Mario Ullan Motellon

GESTIONE QUESTIONARI

Analysis and Design Report

---

1. Introduction

This document describes the analysis and design of the web application Gestione Questionari, developed using Spring Boot.

The system allows administrators to create and manage questionnaires and questions, and allows users (not registered) to fill questionnaires, save drafts, submit final versions and access them through a unique access code.

The application follows a layered architecture based on MVC and applies several software engineering principles.

---

2. Requirements Analysis

2.1 System Description

The system supports two types of actors:

- Administrator
- User (anonymous)

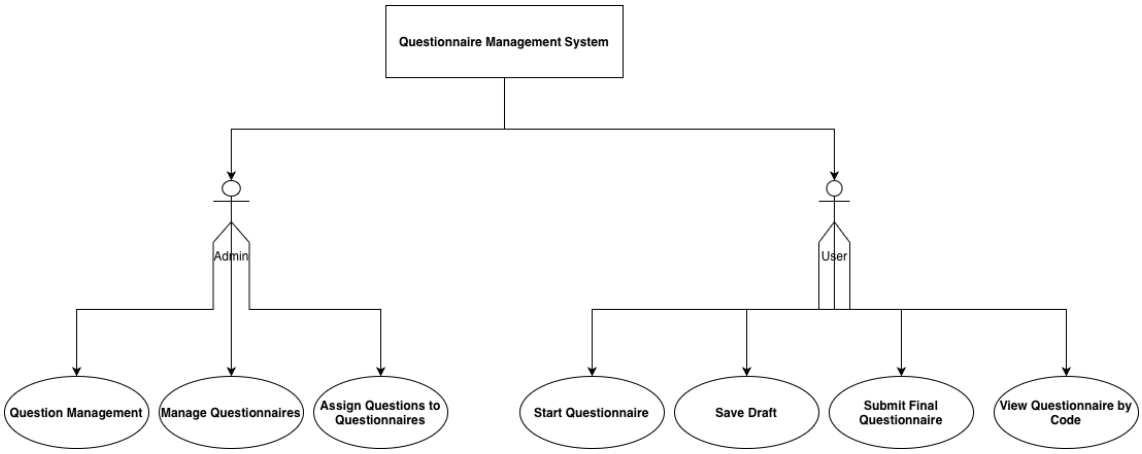
The Administrator can:

- Create, edit and delete questions
- Create and manage questionnaires
- Assign questions to questionnaires

The User can:

- Start filling a questionnaire
- Provide email to create a draft
- Save draft answers
- Submit final questionnaire
- Search submission by access code
- Delete submission

2.2 Use Case Diagram



The use case diagram represents the interaction between the actors and the system.

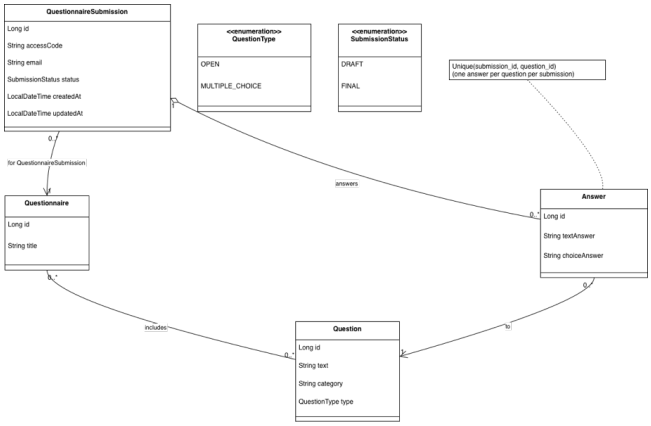
Key use cases:

- Manage Questions
- Manage Questionnaires
- Start Questionnaire
- Save Draft
- Submit Final
- Search Submission
- Delete Submission

---

3. Domain Model

3.1 Domain Class Diagram



The main domain classes are:

- Question
- Questionnaire
- QuestionnaireSubmission
- Answer

The system supports:

- Many-to-many relationship between Questionnaire and Question.
- One submission per user access code.
- One answer per question per submission.

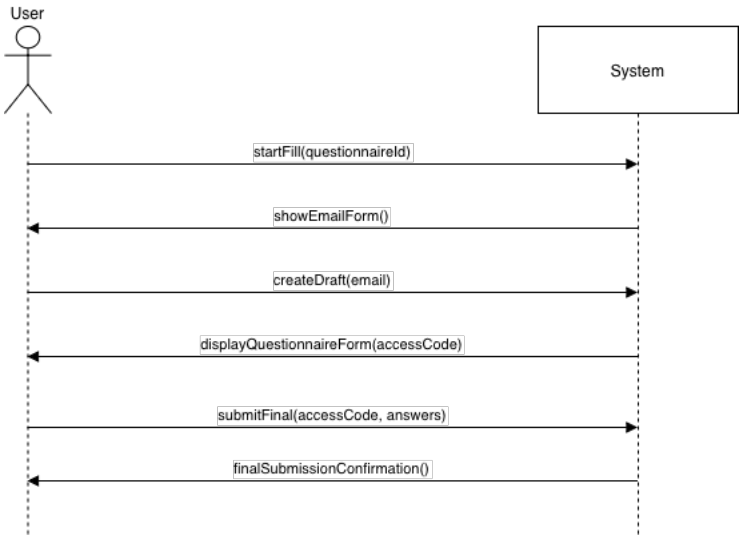
Enumerations:

- QuestionType (OPEN, MULTIPLE_CHOICE)
- SubmissionStatus (DRAFT, FINAL)

---

4. System Sequence Diagrams (SSD)

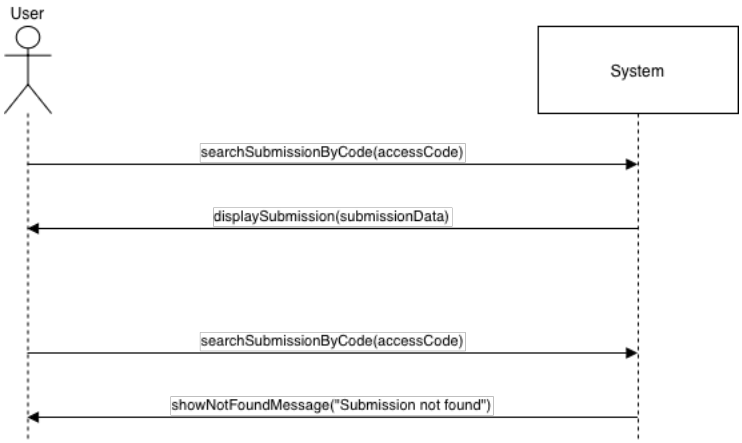
4.1 Compile Questionnaire SSD



This SSD represents the interaction between the User and the System when compiling a questionnaire.

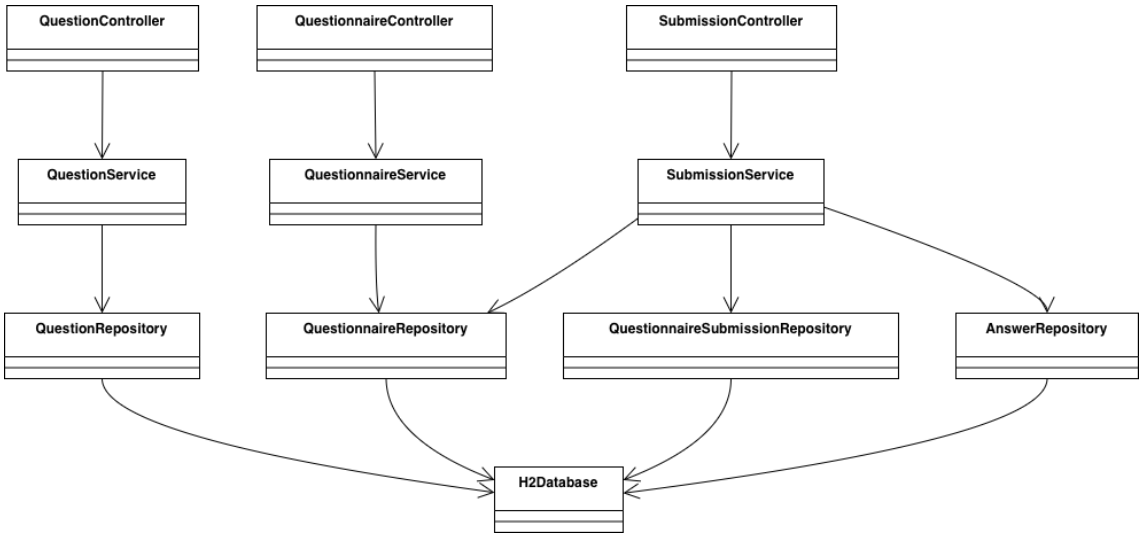
The system is modeled as a black box.

4.2 Search Submission SSD



This SSD describes how the user searches a submission by access code and handles both found and not found cases.

5. Software Architecture



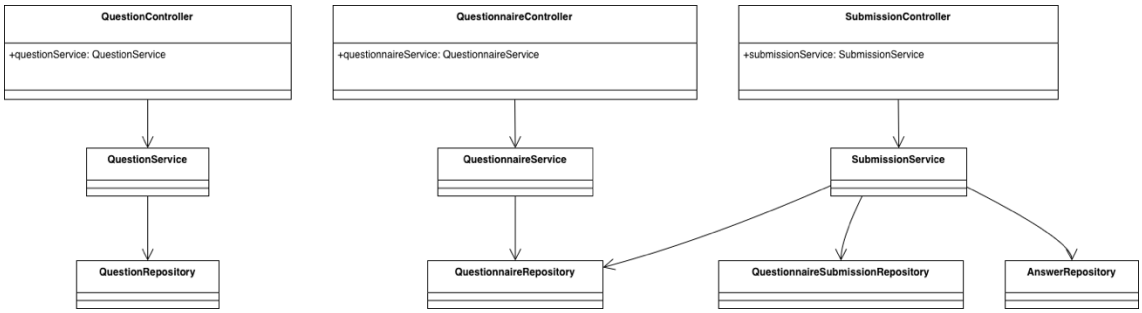
The system follows a layered architecture:

- Controller Layer (MVC Controllers)
- Service Layer (Business logic)
- Repository Layer (Spring Data JPA)
- H2 Database

The architecture separates responsibilities and follows the Separation of Concerns principle.

---

6. Design Class Diagram



The design-level diagram includes:

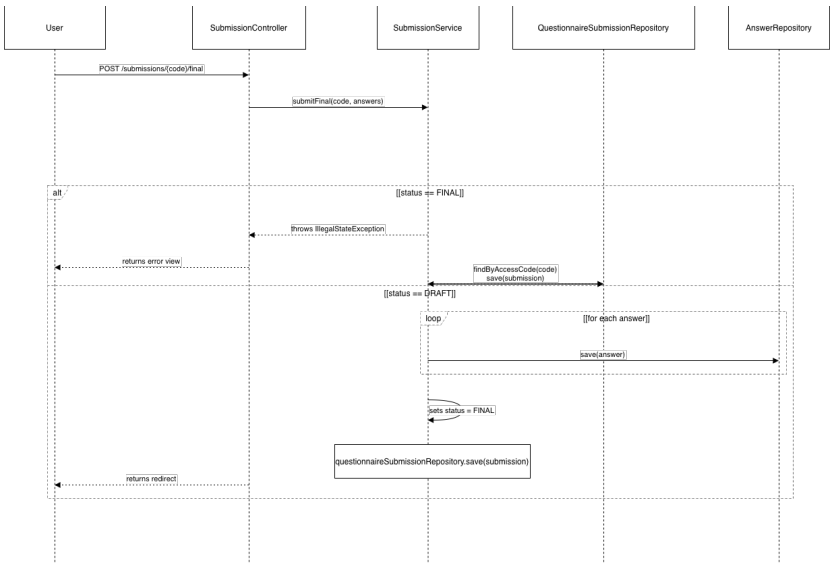
- QuestionController
- QuestionnaireController
- SubmissionController
- Services
- Repositories

Dependencies follow a downward direction:

Controller → Service → Repository

---

7. Design Sequence Diagram



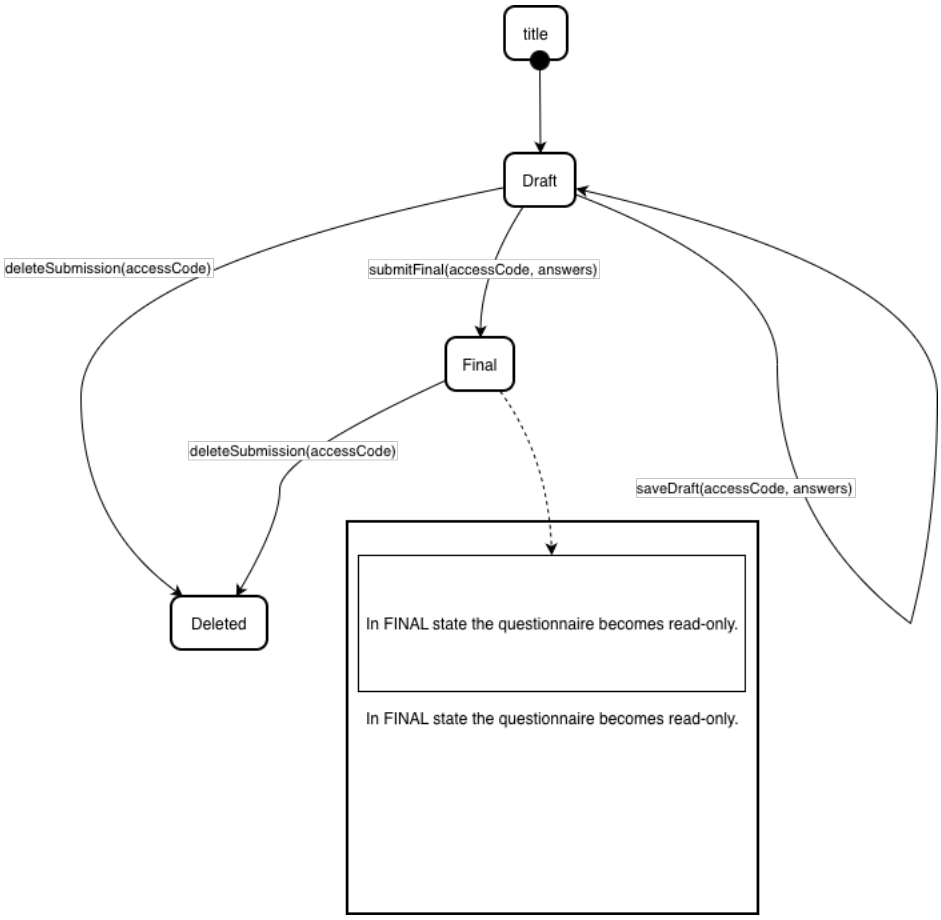
This diagram shows the internal interaction between layers when submitting a final questionnaire.

It includes:

- Validation logic
- Status check
- Repository persistence
- Error handling

---

8. State Diagram



The lifecycle of QuestionnaireSubmission includes:

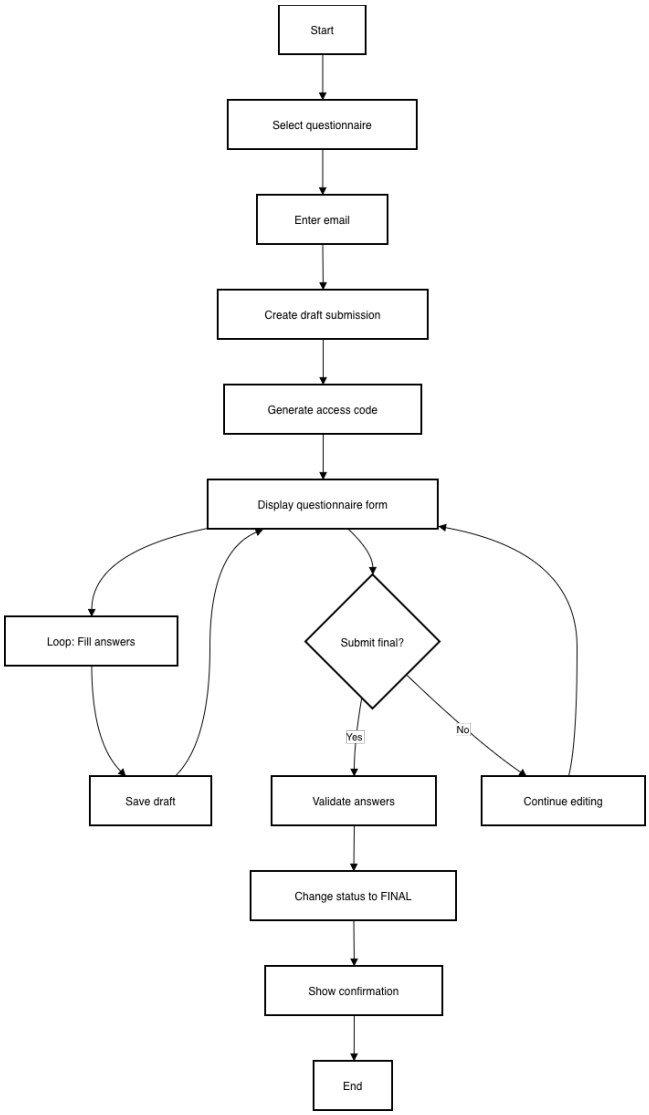
- Draft
- Final
- Deleted

Rules:

- From Draft → Final
- From Draft → Deleted
- From Final → Deleted
- Final state is read-only

---

9. Activity Diagram



The activity diagram represents the full workflow of filling a questionnaire:

- Select questionnaire
- Enter email
- Generate access code
- Fill answers
- Save draft (loop)

- Submit final
- Change status
- Show confirmation

---

10. Design Patterns Applied

The following patterns are applied:

MVC Pattern

Used to separate presentation, business logic and data access.

Repository Pattern

Used to abstract persistence logic through Spring Data JPA repositories.

Service Layer Pattern

Encapsulates business logic between controller and repository layers.

---

11. Design Principles Applied

The project applies the following principles:

- **Single Responsibility Principle:** Each class has a single responsibility.

- **Open/Closed Principle:** Business logic can be extended without modifying existing core structures.

- **Dependency Inversion Principle:** Controllers depend on abstractions (services) instead of concrete implementations.

- **Separation of Concerns:** Layers are clearly separated.

---

12. Code Quality and Analysis

The project was analyzed using SonarCloud. Minor issues related to accessibility were detected and fixed. The system does not show major structural antipatterns.

---

13. Conclusions

The system has been designed following software engineering best practices. The layered architecture ensures maintainability and scalability. UML diagrams were used to support both analysis and design phases.