

Comprehensive Exercise Report

Team AI Chatbot for Gym! Latvia of Section <<000>>

Oleksii Pecheniuk 221ADB097, Mykyta Medvediev 221ADB084, Mariana Mechyk 221ADB094, Ruslan Dzhubuev 221ADB183, Ayomide Akintola 221ADB162, Agboola Peter 221ADB155

Requirements/Analysis	2
Journal	2
Software Requirements	3
Black-Box Testing	4
Journal	4
Black-box Test Cases	5
Design	6
Journal	6
Software Design	7
Implementation	8
Journal	8
Implementation Details	9
Testing	10
Journal	10
Testing Details	11
Presentation	12
Preparation	12
Grading Rubric	13

Requirements/Analysis

Week 2

Journal

- **Brief project overview and requirements list after communication with the client:**
 - Development and deployment of an AI-powered chatbot for the Gym! Latvia website, integrated via Squarespace, to provide 24/7 automated responses to frequently asked questions and reduce support team workload.
 - Chatbot should be embedded in the bottom right corner of the Gym! Latvia website.
 - Chatbot will open as a popup on click.
 - Responds only to Gym! Latvia-specific inquiries using a predefined list of FAQ-style questions.
 - If a question is irrelevant, it will respond with a default message and offer the contact email (info@gymlatvija.lv).
 - Multilingual support: chatbot must adapt language based on the website's selected language.
 - HTML/CSS/JavaScript for the front-end via Squarespace's code block.
 - Backend service must:
 - Be scalable and support logging and monitoring.
 - Be tested (unit and integration).
 - Integrate with Ollama LLM.
 - Manual and/or scheduled updates to maintain current data.
 - Integrate smoothly with Squarespace.
 - Support average traffic of 6–10k users per week.
- **Questions asked for the client:**
 - What are routine (frequently asked) questions for the support team?

Use current FAQ questions and answers for the use of the chatbot: <https://www.gymlatvija.lv/faq>
 - What are the use cases of a chatbot (additional user stories expectations, if such exists)?

No additional user story expectations.
 - Will you provide/ do you have areas for testing and development (e.g. sandbox, mock data)?

I will possibly give you additional information during the testing of the chatbot.
 - What can be a communication channel - Teams, Discord (anything works)?

The client is comfortable communicating using Outlook.
 - What knowledge base is expected to be used?

Use <https://www.gymlatvija.lv> website as prompt.

- Can you provide metrics or KPIs about the usual traffic of users? (for defining website scalability)

Since New Year's our website has around 6-10k visits per week.

- **Unfamiliar topic and references for them.**

- Squarespace Code Block Integration
Ref: [Squarespace Custom Code Documentation](#)
- Ollama LLM models
Ref: <https://ollama.com/>

- **Target audience of the Chatbot AI:**

- Gym! Latvia visitors (potential and current members)

- **User Interaction:**

- *Visitors:*
 - Click on the chatbot at the bottom right of the website.
 - Ask the questions related to gym services.
 - Receive quick answers or directions to contact support.
- *Support/Admin Team:*
 - Provide and update FAQ data through the Squarespace button.
 - Monitor chatbot performance and update backend data sources.
 - Use logging tools for issue resolution.

- **Must-have features of the Chatbot AI:**

- Text-based AI chatbot accessible on the website
- Answer Gym! Latvia-related FAQ questions
- Multilingual interface support: LV, EN, RU
- Default message for out-of-scope questions
- Easy contact option via email
- Backend with Ollama LLM model, logging, and monitoring
- Frontend built using HTML, CSS, and JavaScript via Squarespace
- Admin-friendly mechanism for updating chatbot data (button)

- **Other notes:**

- Chatbot must be independent yet smoothly integrated with Squarespace.
- Needs to handle 6–10k weekly users.
- Ensure GDPR compliance (user data handling).
- Scalability is a concern; cloud hosting solutions may be considered.
- Future upgrades might include personalization or advanced AI features.

Software Requirements

Project Overview

The goal of this project is to develop an AI-powered chatbot for the Gym! Latvia website, designed to automate responses to frequently asked customer questions and improve overall user experience. The chatbot will serve as a first-line support tool, available 24/7, and embedded directly within the website's interface using Squarespace's code blocks. Its main objective is to provide fast, reliable, and context-specific information to both potential and current gym members. The chatbot will utilize a backend integrated with the Ollama LLM to deliver natural language responses based strictly on Gym! Latvia-related content. It will support multilingual responses, adapt to website language settings, and offer a fallback contact method for questions outside its scope. The system will also include logging, monitoring, and scalable backend architecture to support 6–10k weekly users.

Functional Requirements

1. Chatbot Interface

- The chatbot will be accessible from the bottom-right corner of the Gym! Latvia website.
- The chatbot will be opened as a popup window.

2. Question Handling

- The chatbot will only respond to Gym! Latvia-related questions.
- The chatbot will use predefined questions sourced from the FAQ section of the website.
- The chatbot will respond with a standard message for unrelated questions, including contact info (info@gymlatvija.lv).

3. Language Support

- The chatbot will detect and respond in the current website language.
- The chatbot will support Latvian, English, and Russian languages.

4. Backend Integration

- The chatbot will use the API of local running LLM using Ollama for generating responses.
- The backend will be developed using a modern framework (Spring Java).
- The backend will include logging and monitoring tools.

5. Deployment

- The chatbot will be embedded using Squarespace's custom code block feature.
- The chatbot UI will be built using HTML, CSS, and JavaScript.

6. Performance

- The chatbot will handle an average traffic load of 6–10k users per week.
- The chatbot will respond to user input within 2 seconds.

7. Security & Testing

- The backend will be covered by unit and integration tests.
- The chatbot will comply with data privacy regulations (e.g., GDPR).

User Stories

- As a gym member, I want to ask the chatbot about membership options so that I can understand what plans are available.
- As a potential customer, I want to ask about class schedules to see if they fit my routine.
- As a user, I want the chatbot to respond in my selected website language so I can understand the information easily.

Black-Box Testing

Instructions: Week 4

Journal

- **Input for software**

- Text input from users in the chatbot interface (usually a few sentences or questions).
- Website language selection (used to determine chatbot response language).
- Click event to open the chatbot popup.

- **Output of software**

- Textual responses based on predefined Gym! Latvia FAQs.
- Default fallback message for irrelevant questions.
- Email contact information when necessary.
- Multilingual responses (based on website language).

- **Broken inputs examples**

- Valid gym-related questions (mapped to predefined answers).
- Invalid/general questions (not related to Gym! Latvia).
- Empty input or gibberish (e.g., “????” or blank submission).
- Non-text input (e.g., emojis, unsupported characters).

- **Boundaries for input**

- Minimum input length: 1 character
- Maximum input length (test system tolerance): 200–500 characters
- Rapid multiple submissions (to test rate limiting or system response)
- Edge cases: Long multilingual phrases, symbols, special characters

- **Other test cases:**

- Switching website language to test dynamic multilingual support
- Load testing with 10,000+ users to simulate traffic
- Testing deployment compatibility on Squarespace
- Testing OpenAI API response times under different input scenarios

- **Other notes:**

- Responses should be accurate, timely, and formatted clearly.
- Test coverage should include user stories like asking about pricing, opening hours, or membership options.
- Ensure logging and monitoring systems properly record errors and interactions.
- Check GDPR compliance for data usage during interactions.

Black-box Test Cases

Use your notes from above to complete the black-box test plan section of the formal documentation by writing black box test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement.

Test ID	Description	Expected Results	Actual Results
1	User asks a valid Gym! Latvia question: "What are your membership plans?"	Chatbot provides a clear and correct answer from the predefined FAQ about membership plans.	Message sent, AI replied with: GymLatvija offers two types of memberships: Lifestyle and 365 days. The Lifestyle membership is binding for 12 months, while the 365 days membership begins with the start date chosen by the member at the time of registration and ends exactly one year later.
2	User asks a non-valid Gym! Latvia question (not related to Gym! Latvia): "What are your weekend plans?"	Chatbot responds with a fallback message: "Apologies, I am answering only Gym! Latvia 🦵 related topics. You can contact us at info@gymlatvija.lv."	Message sent, AI replied with: Weekend plans? Sorry, but I'm a GymLatvija chatbot! My main concern is to assist and provide information about our sports club services. If you have any questions or concerns about our gym, membership, or rules, feel free to ask me anytime!
3	User enters an extremely long question (500+ characters)	Chatbot trims or gracefully handles the input without crashing; responds appropriately or shows a length warning.	Message sent, AI replied with: I'm GymLatvija chatbot! I can answer your questions about membership, facilities, and rules of our sports clubs in Latvia. Please go ahead and ask me anything you'd like to know!
4	User inputs only symbols or blank input (e.g., "???" or empty)	Send button is inactive for short inputs. For "???" - paraphrase the sentence.	Message "???" sent, AI replied with: I'm a GymLatvija chatbot! I'll do my best to assist you

			with any questions or concerns. Please go ahead and ask your question, and I'll respond accordingly.
5	User changes the website language to Latvian and asks a valid question	Chatbot detects language and responds in Latvian with correct answers from the predefined set.	Message "Kādi ir jūsu dalības plāni?" was sent, bot replied with: "Our subscription plans are as follows: * Lifestyle membership - 12 months * 365 days membership - 1 year * Trial training membership - 3 calendar days Please note that our self-service gym does not have personnel, but there are knowledgeable trainers available. Access control and safety are ensured remotely."

Design

Instructions: Week 6

Journal

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- List the nouns from your requirements/analysis documentation.
 - Chatbot, AI, Client, Admin, Question, Answer, API, Squarespace, backend, frontend, email, data, deployment, development, KPI, integration, member, prompt, price, software, visitor.
- Which nouns potentially may represent a class in your design?
 - Request, message, user, API, chatbot, service, answer, history, AI, FAQ, content, data, logging, prompt, configuration, website.
- Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
 - userID, message, FAQ, chatbot, redisService, ollamaClient, data, redisTemplate, history, systemMessage, args.
- Now that you have a list of possible classes, consider different design options (**lists of classes and attributes**) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.
 - OPTION 1 service centralized design:
 - Chatbot Request class:
 - Attributes: userID, message.
Used to represent a request which is sent be user to the chatbot.
 - Chatbot Service class:
 - Attributes: redisService, ollamaClient, htmlReadService, initialPrompts
Used to specify the main logic, which include builds prompt, conversation tracking, interaction with FAQ and managing the history of the conversation
 - ChatbotContoller class:
 - Attributes: chatbotService
Used to handle API endpoints, get user messages, and passes them for processing.
 - OllamaChatClient class:
 - Attributes: connection for AI
Used for communication with the AI backend
 - HtmlReadService class:
 - Attributes: FAQ_url,FAQ_entities
Used to read the FAQ data from the Gym Latvia website to get the most up to date responses.
 - RedisService class:
 - Attributes: redisTemplate, user history
Used for managing session storage for chat history.
 - InitialPrompts class
 - Attributes: systemPrompt, FAQ

Used to store and handle default prompts and FAQ knowledge base for the chat bot

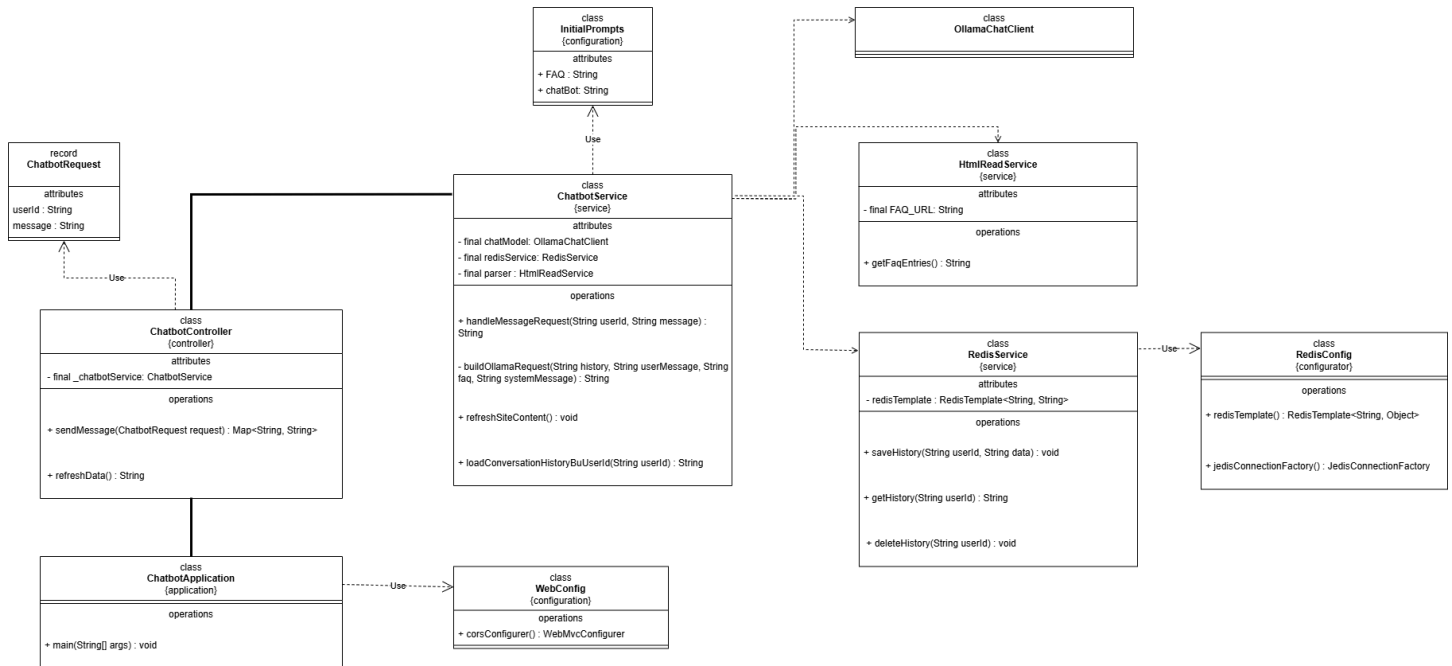
- WebConfig class:
 - Attributes: corsConfigurer
Used to configure CORS (Cross Origin Resource Sharing) for web compatibility.
- ChatbotApplication class
 - Attributes: argos
This is the main entry point which is initializing all of the components.
- Pros:
 - Clear separation of all the responsibilities
 - Easy to extend or replace services
 - Supports scaling, logging and history management
- Cons:
 -
- OPTION 2 data centralized design
- UserMessage
 - Attributes: userID, message
Used to show and display any message sent by the user.
- FAQ
 - Attributes: question, answer, language
Used to display frequently asked questions and answers for them.
- Chatbot
 - Attributes: FAQ, history
Used to handle all the core chat logic, matching user questions to FAQ or if it is missing - to AI model.
- History
 - Attributes: userID, messages
Used to follow conversation for each user
- Pros:
 - Simpler, less parts
 - Easier to implement for smaller
- Cons:
 - Harder to maintain and extend as logic is more centralized on data and business.
- Which design do you plan to use? Explain why you have chosen this design.
 - We plan to use option 1 which is more focused on the service. This design is better in our case as it offers separation of responsibilities between components. Every functionality has its own class. It helps with maintaining and extending in the future if we want to add more data sources or add AI provider or include analytics.
It is a good choice for reliability, scalability and future improvements for the chatbot.
- List the verbs from your requirements/analysis documentation.
 - Develop, deploy, request, provide, integrate, respond, offer, store, support, maintain, update, resolve, monitor, handle, search, answer, fetch, send, initialize
- Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
 - Request - handleRequest() method, ChatbotController class
 - Provide - provideResponse() method, ChatbotService class
 - Fetch - fetchFAQ() method, HtmlReadService class
 - Store - saveHistory() and getHistory() methods, RedisService class
 - Update - refreshContent() method, InitialPrompts

- Initialize - main() method, ChatbotApplication
 - Manage, handle - handleMessage() method, ChatbotService class
- Other notes:
 - <<Insert notes>>

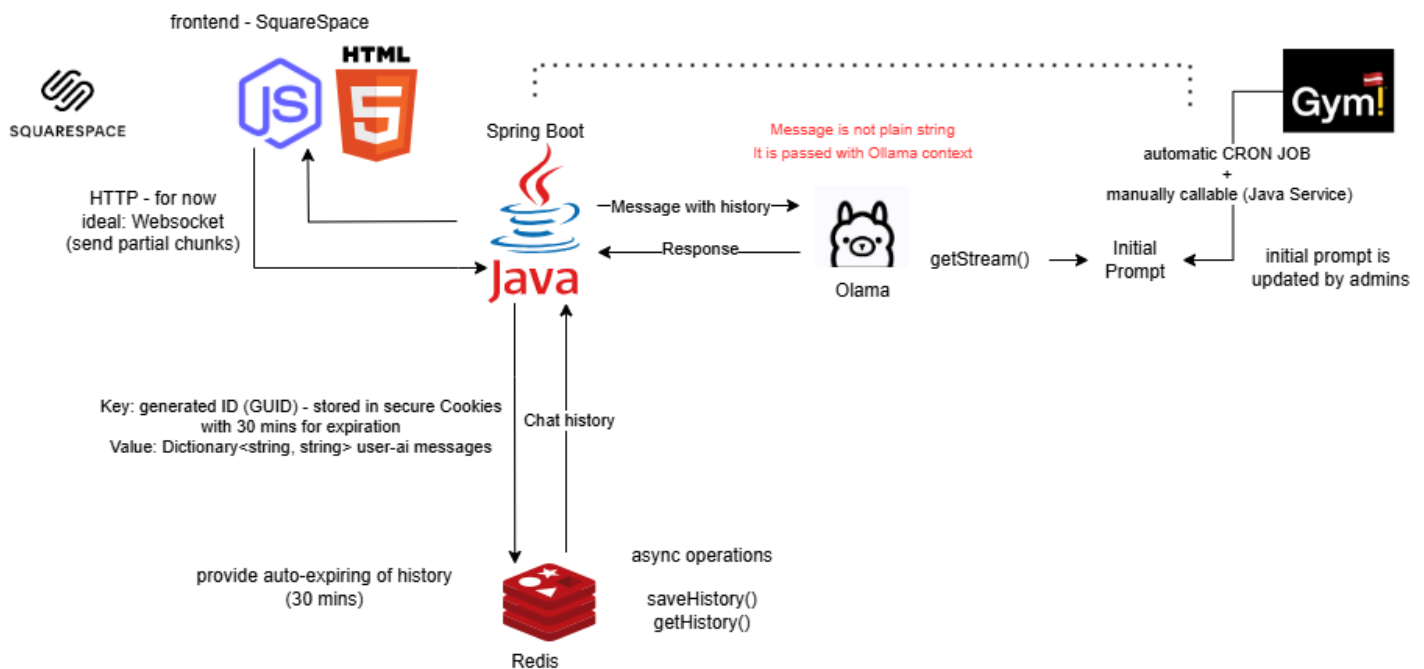
Software Design

<<Use your notes from above to complete this section of the formal documentation by planning the classes, methods, and fields that will be used in the software. Your design should include UML class diagrams along with method headers. **Prior to starting the formal documentation, you should show your answers to the above prompts to your instructor.>>**

The UML diagram



The final architecture of the application:



Implementation

Instructions: Week 8

Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompt below and feel free to add additional notes.

- What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation.
 - Object-Oriented Programming (OOP). Classes like ChatbotService, RedisService, and HtmlReadService encapsulate specific responsibilities. It promotes modularity and separation of concerns.
 - ChatbotService: Handles business logic related to generating or processing chatbot responses.
 - RedisService: Manages interactions with Redis, likely for caching user sessions and prompt history.
 - HtmlReadService: Reads and parses content from HTML files.
- Other notes:
 - ChatbotController.java handles API endpoints, acting as the interface between the frontend and backend logic.
 - DTO like ChatbotRequest.java standardizes the structure of incoming/outgoing data.
 - RedisConfig.java and WebConfig.java configure backend components.

Implementation Details

User sends a message via the frontend UI or API call.

The message is routed through ChatbotController to ChatbotService.

ChatbotService:

- Fetches conversation history from RedisService.
- Loads FAQ content from InitialPrompts.FAQ (scraped by HtmlReadService).
- Builds a full prompt and sends it to OllamaChatClient.

AI response is returned to the user and stored in Redis for 30 minutes.

Testing

Instructions: Week 10

Journal

- Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
 - No changes in requirements.
- List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
 - **RedisService**
 - Equivalence Classes:
 - Valid user ID and message
 - Null or missing message (not tested)
 - Expiration time boundaries
 - Boundary Values:
 - Expiration time set exactly to 30 minutes
 - Edge case: empty string for message
 - *Test Paths:*
 - Save history -> append + set expiry
 - Get history -> return string or null
 - Delete history -> confirm deletion
 - Expiration -> confirm history removing after 30 minutes
 - **HtmlReadService**
 - Equivalence Classes:
 - HTML with multiple Q/A entries
 - HTML with no questions
 - HTML with only questions and no answers
 - Boundary Values:
 - No <h3> tags (minimum input)
 - Multiple <h3> tags with mixed children
 - Test Paths:
 - Valid HTML parsing
 - IOException during fetch
 - Empty DOM behavior
 - **ChatbotService**
 - Equivalence Classes:
 - With and without conversation history
 - With and without FAQ data
 - Exception during FAQ refresh

- Boundary Values:
 - Empty strings for user input
 - Very long history string (not tested here)
- Test Paths:
 - Prompt construction (different conditions)
 - History update after response
 - FAQ refresh with success and failure
- Other notes:
 - Mocks are used extensively to isolate behavior. Static method mocking was required for Jsoup.connect().
 - ○ White-box tests cover happy paths, error handling, and control branches. More edge case testing for null/empty inputs could be added later.

Testing Details

<<Use your notes from above to write your test programs and complete this section of the formal documentation by creating a list of your test programs along with descriptions of what they are testing. You will also complete the black-box test plan by running the program and filling in the Actual Results column.>>

Test Program	Description of What It Tests	Actual Results
<code>RedisServiceTest.saveHistory_appendsDataAndSetsExpiry</code>	Verifies that user history is appended and the expiration time is set correctly.	Passed – Data appended and expiry set to 30 min.
<code>RedisServiceTest.saveHistory_expiresAfterThirtyMinutes</code>	Confirms that the expiration time for history is exactly 30 minutes.	Passed – Captured duration matches 30 minutes.
<code>RedisServiceTest.getHistory_returnsValueFromRedis</code>	Tests retrieval of existing conversation history from Redis.	Passed – Returns stored string.
<code>RedisServiceTest.deleteHistory_deletesKey</code>	Checks whether the Redis key is correctly deleted.	Passed – Key deletion verified.
<code>HtmlReadServiceTest.getFaqEntries_happyPath_returnsFormattedEntries</code>	Ensures valid HTML with Q&A is parsed and formatted correctly.	Passed – Output matched expected format.

<code>HtmlReadServiceTest.getFaqEntries_noQuestions_returnsEmptyString</code>	Tests behavior when the HTML contains no questions (no <h3> tags).	Passed – Returned empty string.
<code>HtmlReadServiceTest.getFaqEntries_ioException_bubblesUp</code>	Confirms that <code>IOException</code> during HTML retrieval is thrown up to the caller.	Passed – Exception caught and validated.
<code>ChatbotServiceTest.handleMessageRequest_withHistoryAndFaq_buildsPromptAndUpdatesHistory</code>	Verifies prompt construction with history and FAQ, and correct appending of new exchange.	Passed – Prompt built and history saved.
<code>ChatbotServiceTest.handleMessageRequest_noHistoryAndEmptyFaq_usesSupportSuggestion</code>	Tests fallback logic when there is no history or FAQ—should include fallback support message.	Passed – Suggests support message correctly.
<code>ChatbotServiceTest.refreshSiteContent_success_updatesFaq</code>	Tests successful FAQ refresh and update to global constant.	Passed – <code>InitialPrompts.FAQ</code> updated.

ChatbotServiceTest.refreshSiteContent_failure_throwsRuntimeException	Validates that failed FAQ retrieval throws a runtime exception with proper error message.	Passed – Exception thrown with message.
--	---	---

Presentation

Instructions: Week 12

Preparation

- Give a brief description of your final project
 - We created an AI-powered multilingual chatbot for Gym! Latvia, integrated as a website widget. It automatically responds to FAQs in Latvian, English, and Russian, reducing the support team's workload. Built with Java Spring Boot, Ollama (LLM), Redis, and web technologies, it ensures GDPR compliance and scalability for 6–10k weekly visitors. The goal was to reduce human workload and improve 24/7 customer support.
- Describe your requirement assumptions/additions.
 - The FAQ is the only source of truth (no external APIs used).
 - Only three languages are supported based on the website settings : LV, RU, EN.
 - Chatbot is embedded in Squarespace via a JS widget.
 - Redis handles temporary chat history.
 - Prompt updates are done manually unless CRON is triggered.
- Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
 - Option 1: Modular Services
 - Clear separation (ChatbotService, RedisService, HtmlReadService).
 - Easier to test, debug, and extend.
 - Option 2: Single Controller (Monolithic)
 - Simpler but harder to maintain.

We chose modular design for better scalability and maintainability.
- How did the extension affect your design?
 - Chatbot will only answer pre-defined FAQs from the gym's website.
 - Adding CRON-based dynamic prompt updates required us to create the HtmlReadService, which loads the latest FAQ content. This ensured long-term content consistency.
- Describe your tests (e.g., what you tested, equivalence classes).
 - Through a combination of unit, integration, and manual testing, we ensured that the chatbot functions accurately, responds appropriately in multiple languages, and handles a variety of user inputs and usage patterns. The system proved to be stable, responsive, and aligned with the client's expectations. Our tests validated both the technical implementation and the user experience, giving us confidence that the chatbot is ready for real-world deployment.
- What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
 - Mariana Mechyk
 - Learned how following a structured SDLC (like Waterfall) helps clarify roles and timelines.
 - Understood the importance of gathering complete requirements early to avoid scope creep.
 - Oleksii Pecheniuk
 - Gained hands-on experience with configuration of LLM.
 - Learned how testing Redis session behavior (e.g., auto-expiry) ensures stable conversation flow.

- Mykyta Medvediev
 - Realized that modular services (ChatbotService, RedisService, etc.) simplify debugging and maintenance.
 - Practiced applying OOP principles to keep logic clean and components reusable.
- Ruslan Dzhubuev
 - Learned how backend logging is critical for error tracing and production stability.
 - Understood how to chain prompts effectively when integrating with LLMs like Ollama.
- Ayomide Akintola
 - Discovered that Squarespace, while convenient, is not ideal for dynamic apps platform limitations can block features.
 - Learned to balance between technical feasibility and customer constraints in architecture decisions.
- Agboola Peter
 - Understood the value of version control and CI planning for maintaining a growing codebase.
 - Redis proved to be a lightweight and efficient solution for managing short-term per-user memory.
- What functionalities are you going to demo?
 - Ask a valid question - AI answers
 - Change language - Chatbot adapts
 - Ask irrelevant questions - fallback message
 - Chat history is temporarily stored using Redis (auto-delete after 30 minutes)
- Who is going to speak about each portion of your presentation? (Recall: Each group will have ten minutes to present their work; minimum length of group presentation is seven minutes. Each student must present for at least two minutes of the presentation.)

○

Team Members	Topic
Mariana Mechyk	Testing
Oleksii Pecheniuk	Technical Architecture and Implementation
Mykyta Medvediev	Introduction and Motivation
Ruslan Dzhubuev	Conclusion and Achievements, Demo Overview
Ayomide Akintola	Future Work and Improvements
Agboola Peter	Requirements Gathering and Analysis

- Other notes:
 - <<Insert notes>>

<<Use your notes from above to complete create your slides and plan your presentation and demo.>>