Faculty of Engineering of the University of Porto

Master in Informatics and Computing Engineering

Software Systems Architecture

Questions Management Tool

Homework 03

Team 32

José Francisco Veiga up202108753@up.pt

Marco Vilas Boas up202108774@up.pt

Pedro Lima up202108806@up.pt

Pedro Januário up202108768@up.pt

Pedro Marcelino up202108754@up.pt

March 2025

Part 1

Below are the questions our group developed during the class:

- How should we prioritize each question?
- Should we have different metrics to evaluate question priority?
- Who are the decision makers and how are they chosen?
- Can the question maker reject the answer?
- Can the question maker continue the conversation after receiving an answer?

- Should we use AI to try to help answer the questions asked?
- Should everyone have access to all questions or are they restricted?
- Can we direct/tag a person/team to answer the question?
- Which fields in a question should be queryable in order to retrieve past questions?
- What is the question backlog and how is it presented?
- Can the user choose to follow an unanswered question in order to receive a notification when it is answered?

- Are ‘Developers / Researchers’ and the ‘Decision Makers’ group completely separate or can they have common people?
- Why do we need both ‘Developers / Researchers’ and the ‘Decision Makers’ groups?
- Can this architecture be simplified?
- When a question is not resolved, should its priority be reevaluated and go again through research?
- Can the user choose that AI answers their question?
- Can the user ask AI to read the question before submission in order to find duplicates?

Part 2

Introduction

In the first part of the assignment, we developed a series of so-called *natural questions* aimed at clarifying the requirements and functionalities of a system that manages architectural questions and knowledge. Those *natural questions* address aspects such as prioritization of questions (the ones that the system will handle, not to be confused with our *natural questions*), access control, the role of AI, user interaction, and the handling of unanswered or duplicate content.

Using those *natural questions* as a starting point, in the second part, we define the system's architecture. Our architecture follows a layered structure with a clear separation of concerns, ensuring modularity and scalability while maintaining security and access control. The system is designed to support efficient question and answer management, with modules for question categorization, AI-assisted recommendations, user authentication, and notifications.

The following sections provide the details of the system's architecture, including its key modules and interactions between them, as well as possible usage scenarios that illustrate how the system would behave in such cases.

General Vision of the System

Our architecture follows a layered structure with separation of concerns. It aims at modularity and scalability while ensuring security and access control to every aspect of the system.

It starts with a frontend responsible for showing the UI to the user and that communicates with the business layer, specifically with the Question Management and Categorization Module (QMCM) for asking or answering questions addressed to a specific user or with the Authorization and Authentication module in the middleware for user profile and permissions related issues, which in turn communicates with the user repository on the data layer, where user data on registration or user profiles changes are persisted or retrieved from for each login and permission check.

The QMCM is responsible for handling the entire process of creating, editing or deleting answers and questions. It does so by communicating with the Q&A repository in the data layer and with the help of the LLM recommendation module, responsible for sorting and tagging questions and answers while merging duplicate ones resorting to an LLM. Each user request to the QMCM is always synchronously validated by the authentication & Authorization module.

The last module the business layer includes is the Notification Module. When the QMCM finishes a user-related action, it triggers an asynchronous event, the Notification Module is then responsible for handling this event by pushing a notification to the frontend UI in order to alert the user for the need to answer a question or to read the new answer they had required.

This architecture ensures efficient questions and answers by keeping essential processes synchronous and optional modules asynchronous like notifications to avoid delays while maintaining full functionality. Its modularity also allows easy expansion and addition of new independent modules that can introduce new features, such as analytics & logs for user statistics or even a questions search engine to further enhance the user's experience or to integrate with existing systems.

The diagram below shows the final architecture. It is composed of blocks representing the layers with various components inside represented as boxes, the boxes have different sizes, proportional to their importance and responsibility for each layer, for instance the QMCM being the core of the business layer is larger than the Notification Module whose objective is secondary. The lines in the diagram represent interactions and dependencies between different system components. Each line represents a communication flow, indicating how data or requests travel between modules. Their direction reflects whether the flow is unidirectional (one-way communication) or bidirectional (mutual interaction). This communication can be either synchronous or asynchronous, represented by full and dashed lines respectively.

Note that Authentication & Authorization is within a package called middleware, this is because it acts as an intermediary in each request, this is done so, we can separate the modules based on behaviour and later add new ones that fit this description, allowing for better project expansion and system maintenance.

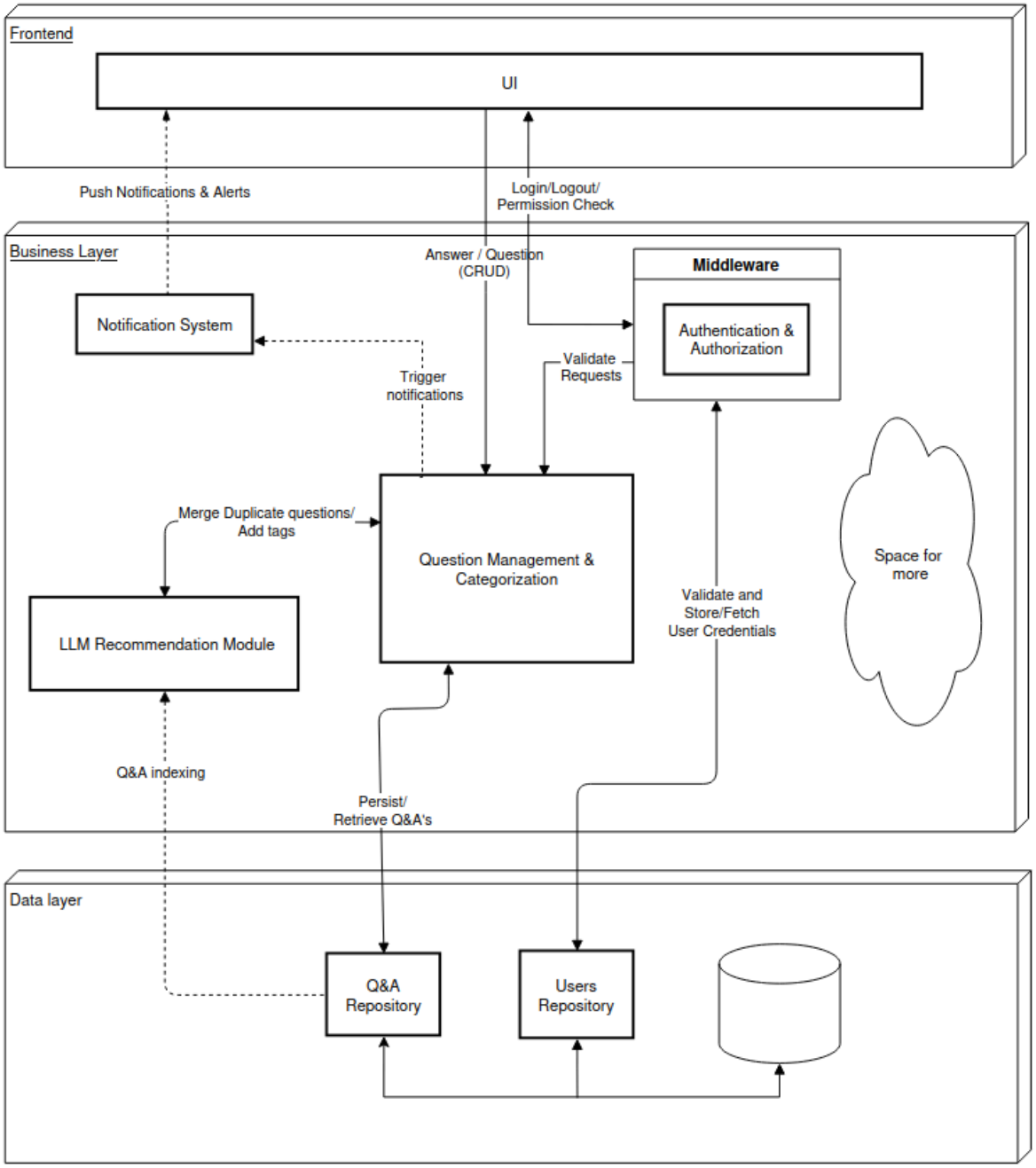

Fig 1. System architecture diagram

Example Scenarios

Scenario 1 - A user logs in and wants to see the list of open questions they asked

Consider an instance of the system we described. A user logs in and wants to check the status of the questions they previously submitted. As they access their personal dashboard in the frontend module, a request is sent to the question management module, which retrieves the list of open (unanswered) questions associated with their account.

After passing through the Authentication and Authorization module to ensure the user has the necessary permissions, the list is displayed instantly on their screen.

This integration between modules allows users to efficiently track their pending inquiries, eliminating the need to manually search for them and ensuring they stay informed about the status of their requests.

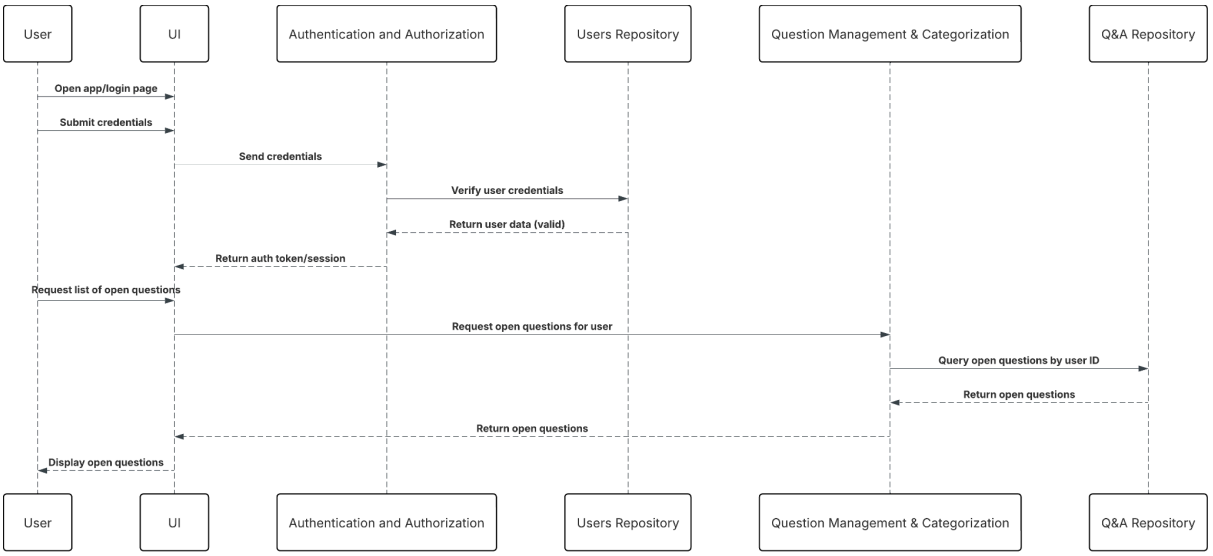

Fig 2. Scenario 1 Sequence diagram

Scenario 2 - A user asks a question that is a duplicate of an old one.

In this scenario, let's say that the system is being used by a company Y to answer general, day to day questions. The user, who is already logged in the system, creates a new question that asks about how they should proceed to impute an expense related to a company event. This question, written in the frontend module, will go through the question management and categorization module first, where, together with the LLM module, it detects that this question is a duplicate from an already existing (and answered) one in the database. After going through the Authentication and Authorization module to verify that this information can in fact be retrieved by the user, the answer is sent back to the user frontend.

As we can see by this scenario, the clever design and modularity of the architecture allows this question to be answered instantly, instead of consuming some time and effort by another colleague of the user having to answer it!

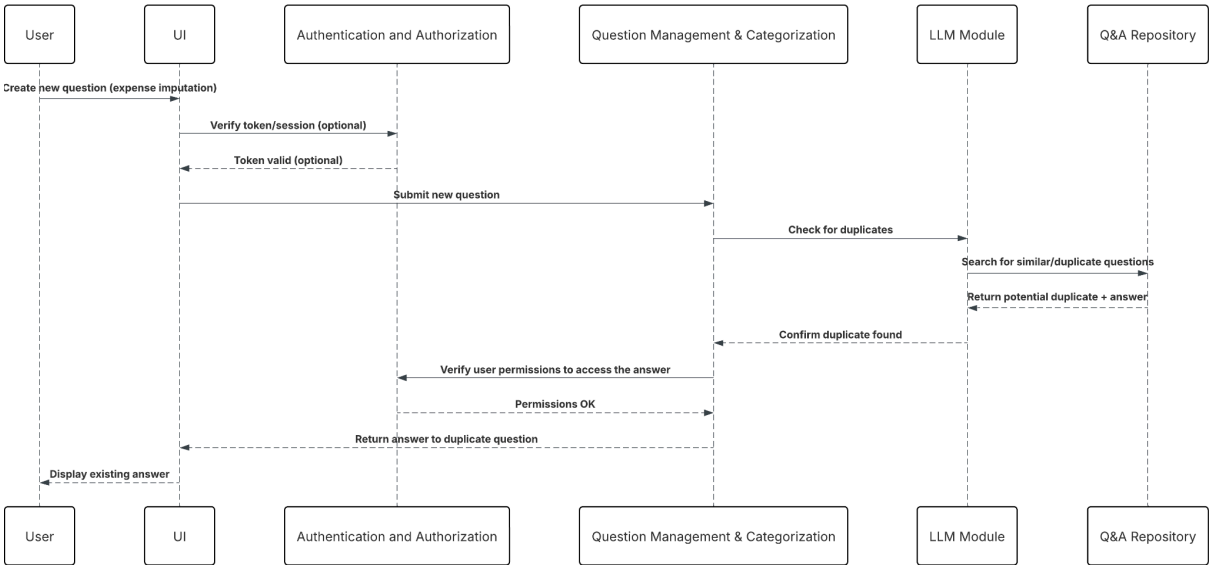

Fig 3. Scenario 2 Sequence diagram

Scenario 3 - A user sees an unanswered question in the public forum that they also don't know the answer to.

For the sake of simplicity, let's take the same system and company Y from the previous scenario. The user, who just logged in, sees the following open (unanswered) question in the public forum: *Can I buy and sell Y's stocks freely as a Y employee?*

The user, who is starting to invest in the stock market, realizes that he also does not know if he is legally allowed to buy or sell stocks of the company he works at. But the question is still unanswered, so he clicks the 'Follow' button in the question. After a few hours, another colleague from the legal team answers the question, and the user receives a notification with the answer.

Using the notification module in the system allows for the instant, seamless and effortless distribution of information in this type of scenario, providing a positive user experience and improving the efficacy of the communication between colleagues.

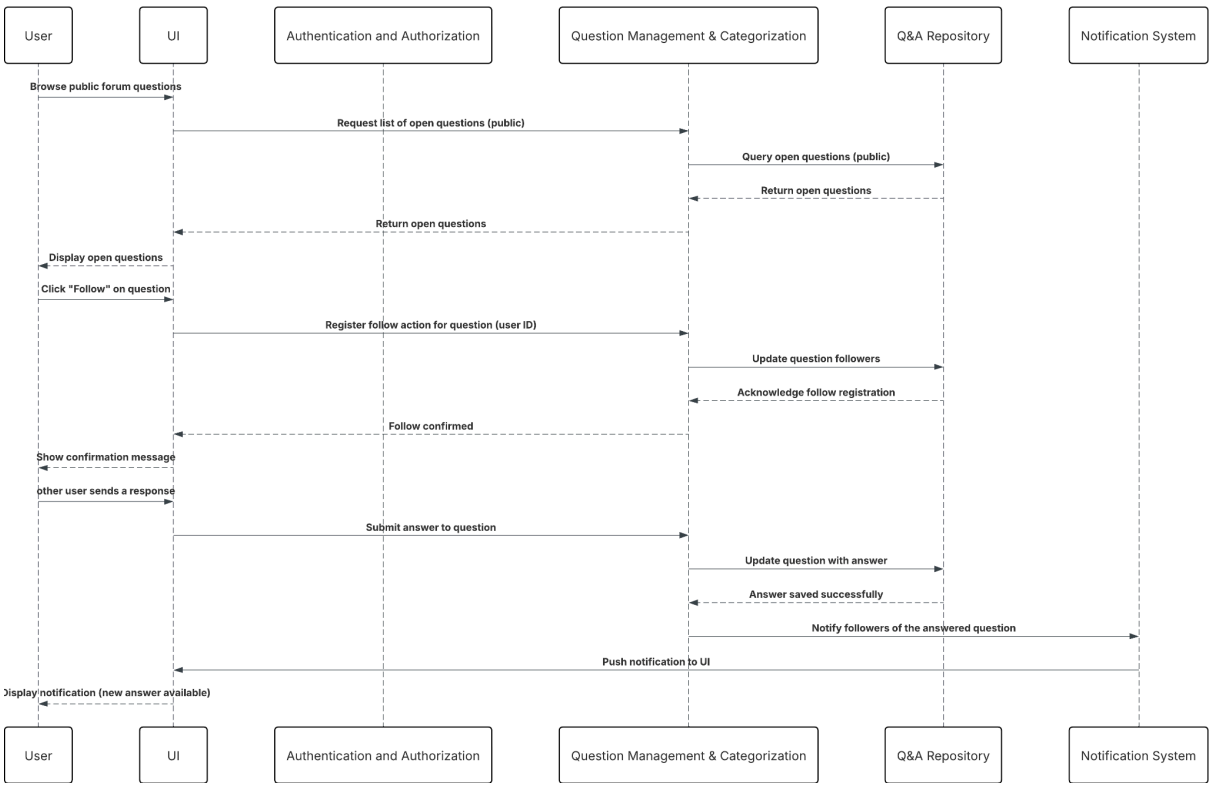

Fig 4. Scenario 3 Sequence diagram

How we dealt with ambiguity

When designing our system, we encountered several ambiguities in the specification. This is common in real-world projects, where not everything is clearly defined from the start. To move forward, we made some assumptions and set guiding principles to ensure the system would work effectively. Below, we explain our reasoning for resolving some of these ambiguities.

1. Access control for questions

It was unclear whether all questions should be publicly accessible or restricted to specific users. To maintain flexibility, we assumed that questions could be tagged with access controls, allowing the Question Owner to decide whether a question is private (restricted to specific teams) or public (visible to everyone). We decided that some questions should be posted in public forums, making them available to the whole company, while others should only be visible to a restricted group of people. This access control is enforced by the Authentication & Authorization module, ensuring that only authorized users can view or contribute to restricted discussions.

2. Who is allowed to answer questions

This answer seems simple, the Decisions-Makers, but who are they? We assumed that anyone capable of seeing the question should be allowed to answer it, therefore we allowed for some overlap between developers/researchers and decision makers, maintaining flexibility within project teams.

3. Questions' priority

We assumed that, while answering, a user can set a priority, but it is up to the person who is answering them to follow them or not, making priorities purely indicative. We also considered implementing different metrics, such as urgency, relevance, and impact, but left the final decision open for future refinement.

4. Handling Unanswered Questions

In scenarios where a user encounters an unanswered question in a public forum and is also seeking the same information, we designed a simple yet effective solution. Users can follow an open question to receive a notification when it gets answered. The "Notification System" is responsible for this.

5. Role of AI in Answering Questions

We considered whether AI should automatically answer questions or only assist in organizing them. Since AI-generated answers may not always be reliable, we decided to use AI mainly for recommendations, such as merging duplicate questions and tagging them for better searchability.

6. Querying Past Questions

To ensure knowledge preservation, we assumed that past questions should be searchable by full text search, within the "Question Management & Categorization Module". This helps users find relevant information without re-asking similar questions.

7. Scalability and Future Modifications

Given all the uncertainties, we designed our architecture with modularity in mind. By keeping different components separate, such as authentication, question management, and notifications, we ensured that new features or changes could be added without requiring a complete system overhaul.

Conclusion

The system architecture we designed provides a modular approach to the management and storing of questions and answers efficiently. By structuring the system with distinct modules, we ensure that interactions provide a friendly user experience and an efficient flow of information. The integration of AI in the system also allows to optimize and speed up some tasks, while reducing human effort at the same time.

Additionally, like with most system requirements, the specification of the wanted system was not very clear, and the requirements not clearly stated, which means that we had to make some assumptions, decisions and design choices that, in a real world scenario, could potentially go against the product that the client had in mind. Ambiguity is a natural part of system design, but by making these assumptions, we ensured that our system was functional without waiting for perfect clarity.