(keyword.toLowerCase())) {
          foundProducts = knowledgeBase.getProducts();
       } else {
          foundProducts = knowledgeBase.findProductsByKeyword(keyword);
       }
       if (foundProducts.isEmpty()) {
          return new ChatResponse("Sorry, I couldn't find any products matching '" + keyword +
"'.", false, BotState.NORMAL, false);
       }
       String productList =
foundProducts.stream().map(Product::toString).collect(Collectors.joining("\n---\n"));
       String responseHeader = ALL_KEYWORDS.contains(keyword.toLowerCase())
          ? "Here are all the products we have:\n"
          : "Here's what I found for '" + keyword + "':\n";
       return new ChatResponse(responseHeader + productList, false, BotState.NORMAL,
false);
    }

    private ChatResponse findProductPrice(String keyword) {
       List<Product> foundProducts = knowledgeBase.findProductsByKeyword(keyword);
       if (foundProducts.isEmpty()) {
          return new ChatResponse("Sorry, I couldn't find a product named '" + keyword + "'.",
false, BotState.NORMAL, false);
       }
       if (foundProducts.size() > 1) {
          String suggestions =
foundProducts.stream().map(Product::name).limit(5).collect(Collectors.joining(", "));
          return new ChatResponse("I found multiple products matching '" + keyword + "': " +
suggestions + "... Can you be more specific?", false, BotState.NORMAL, false);
       }
       Product p = foundProducts.get(0);
       return new ChatResponse(String.format("The price of the %s is $%.2f.", p.name(),
p.price()), false, BotState.NORMAL, false);
    }
```

```
    private ChatResponse checkStock(String keyword) {
        List<Product> foundProducts = knowledgeBase.findProductsByKeyword(keyword);
        if (foundProducts.isEmpty()) {
            return new ChatResponse("Sorry, I couldn't find a product named '" + keyword + "'.",
false, BotState.NORMAL, false);
        }
        if (foundProducts.size() > 1) {
            String suggestions =
foundProducts.stream().map(Product::name).limit(5).collect(Collectors.joining(", "));
            return new ChatResponse("I found multiple products matching '" + keyword + "': " +
suggestions + "... Can you be more specific?", false, BotState.NORMAL, false);
        }
        Product p = foundProducts.get(0);
        String stockStatus = p.stock().equalsIgnoreCase("in stock") ? "is in stock" : "is currently
out of stock";
        return new ChatResponse(String.format("The %s %s.", p.name(), stockStatus), false,
BotState.NORMAL, false);
    }

    private String extractKeyword(String input, String... prefixes) {
        String result = input;
        for (String prefix : prefixes) {
            result = result.replace(prefix, "");
        }
        return result.replace("?", "").trim();
    }
}
```

8. src/main/java/com/computerx/chatbot/gui/ChatFrame.java

```
package com.computerx.chatbot.gui;

import com.computerx.chatbot.model.BotState; import
com.computerx.chatbot.model.ChatResponse; import
com.computerx.chatbot.service.ChatService; import
org.springframework.stereotype.Component;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
import java.net.URL;

@Component
public class ChatFrame extends JFrame {

private final ChatService chatService;
private JTextArea chatArea;
private JTextField inputField;
private JButton submitButton;
private JLabel avatarLabel;

private ImageIcon normalAvatar;
private ImageIcon annoyedAvatar;

public ChatFrame(ChatService chatService) {
    this.chatService = chatService;
    loadAvatars();
    initComponents();
}

private void loadAvatars() {
    URL normalUrl = getClass().getClassLoader().getResource("images/nayana-normal.png");
```

```
    URL annoyedUrl = getClass().getClassLoader().getResource("images/nayana-
annoyed.png");

    if (normalUrl != null) {
        normalAvatar = new ImageIcon(new
ImageIcon(normalUrl).getImage().getScaledInstance(200, 200, Image.SCALE_SMOOTH));
    } else {
        System.err.println("Could not find normal avatar image! Make sure it's in
src/main/resources/images/");
    }

    if (annoyedUrl != null) {
        annoyedAvatar = new ImageIcon(new
ImageIcon(annoyedUrl).getImage().getScaledInstance(200, 200, Image.SCALE_SMOOTH));
    } else {
        System.err.println("Could not find annoyed avatar image! Make sure it's in
src/main/resources/images/");
    }
}

private void initComponents() {
    setTitle("Virtual Assistant for Computer X");
    setSize(800, 600);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new BorderLayout(10, 10));

    JPanel mainPanel = new JPanel(new BorderLayout(10, 10));
    mainPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

    JPanel centerPanel = new JPanel(new BorderLayout(10, 10));

    chatArea = new JTextArea();
    chatArea.setEditable(false);
    chatArea.setLineWrap(true);
    chatArea.setWrapStyleWord(true);
    chatArea.setFont(new Font("Monospaced", Font.PLAIN, 14));
    JScrollPane scrollPane = new JScrollPane(chatArea);
    centerPanel.add(scrollPane, BorderLayout.CENTER);

    avatarLabel = new JLabel();
    setAvatar(BotState.NORMAL);
    centerPanel.add(avatarLabel, BorderLayout.EAST);

    mainPanel.add(centerPanel, BorderLayout.CENTER);

    JPanel bottomPanel = new JPanel(new BorderLayout(10, 10));
    inputField = new JTextField();
    inputField.setFont(new Font("SansSerif", Font.PLAIN, 14));
    submitButton = new JButton("Submit");

    bottomPanel.add(new JLabel("You: "), BorderLayout.WEST);
    bottomPanel.add(inputField, BorderLayout.CENTER);
    bottomPanel.add(submitButton, BorderLayout.EAST);

    mainPanel.add(bottomPanel, BorderLayout.SOUTH);
    add(mainPanel);

    ActionListener actionListener = e -> processUserInput();
    submitButton.addActionListener(actionListener);
    inputField.addActionListener(actionListener);

    appendToChat("Nayana", "Hello! I'm Nayana, the virtual assistant for Computer X. How
can I help you today?");
}
```

```
private void processUserInput() {
    String userInput = inputField.getText().trim();
    if (userInput.isEmpty()) {
        return;
    }

    appendToChat("You", userInput);
    inputField.setText("");

    ChatResponse response = chatService.getResponse(userInput);

    if (response.botState() == BotState.LEARNING) {
        String teachingAnswer = JOptionPane.showInputDialog(
                this,
                response.message(),
                "Teach Nayana",
                JOptionPane.QUESTION_MESSAGE
        );
        if (teachingAnswer != null && !teachingAnswer.isBlank()) {
            chatService.learn(userInput, teachingAnswer);
            appendToChat("Nayana", "Thank you! I've learned that.");
        } else {
            appendToChat("Nayana", "Okay, I won't learn that for now.");
        }
    } else {
        appendToChat("Nayana", response.message());
    }

    setAvatar(response.botState());

    if (response.endConversation()) {
        JOptionPane.showMessageDialog(this, "Goodbye! The application will now close.");
        System.exit(0);
    }
}

private void appendToChat(String user, String message) {
    chatArea.append(String.format("%s: %s\n\n", user, message));
    chatArea.setCaretPosition(chatArea.getDocument().getLength());
}

private void setAvatar(BotState state) {
    if (state == BotState.ANNOYED && annoyedAvatar != null) {
        avatarLabel.setIcon(annoyedAvatar);
    } else if (normalAvatar != null) {
        avatarLabel.setIcon(normalAvatar);
    }
    this.revalidate();
    this.repaint();
}
}
```