



## INGEGNERIA DEL SOFTWARE - TRIENNALE INFORMATICA

PROGETTO ESAME

---

# Web Surveys

---

### **Studenti :**

Polonia LUCA  
Salami ILARIA  
Enobo FRANCK  
Sereno LEONARDO

### **Professori :**

Arcelli Fontana FRANCESCA  
Riganelli OLIVIERO

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Descrizione Progetto . . . . .	3
1.2	Tecnologie Usate . . . . .	3
<b>2</b>	<b>Diagrammi UML</b>	<b>4</b>
2.1	Diagramma dei Casi d'Uso . . . . .	4
2.2	Diagramma delle Classi di Dominio . . . . .	5
2.3	Diagramma dell'Architettura Software . . . . .	6
2.4	Diagramma delle Classi a livello di Progettazione . . . . .	7
2.4.1	Model . . . . .	7
2.4.2	View . . . . .	8
2.4.3	Controller . . . . .	9
2.4.4	Services . . . . .	10
2.4.5	Completo . . . . .	11
2.5	Diagramma SSD . . . . .	12
2.6	Diagrammi SD . . . . .	13
2.7	Diagrammi delle Macchine a Stati . . . . .	14
2.8	Diagrammi delle Attività . . . . .	15
2.9	Diagramma ER . . . . .	16
<b>3</b>	<b>Organizzazione del Progetto</b>	<b>17</b>
3.1	Diagramma di Gantt . . . . .	17
3.1.1	Diagramma Iniziale . . . . .	17
3.1.2	Diagramma Finale . . . . .	18
3.2	Ideazione . . . . .	19
3.3	Metodologia di Sviluppo . . . . .	20
<b>4</b>	<b>Implementazione</b>	<b>22</b>
4.1	Gestione Repository . . . . .	22
4.2	Gestione CI/CD . . . . .	22
4.2.1	Github Actions . . . . .	22
<b>5</b>	<b>Documentazione Backend</b>	<b>23</b>
5.1	Controller . . . . .	23
5.2	Autenticazione . . . . .	23
5.3	Servizi . . . . .	23
5.4	Persistence . . . . .	24
<b>6</b>	<b>Documentazione Frontend</b>	<b>25</b>

6.1	Model . . . . .	25
6.2	Pages . . . . .	25
6.3	Components . . . . .	25
6.4	Servizi . . . . .	25
<b>7</b>	<b>Pattern</b>	<b>26</b>
7.1	Design Pattern . . . . .	26
7.1.1	DTO (Data Transfer Object) . . . . .	26
7.1.2	Builder . . . . .	26
7.1.3	Repository . . . . .	26
7.1.4	Singleton . . . . .	27
7.2	Pattern Architetturali . . . . .	27
7.2.1	MVC (Model View Controller) . . . . .	27
<b>8</b>	<b>Analisi</b>	<b>28</b>
8.1	Sonar Cloud . . . . .	28
8.2	Understand . . . . .	28
8.2.1	Dependency Graph . . . . .	28
8.2.2	TreeMap Complexity Metrics . . . . .	29
<b>9</b>	<b>READ ME</b>	<b>31</b>
9.1	Tecnologie . . . . .	31
9.2	Installazione . . . . .	31
9.2.1	Clonare il repository . . . . .	31
9.2.2	Creazione del database in MySQL . . . . .	31
9.2.3	Avvio di XAMPP . . . . .	31
9.2.4	Backend . . . . .	31
9.2.5	Frontend . . . . .	32
9.3	Test Applicazione . . . . .	32

# Capitolo 1

## Introduzione

### 1.1 Descrizione Progetto

L'obiettivo del progetto è sviluppare un'applicazione Web per la gestione di questionari su vari argomenti. L'applicazione consente la creazione, archiviazione, ricerca e compilazione di questionari, offrendo un'interfaccia intuitiva per utenti registrati e non.

Le principali funzionalità includono:

- **Gestione delle domande:** creazione di domande testuali o con immagini, con risposte aperte (limitate in caratteri) o chiuse (scelte multiple), categorizzazione e archiviazione nel database.
- **Gestione dei questionari:** creazione di questionari basati su domande archiviate, modifica, cancellazione e ricerca avanzata tramite parole chiave o codice identificativo.
- **Compilazione e salvataggio:** gli utenti possono compilare questionari con possibilità di salvataggio temporaneo e finale. I questionari completati vengono salvati nel database e possono essere esportati in PDF con notifica via email.
- **Accesso utenti:** gli utenti registrati possono gestire i propri questionari compilati, mentre gli utenti non registrati ricevono un codice univoco per accedere, modificare o cancellare il proprio questionario.

L'applicazione fornisce un'interfaccia web intuitiva e responsive per la gestione e la compilazione dei questionari, garantendo un'esperienza utente fluida ed efficace.

### 1.2 Tecnologie Usate

L'applicazione è stata sviluppata utilizzando un'architettura full-stack basata su tecnologie moderne per garantire scalabilità, efficienza e manutenibilità.

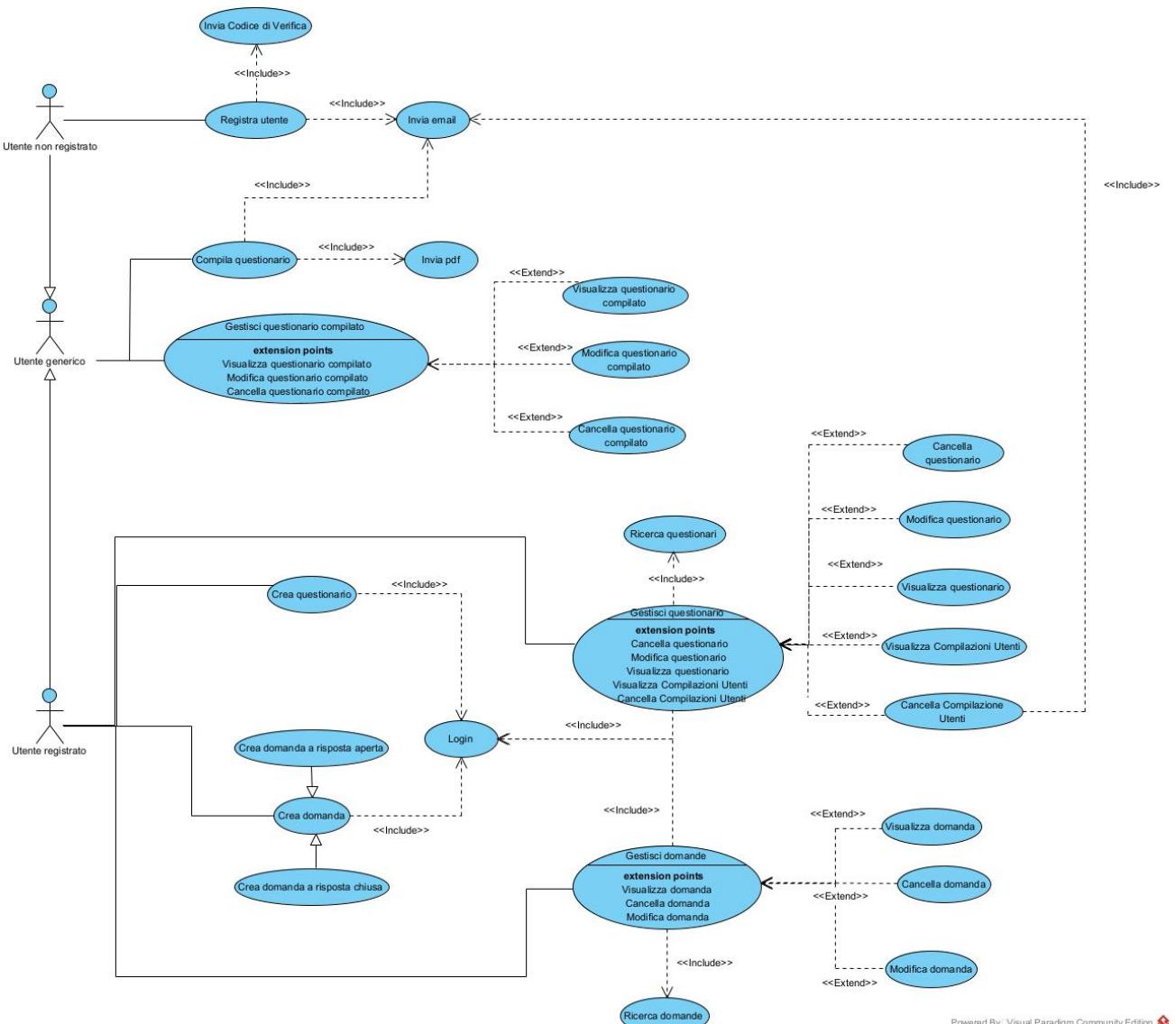
- **Frontend:** sviluppato con React, utilizzando HTML, JavaScript e lo stile gestito tramite Tailwind CSS per un'interfaccia reattiva e intuitiva.
- **Backend:** implementato con Spring Boot in Java, sfruttando Hibernate per la gestione della persistenza dei dati.
- **Database:** basato su SQL, con gestione tramite ORM Hibernate per garantire un'integrazione efficiente con il backend.

# Capitolo 2

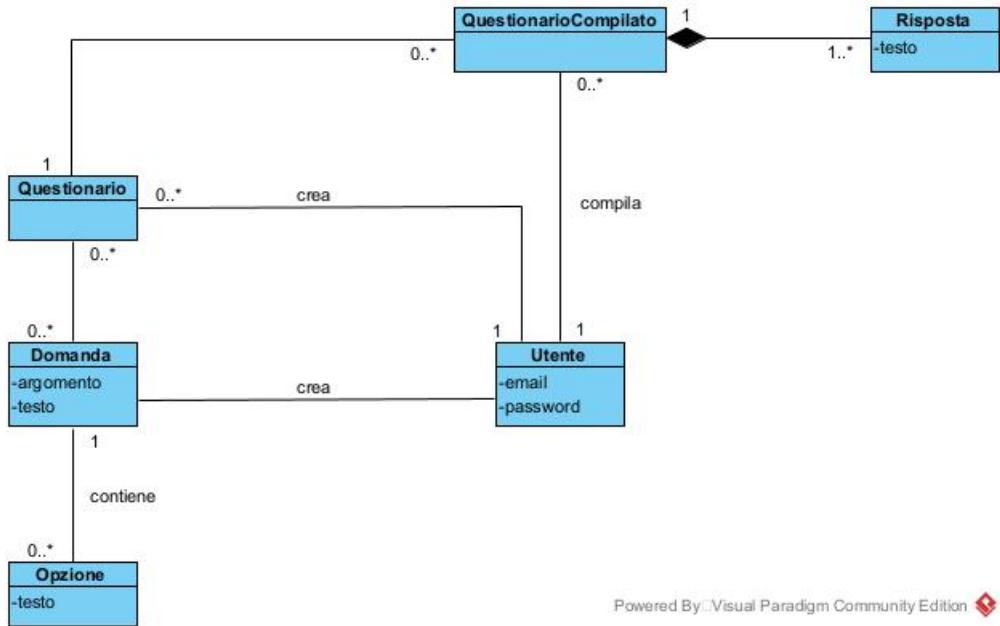
## Diagrammi UML

### 2.1 Diagramma dei Casi d'Uso

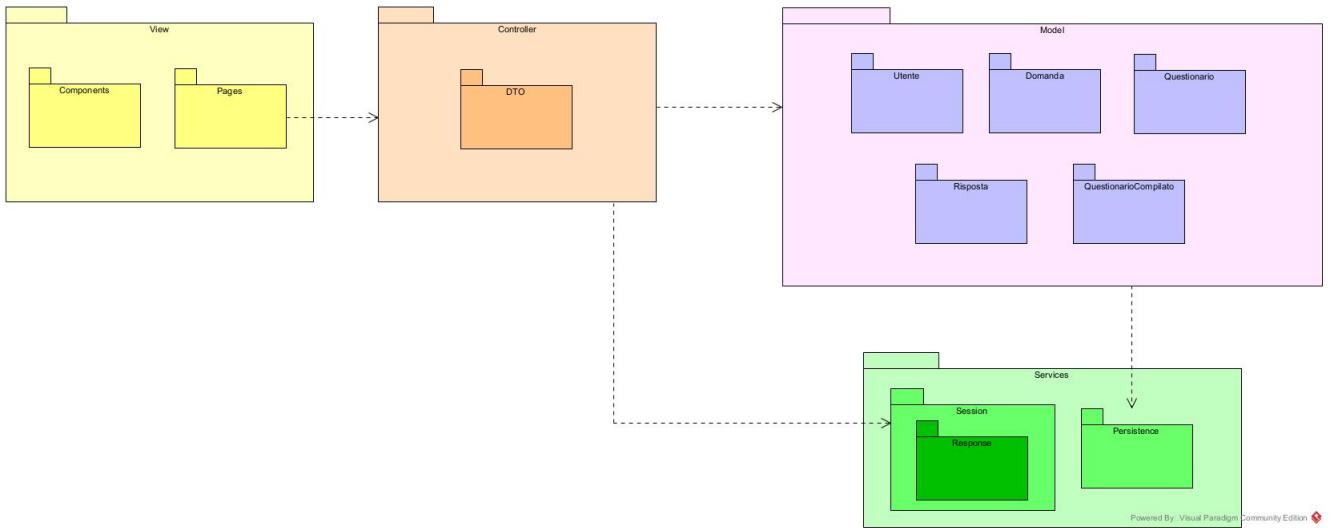
Link Google Drive con cartella con tutte le foto presenti nella relazione e il file di visual paradigm [https://drive.google.com/drive/folders/1CXdbUMelZnqai-AvXRk1Ko3Wv8UkJ4Z5?usp=drive\\_link](https://drive.google.com/drive/folders/1CXdbUMelZnqai-AvXRk1Ko3Wv8UkJ4Z5?usp=drive_link)



## 2.2 Diagramma delle Classi di Dominio

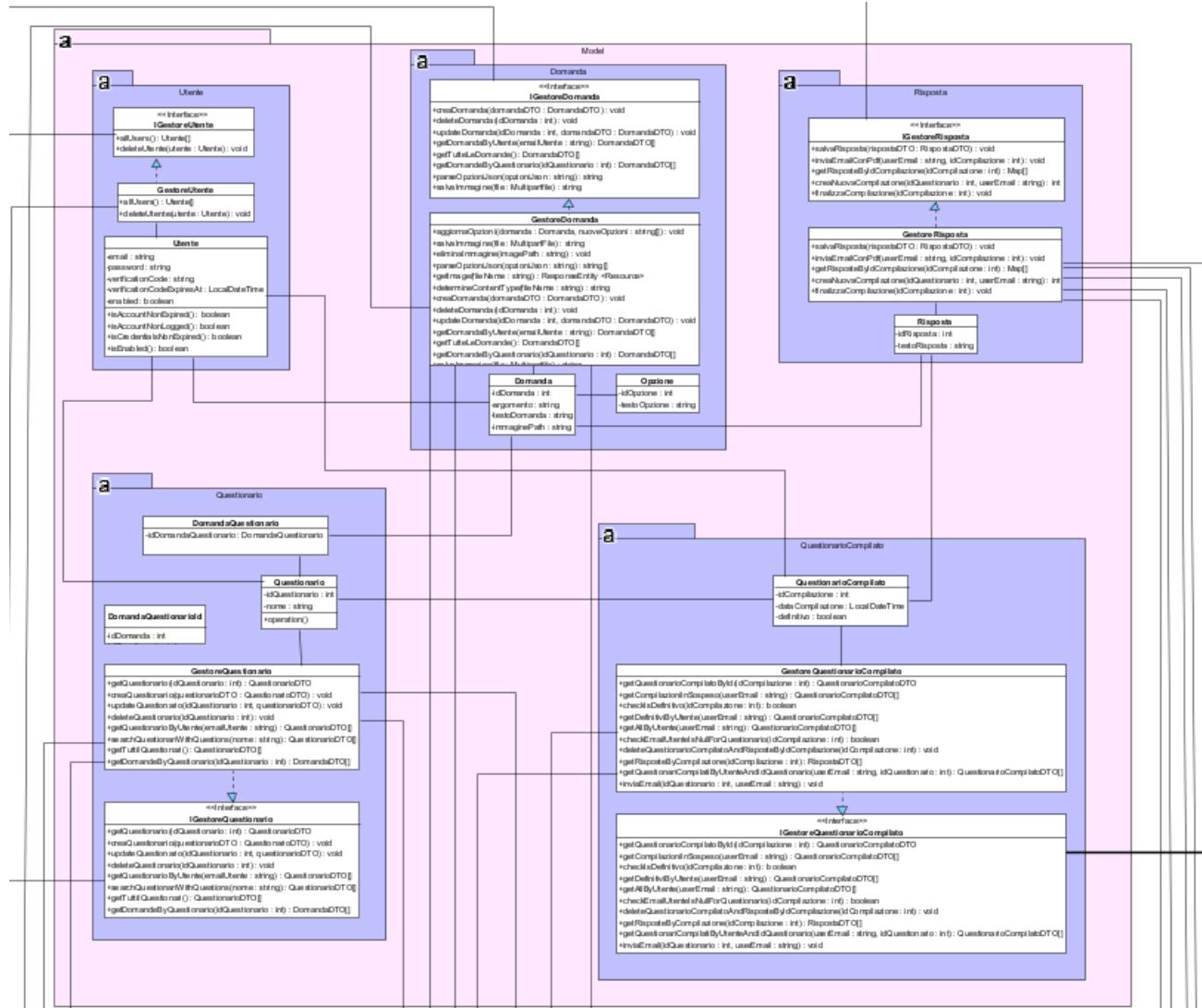


## 2.3 Diagramma dell'Architettura Software

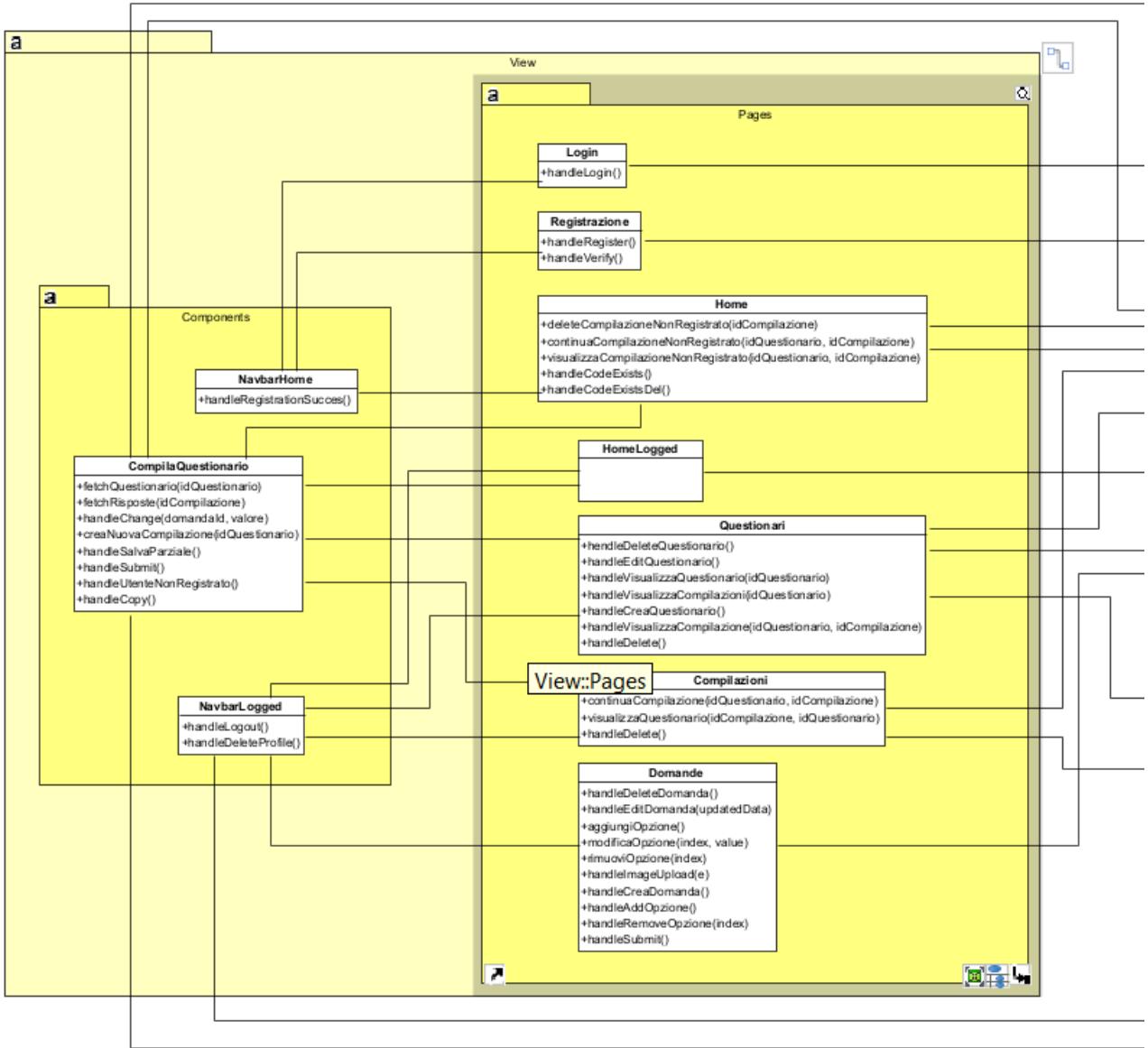


## 2.4 Diagramma delle Classi a livello di Progettazione

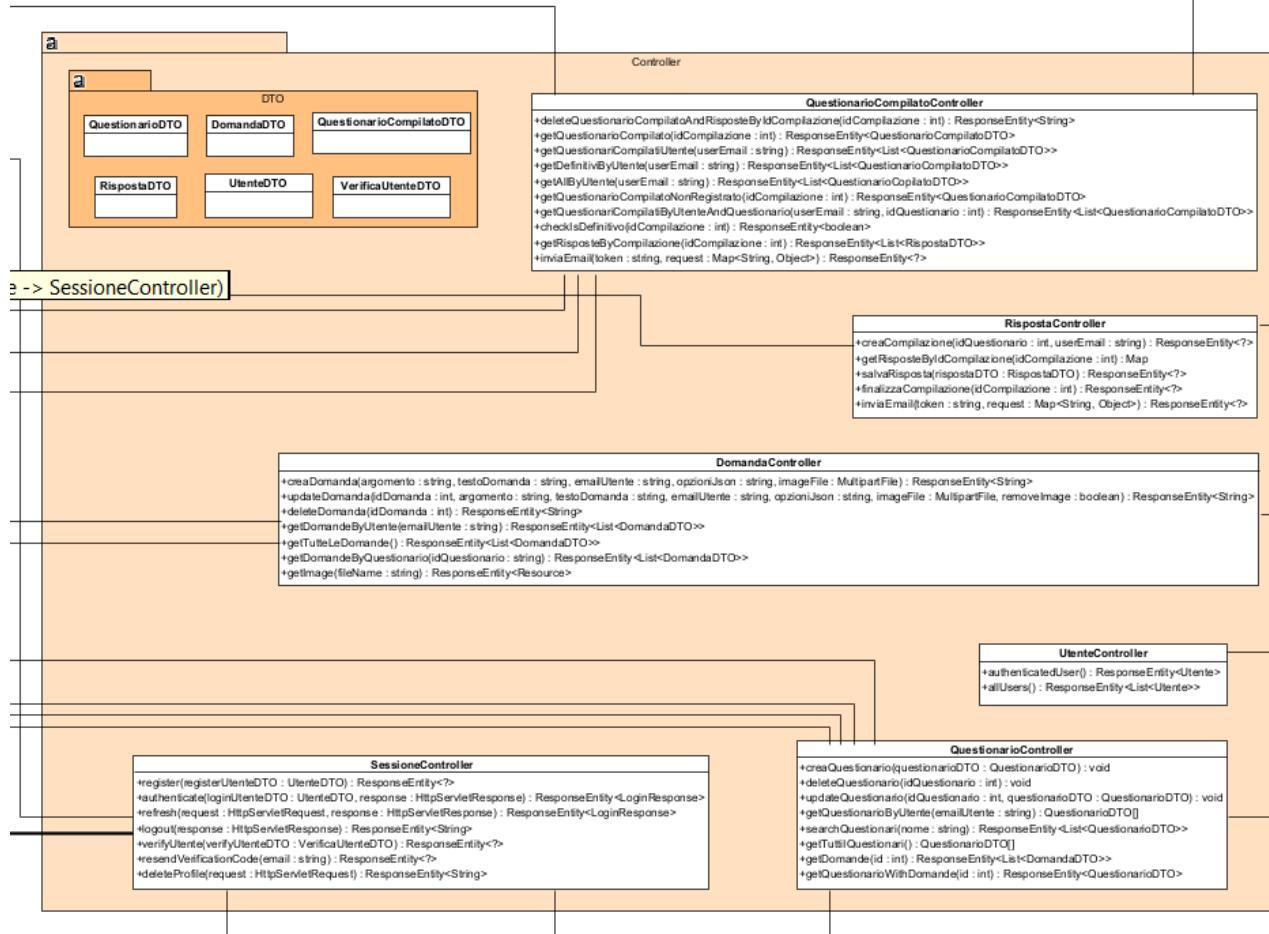
### 2.4.1 Model



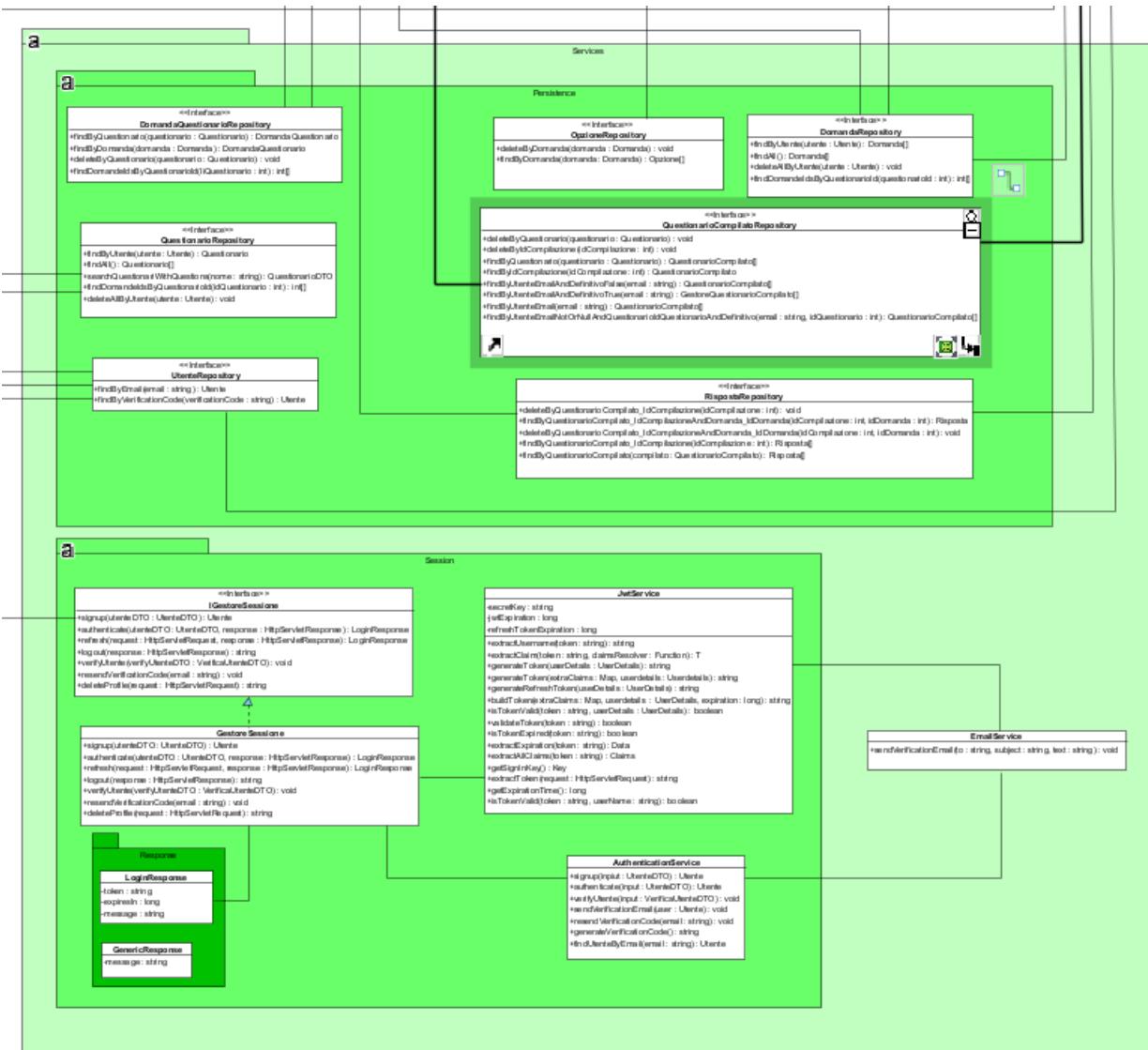
## 2.4.2 View



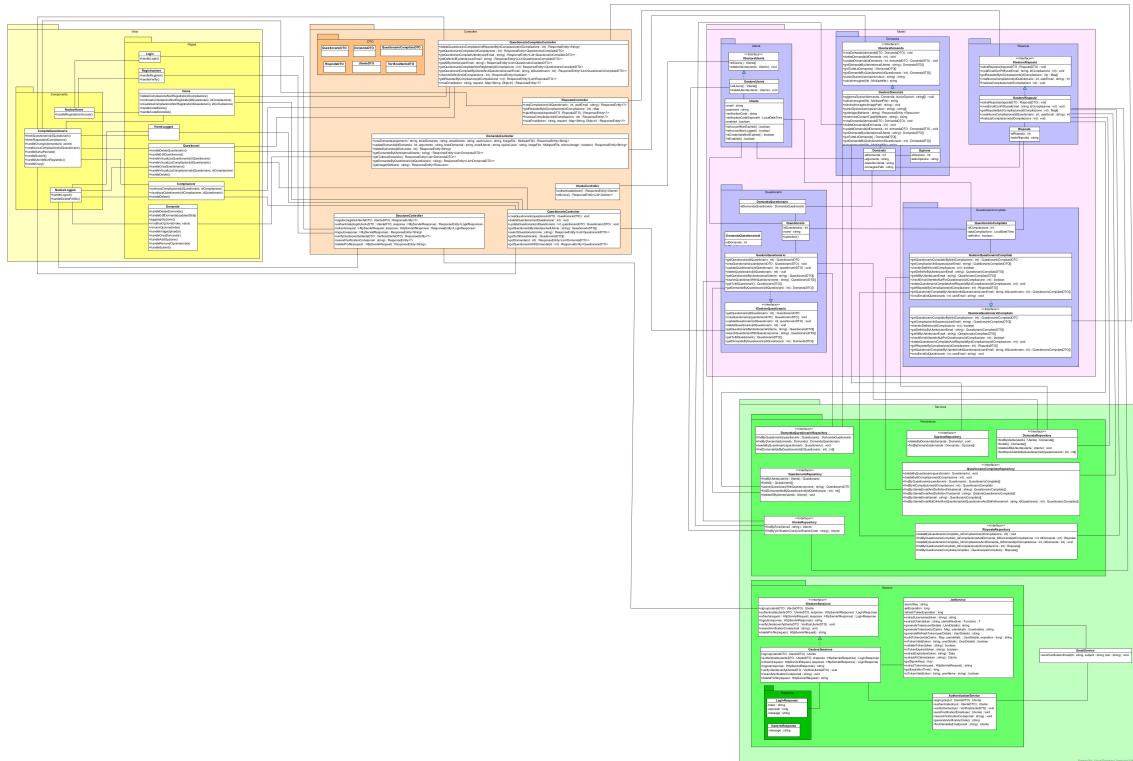
### 2.4.3 Controller



#### 2.4.4 Services



## 2.4.5 Completo



## 2.5 Diagramma SSD

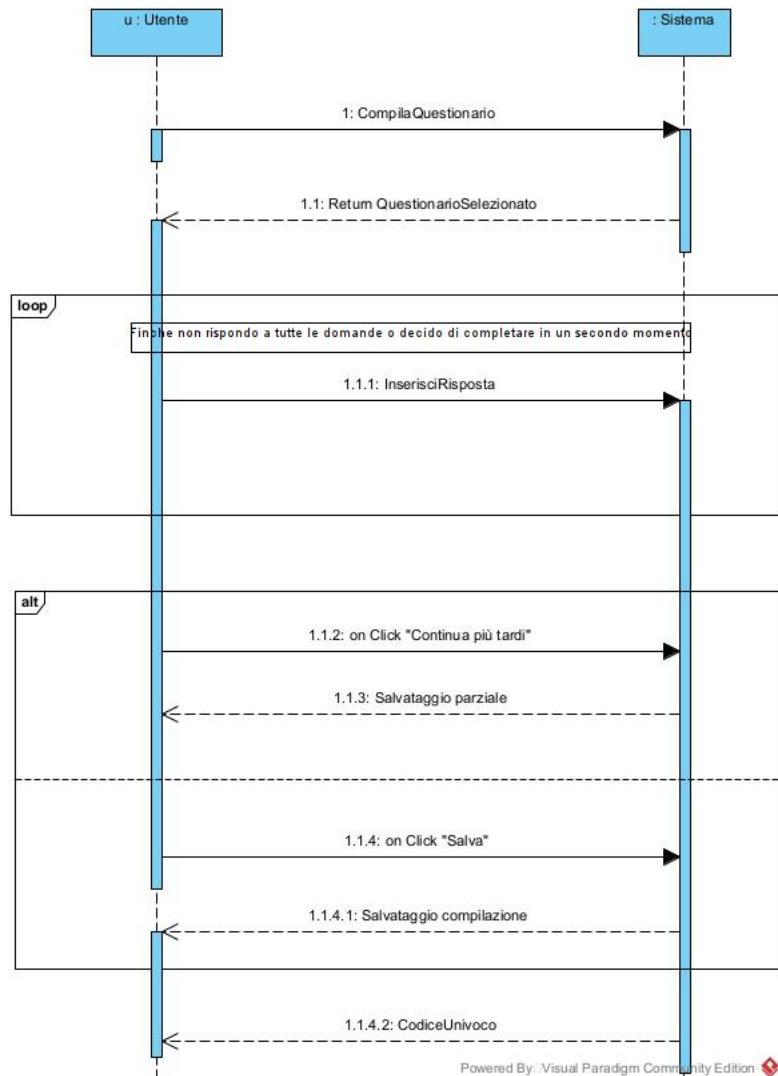


Figura 2.1: Compilazione questionario utente non registrato.

## 2.6 Diagrammi SD

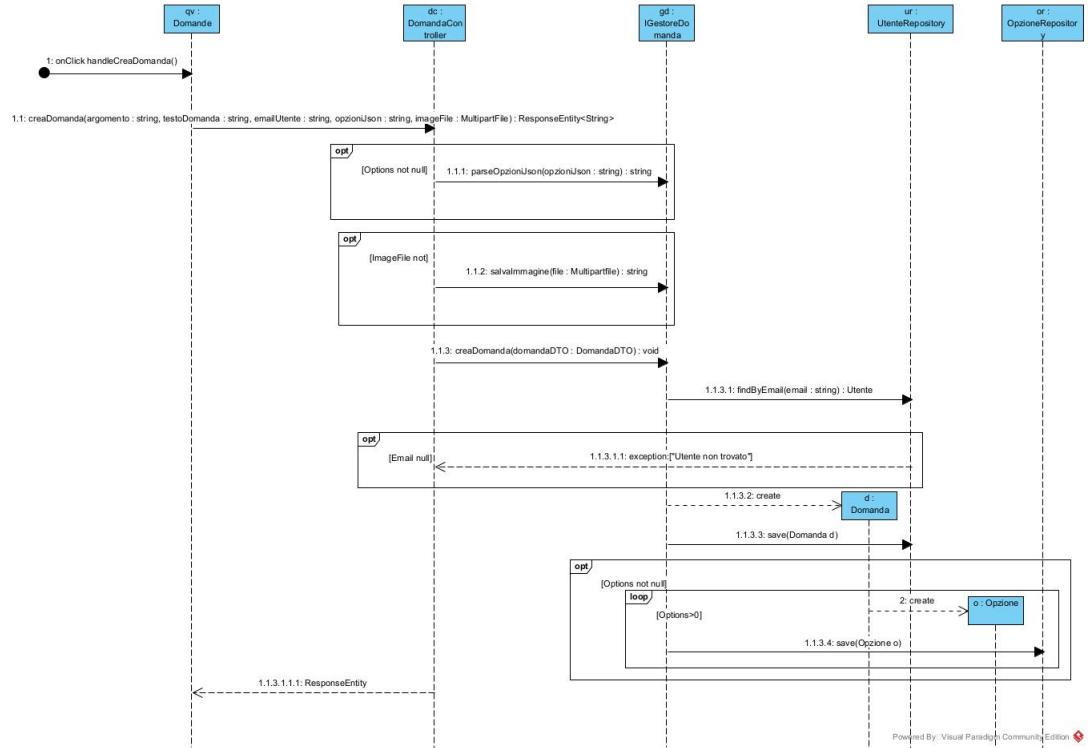


Figura 2.2: salvataggio domanda.

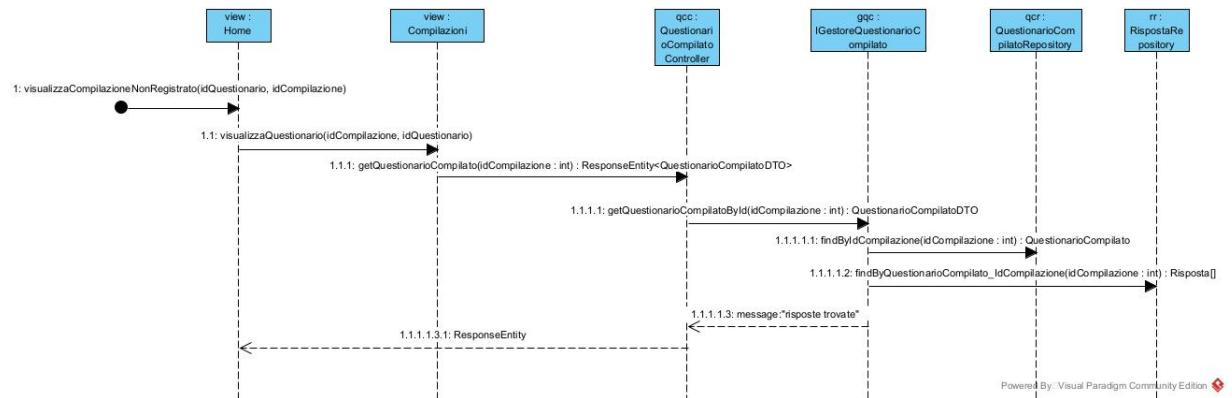


Figura 2.3: visualizzazione di un questionario compilato per un utente non loggato.

## 2.7 Diagrammi delle Macchine a Stati

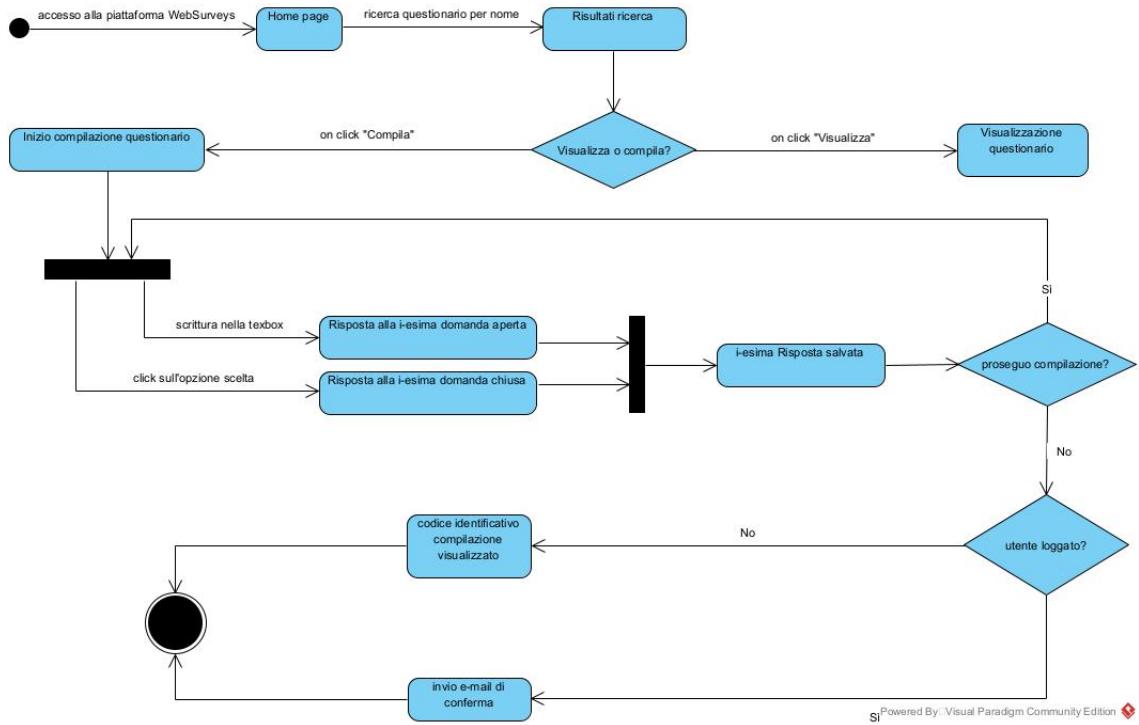


Figura 2.4: Compilazione questionario.

## 2.8 Diagrammi delle Attività

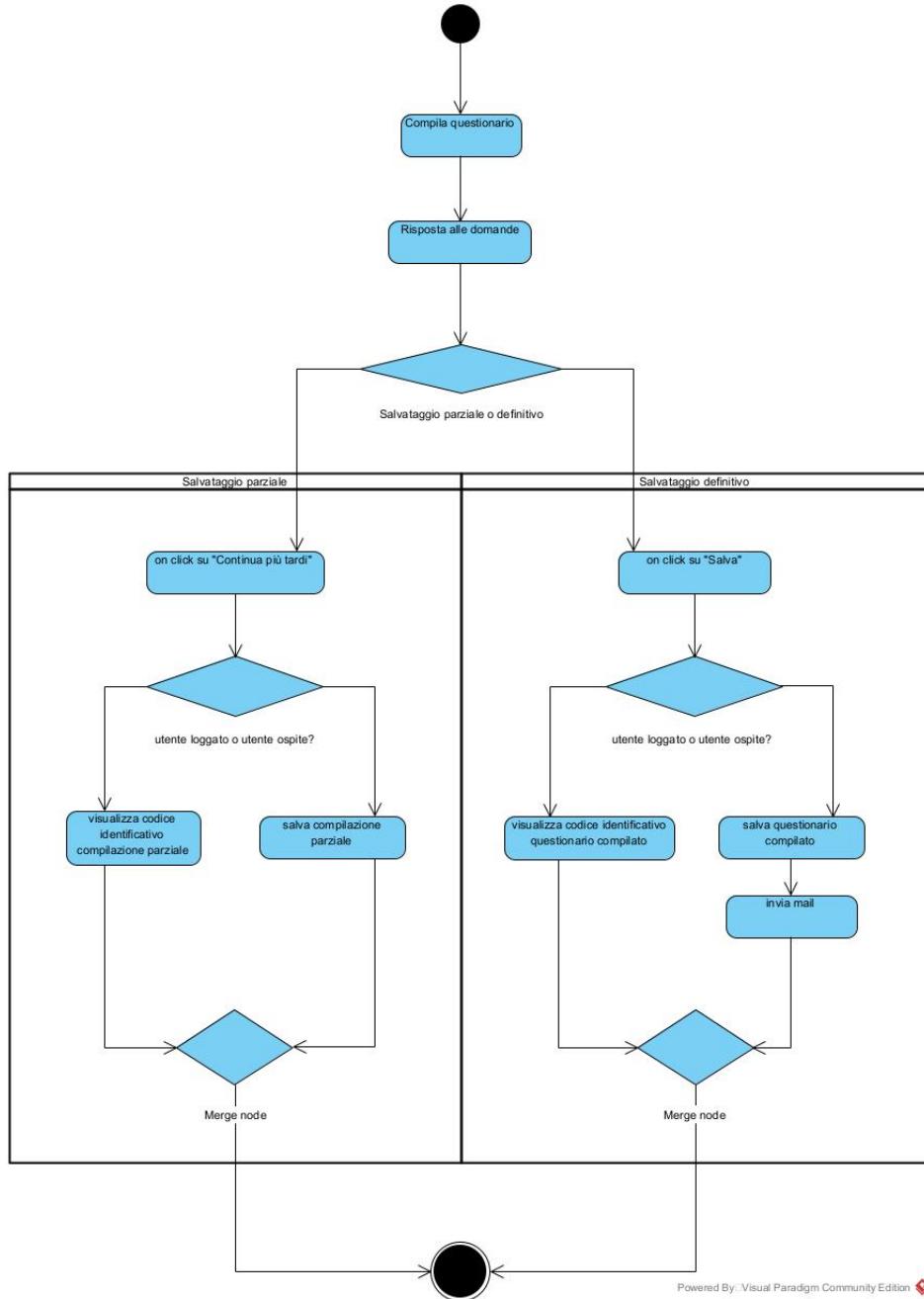
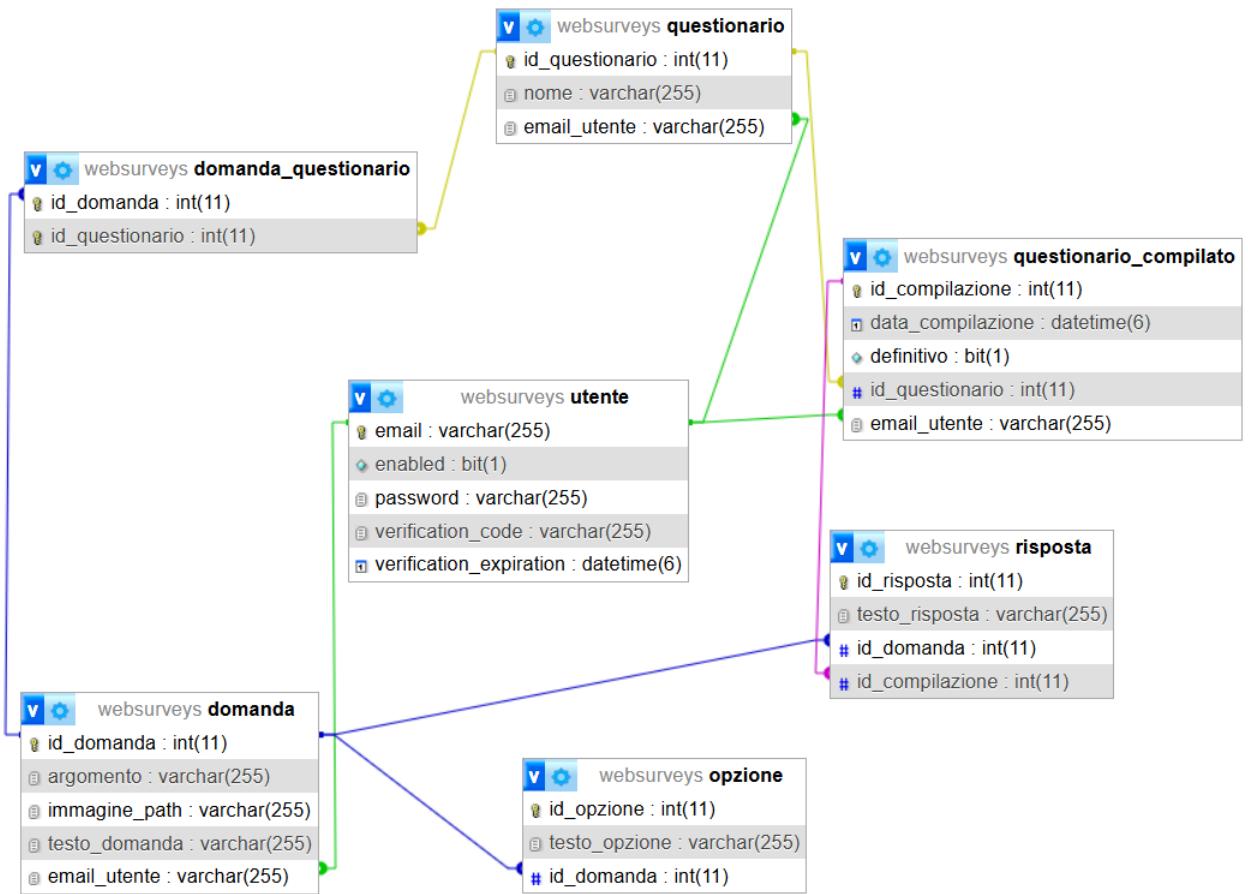


Figura 2.5: Compilazione questionario.

## 2.9 Diagramma ER



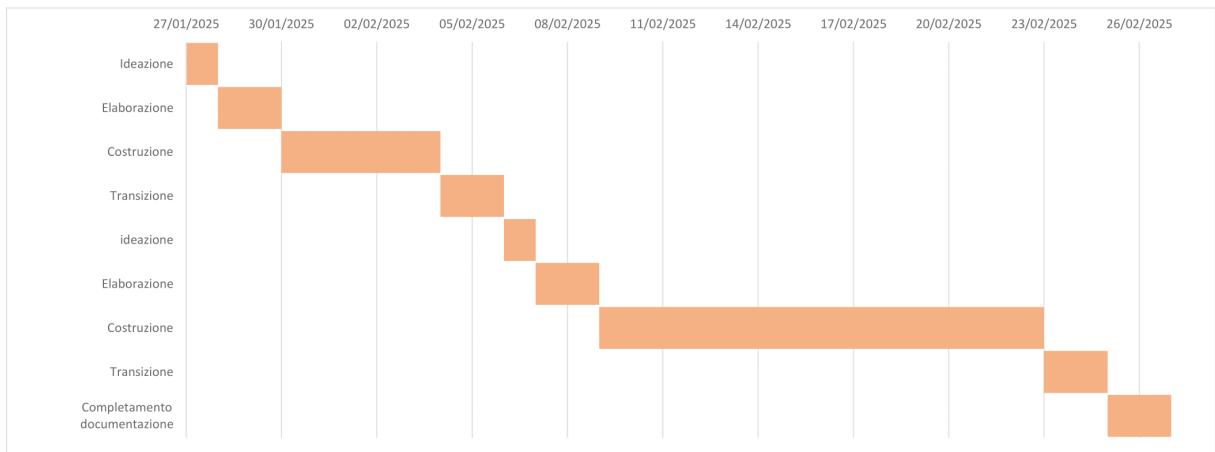
# Capitolo 3

## Organizzazione del Progetto

### 3.1 Diagramma di Gantt

#### 3.1.1 Diagramma Iniziale

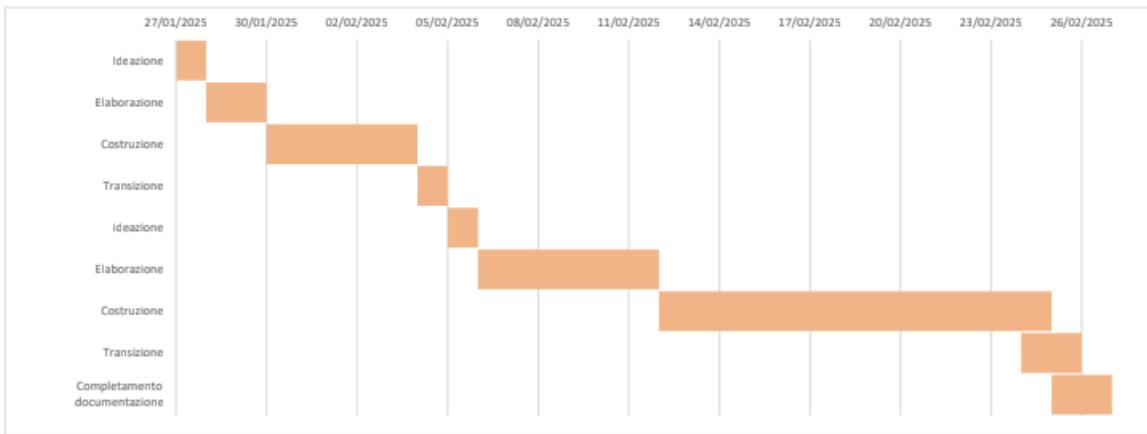
Fasi	Data Inizio	Data Fine	durata	note
Ideazione	27/01/2025	28/01/2025	1	Avvio del progetto, visione approssimativa, stime approssimative dei tempi
Elaborazione	28/01/2025	30/01/2025	2	Realizzazione del nucleo dell'architettura, identificazione di gran parte dei requisiti
Costruzione	30/01/2025	04/02/2025	5	Realizzazione delle capacità operative iniziali
Transizione	04/02/2025	06/02/2025	2	Test del nucleo dell'architettura
ideazione	06/02/2025	07/02/2025	1	Discussione e studio del progetto dettagliata
Elaborazione	07/02/2025	09/02/2025	2	Continuazione dell'architettura, conclusione della definizione dei requisiti
Costruzione	09/02/2025	23/02/2025	14	Continuazione dell'implementazione fino alla conclusione
Transizione	23/02/2025	25/02/2025	2	Revisione della correttezza e completezza del progetto
Completamento documentazione	25/02/2025	27/02/2025	2	



### 3.1.2 Diagramma Finale

Fasi	Data Inizio	Data Fine	durata	note
Ideazione	27/01/2025	28/01/2025	1	Avvio del progetto, visione approssimativa, stime approssimazione dei tempi
Elaborazione	28/01/2025	30/01/2025	2	Realizzazione del nucleo dell'architettura, identificazione di gran parte dei requisiti
Costruzione	30/01/2025	04/02/2025	5	Realizzazione delle capacità operative iniziali
Transizione	04/02/2025	05/02/2025	1	Test del nucleo dell'architettura
ideazione	05/02/2025	06/02/2025	1	Discussione e studio del progetto dettagliata
Elaborazione	06/02/2025	12/02/2025	6	Continuazione dell'architettura, conclusione della definizione dei requisiti
Costruzione	12/02/2025	25/02/2025	13	Continuazione dell'implementazione fino alla conclusione
Transizione	24/02/2025	26/02/2025	2	Revisione della correttezza e completezza del progetto
Completamento documentazione	25/02/2025	27/02/2025	2	

Iter  
1  
2



## 3.2 Ideazione

Nella fase iniziale del progetto, è stato utilizzato Figma per delineare in modo preliminare l'interfaccia utente, al fine di ottenere una visione generale delle funzionalità necessarie per lo sviluppo della piattaforma. Di seguito, abbiamo riportato alcune schermate.

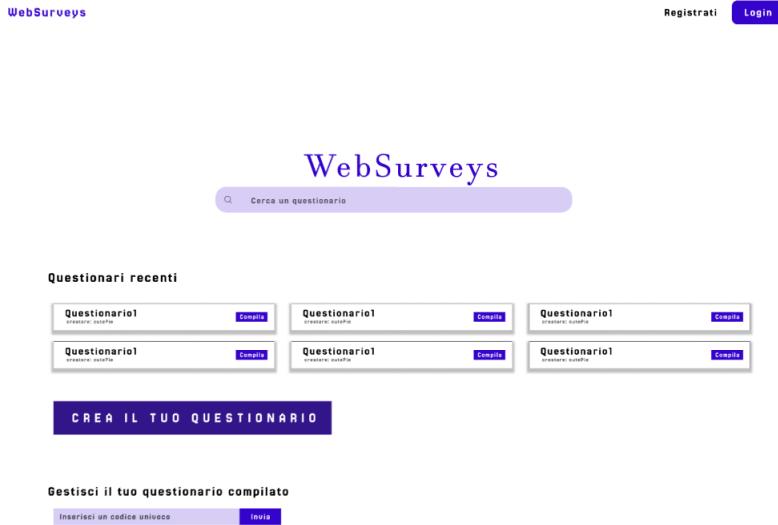


Figura 3.1: Schermata iniziale della piattaforma.

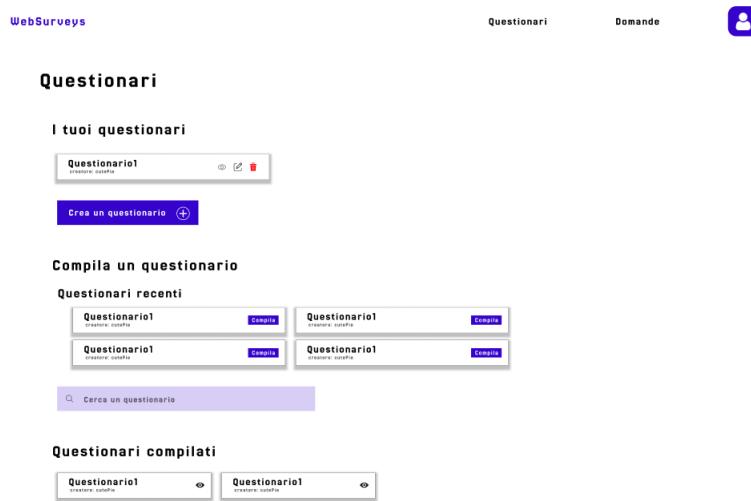


Figura 3.2: Sezione dedicata ai questionari.

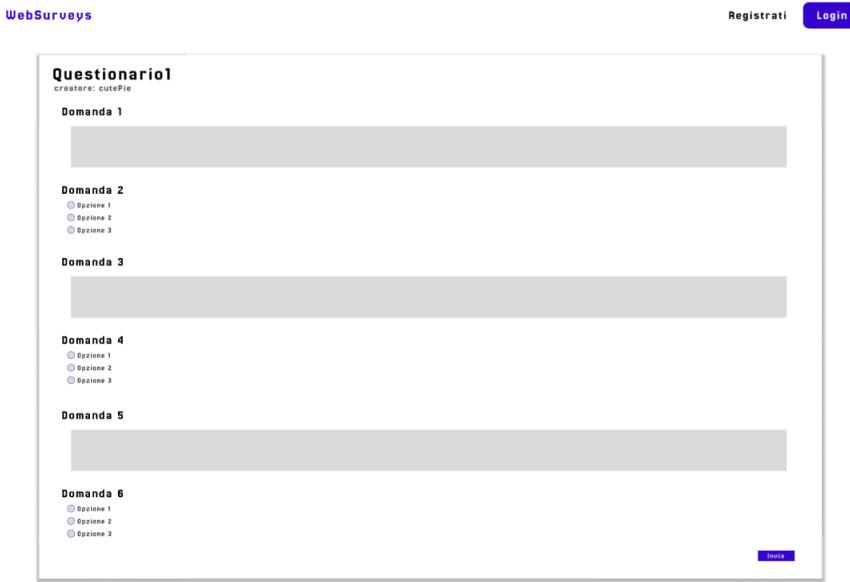


Figura 3.3: Interfaccia per la compilazione di un questionario.

### 3.3 Metodologia di Sviluppo

Lo sviluppo dell'applicazione è stato gestito seguendo il metodo Agile, con particolare riferimento alla metodologia **Scrum** e all'**Agile Unified Process** (Agile UP). Questo approccio ha permesso di incrementare la flessibilità del processo, favorendo adattamenti continui in base alle necessità e migliorando la gestione delle funzionalità richieste.

Le fasi principali dello sviluppo sono state:

- **Suddivisione del lavoro in Sprint:** il progetto è stato diviso in iterazioni (Sprint) della durata di [specifica durata, es. 1 o 2 settimane], ognuna con obiettivi chiari e misurabili.
- **Backlog del prodotto:** creazione e gestione di un elenco di funzionalità da implementare, con priorità assegnate in base all'importanza e alla complessità.
- **Daily Stand-up Meeting (se applicabile):** incontri brevi e regolari per monitorare i progressi e risolvere eventuali ostacoli.
- **Revisione e testing continuo:** al termine di ogni Sprint, le funzionalità sviluppate sono state testate e, se necessario, migliorate sulla base del feedback.
- **Integrazione e deployment progressivo:** le diverse parti del progetto sono state integrate gradualmente, evitando problemi legati allo sviluppo monolitico.

In particolare, nell'Agile Unified Process sono state seguite queste fasi:

1. **Inception:** definizione degli obiettivi principali del progetto e raccolta dei requisiti iniziali.

2. **Elaboration:** analisi e progettazione delle funzionalità chiave, con una valutazione dei rischi e una pianificazione dettagliata.
3. **Construction:** sviluppo iterativo delle funzionalità, con test e miglioramenti continui.
4. **Transition:** rilascio del prodotto e gestione del deployment, assicurando la stabilità della soluzione.

L'approccio Agile, combinando Scrum e Agile UP, ha permesso di migliorare la gestione del tempo e delle risorse, garantendo un processo iterativo e incrementale che ha portato a un prodotto finale più stabile e aderente ai requisiti iniziali.

# Capitolo 4

## Implementazione

### 4.1 Gestione Repository

Lo sviluppo del progetto è stato gestito tramite GitHub, seguendo un workflow che garantisse un'organizzazione chiara del codice e un'integrazione efficiente delle funzionalità.

L'approccio adottato è stato il seguente:

- **Branching Strategy:** il branch principale di sviluppo è stato il master, utilizzato come branch develop.
- **Gestione delle funzionalità:** per lo sviluppo di funzionalità complesse, sono stati creati branch specifici che, una volta completati e testati, venivano uniti nel branch principale.
- **Utilizzo dei tag:** per identificare versioni e milestone importanti, sono stati utilizzati tag associati alle funzionalità implementate.
- **Deploy delle milestone:** le milestone raggiunte sono state pushate sul branch main, assegnando a ciascuna il relativo tag di versione per tracciare l'evoluzione del progetto nel tempo.

Questa gestione ha permesso di mantenere un codice ben strutturato, facilitando il tracciamento delle modifiche e il *versionamento* dell'applicazione.

### 4.2 Gestione CI/CD

#### 4.2.1 Github Actions

Nel nostro progetto, abbiamo implementato **GitHub Actions** per automatizzare il processo di Continuous Integration (CI), includendo analisi del codice con **SonarCloud** per garantire qualità e sicurezza.

L'obiettivo della pipeline CI è:

1. Eseguire build e test per il backend (Spring Boot) e il frontend (React).
2. Eseguire l'analisi statica del codice con SonarCloud per rilevare code smells, vulnerabilità e bug. Garantire che il codice soddisfi standard di qualità prima di essere integrato nel repository principale.

# Capitolo 5

## Documentazione Backend

Il backend dell'applicazione è stato sviluppato utilizzando Spring Boot, seguendo l'architettura MVC per separare la gestione delle richieste, la logica di business e l'accesso ai dati. Questa sezione descrive i principali componenti del backend e il loro ruolo all'interno del sistema.

### 5.1 Controller

I Controller gestiscono le richieste HTTP provenienti dal frontend e smistano le operazioni verso i servizi appropriati. Ogni controller è responsabile di un'entità o di un gruppo di funzionalità correlate.

Le principali responsabilità dei controller sono:

- Ricevere e gestire le richieste HTTP (GET, POST, PUT, DELETE).
- Validare i dati in ingresso.
- Interagire con i servizi per elaborare le richieste.
- Restituire risposte JSON al frontend.

### 5.2 Autenticazione

L'autenticazione è stata implementata per garantire la protezione delle API e la gestione degli utenti. Il sistema utilizza JWT (JSON Web Token) per autenticare e autorizzare gli utenti, evitando sessioni basate su stato.

Le principali componenti dell'autenticazione sono:

- **Login API:** verifica le credenziali dell'utente e genera un token JWT.
- **Filtri di sicurezza:** intercettano le richieste e validano il token JWT.
- **Gestione dei ruoli e permessi:** utilizzo di Spring Security per definire le autorizzazioni.

### 5.3 Servizi

I Service contengono la logica di business e si interfacciano con i repository per accedere ai dati. Separare la logica di business dai controller garantisce un codice più pulito e testabile

## 5.4 Persistence

Il livello di persistenza gestisce l'accesso ai dati tramite Spring Data JPA e Hibernate, utilizzando il pattern Repository. Il database utilizzato è basato su SQL, garantendo integrità e prestazioni nelle operazioni sui dati.

Le principali caratteristiche del livello di persistenza sono:

- Utilizzo delle Entity per mappare le tabelle del database.
- Gestione delle operazioni CRUD tramite repository.
- Uso di DTO e mapper per separare le entità dal livello di presentazione.

# Capitolo 6

## Documentazione Frontend

Il frontend dell'applicazione è stato sviluppato utilizzando React, seguendo un'architettura modulare che suddivide il codice in model, pagine, componenti e servizi. Questa suddivisione consente una maggiore manutenibilità, riusabilità e organizzazione del codice.

### 6.1 Model

La sezione Model contiene le definizioni delle strutture dati utilizzate nel frontend, che rappresentano gli oggetti gestiti dall'applicazione. I modelli vengono usati per tipizzare i dati e garantire una corretta gestione delle informazioni.

### 6.2 Pages

Le Pages rappresentano le principali viste dell'applicazione, ovvero le pagine accessibili dagli utenti. Ogni pagina contiene la logica relativa alla sua visualizzazione e organizza i componenti necessari per renderizzare il contenuto.

### 6.3 Components

I componenti sono elementi riutilizzabili dell'interfaccia utente, che vengono combinati all'interno delle pagine per costruire il layout dell'applicazione. L'approccio basato su componenti migliora la modularità e facilita la gestione del codice.

### 6.4 Servizi

I servizi sono responsabili della comunicazione con il backend, eseguendo richieste HTTP e gestendo i dati. Questi moduli separano la logica di recupero dei dati dalla presentazione, migliorando la manutenibilità del codice.

# Capitolo 7

## Pattern

Nel progetto sono stati adottati diversi design pattern per garantire una struttura modulare, manutenibile e scalabile. L'utilizzo di questi pattern ha facilitato l'organizzazione del codice, migliorando la separazione delle responsabilità e la leggibilità.

### 7.1 Design Pattern

#### 7.1.1 DTO (Data Transfer Object)

Il **DTO** è stato utilizzato per separare la logica di business dai dati trasmessi tra il frontend e il backend tramite API REST.

Questo pattern ha permesso di:

- Ridurre la dipendenza diretta dalle entità del database.
- Ottimizzare le operazioni di serializzazione e deserializzazione.
- Garantire maggiore sicurezza e controllo sui dati esposti.

#### 7.1.2 Builder

Per semplificare la creazione di oggetti complessi, è stato adottato il **Builder Pattern**, implementato tramite Lombok.

Questo ha permesso di:

- Migliorare la leggibilità del codice grazie alla costruzione fluida degli oggetti.
- Evitare costruttori lunghi e sovraccarichi, migliorando la gestione delle istanze.
- Ridurre il rischio di errori dovuti alla creazione manuale degli oggetti.

#### 7.1.3 Repository

Per la gestione dell'accesso ai dati, è stato implementato il **Repository Pattern**.

Questo approccio ha consentito di:

- Separare la logica di accesso ai dati dalla logica di business.
- Facilitare la gestione delle operazioni CRUD tramite Spring Data JPA.
- Rendere il codice più testabile e scalabile.

### 7.1.4 Singleton

Abbiamo applicato il Singleton Pattern in particolare per:

- **Servizi di autenticazione** (es. *JwtService*, *GestoreSessione*): devono gestire token JWT e sessioni utente in modo centralizzato.
- **Servizi di gestione dati** (es. *GestoreQuestionario*, *GestoreRisposta*): per evitare di creare nuove istanze a ogni richiesta, migliorando le prestazioni e riducendo il consumo di memoria.

Nel nostro progetto, il Singleton Pattern viene automaticamente applicato da Spring, grazie all'annotazione `@Service`

## 7.2 Pattern Architetturali

### 7.2.1 MVC (Model View Controller)

Il progetto segue l'architettura MVC, separando chiaramente:

- **Model:** rappresenta i dati e la logica di business. (Contiene le Entità e i Gestori)
- **View:** gestisce l'interfaccia utente nel frontend. (Contiene i componenti React che gestiscono la presentazione)
- **Controller:** gestisce le richieste, interagendo con il Model e restituendo la View appropriata. (Espone le API REST)

Questa struttura ha garantito un codice più organizzato e facilmente manutenibile, favorendo la scalabilità dell'applicazione.

# Capitolo 8

## Analisi

Per garantire la qualità del codice e l'affidabilità dell'applicazione, sono stati utilizzati due strumenti principali, suddivisi nelle seguenti aree di analisi

### 8.1 Sonar Cloud

L'integrazione con SonarCloud ha permesso di eseguire un'analisi statica del codice e di monitorare metriche come:

- Code Smells e Best Practices
- Complessità ciclomatica
- Copertura dei test
- Identificazione di bug e vulnerabilità di sicurezza

L'analisi è stata automatizzata tramite GitHub Actions, eseguendo controlli ad ogni build per garantire il rispetto degli standard di qualità.

### 8.2 Understand

Per un'analisi più approfondita della qualità del codice, è stato utilizzato Understand, che ha fornito insight dettagliati su:

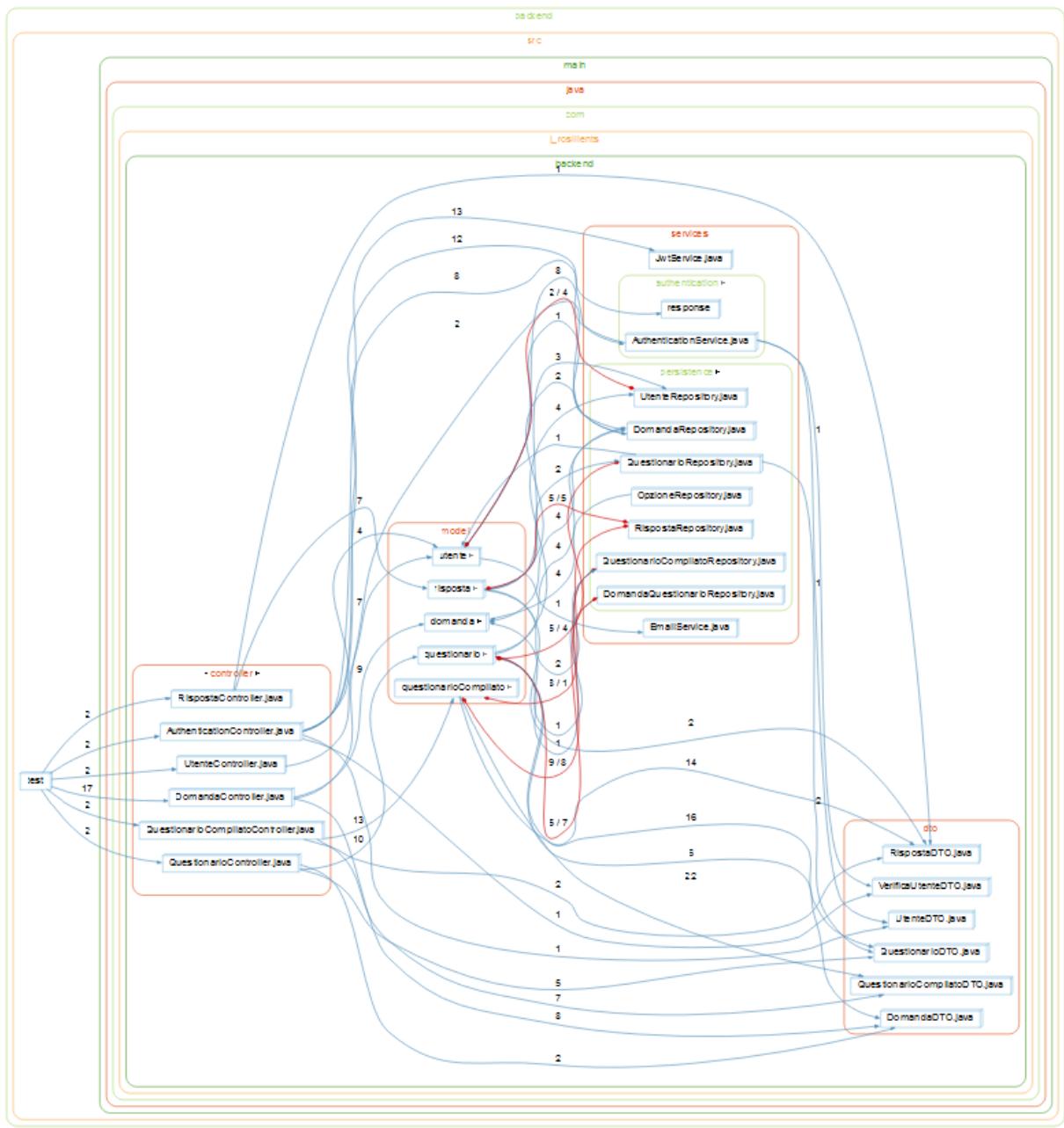
- Struttura e dipendenze del codice
- Complessità delle funzioni e delle classi
- Metriche di manutenibilità e ottimizzazione

Grazie a questo approccio, è stato possibile migliorare la leggibilità, la manutenibilità e le performance del codice, garantendo uno sviluppo più efficiente e controllato.

#### 8.2.1 Dependency Graph

Il **Dependency Graph** mostra le relazioni tra i diversi file e package del progetto. Questo grafico aiuta a identificare:

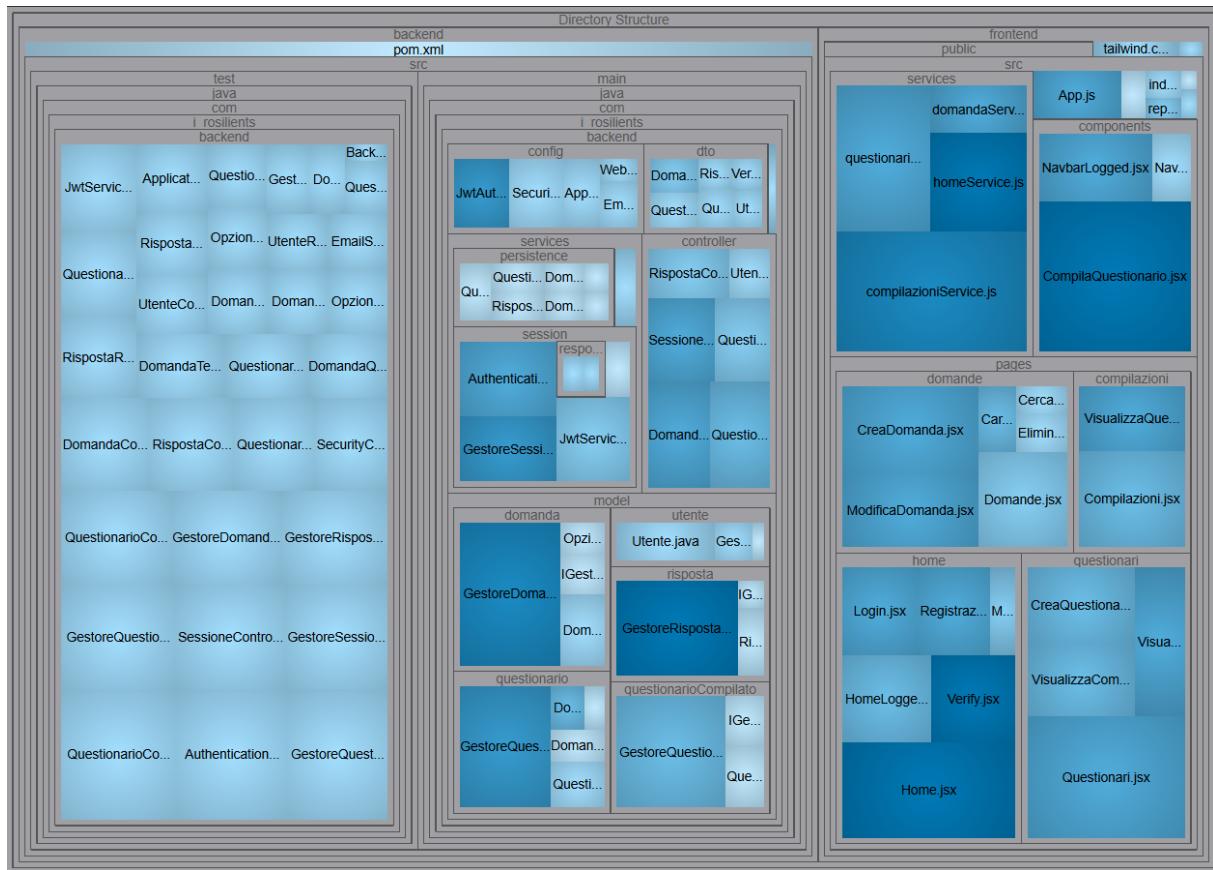
- **Dipendenze eccessive tra i moduli**, che possono indicare un alto grado di accoppiamento.
- **Cicli di dipendenza**, che potrebbero causare problemi di manutenibilità.
- **Struttura del progetto**, evidenziando i package principali e le loro interazioni.



### 8.2.2 TreeMap Complexity Metrics

Il **Treemap Metrics** rappresenta visivamente la complessità delle classi e dei file all'interno del progetto. Ogni rettangolo rappresenta un file, con dimensioni e colori basati sulle metriche selezionate (es. Linee di Codice - LOC, Complessità Ciclomatica - CC).

- **Blu scuro:** Indica file con valori più elevati nella metrica scelta (es. più righe di codice o maggiore complessità ciclomatica). Questi file potrebbero essere candidati per un'ottimizzazione o un refactoring.
- **Blu chiaro:** Rappresenta file con valori più bassi nella metrica selezionata, suggerendo che siano più semplici e modulari.



Questa rappresentazione aiuta a individuare rapidamente:

- **Zone critiche del codice**, ovvero file molto grandi o complessi che potrebbero essere difficili da mantenere.
- **Possibili punti di refactoring**, come la suddivisione di file molto grandi in moduli più piccoli.
- **Bilanciamento della complessità** tra le varie parti del progetto, permettendo di ottimizzare la struttura del codice.

Nel nostro caso, il Treemap ci ha permesso di individuare le classi che avevano complessità ciclomatica elevata e, di conseguenza, di migliorare il codice per bilanciare la complessità. Il Treemap in figura è il risultato del refactoring: tutte le classi presentano maxcyclomatic minore o uguale di 7.

# Capitolo 9

## READ ME

### 9.1 Tecnologie

- **Node.js** (per React)
- **Java 17** (per Spring Boot)
- **Maven** (per gestire le dipendenze di Spring Boot)
- **Database** (MySQLWorkBench)
- **XAMPP**
- **IDE** consigliato (IntelliJ, VSCode, ecc.)

### 9.2 Installazione

#### 9.2.1 Clonare il repository

```
git clone <repository-url>
```

#### 9.2.2 Creazione del database in MySQL

Aprire MySQL, creare un nuovo database chiamato *websurveys*, esempio:

```
CREATE SCHEMA IF NOT EXISTS websurveys;
USE websurveys;
```

#### 9.2.3 Avvio di XAMPP

Avviare moduli *Apache* e *MySQL*, verificare che entrambe i servizi siano attivi e funzionanti.

#### 9.2.4 Backend

##### 1. Configurazione Database

- Aggiungi le credenziali del tuo DBMS nel file di configurazione di Spring Boot *application.properties*

```
spring.datasource.url=jdbc:mysql://localhost:3306/websurveys
spring.datasource.username=<tuo-username>
spring.datasource.password=<tua-password>
spring.jpa.hibernate.ddl-auto=update
```

2. Aprire il terminale nella directory *backend*

- Eseguire il seguente comando

```
mvn clean install
```

3. Avviare il backend

- Usa Maven per costruire il progetto

```
./mvnw spring-boot:run
```

- il backend si avvierà su una porta predefinita (es. 8080)

### 9.2.5 Frontend

1. Spostarsi, nel terminale, nella directory *frontend*

2. Installazione delle dipendenze

```
npm install
```

3. Avviare il frontend

- Una volta installate le dipendenze

```
npm start
```

- L'app React sarà accessibile su *http://localhost:3000*

## 9.3 Test Applicazione

Dopo aver avviato sia il backend che il frontend, puoi testare l'applicazione aprendo il browser e accedendo a *http://localhost:3000*. Assicurati che tutte le funzionalità funzionino correttamente.