

**UNIVERSITÀ DEGLI STUDI DI NAPOLI  
FEDERICO II**



**SCUOLA POLITECNICA E DELLE SCIENZE DI BASE  
CORSO DI LAUREA IN INGEGNERIA  
INFORMATICA**

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE  
TECNOLOGIE DELL'INFORMAZIONE

**DOCUMENTAZIONE GRUPPO G24**

**Professoressa:**

Anna Rita Fasolino

**Gruppo:**

Bisogno Francesco	M63001449
Galiero Ilaria	M63001425
Porcellini Pierpaolo	M63001474
Scognamiglio Andrea	M63001431

ANNO ACCADEMICO 2022/2023

# Indice

<b>1 Analisi dei requisiti</b>	<b>4</b>
1.1 Requisiti funzionali . . . . .	4
1.2 Requisiti non funzionali . . . . .	4
1.2.1 Sicurezza . . . . .	4
1.2.2 Interoperabilità . . . . .	5
1.2.3 Usabilità . . . . .	5
1.2.4 Affidabilità . . . . .	5
1.2.5 Prestazioni . . . . .	5
1.2.6 Manutenibilità . . . . .	5
1.2.7 Portabilità . . . . .	6
1.3 Glossario . . . . .	6
1.4 Storie utente . . . . .	6
1.4.1 Vista di alto livello del sistema . . . . .	6
<b>2 Processo di sviluppo</b>	<b>8</b>
2.1 SCRUM . . . . .	8
2.2 Tool per la condivisione del lavoro . . . . .	10
2.2.1 Tool e tecnologie per lo sviluppo . . . . .	10
2.3 Stima dei costi . . . . .	11
2.4 Diario del team . . . . .	12
<b>3 Attività di analisi</b>	<b>13</b>
3.1 Use Case . . . . .	13
3.1.1 Scenari . . . . .	14
3.2 Sequence Diagram . . . . .	17
3.2.1 Registrazione . . . . .	17
3.2.2 Login . . . . .	18
3.2.3 Recupera password . . . . .	19
3.3 Activity Diagram . . . . .	20

3.3.1	Registrazione . . . . .	20
3.3.2	Login . . . . .	21
3.3.3	Recupera password . . . . .	21
3.4	System Domain Model . . . . .	22
<b>4</b>	<b>Attività di progettazione</b>	<b>23</b>
4.1	Spring . . . . .	23
4.1.1	Spring MVC . . . . .	23
4.1.2	Spring Data . . . . .	25
4.1.3	Spring Security . . . . .	25
4.2	Controller-Service-Repository . . . . .	26
4.3	DTO . . . . .	27
4.4	Sequence Diagram . . . . .	27
4.4.1	Login . . . . .	27
4.4.2	Register . . . . .	28
4.4.3	Confirm register . . . . .	29
4.4.4	Forgot password . . . . .	29
4.4.5	Reset password . . . . .	30
4.5	Component Diagram . . . . .	30
4.6	Package Diagram . . . . .	33
4.7	Class Diagram . . . . .	34
4.7.1	Register . . . . .	35
4.7.2	Authentication . . . . .	35
4.7.3	Forgot Password . . . . .	36
4.7.4	Reset Password . . . . .	36
4.8	Modello ER . . . . .	37
<b>5</b>	<b>Attività di implementazione</b>	<b>38</b>
5.1	Sequence Diagram . . . . .	38
5.1.1	Register . . . . .	38
5.1.2	Confirm register . . . . .	40
5.1.3	Login . . . . .	41
5.1.4	Forgot Password . . . . .	43
5.1.5	Reset password . . . . .	44
5.1.6	Logout . . . . .	45
5.2	Deployment Diagram . . . . .	46
<b>6</b>	<b>Integrabilità</b>	<b>47</b>
6.1	API locali . . . . .	47

6.1.1	RegisterController . . . . .	47
6.1.2	AuthController . . . . .	48
6.2	ForgotController . . . . .	49
6.2.1	ResetController . . . . .	50
6.3	REST API . . . . .	51
6.3.1	Swagger . . . . .	53
<b>7</b>	<b>Testing</b>	<b>54</b>
7.1	Testing basato sulle GUI . . . . .	54
7.1.1	Registrazione . . . . .	54
7.1.2	Login . . . . .	58
7.1.3	Forgot password . . . . .	59
7.1.4	Reset Password . . . . .	60
7.2	Testing basato sui servizi API REST . . . . .	62
7.2.1	Registrazione . . . . .	62
7.2.2	Conferma registrazione . . . . .	63
7.2.3	Verifica credenziali . . . . .	64
7.2.4	Identificativo utente . . . . .	64
7.2.5	Password dimenticata . . . . .	65
7.2.6	Reset password . . . . .	66
<b>8</b>	<b>Installazione</b>	<b>67</b>
8.1	Esecuzione con Docker Desktop . . . . .	68
<b>9</b>	<b>Manuale d'uso</b>	<b>71</b>
9.1	Pagina index . . . . .	71
9.2	Registrazione . . . . .	71
9.3	Login . . . . .	73

# Capitolo 1

## Analisi dei requisiti

### 1.1 Requisiti funzionali

1. Il sistema deve consentire agli utenti di potersi registrare inserendo i propri dati personali e le credenziali di accesso. Un utente si registrerà al sistema inserendo nome, cognome, percorso di studi, università, email e password. All'atto della registrazione (temporanea), il sistema invierà all'utente un'email per consentirgli di confermare la registrazione. In seguito alla conferma, l'utente risulterà registrato ed eventualmente potrà autenticarsi;
2. L'utente inserisce la propria e-mail e password per autenticarsi. Il sistema controlla la validità dei dati: se corretti, l'utente viene autenticato;
3. Il sistema deve consentire all'utente autenticato di accedere alle funzionalità di gioco o alla pagina di consultazione delle sessioni passate;
4. Il sistema deve consentire all'utente autenticato di effettuare il logout;
5. Il sistema deve consentire all'utente registrato di recuperare la password;
6. Il sistema deve consentire all'Admin di visualizzare un riepilogo di tutti gli utenti registrati al sistema.

### 1.2 Requisiti non funzionali

#### 1.2.1 Sicurezza

- La password verrà criptata prima di essere salvata nel DB;
- La password deve avere una lunghezza minima di 8 caratteri, di cui almeno una maiuscola, una minuscola ed un carattere speciale (oppure un numero).

- Un utente deve confermare la propria email per concludere la registrazione.
- Un utente deve confermare la propria email per effettuare il recupero della password.
- Il sistema deve garantire che la password non sia visibile all'esterno di esso.

### **1.2.2 Interoperabilità**

Il sistema deve essere in grado di interagire con altri sottosistemi per lo scambio di informazioni (ad esempio fornendo l'ID dell'utente registrato).

A tal proposito abbiamo previsto l'utilizzo di REST API per l'integrabilità con i task degli altri team. Per maggiori approfondimenti leggere il Capitolo 6.

### **1.2.3 Usabilità**

Il sistema deve essere facilmente utilizzabile e comprensibile dagli utenti senza richiedere un addestramento o una conoscenza approfondita del sistema stesso.

Per maggiori approfondimenti leggere il “Manuale d'uso” al Capitolo 9.

### **1.2.4 Affidabilità**

Il sistema deve funzionare in modo consistente e senza guasti per un determinato periodo di tempo, rispettando le specifiche di funzionamento.

Queste ultime sono state verificate tramite il testing funzionale. Per maggiori approfondimenti leggere il Capitolo 7.

### **1.2.5 Prestazioni**

Il sistema deve poter essere utilizzato da più utenti contemporaneamente e deve essere scalabile. In particolare abbiamo testato le varie funzionalità del sistema contemporaneamente (tramite diversi utenti) sui nostri dispositivi e risultano rispettare le specifiche funzionali.

Inoltre, non è possibile creare due sessioni contemporaneamente per lo stesso utente, in quanto una delle due verrà chiusa automaticamente.

### **1.2.6 Manutenibilità**

Il sistema deve essere modulare ed estensibile. Inoltre, il codice deve essere ben organizzato. Per soddisfare questo requisito, abbiamo scelto di utilizzare l'architettura Spring MVC (si rimanda al capitolo 4 per maggiori informazioni) con l'aggiunta del pattern Controller-Service-Repository. In generale, il pattern CSR è considerato un pattern architettonale di tipo "Service-Oriented Architecture" (SOA), che si concentra sulla separazione delle responsabilità e sulla modularità del sistema per migliorare la manutenibilità e la scalabilità dell'applicazione.

### 1.2.7 Portabilità

Il sistema deve poter essere eseguito su diverse piattaforme o ambienti.

In questo caso abbiamo solo verificato che l'applicazione funzioni anche su dispositivi mobili con sistema operativo Android.

## 1.3 Glossario

<b>Utente</b>	E' un utente che non ha accesso alle funzionalità del sistema, ma può richiedere la registrazione.
<b>Utente Registrato</b>	E' uno utente registrato, che ha accesso alle funzionalità del sistema.
<b>Utente Autenticato</b>	E' un utente (registrato) che ha effettuato il Login.
<b>Admin</b>	E' l'amministratore del sistema, dunque può accedere a tutte le funzionalità ed alle informazioni degli utenti registrati.

## 1.4 Storie utente

<b>Come</b> utente non registrato, <b>voglio</b> accedere alla pagina di registrazione dell'applicazione, <b>in modo da</b> potermi registrare.
<b>Come</b> utente registrato, <b>voglio</b> accedere alla pagina di autenticazione dell'applicazione, <b>in modo da</b> potermi autenticare.
<b>Come</b> utente registrato, <b>voglio</b> accedere alla pagina di password dimenticata dell'applicazione, <b>in modo da</b> poter recuperare l'account.
<b>Come</b> admin, <b>voglio</b> accedere alla pagina di gestione utenti, <b>in modo da</b> poter visualizzare l'elenco degli utenti registrati.

### 1.4.1 Vista di alto livello del sistema

Nella figura seguente si mostra il context diagram del sistema, in modo da evidenziare le User Interface tramite le quali i servizi esterni e gli attori possono interagire con il sistema.

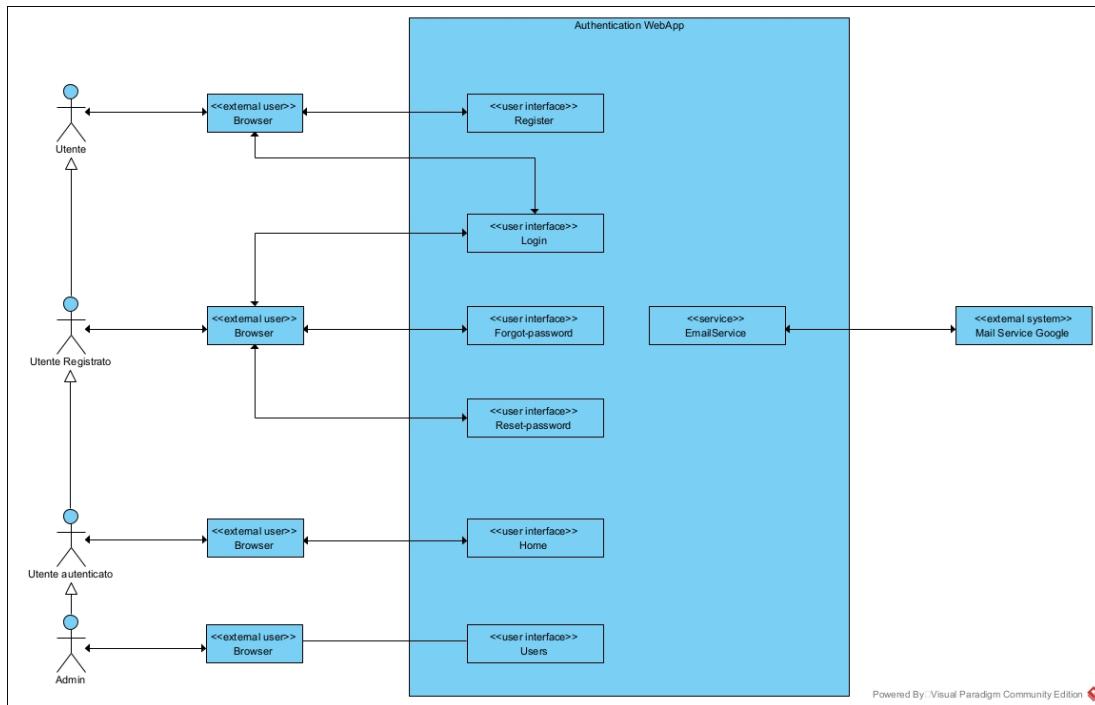


Figura 1.1: *Context diagram*

Analizzando nel dettaglio:

- **Register:** interfaccia utente dedicata alla registrazione. L'utente potrà inserire i suoi dati personali.
- **Login:** interfaccia utente dedicata all'autenticazione. In particolare un utente registrato potrà effettuare il login inserendo username e password (eventualmente potrà anche richiedere il recupero password), mentre un utente non registrato verrà ricondotto alla pagina di registrazione.
- **Forgot password:** consente all'utente registrato l'inserimento dell'email per procedere al recupero della password.
- **Reset password:** consente all'utente registrato di inserire la nuova password.
- **Home:** è la pagina che viene mostrata ad un utente dopo che l'autenticazione è andata a buon fine.
- **Users:** è la pagina visibile solo all'admin, dalla quale si possono leggere i dati (non sensibili) degli utenti registrati.
- **Email Service:** è un servizio esterno (Mail Service Google) per l'invio dell'email (nel caso di conferma registrazione e cambio password).

## Capitolo 2

# Processo di sviluppo

### 2.1 SCRUM

**SCRUM** è uno dei principali framework per l’organizzazione del lavoro agile, grazie alla sua flessibilità ed adattamento a cambiamenti e nuove priorità durante lo sviluppo. Il framework prevede un approccio iterativo ed incrementale alla gestione dei progetti costituito da diverse fasi, con un forte focus sulla collaborazione tra i membri del team e il **Product Owner**.

La prima è la fase di *avvio* o pianificazione in cui si stabiliscono gli obiettivi generali del progetto e si progetta una architettura software preliminare, questa fase risulta essere uno studio di fattibilità.

Successivamente si procede con le iterazioni (*cicli di Sprint*), che solitamente durano da due a quattro settimane. Ogni Sprint inizia con una riunione di pianificazione, in cui il Product Owner (il rappresentante del cliente o del committente) e il team di sviluppo definiscono gli obiettivi dello Sprint e scelgono le attività che verranno svolte in quell’iterazione. Il team seleziona un **backlog**, ovvero un insieme di voci da sviluppare in quella iterazione per raggiungere il *goal*.

Nello **Sprint Backlog** vengono infatti suddivisi i lavori ancora da fare, quelli in progresso e quelli completati in maniera tale da tracciare l’andamento dell’iterazione. Durante lo Sprint vengono organizzate dallo Scrum Master delle brevi riunioni quotidiane chiamate Scrum Daily, in cui si valuta lo stato di avanzamento del lavoro, si condividono informazioni sullo stato di avanzamento del lavoro e per identificare eventuali impedimenti o problemi. Alla fine di ogni Sprint vi è la **Sprint Review**, che consiste in una riunione con il Product Owner in cui il team di sviluppo presenta il lavoro compiuto, in modo che possa essere valutato e approvato, al termine di questa il team si riunisce per una Sprint Retrospective, in cui si confronta sullo Sprint appena concluso e discute cosa fare per il prossimo.

Dopo l’esecuzione dei vari Sprint si ha la fase conclusiva del progetto in cui viene completata la

documentazione richiesta e viene contrassegnato il manuale utente. Scrum prevede la distinzione di ruoli all'interno del team:

1. Il Product Owner è colui che rappresenta il fornitore dei requisiti (come ad esempio lo stakeholder).
2. Team di sviluppo che viene realizzato con un numero limitato di sviluppatori così da garantire una facile organizzazione del lavoro.
3. Scrum Master è il tutor che aiuta il gruppo ad apprendere e ad applicare le metodologie di Scrum.

Di seguito si riporta una tabella rappresentativa del nostro approccio a tale framework.

<b>Iterazione</b>	<b>Durata</b>	<b>Obiettivi prefissati</b>	<b>Documenti prodotti</b>
<i>Iter. n°1</i>	8 Apr - 21 Apr	1. Individuazione dei requisiti funzionali e non funzionali; 2. Tecnologie da impiegare per la gestione della registrazione e dell'autenticazione; 3. Diagrammi di analisi; 4. Introduzione ai servizi di Spring; 5. Tool da impiegare per lo sviluppo.	- Specifica dei requisiti - Use Case diagram con scenari - System Domain Model - Sequence Diagram - Scelta dei tool
<i>Iter. n°2</i>	27 Apr - 15 Mag	1. Scelta del DB; 2. Crittografia dei dati; 3. Deployment Diagram - Prototipo; 4. Prototipi delle funzioni.	- Deployment diagram; - Creazione del DB: MySQL - Scelta della crittografia: BCrypt. - Creazione del progetto. - Primo approccio al codice di Spring Security.
<i>Iter. n°3</i>	16 Mag - 5 Giu	1. Implementazione delle varie funzionalità prototipali: Registrazione, Login e RecuperaPassword; 2. Diagrammi di progettazione; 3. Creazione di un database remoto.	- Sviluppo di codice - Activity Diagram

<i>Iter. n°4</i>	23 Giu - 13 Lug	<ol style="list-style-type: none"> <li>1. Diagrammi di progettazione e di implementazione;</li> <li>2. Sviluppo completo del codice a partire dal prototipo;</li> <li>3. Documentazione finale;</li> <li>4. Verifica degli attributi di qualità.</li> </ol>	<ul style="list-style-type: none"> <li>- Sequence diagram di dettaglio</li> <li>- Component diagram</li> <li>- Package diagram</li> <li>- Class Diagram</li> <li>- Deployment diagram</li> <li>- REST API</li> <li>- Testing</li> <li>- Installazione</li> <li>- Documentazione con manuale d'uso</li> </ul>
------------------	-----------------	---	--

## 2.2 Tool per la condivisione del lavoro

Il team si è riunito periodicamente tramite lo strumento **Microsoft Teams**, che consente di organizzare e schedulare riunioni e meeting online. In particolare tramite lo SharePoint siamo riusciti a condividere i diversi file per lo sviluppo della nostra applicazione.

Inoltre, è stato utilizzato anche **GitHub**, un servizio di hosting per progetti software, tramite il quale i componenti del gruppo hanno sviluppato differenti funzionalità utilizzando il *pair programming* (tramite creazione di *branch*), per poi *pushare* nel repository comune i vari artefatti, rendendoli visibili e disponibili a tutti gli altri componenti del gruppo.

### 2.2.1 Tool e tecnologie per lo sviluppo

- **IDE e Linguaggio Implementativo:** Eclipse IDE, Java 17;
- **Database:** Database relazionale MySQL 8.0.33;
- **Build Automation:** Maven;
- **Supporto allo sviluppo e Servlet:** Spring Tools Suite 4, Spring Boot 3.1.0, Tomcat Apache, Lombok 1.18.26;
- **Pattern Architetturali:** Spring Web MVC;
- **Autenticazione:** Spring Security 6.1.0;
- **Accesso al Database:** Spring Data JPA;
- **Template Engine:** Thymeleaf;
- **Template:** HTML, CSS;
- **Servizio Email:** Spring Java Mail Sender;

- **Diagramma UML:** Visual Paradigm Community Edition;
- **Account Email:** Google;
- **Container:** Docker;
- **Strumento di Testing:** Postman v10.15.

### 2.3 Stima dei costi

Storia Utente	Descrizione	Stima	Priorità
<b>US1</b>	<i>Come utente non registrato, voglio accedere alla pagina di registrazione dell'applicazione, in modo da potermi registrare.</i>	5	Alta
<b>US2</b>	<i>Come utente registrato, voglio accedere alla pagina di autenticazione dell'applicazione, in modo da potermi autenticare.</i>	2	Alta
<b>US3</b>	<i>Come utente registrato, voglio accedere alla pagina di password dimenticata dell'applicazione, in modo da poter recuperare l'account.</i>	3	Alta
<b>US4</b>	<i>Come admin, voglio accedere alla pagina di gestione utenti, in modo da poter visualizzare l'elenco degli utenti registrati.</i>	2	Media

Attività	Risorse Necessarie	Tempo	Stima	Priorità
Analisi dei requisiti	Team di sviluppo	20	3	Alta
Sprint Planning	Team di sviluppo	10	2	Alta
Sprint Review	Team di sviluppo	7	1	Media (once a week)
Sviluppo delle storie utente	Team di sviluppo	50	4	Alta
Documentazione del progetto	Team di sviluppo	45	4	Alta
Studio delle tecnologie	Team di sviluppo	10	2	Alta

## 2.4 Diario del team

- **Aprile: 10h 30min**

- 7. 2h
- 13. 2h
- 15. 3h
- 17. 1h
- 27. 2h 30min

- **Maggio: 36h 45min**

- 6. 3h 30min
- 7. 2h
- 8. 1h 45min
- 15. 2h
- 16. 1h 30min
- 18. 3h 30min
- 19. 2h 50min
- 20. 1h 30min
- 22. 4h
- 23. 1h 30min
- 25. 4h 40min
- 26. 2h
- 29. 2h 50min
- 30. 3h 10min

- **Giugno: 20h 50min**

- 5. 2h 50min
- 23. 4h 55min
- 24. 4h 40min
- 26. 3h 50min
- 27. 5h 50min
- 30. 2h 45min

- **Luglio: 74h**

- 1. 6h 30min
- 2. 2h 45min
- 3. 7h 25min
- 4. 6h
- 5. 6h 35min
- 6. 6h 20min
- 7. 6h 10min
- 8. 6h 45min
- 9. 5h 50min
- 10. 2h 50min
- 11. 7h
- 12. 6h 50min
- 13. 3h

**Ore di lavoro totali:** 142h 5min

# Capitolo 3

## Attività di analisi

### 3.1 Use Case

Lo use case diagram, chiamato anche diagramma dei casi d'uso, nell'analisi dei requisiti permette di rappresentare le interazioni tra un sistema e gli attori esterni ad esso. Gli attori sono rappresentati come omini, mentre i casi d'uso sono rappresentati come ovali.

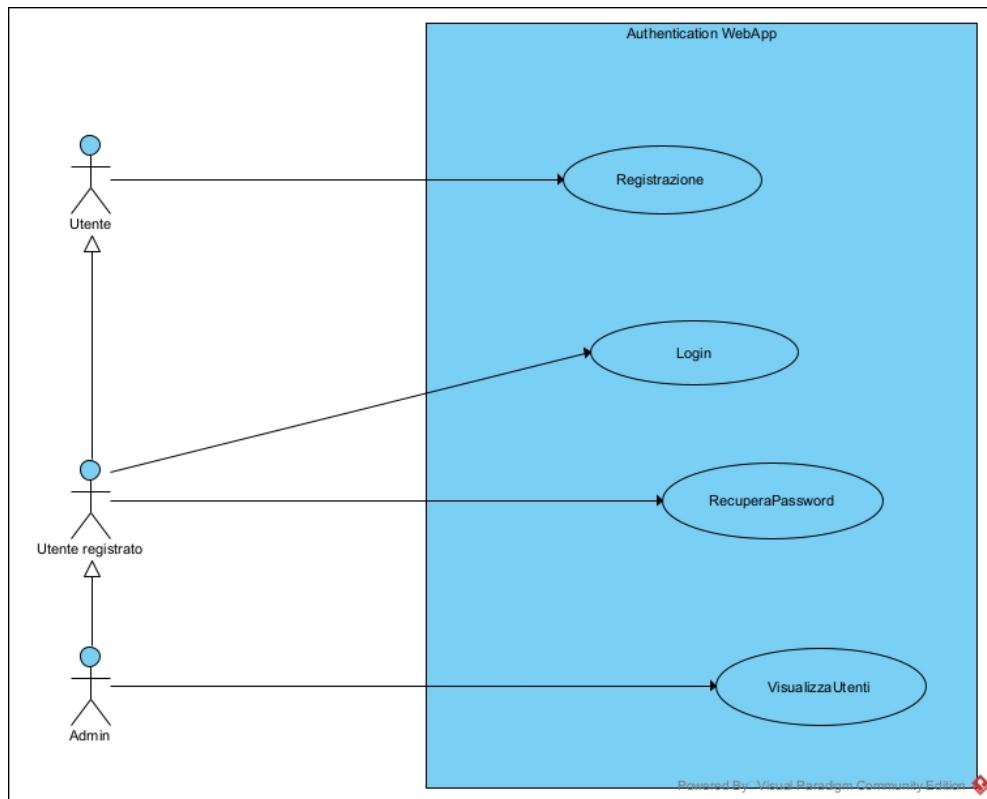


Figura 3.1: Use Case Diagram

Osservando il diagramma, si possono notare:

- *Attori:*

- **Utente:** utente che intende registrarsi al sistema;
- **Utente Registrato:** utente registrato al sistema.
- **Admin:** utente registrato con privilegi.

- *Casi d'uso:*

- **Registrazione:** l'utente non registrato inserisce i propri dati. Il sistema invierà una email di conferma all'atto della registrazione;
- **Login:** l'utente registrato inserisce le proprie credenziali per accedere alle funzionalità di gioco.
- **RecuperaPassword:** l'utente registrato può richiedere al sistema il recupero della password. Il sistema invierà una email di conferma per verificare l'email.
- **VisualizzaUtenti:** l'admin può richiedere di visualizzare i dati (non sensibili) di tutti gli utenti registrati nel sistema.

### 3.1.1 Scenari

Caso d'uso	Registrazione
Attore primario	Utente
Attore secondario	-
Descrizione	Consente a uno studente di effettuare la registrazione per usufruire delle funzionalità del gioco.
Pre - condizioni	L'utente non è registrato.
Sequenza di eventi principale	<ol style="list-style-type: none"> <li>1. L'utente accede alla schermata di registrazione;</li> <li>2. L'utente inserisce i dati e richiede la registrazione;</li> <li>3. Il sistema controlla la validità dei dati;</li> <li>4. Il sistema memorizza l'utente;</li> <li>5. Il sistema notifica l'avvenuta registrazione tramite l'invio di una mail.</li> </ol>
Post - condizioni	I dati dell'utente vengono memorizzati, viene inviata una mail.
Casi d'uso correlati	-
Sequenza di eventi alternativi	Se i dati inseriti non sono corretti, la registrazione non andrà a buon fine e il sistema mostrerà un messaggio di errore.

Caso d'uso	Login
Attore primario	Utente registrato
Attore secondario	-
Descrizione	Offre la possibilità ad un utente di effettuare l'accesso ed iniziare a giocare.
Pre - condizioni	-
Sequenza di eventi principale	<ol style="list-style-type: none"> <li>1. L'utente accede alla schermata di login;</li> <li>2. L'utente inserisce le credenziali e richiede l'accesso;</li> <li>3. Il sistema controlla la validità dei dati;</li> <li>4. Il sistema autentica l'utente.</li> </ol>
Post - condizioni	L'utente può iniziare a giocare.
Casi d'uso correlati	-
Sequenza di eventi alternativi	Se i dati inseriti non sono corretti, il login non andrà a buon fine e il sistema mostrerà un messaggio di errore.

Caso d'uso	Recupera password
Attore primario	Utente registrato
Attore secondario	-
Descrizione	Consente ad un utente registrato di reimpostare la propria password.
Pre - condizioni	-
Sequenza di eventi principale	<ol style="list-style-type: none"> <li>1. L'utente accede alla schermata di login;</li> <li>2. L'utente richiede di reimpostare la password;</li> <li>3. L'utente inserisce il proprio indirizzo email;</li> <li>3. Il sistema controlla la validità dell'email;</li> <li>4. Il sistema invia una email;</li> <li>5. L'utente clicca sul link ricevuto;</li> <li>6. L'utente inserisce la nuova password;</li> <li>7. Il sistema controlla la validità dei dati;</li> <li>8. Il sistema aggiorna la password.</li> </ol>
Post - condizioni	L'utente può accedere impiegando la nuova password.
Casi d'uso correlati	-
Sequenza di eventi alternativi	Se i dati inseriti non sono corretti, la reimpostazione della password non andrà a buon fine e il sistema mostrerà un messaggio di errore.

Caso d'uso	VisualizzaUtenti
<i>Attore primario</i>	Admin
<i>Attore secondario</i>	-
<i>Descrizione</i>	Consente ad un admin di visualizzare tutti gli utenti registrati.
<i>Pre - condizioni</i>	L'admin deve essere autenticato.
<i>Sequenza di eventi principale</i>	<ol style="list-style-type: none"> <li>1. L'admin accede alla schermata di gestione degli utenti;</li> <li>2. Il sistema visualizza l'elenco degli utenti registrati.</li> </ol>
<i>Post - condizioni</i>	-
<i>Casi d'uso correlati</i>	-
<i>Sequenza di eventi alternativi</i>	-

## 3.2 Sequence Diagram

Il sequence diagram, chiamato anche diagramma di sequenza, permette di rappresentare l'interazione tra oggetti o entità all'interno di un sistema, mostrando l'ordine con cui gli oggetti interagiscono tra loro e le azioni che eseguono durante l'interazione.

L'asse verticale rappresenta il tempo, mentre l'asse orizzontale rappresenta gli oggetti coinvolti nell'interazione. Ogni oggetto è rappresentato da un rettangolo, il quale contiene il nome dell'oggetto stesso. Le frecce nel diagramma di sequenza rappresentano i messaggi scambiati tra gli oggetti.

### 3.2.1 Registrazione

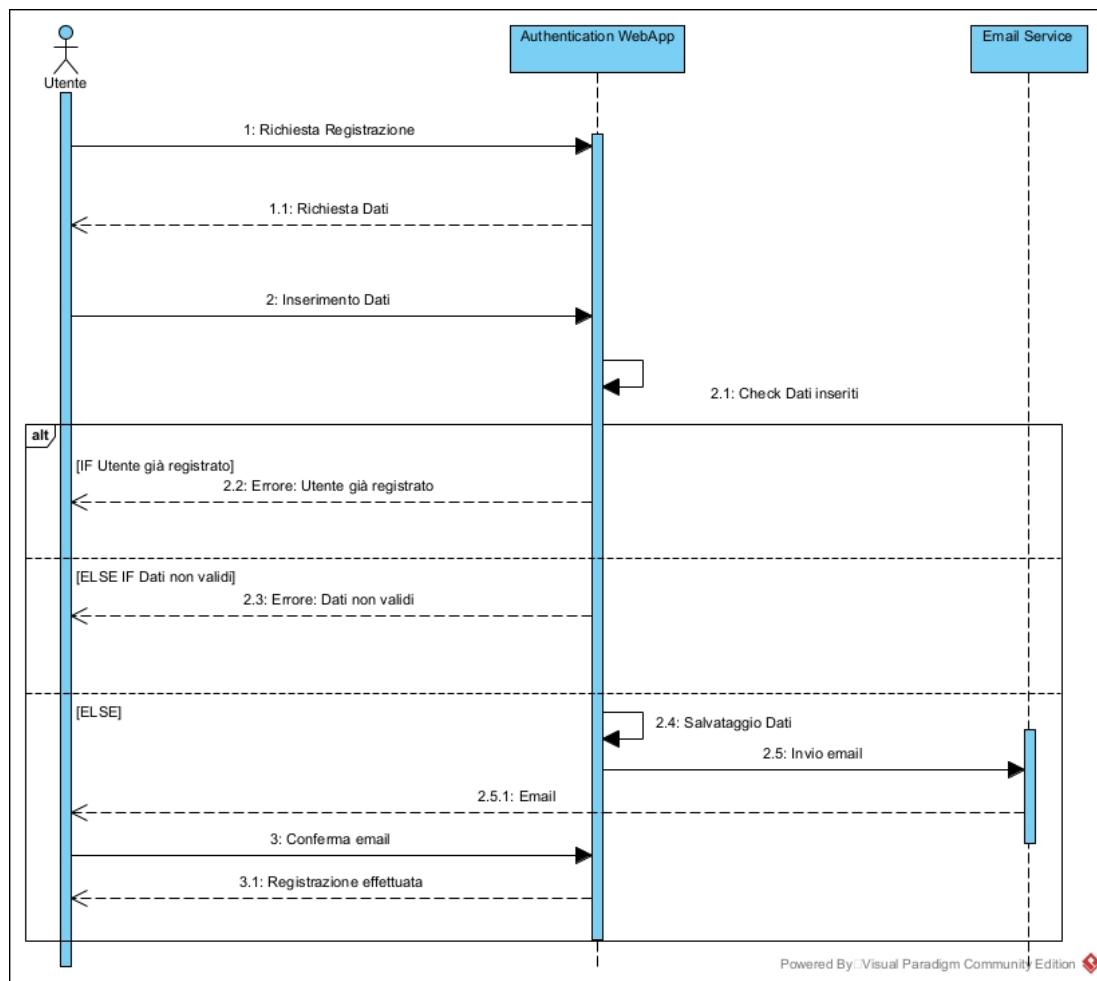


Figura 3.2: *Registrazione* sequence diagram

L'Utente richiede la pagina Registrazione. Il sistema restituisce la pagina e lo studente inserisce i propri dati. Il sistema controlla la correttezza dei dati (ad esempio i vincoli sull'email e sulla password). Se l'Utente è già presente nel database, il sistema restituisce un errore e ritorna alla

pagina di Registrazione. Stessa cosa nel caso in cui il check sui dati inseriti dall'utente non va a buon fine.

Nel caso in cui il check sui dati vada a buon fine, l'utente viene registrato nel database, ma non risulterà ancora abilitato al login (registrazione non completa) finché non confermerà l'email ricevuta dal sistema (passo 2.5).

### 3.2.2 Login

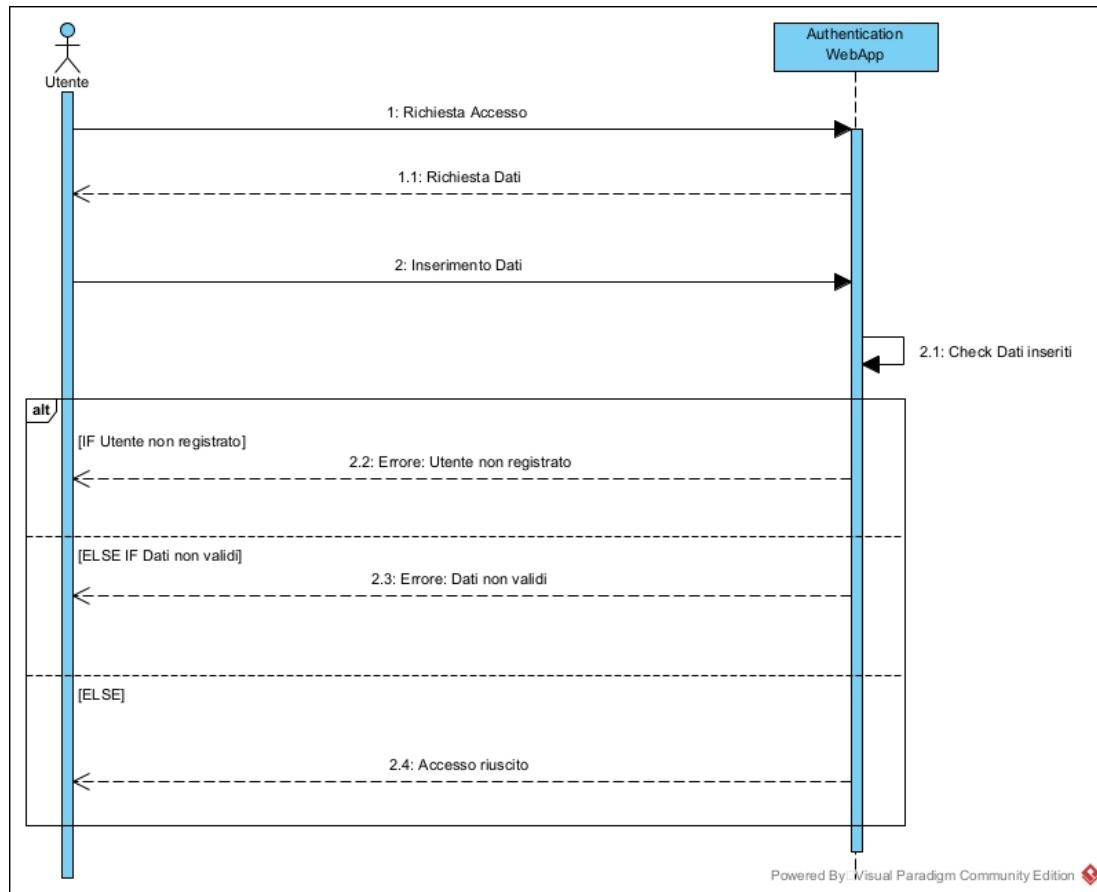


Figura 3.3: *Login* sequence diagram

L'utente registrato inserisce username e password per autenticarsi al sistema. Il sistema controlla se l'email e la password sono coerenti con quelli salvati nel DB. In tal caso il login andrà a buon fine e verrà mostrata all'utente la propria home.

Altrimenti nel caso in cui vi siano errori nell'inserimento delle credenziali oppure l'utente non è ancora registrato, il sistema ritornerà la pagina di login.

### 3.2.3 Recupera password

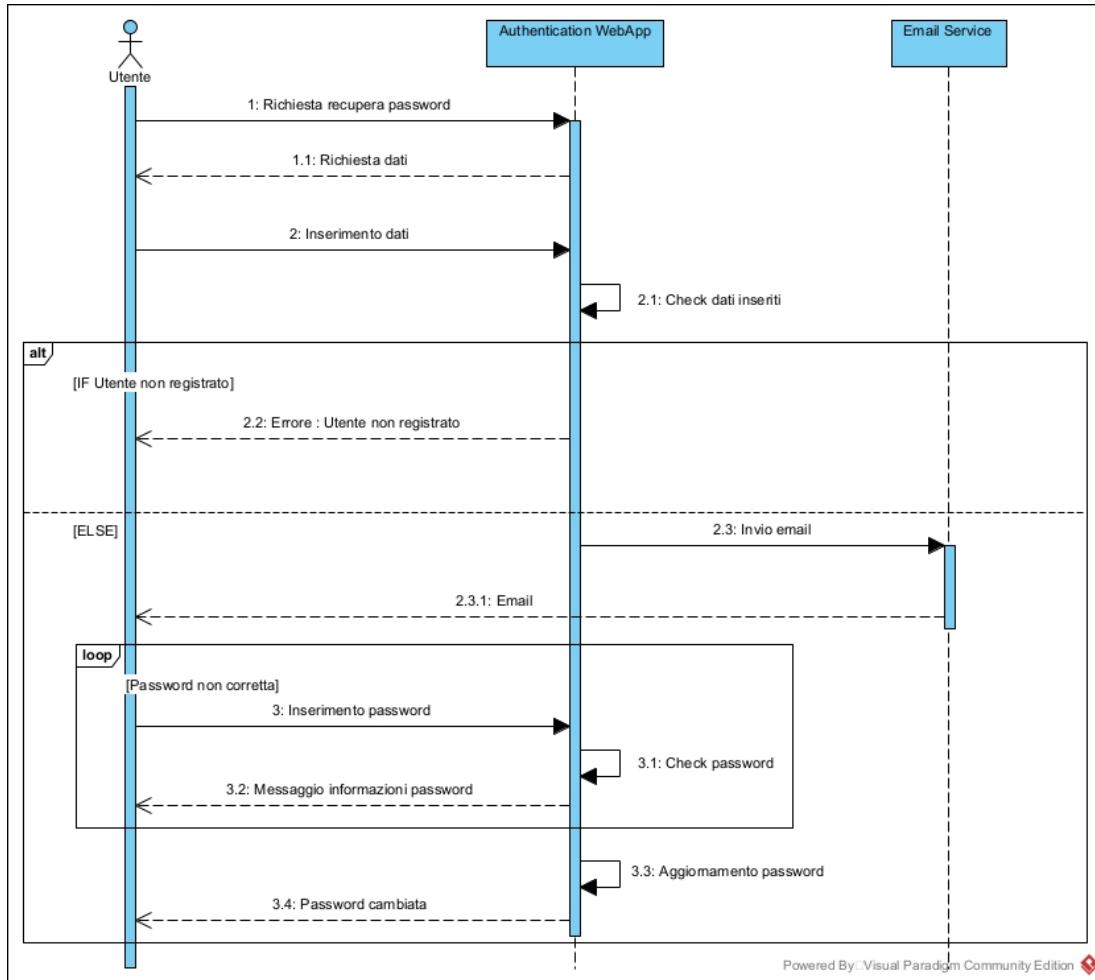


Figura 3.4: *Recupera password* sequence diagram

L’utente registrato che desidera recuperare la password, accede alla pagina “Recupera Password” e inserisce la propria email. Il sistema controlla l'email inserita, in particolare:

- Se l’utente non è registrato, ritorna la pagina di “Recupera Password”;
- Se il check va a buon fine, invia un'email all’utente, il quale potrà collegarsi alla pagina “Reset Password” ed inserire la nuova password. Il sistema controllerà se la nuova password rispetta i vincoli imposti nei requisiti di sicurezza ed in caso affermativo restituirà all’utente la pagina di login.

### 3.3 Activity Diagram

L'activity diagram, chiamato anche diagramma delle attività, permette di rappresentare il flusso di lavoro o di attività all'interno di un processo o di un sistema.

Il diagramma delle attività è composto da una serie di forme geometriche (rettangoli, rombi, cerchi, frecce) che rappresentano le attività, le decisioni, le transizioni e i flussi di dati all'interno del processo. Le attività sono le azioni che vengono eseguite durante il processo, mentre le decisioni rappresentano i punti in cui si prendono scelte basate su determinati criteri.

#### 3.3.1 Registrazione

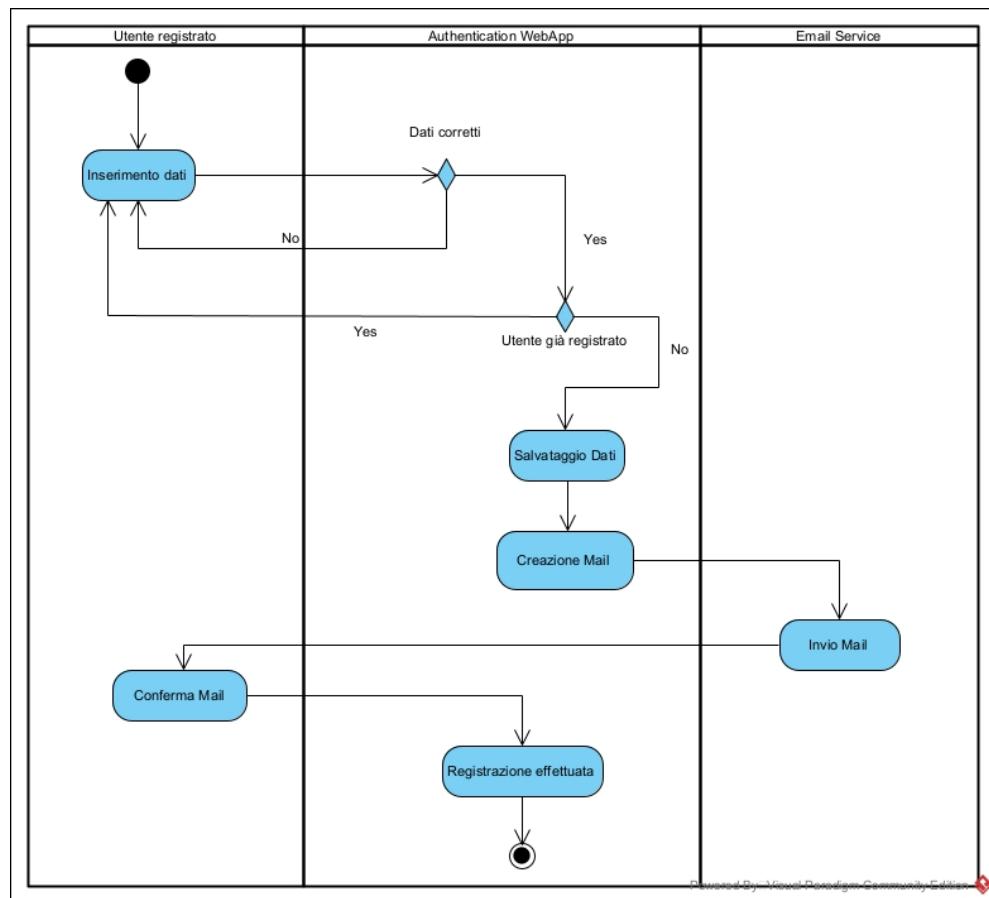


Figura 3.5: *Registrazione* activity diagram

### 3.3.2 Login

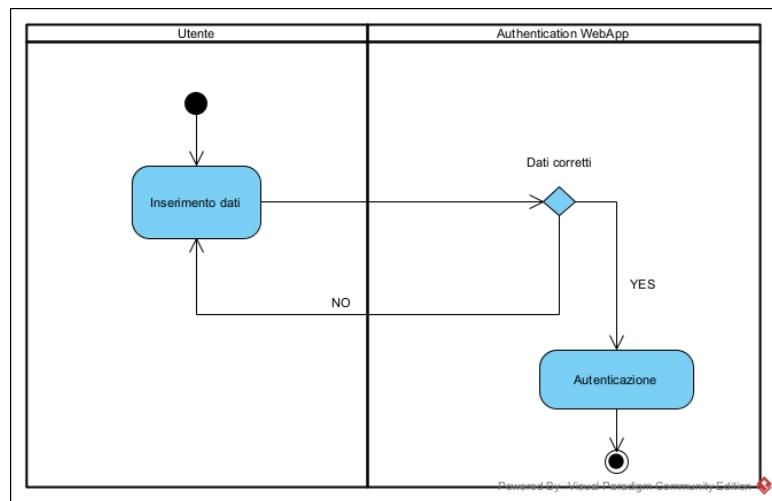


Figura 3.6: *Login* activity diagram

### 3.3.3 Recupera password

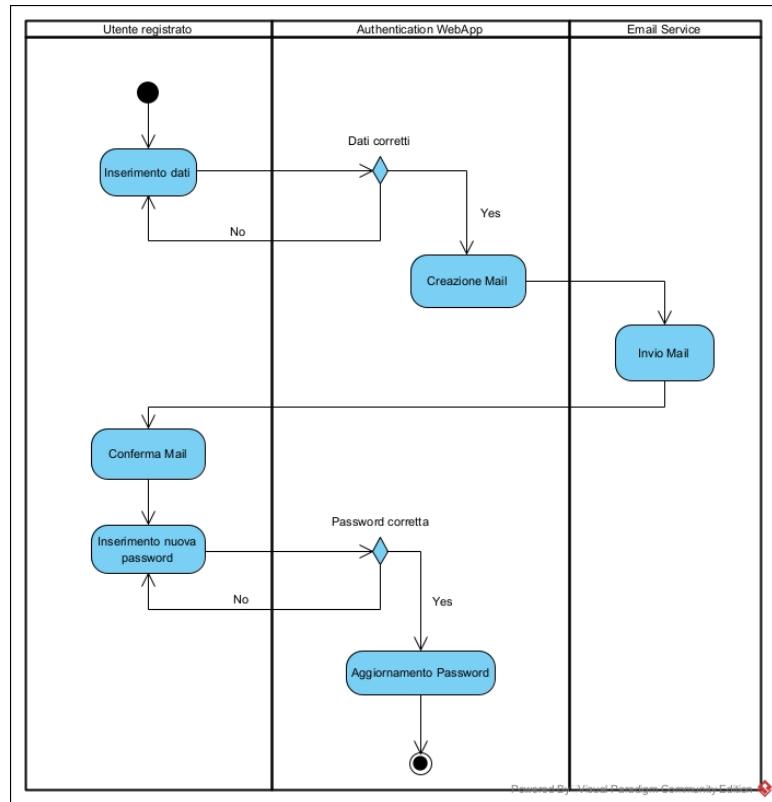


Figura 3.7: *Recupera password* activity diagram

### 3.4 System Domain Model

Il System Domain Model (SDM) è un modello che permette di rappresentare il dominio del sistema, descrivendo le classi, gli oggetti e le relazioni esistenti all'interno del sistema.

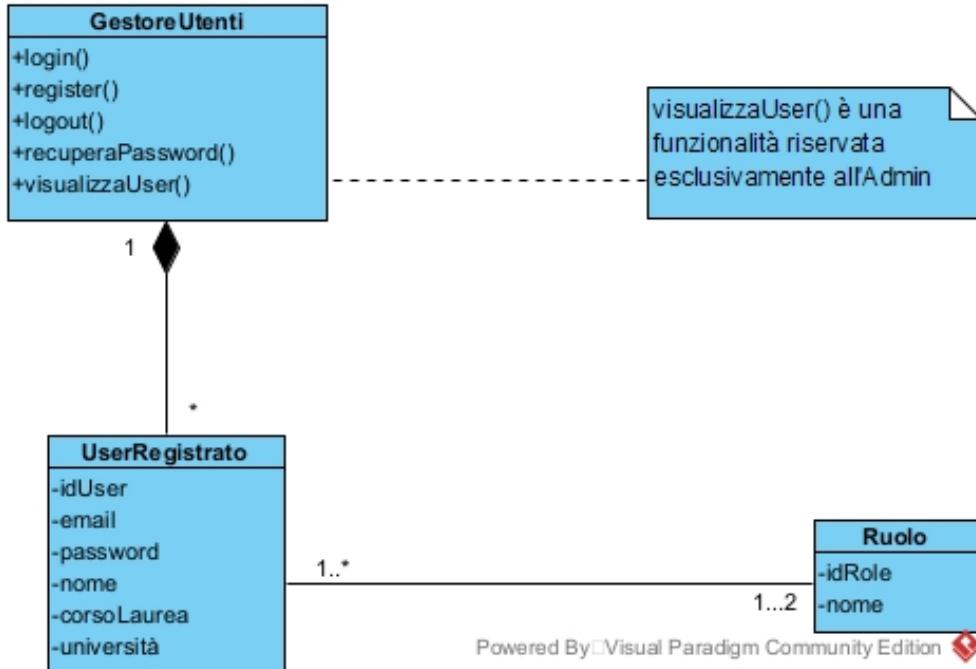


Figura 3.8: System Domain Model

Le principali entità del nostro dominio sono gli Utenti (**UserRegistrato**) che utilizzano le funzionalità del gestore **GestoreUtenti**.

Il sistema non è modellato per mantenere informazioni riguardo utenti non registrati. A questi ultimi offre la funzionalità `register()`, che consente di effettuare la registrazione.

È stato previsto un utente registrato con privilegi (admin) che ha un ruolo aggiuntivo per essere contraddistinto dall'utente normale.

Il sistema offre esclusivamente all'admin la funzionalità `visualizzaUser()`.

# Capitolo 4

## Attività di progettazione

Per la nostra applicazione abbiamo utilizzato il framework Spring.

Per l'architettura abbiamo utilizzato MVC, il pattern controller service repository e le DTO.

Per la persistenza dei dati abbiamo utilizzato Spring Data JPA, per la sicurezza Spring Security, in particolare per crittografare la password abbiamo utilizzato BcryptPasswordEncoder.

### 4.1 Spring

Il framework Spring fornisce un modello completo di programmazione e configurazione per le moderne applicazioni aziendali basate su Java, fornendo una serie di componenti e strumenti riutilizzabili.

Spring offre infatti un modo semplice e coerente per gestire le dipendenze tra i diversi componenti di un'applicazione, facilitando l'organizzazione e il riutilizzo del codice. Spring fornisce anche una serie di funzionalità integrate per la gestione di attività comuni. Inoltre, è modulare, il che significa che puoi scegliere le funzionalità di cui hai bisogno senza utilizzare l'intero framework.

Spring fornisce anche supporto per diverse tecnologie di accesso ai dati attraverso il proprio livello di accesso/integrazione dati e fornisce un framework Web chiamato Spring MVC per creare applicazioni web.

#### 4.1.1 Spring MVC

Il framework Spring Web MVC fornisce un'architettura Model-View-Controller (MVC) e componenti già pronti che possono essere utilizzati per sviluppare applicazioni web flessibili. Il pattern MVC consente di separare i diversi aspetti dell'applicazione (logica di input, logica di business e logica dell'interfaccia utente), fornendo al contempo un accoppiamento lasco tra questi elementi.

- **Model:** rappresenta il modello dati dell'applicazione e gestisce l'accesso e la manipolazione dei dati.
- **View:** si occupa della visualizzazione dei dati dell'applicazione e della presentazione all'utente finale.
- **Controller:** gestisce il flusso di esecuzione dell'applicazione, riceve le richieste HTTP dal client, interagisce con il modello per recuperare i dati e li passa alla view per la visualizzazione.

Il framework MVC (Model-View-Controller) di Spring Web è progettato attorno a un DispatcherServlet che gestisce tutte le richieste e le risposte HTTP.

Il funzionamento di Spring MVC con il DispatcherServlet si articola in diversi passaggi, che vengono eseguiti quando il client invia una richiesta HTTP al server. I passaggi principali sono i seguenti:

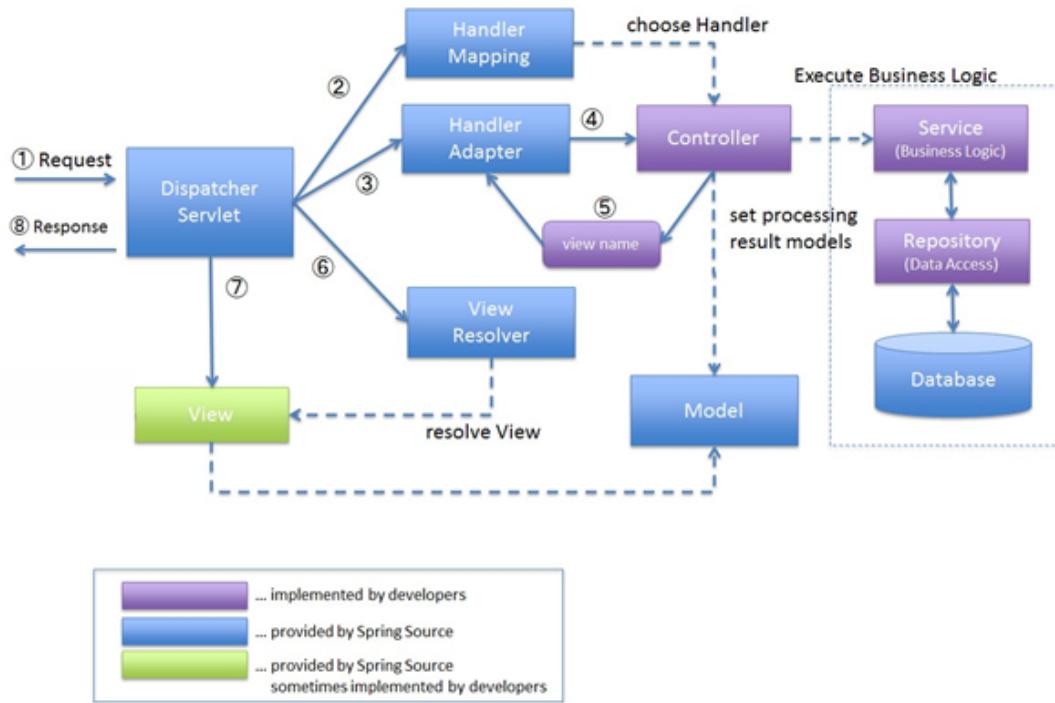


Figura 4.1: Architettura MVC

1. Il client invia una richiesta HTTP al server. Il server riceve la richiesta e la passa al DispatcherServlet di Spring MVC.
2. Il DispatcherServlet cerca il Controller appropriato per gestire la richiesta, utilizzando l'HandlerMapping.
3. L'HandlerMapping seleziona il Controller che è mappato all'URL della richiesta in arrivo e restituisce l'Handler corrispondente al DispatcherServlet.

4. Il Controller riceve la richiesta tramite l'Handler e la elabora.
5. Il Controller restituisce il nome della View ed il Model associato ad essa al DispatcherServlet.
6. Il DispatcherServlet inoltra il compito di risolvere la View al ViewResolver, che restituisce la vista corretta.
7. Il DispatcherServlet passa i dati del Model alla View, che li utilizza per generare la risposta HTML.
8. Il DispatcherServlet restituisce la risposta HTTP al client.

#### 4.1.2 Spring Data

Spring Data permette di semplificare lo stato di persistenza rimuovendo completamente l'implementazione dei Repository dalla nostra applicazione. Per fare ciò, l'interfaccia dei Repository deve estendere JpaRepository e Spring Data creerà automaticamente un'implementazione dotata dei metodi CRUD più rilevanti per l'accesso ai dati.

#### 4.1.3 Spring Security

Spring Security è un framework di sicurezza per applicazioni Java basate su Spring, che fornisce una serie di funzionalità per la gestione dell'autenticazione, dell'autorizzazione e della sicurezza in generale.

Il suo funzionamento può essere suddiviso in tre fasi principali:

- *Autenticazione*: La fase di autenticazione verifica l'identità dell'utente. Durante questa fase, Spring Security verifica le credenziali fornite dall'utente (username e password) e crea una sessione.
- *Autorizzazione*: La fase di autorizzazione determina se l'utente ha il permesso di accedere alle risorse protette dall'applicazione. Spring Security gestisce questa fase utilizzando il concetto di ruoli, ad ogni utente vengono assegnati uno o più ruoli. Durante la fase di autorizzazione, Spring Security verifica se l'utente ha i permessi necessari per accedere alle risorse protette.
- *Gestione della sicurezza*: La fase di gestione della sicurezza si occupa di gestire e prevenire gli attacchi alla sicurezza dell'applicazione. Spring Security fornisce una serie di funzionalità per la gestione della sicurezza, come ad esempio la gestione delle sessioni, la crittografia dei dati sensibili e altri ancora.

L'architettura di Spring Security è basata su un insieme di filtri (*filters*) che vengono applicati alle richieste in ingresso all'applicazione. I filtri sono organizzati in una catena (*filter chain*)

che viene eseguita in sequenza per gestire l'autenticazione, l'autorizzazione e la sicurezza dell'applicazione.

La catena di filtri può essere personalizzata e configurata a seconda delle esigenze specifiche dell'applicazione e Spring Security fornisce anche una serie di classi di configurazione per semplificare questo processo.

Per memorizzare sul database le informazioni di accesso degli utenti, è stato utilizzato un Password encoder per crittografare la password.

L'interfaccia PasswordEncoder di Spring Security viene utilizzata per eseguire una trasformazione unidirezionale di una password, in modo da consentirne la memorizzazione sicura. In genere, PasswordEncoder viene utilizzato per memorizzarne una che deve essere confrontata con quella fornita dall'utente al momento dell'autenticazione.

L'implementazione di BCryptPasswordEncoder utilizza l'algoritmo bcrypt, ampiamente supportato, per eseguire l'hash delle password.

## 4.2 Controller-Service-Repository

Il pattern Controller-Service-Repository è un'architettura comune utilizzata nello sviluppo del software per separare la logica dell'applicazione dalla gestione dei dati e delle risorse. Esso prevede l'utilizzo di tre componenti principali:

- **Controller:** si occupa di esporre le funzionalità all'esterno, ricevendo le richieste degli utenti, elaborandole e restituendo le risposte appropriate. Comunica con i servizi per effettuare le operazioni necessarie.
- **Service:** incapsula tutta la logica di business in un unico luogo per promuovere il riutilizzo del codice. Se la logica di business richiede l'acquisizione/il salvataggio di dati, il livello Service può utilizzare il livello Repository per implementare alcune logiche che coinvolgono il database.
- **Repository:** si occupa di archiviare e gestire i dati dell'applicazione, estendendo le operazioni CRUD sul database.

In un'applicazione, è meglio avere un livello Repository e un livello Service separati, cosicché anche cambiando la tipologia di database il codice dei livelli superiori non avrà bisogno di modifiche.

L'utilizzo del pattern Controller-Service-Repository consente di separare in modo efficace la logica dell'applicazione dalla gestione dei dati e delle risorse, rendendo l'applicazione più modulare, scalabile e facile da gestire. Inoltre, esso favorisce la riusabilità del codice e la manutenibilità del software.

## 4.3 DTO

Il DTO o *Data Transfer Objects* è un design pattern usato per trasferire dati tra sottosistemi di un'applicazione software.

Con i DTO, è possibile costruire viste diverse dai domain models, consentendo di creare altre rappresentazioni dello stesso dominio, ma ottimizzandole in base alle esigenze dei client senza alterare il design del dominio. Inoltre, disaccoppia i design model dal livello di presentazione, consentendo a entrambi di cambiare in modo indipendente.

Possono essere usati anche per nascondere i dettagli di implementazione delle entity del livello database, in quanto esporle a livello web senza gestire correttamente la risposta può diventare un problema di sicurezza. Se la risposta non viene gestita correttamente, si possono ottenere tutti i campi della classe, che possono contenere informazioni sensibili.

I dati vengono mappati dai modelli di dominio ai DTO, normalmente attraverso un componente **mapper**.

## 4.4 Sequence Diagram

### 4.4.1 Login

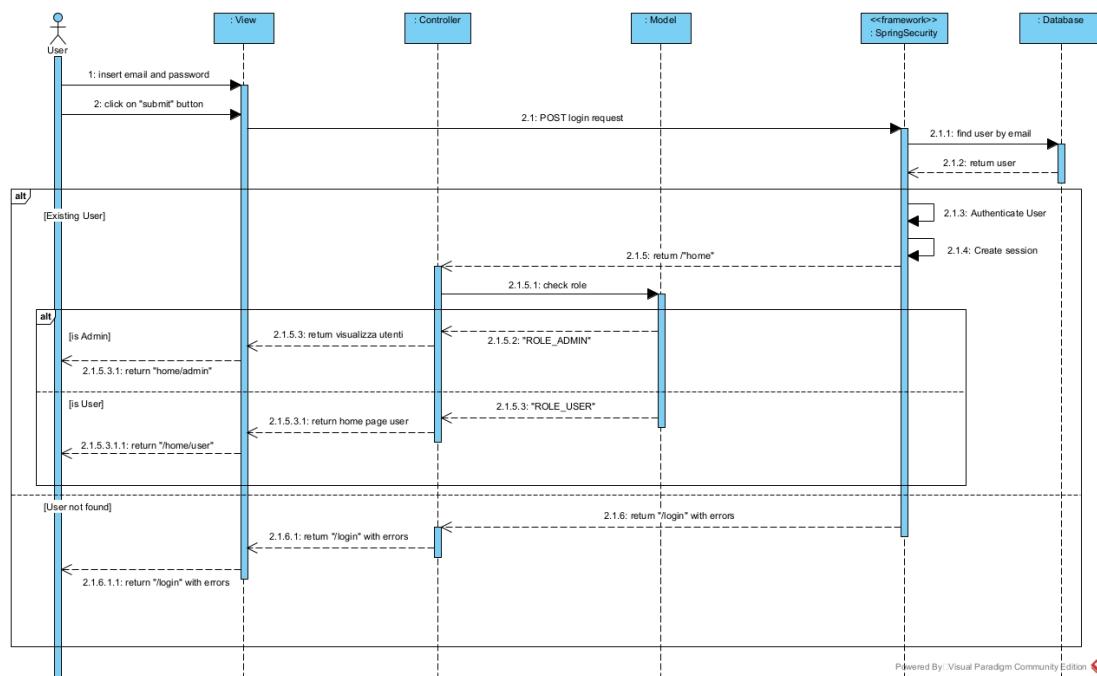


Figura 4.2: *Login* sequence diagram

L'utente inserisce email e password nel form di login. La richiesta HTTP POST viene raccolta dai filtri interni di Spring e processata. Il framework Spring, recupera i dati dell'utente dal database ricercandolo in base all'email.

Se l'utente non esiste, il login non va a buon fine ed il controller restituisce la view “/login” stampando a video l'errore.

Se l'utente esiste, viene autenticato e viene creata una sessione. Il controller restituirà viste differenti a seconda della tipologia di autenticazione. Se l'utente è un admin (“ROLE\_ADMIN”) verrà restituita la view “/home/admin” che mostrerà a video la lista degli utenti registrati.

Se l'utente è uno user (“ROLE\_USER”), verrà restituita la view “/home/user”.

#### 4.4.2 Register

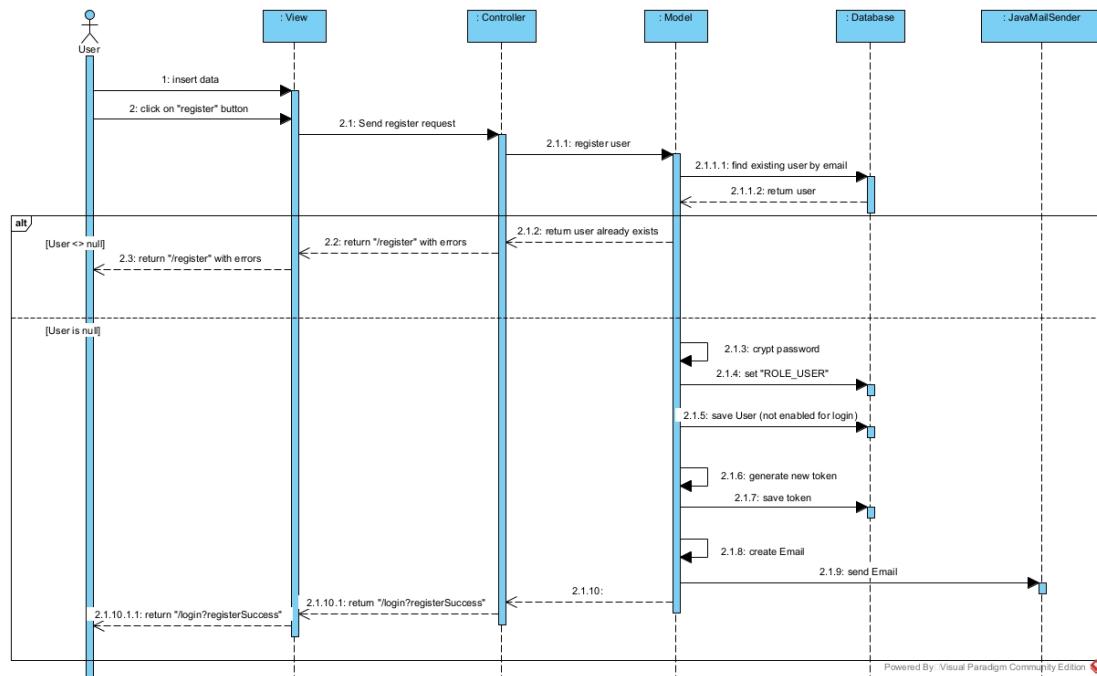


Figura 4.3: *Register* sequence diagram

L'utente inserisce i dati nel form di registrazione e clicca sul pulsante “register”. Il controller gestisce la richiesta e la inoltra al model, il quale userà dei servizi per eseguire le seguenti operazioni:

1. Cerca l'utente nel database accedendo tramite il campo email;
2. Se l'utente è già presente, la registrazione fallisce e viene restituita all'utente la vista “/register” con relativo messaggio di errore.
3. Se l'utente non è presente nel database:
  - 1) Viene criptata la password;
  - 2) Vengono salvate le informazioni dell'utente nel database;
  - 3) Viene generato un nuovo token (associato all'utente);

- 4) Viene generata ed inviata una email con il token creato al passo precedente.

Il controller restituirà la vista di Login.

L'utente riceverà una mail contenente un link (token) che dovrà aprire per completare la fase di registrazione (vedere il sequence di Confirm Registration).

#### 4.4.3 Confirm register

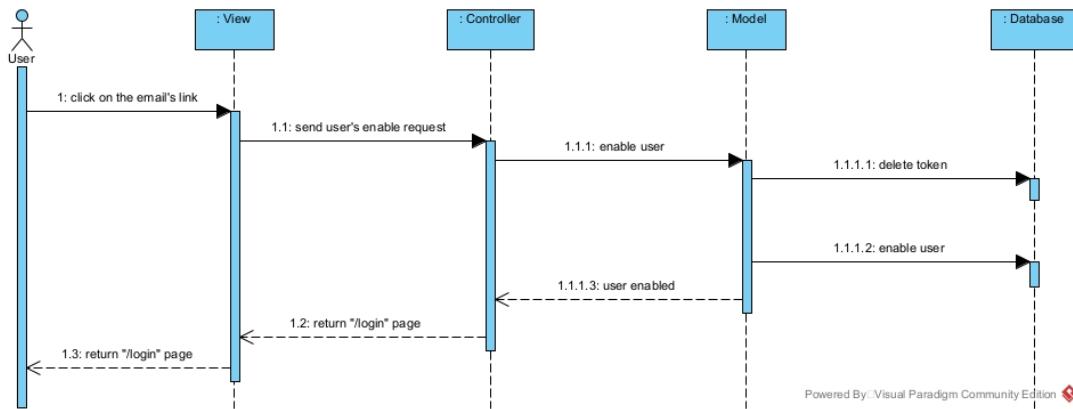


Figura 4.4: *Confirm register* sequence diagram

L'utente clicca sul link ricevuto tramite email. Viene inviata una richiesta al controller, che tramite i servizi del Model provvederà ad abilitare l'utente e a cancellare il token ad esso associato dal database. Dopodiché verrà restituita la vista “/login”.

#### 4.4.4 Forgot password

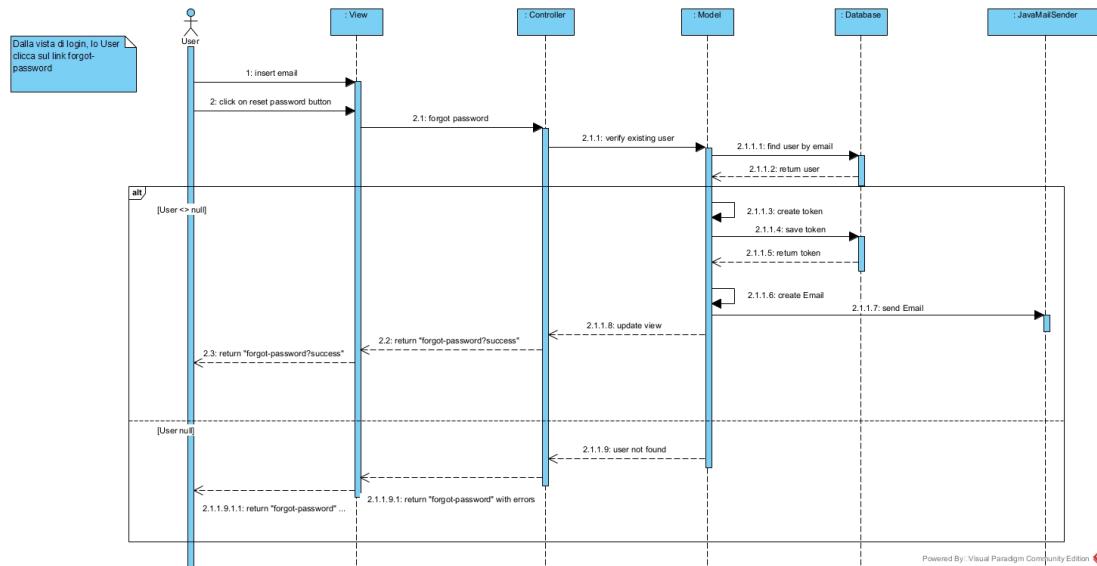


Figura 4.5: *Forgot password* sequence diagram

Dalla view “/login” l’utente clicca su forgot password. A questo punto dovrà inserire la propria email nel form comparso.

Il controller tramite i servizi del model controlla se l’utente è registrato nel database (accedendo tramite email).

Se l’utente è presente, verrà creato (e salvato nel database) un nuovo token associato ad esso.

In seguito verrà generata ed inviata una email all’indirizzo specificato nel form. Il controller provvederà ad aggiornare la vista mostrando all’utente la pagina “/forgot-password?success”.

Se l’utente non è presente nel database, il controller mostrerà la vista “/forgot-password” stampando a video il messaggio di errore.

#### 4.4.5 Reset password

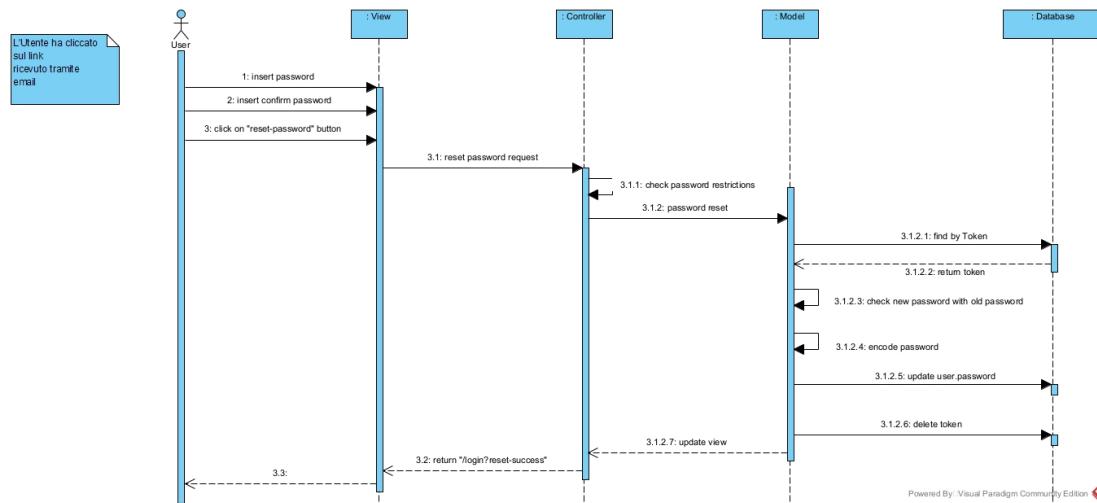


Figura 4.6: *Reset password* sequenze diagram

L’utente ha cliccato sul link ricevuto tramite email per il reset della password. La vista “/reset-password” mostrerà un form all’interno del quale l’utente dovrà inserire la nuova password e confermarla (rispettando i vincoli descritti nei requisiti). La view inoltrerà la richiesta al controller, il quale, utilizzando i servizi del Model, tramite il token preleva l’utente ad esso associato dal database e se la nuova password è diversa da quella vecchia, verrà criptata e aggiornata nel database.

Dopodiché il token verrà cancellato ed il controller restituirà la vista “/login?reset-success”.

## 4.5 Component Diagram

Un Component diagram è un diagramma UML utilizzato per visualizzare la struttura interna di un sistema software e le relazioni tra i suoi componenti per fornire le funzionalità richieste.

Nel nostro caso le funzionalità che esponiamo all'esterno sono autenticazione, registrazione, recupero password, conferma registrazione e restituzione dell'Id dell'Utente.

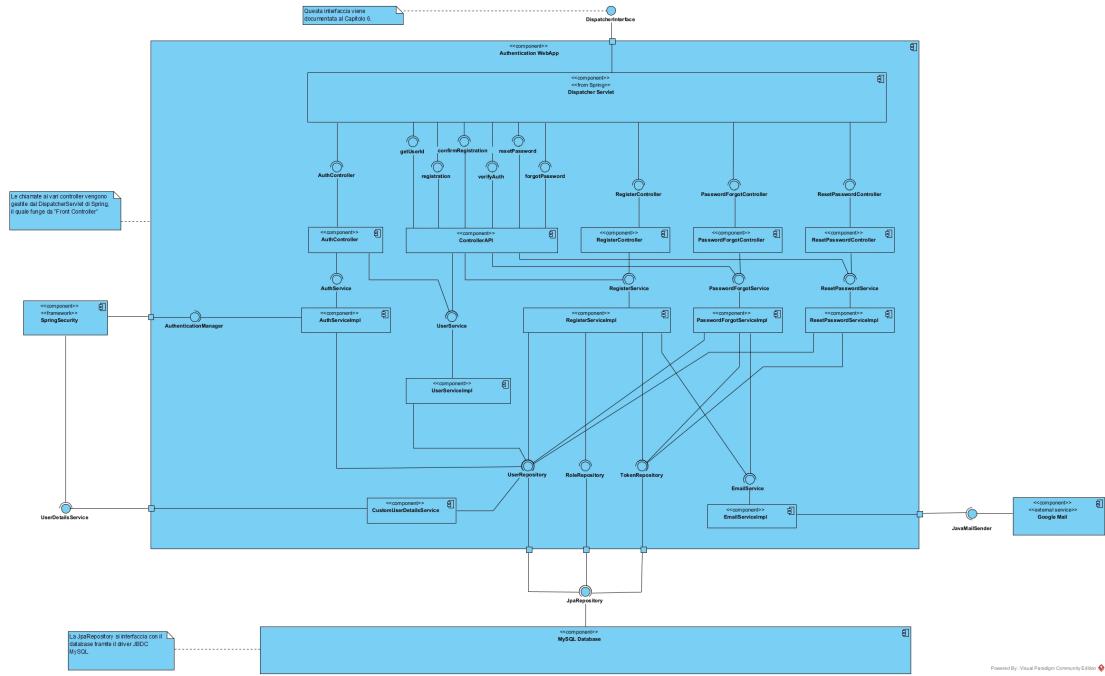


Figura 4.7: Component Diagram

I componenti di cui il nostro sistema si compone sono:

- **ControllerAPI:** componente che espone i servizi: getUserId, forgotPassword, registration, confirmPassword, resetPassword e verifyAuth e utilizza i servizi: userService fornito dal component UserServiceImpl, registerService fornito dal component RegisterServiceImpl, passwordForgotService fornito dal component PasswordForgotServiceImpl e infine, il passwordResetService fornito dal component PasswordResetServiceImpl.
- **AuthController:** componente che si occupa di gestire l'autenticazione utilizzando il servizio fornito da userServiceImpl e l'authServiceImpl.
- **RegisterController:** componente che si occupa di gestire la registrazione utilizzando il servizio fornito da registerServiceImpl.
- **PasswordForgotController:** componente che si occupa di gestire la richiesta di recupera password utilizzando il servizio fornito da passwordForgotServiceImpl.
- **ResetPasswordController:** componente che si occupa di gestire la richiesta di reset password utilizzando il servizio fornito da resetPasswordServiceImpl.
- **DispatcherServlet:** componente messo a disposizione da Spring che riceve (filtra) le richieste dall'esterno, tramite l'interfaccia DispatcherInterface, andando a richiamare il

controller corrispondente alla richiesta ricevuta, fungendo da *front controller*.

- **EmailServiceImpl:** componente che offre i servizi di gestione delle email e utilizza il servizio offerto da Spring JavaMailSender.
- **AuthServiceImpl:** componente che offre i servizi di gestione dell'autenticazione interfacciandosi con l'AuthenticationManager offerto da Spring Security ed interfacciandosi con il database tramite UserRepository.
- **RegisterServiceImpl:** componente che offre i servizi di gestione della registrazione utilizzando il servizio offerto da EmailServiceImpl e interfacciandosi con il database tramite UserRepository, RoleRepository e TokenRepository ed utilizzando per la crittografia della password il passwordEncoder offerto da Spring.
- **PasswordForgotServiceImpl:** componente che offre i servizi di gestione del recupero password utilizzando il servizio offerto da EmailServiceImpl e interfacciandosi con il database tramite UserRepository e TokenRepository.
- **PasswordResetServiceImpl:** componente che offre i servizi di gestione del reset password interfacciandosi con il database tramite UserRepository e TokenRepository ed utilizzando per la crittografia della password il passwordEncoder offerto da Spring.
- **UserServiceImpl:** componente che offre i servizi di gestione dell'utente interfacciandosi con il database tramite UserRepository ed utilizzando per la crittografia della password il passwordEncoder offerto da Spring.
- **CustomUserDetailsService:** componente utilizzato da Spring Security per effettuare l'autenticazione dell'utente interfacciandosi con il database tramite UserRepository.
- **MySQL Database:** componente esterno che indica la nostra base dati.
- **SpringSecurity:** componente esterno che ci offre funzionalità per l'autenticazione, il logout e per la crittografia della password con il passwordEncoder.
- **Google Mail:** componente esterno per l'invio della email utilizzando JavaMailSender.

## 4.6 Package Diagram

Il package diagram mostra l'organizzazione e le dipendenze tra i package in un sistema o un'applicazione software. Di seguito si mostra come i moduli sono stati ripartiti all'interno del progetto rispettando il pattern MVC.

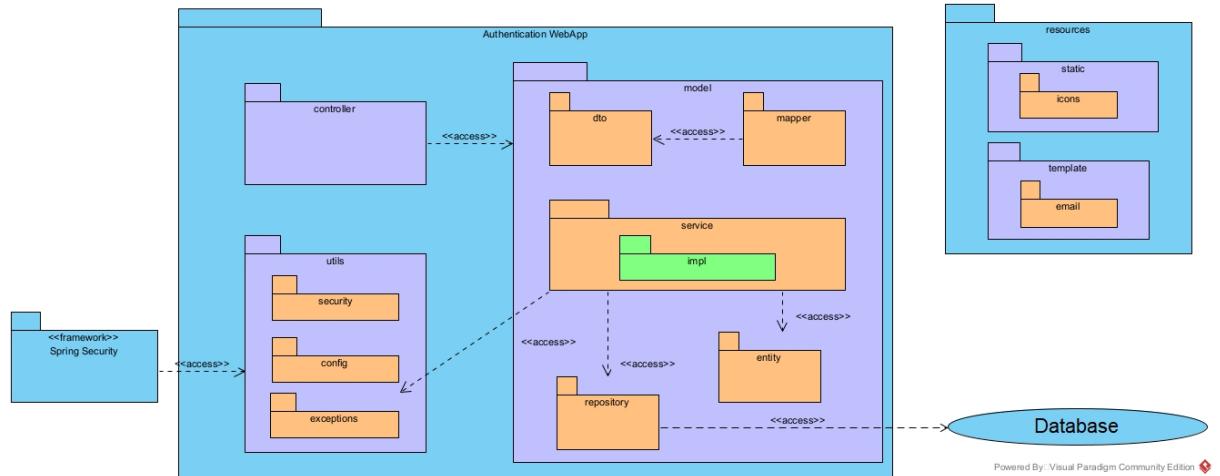


Figura 4.8: Package Diagram

Tale pattern permette una separazione netta tra i componenti di controllo, presentazione e gestione dei dati.

Nel caso specifico, le **View** sono contenute nel package "resources", che contiene tutte le interfacce con le quali il generico User interagisce (nel package "template").

Il package "static" contiene il file di stile .css, mentre il package "icons" contiene tutte le immagini che abbiamo utilizzato per modellare le pagine html.

All'interno del package "template" troviamo il package "email" che contiene il template utilizzato per la generazione dell'email automatica che l'utente riceverà in fase di registrazione o nel caso in cui richiedesse di recuperare la password. Inoltre, troviamo:

- **Model:** Nell'MVC, il Model rappresenta la logica di gestione dei dati, in quanto fornisce al controller i servizi ("service" package) e la logica di implementazione. Il package Model, oltre ai servizi, include diversi componenti:

1. *DTO (Data Transfer Objects):* I DTO (package "dto") sono oggetti utilizzati per trasferire dati tra le diverse componenti dell'applicazione. Servono come contenitori per le informazioni necessarie durante la comunicazione tra il Model, la View e il Controller.
2. *Repository (package "repository"):* I repository sono responsabili dell'accesso ai dati persistenti, come ad esempio i dati presenti in un database. Questi oggetti forniscono metodi per recuperare, salvare, aggiornare o eliminare dati dallo storage persistente.

Nel nostro progetto, il package “repository” implementa (ed estende) l’interfaccia JpaRepository fornita dal modulo Spring Data.

3. *Entity (package “entity”)*: sono le entità vere e proprie del nostro dominio. In particolare (verrà mostrato nel class diagram di dettaglio) nel nostro progetto esse sono: User, Role, Token.
4. *Mapper (package “mapper”)*: è un componente talvolta utilizzato nel pattern DTO (non è sempre necessario). In particolare si occupa di gestire la conversione di oggetti di tipo DTO in entity e viceversa. Solitamente viene utilizzato dal controller ed ha lo scopo di separare la logica di conversione dei dati dalla logica di business, in modo da mantenere un’architettura pulita e modulare, in cui ogni componente si occupa di un singolo compito specifico.

- **Controller:** Il suo compito principale è quello di ricevere le richieste dell’utente provenienti dalla View, elaborarle e richiamare i servizi corretti per soddisfarle.  
A valle dei risultati forniti dai service, il Controller si occuperà di aggiornare correttamente la View da restituire all’Utente.

Troviamo, infine, il package “utils” all’interno del quale abbiamo importato moduli di Spring Security, moduli di configurazione delle connessioni (e di Spring) e le eccezioni. Il tutto nei rispettivi package.

## 4.7 Class Diagram

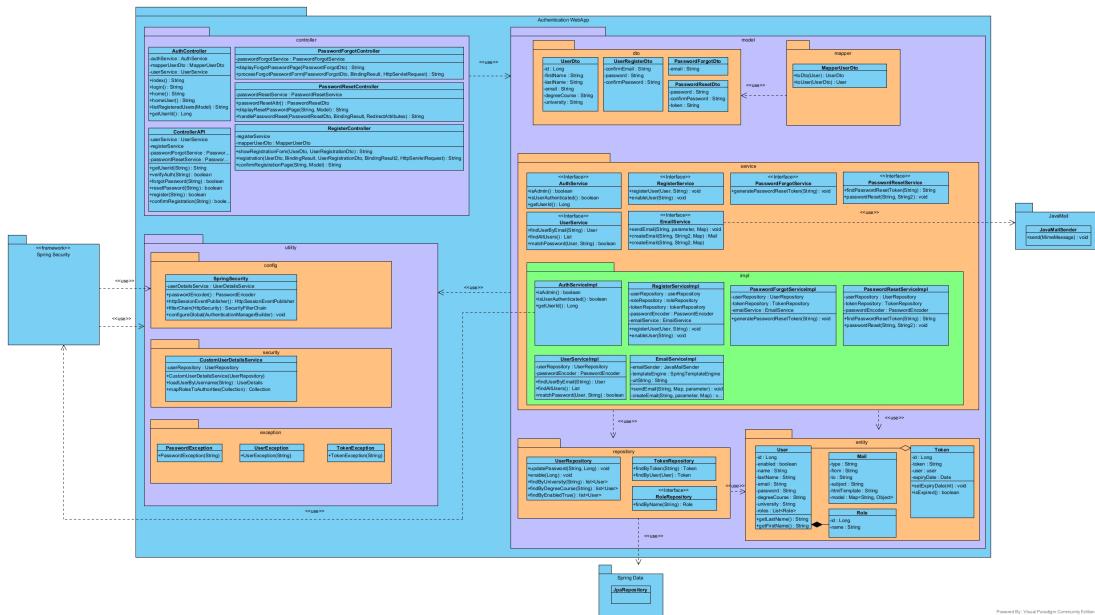


Figura 4.9: Class Diagram

#### 4.7.1 Register

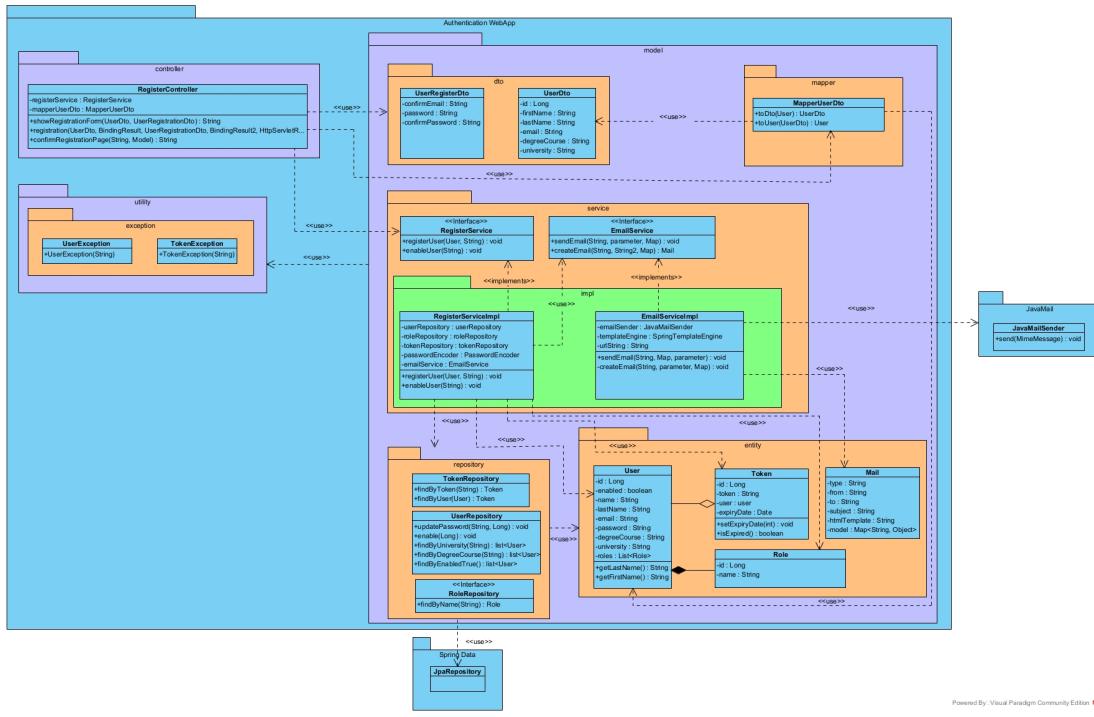


Figura 4.10: *Register* class diagram

#### 4.7.2 Authentication

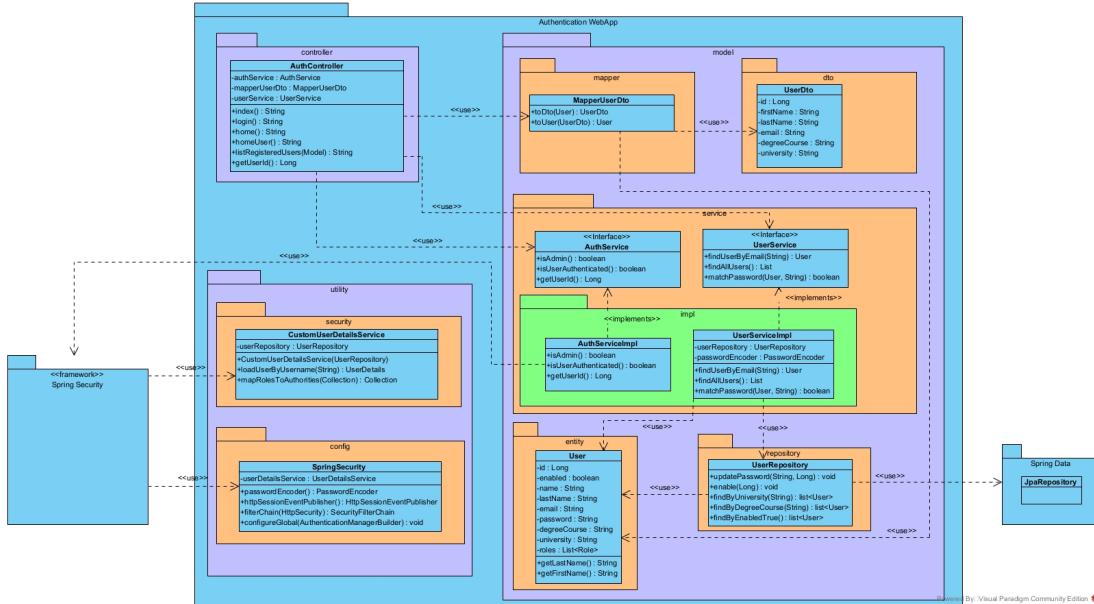


Figura 4.11: *Authentication* class diagram

#### 4.7.3 Forgot Password

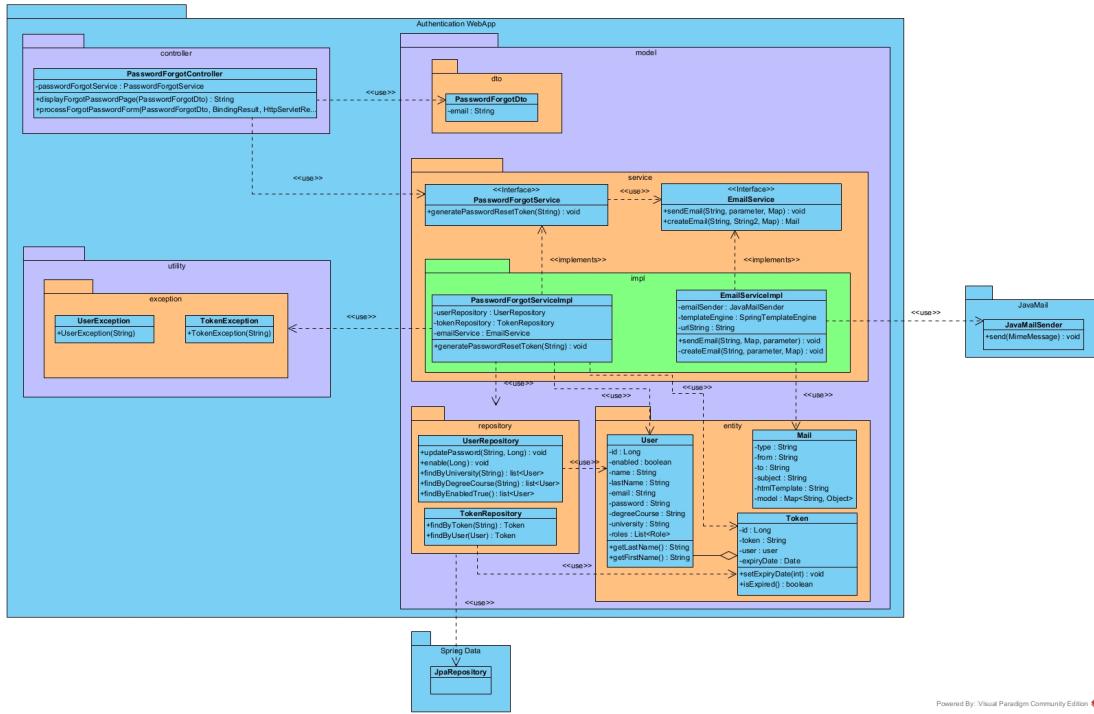


Figura 4.12: Forgot password class diagram

#### 4.7.4 Reset Password

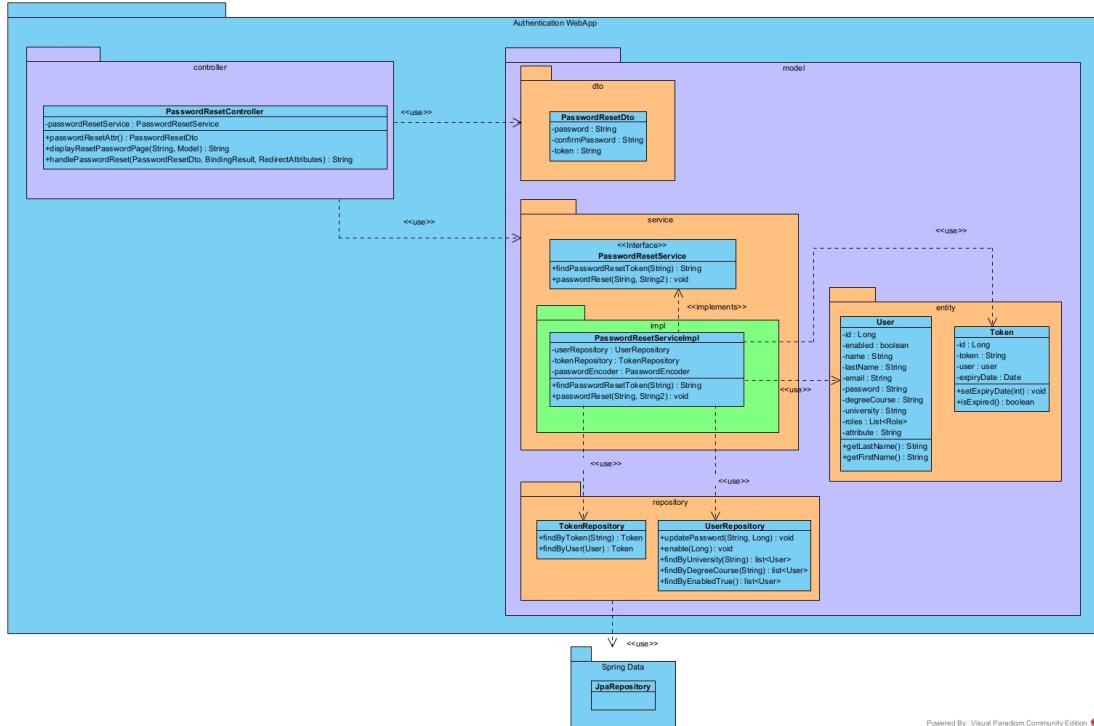


Figura 4.13: Reset password class diagram

## 4.8 Modello ER

L'Entity-Relationship Diagram (ERD) è uno strumento di modellizzazione dei dati utilizzato per descrivere le relazioni tra le entità di un sistema, ad esempio un database.

E' una vista molto importante in quanto consente di visualizzare e progettare la struttura del database in modo chiaro e conciso.

Inoltre, può aiutare a identificare potenziali problemi di progettazione del database e a prevenire errori che potrebbero renderlo inefficiente.

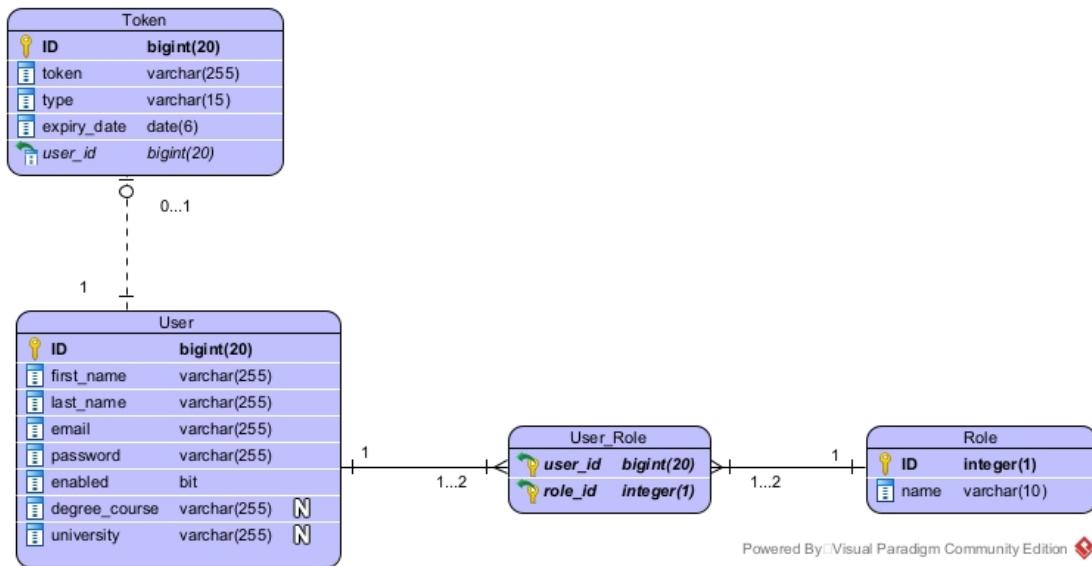


Figura 4.14: DB entity relationship

La tabella User ha come chiave primaria *ID*. Lo User è relazionato con la tabella Role tramite un'associazione molti a molti. In particolare una istanza User può avere uno o due ruoli (nel caso in cui l'utente sia anche admin).

La tabella Token partecipa con cardinalità 0..1 alla relazione con la tabella User, in quanto ogni utente può avere un solo token ad esso associato. Il token può essere di due tipologie: "Registration" e "Password Reset". In questo caso abbiamo preferito che la tabella Token contenesse la FK *user\_id*.

In tal modo ogni volta che si va a prelevare un token dal database, sarà possibile risalire subito all'utente ad esso associato.

# Capitolo 5

## Attività di implementazione

### 5.1 Sequence Diagram

Il sequence diagram di dettaglio è un diagramma UML che viene utilizzato per mostrare la sequenza dettagliata di azioni e scambi di messaggi tra oggetti o componenti durante l'esecuzione di un particolare scenario o caso d'uso. I sequence di dettaglio sono spesso utilizzati per analizzare e migliorare le prestazioni di un sistema, in quanto consentono di identificare eventuali ritardi o inefficienze nelle comunicazioni tra gli oggetti o componenti. Sono anche utili per documentare in dettaglio il comportamento del sistema e per comunicare le informazioni tecniche a sviluppatori, tester e altri membri del team tecnico.

#### 5.1.1 Register

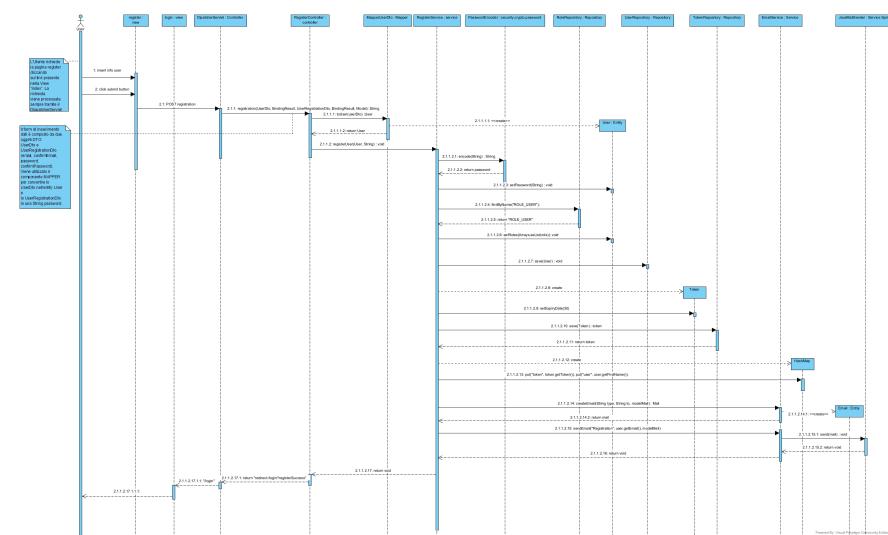


Figura 5.1: *Register* sequence diagram di dettaglio

**Scenario di successo:** L'utente compila il form di registrazione e clicca su “register”. Il DispatcherServlet di Spring, raccoglie la richiesta HTTP di tipo POST inviata dall'utente e la indirizza verso il RegisterController, richiamandone la funzione *registration(UserDto, BindingResult, UserRegistrationDto, BindingResult, Model)*. Il controller converte l'UserDto in User («create» new User) utilizzando il componente MapperUserDto. Viene invocata la funzione *registerUser(User, String) : void* del RegisterService, la quale cripta la password (tramite il PasswordEncoder di Spring), setta il ruolo “ROLE\_USER” e invoca lo *UserRepository (save(User))* per memorizzare il nuovo utente nel DB. Dopodichè viene creato un nuovo token associato allo User tramite: *new Token(null, UUID.randomUUID().toString(), "Registration", user, null)*. Il token sarà valido per 30 minuti.

Tramite un oggetto di tipo HashMap costruiamo il modelMail, il quale conterrà il token appena creato ed il nome dello User relativo ad esso.

Il modelMail verrà inviato come parametro alla funzione *createEmail (String type, String to, Modelmail): Mail* del modulo EmailService che si occuperà di creare un oggetto di tipo Email e di inviarlo tramite la *sendEmail*. L'EmailService si interfaccia con il JavaMailSender tramite la funzione *send(email)*. A questo punto l'utente verrà reindirizzato alla pagina di login.

**Scenario alternativo - Utente compila erroneamente i campi del form di registrazione:** L'utente inserisce i dati della registrazione. Se rispetta i vincoli del form (vedere sezione “Testing” per maggiori dettagli) lo scenario procede come quello di successo. Altrimenti l'utente viene rimandato alla pagina di registrazione e visualizzerà gli errori commessi nella compilazione del form.

**Scenario alternativo - L'utente è già registrato:** Se i dati nel form rispettano i vincoli, il sistema controlla se esiste già un utente registrato con la stessa email (*userRepository.findByEmail(user.getEmail())*).

- Se esiste si cerca il token relativo all'utente(*tokenRepository.findByUser(existingUser)*). Se il token esiste, è di tipo “Registration” ed è scaduto, allora verrà eliminato il token (*tokenRepository.delete(token)*) ed il relativo utente (*userRepository.delete(existingUser)*). L'esecuzione continuerà come nel caso di successo. In altre parole, l'utente ha ricevuto l'email di conferma registrazione ma non ha mai completato la procedura.
- Altrimenti viene lanciata l'eccezione *UserException("There is already an user registered with the same email")*.

### 5.1.2 Confirm register

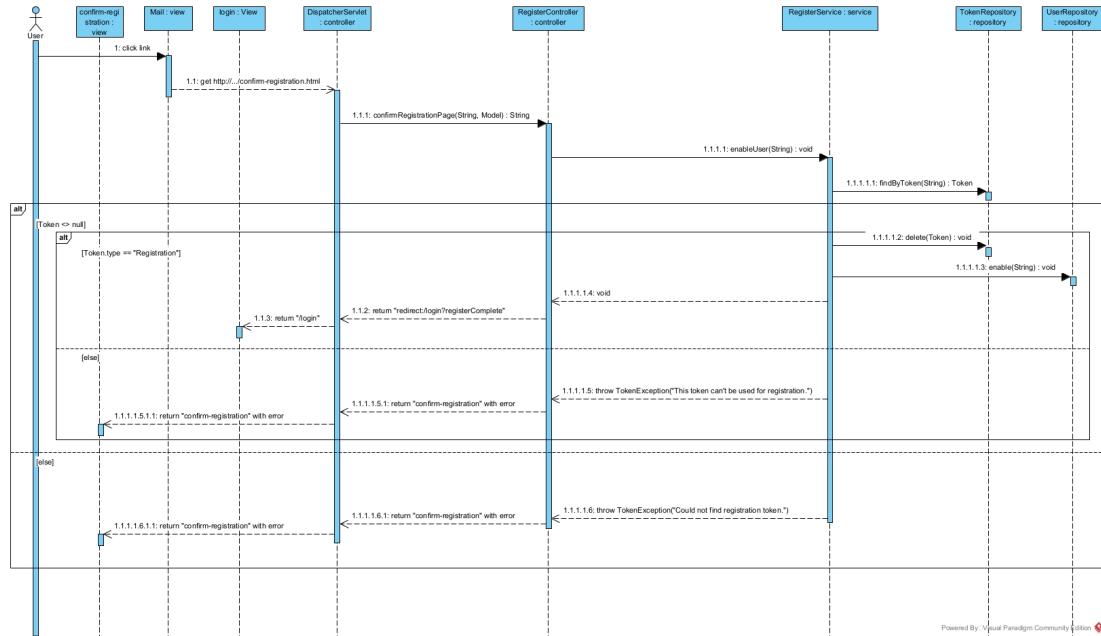


Figura 5.2: *Confirm registration* sequence diagram di dettaglio

**Scenario di successo:** L’utente, dopo aver ricevuto la mail, clicca sul link, generando una chiamata di tipo HTTP di tipo GET verso il DispatcherServlet. Quest’ultimo indirizza la chiamata al RegisterController tramite il metodo *confirmRegistrationPage(String, Model)*: *String*.

Il controller richiama i metodi della classe RegisterService la quale prima cancella il token associato all’utente tramite il TokenRepository (*delete(Token) : void*), poi abilita l’utente ad effettuare il login (*enableUser(String)*). Il controllo ritorna al RegisterController che tramite il DispatcherServlet restituisce all’utente la pagina di login.

**Scenario alternativo - L’utente clicca sul link ricevuto tramite email, ma il token è già scaduto:** Il RegisterService lancia l’eccezione *TokenException*(“*Token has expired, please request a new registration.*”). Viene restituita all’utente la pagina di registrazione.

**Scenario alternativo - Esiste il token associato all’utente ma non è di tipo “Registration”:** In tal caso il RegisterService lancia l’eccezione *TokenException*(“*This token can’t be used for registration*”), che viene raccolta dal RegisterController, il quale aggiorna la vista “confirm-registration” stampando a video gli errori.

**Scenario alternativo - Il token non esiste:** Il RegisterService lancia l’eccezione *TokenException*(“*Could not find registration token*”), che viene raccolta dal RegisterController, il quale

aggiorna la vista “confirm-registration” stampando a video gli errori.

### 5.1.3 Login

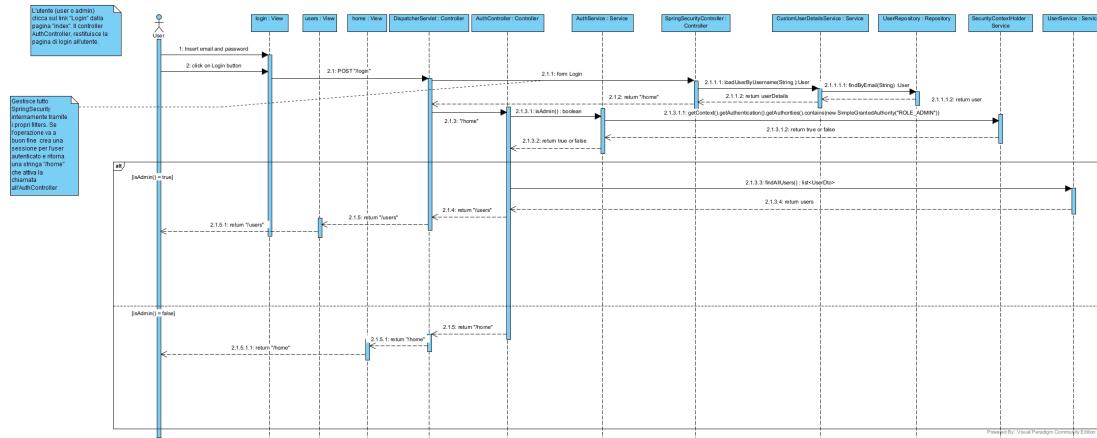


Figura 5.3: *Login* sequence diagram di dettaglio

**Scenario di successo:** L’utente si trova nella pagina “login” ed inserisce username e password nell’apposito form. Il Dispatcher Servlet riceve una chiamata di tipo HTTP POST e la gira allo SpringSecurityController. Da questo momento il controllo passa a Spring, in quanto grazie ai suoi filtri interni gestisce l’autenticazione, l’assegnazione dei ruoli e la creazione di una sessione per quel determinato utente.

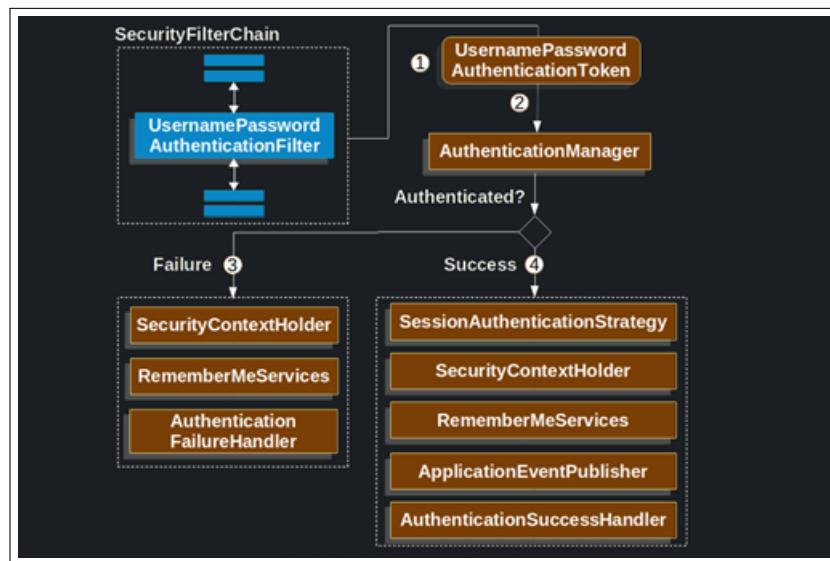


Figura 5.4: Autenticazione con username e password

Quando l’utente inserisce username e password, Spring estrae un UsernamePasswordAuthenticationToken dalla richiesta HttpServletRequest.

Lo UsernamePasswordAuthenticationToken viene passato all'AuthenticationManager che provvederà all'autenticazione prelevando dal database (tramite la funzione *loadUserByUsername(String):User*) l'utente (cercandone l'email).

Inoltre, viene effettuato (automaticamente da Spring) il controllo del campo *enable* dell'user. Questo campo indica se l'utente ha completato correttamente la registrazione (confermando l'email) o meno.

Se l'autenticazione va a buon fine allora viene settata sull'oggetto SecurityContextHolder. Quest'ultimo mette a disposizione diversi metodi come: *getContext(),getAuthentication(), getAuthorities()* che rispettivamente forniscono contesto, sessione e ruolo dell'utente.

Nel caso in cui l'autenticazione fallisse, viene cancellato il SecurityContextHolder e viene ritornata la pagina indicata nella configurazione delle richieste http (*SecurityFilterChain(HttpSecurity)*).

Al termine di questi controlli viene restituito come successUrl “/home” che tramite il dispatcher arriva all'AuthController. Quest'ultimo richiama la funzione *isAdmin()* dell'AuthService, per controllare se ad autenticarsi è stato l'admin o un utente.

Nel caso in cui ad autenticarsi è l'admin, l'AuthController richiama la funzione *findAllUsers():list<UserDto>* tramite lo UserService, per ottenere una lista di tutti gli utenti registrati e mostrarla a video all'admin.

Altrimenti, nel caso in cui ad autenticarsi è lo user, l'AuthController restituirà la dashboard personalizzata (home dell'utente).

**Scenario alternativo - L'utente tenta il login senza aver completato la registrazione (non ha cliccato sul link ricevuto tramite email):** In questo caso, i controlli effettuati da Spring, generano un errore. Il SecurityContextHolder viene cancellato e viene invocato l'AuthenticationFailureHandler che gestirà l'eccezione.

Infine l'AuthController restituirà (tramite il Dispatcher) all'utente la vista “/login?error”.

Questo scenario vale anche per gli utenti non registrati che tentano di effettuare il login.

### 5.1.4 Forgot Password

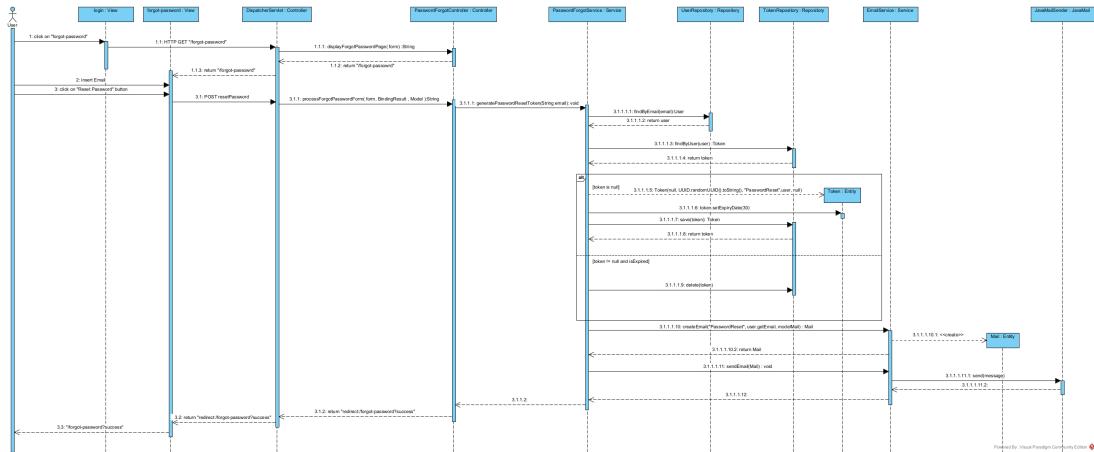


Figura 5.5: *Forgot Password* sequence diagram di dettaglio

**Scenario di successo:** L'utente si trova nella pagina “/forgot-password” (dopo averla richiesta) ed inserisce la propria email nel form. Tramite il Dispatcher viene richiamata la funzione *processForgotPasswordForm(PasswordResetDto, BindingResult, Model) : String* del PasswordForgotController. Attraverso il PasswordForgotService viene richiamata la funzione *generatePasswordResetToken(String email) : void*, la quale preleva l'utente dallo UserRepository (per controllare se ad esso è associato già un token). Dato che siamo nello scenario di successo, il token viene generato ex novo e salvato nel repository tramite la *save(token):Token*.

A questo punto viene creata l'email con all'interno un link (token) tramite la funzione *createEmail(String type, user.getEmail, modelMail)* dell' EmailService che a sua volta delegherà l'invio dell'email al servizio JavaMailSender tramite la funzione *send(email)*.

L'utente verrà reindirizzato alla pagina “/forgot-password?success”.

**Scenario alternativo - Utente non registrato:** La chiamata *findByEmail(email): User* restituisce un Utente null. Vuol dire che non è stato trovato alcun utente registrato corrispondente alla email inserita nel form.

Dunque viene lanciata l'eccezione *UserException* (“We could not find an user for that e-mail address.”). L'utente viene reindirizzato alla pagina “/forgot-password”.

**Scenario alternativo - Utente non autenticato (registrazione incompleta):** Viene prelevato l'utente, viene prelevato il token e si controlla se è di tipo “PasswordReset”. Se non lo è viene lanciata l' eccezione *TokenException* (“You must confirm your registration before you can change your password.”) e l'utente viene reindirizzato alla pagina “forgot-password”.

**Scenario alternativo - Token scaduto:** Viene prelevato l'utente e in seguito il token ad esso associato. Il token già esistente viene eliminato e subito ne viene creato uno nuovo. Lo scenario continua come quello di successo.

### 5.1.5 Reset password

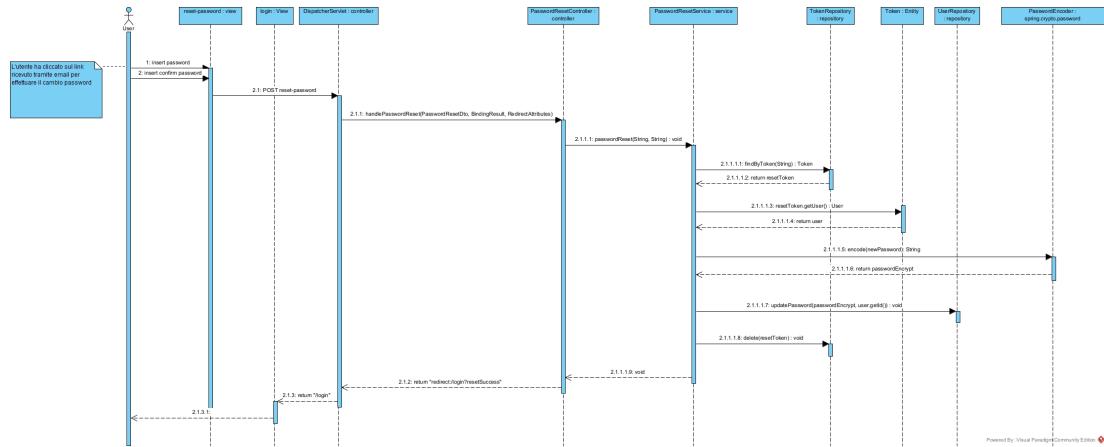


Figura 5.6: *Reset Password* sequence diagram di dettaglio

**Scenario di successo:** L'utente ha cliccato sul link ricevuto tramite email ed è stato rein- dirizzato dal dispatcher servlet alla pagina “/reset-password”. L'utente inserisce *password* e *confirm password* nel form. A questo punto viene chiamato l'handlePasswordReset del PasswordResetController, il quale tramite il servizio PasswordResetService preleva prima il token dal TokenRepository e in seguito l'utente ad esso associato (dallo UserRepository). Tramite il PasswordEncoder viene criptata la nuova password inserita e viene aggiornata nello UserRepository (se soddisfa determinati criteri spiegati negli scenari alternativi). A questo punto viene cancellato il token e l'utente viene reindirizzato alla pagina di “login?resetSuccess”.

**Scenario alternativo - Il token è scaduto:** Nella funzione displayResetPasswordPage() viene effettuato un controllo sul token (se è ancora valido). Prende anche il model perché nel caso il token fosse scaduto verrà restituita una pagina con errore “reset-password”. L'eccezione che visualizzerà l'utente è TokenException(“Token has expired, please request a new password reset.”);

**Scenario alternativo - Il token non è di tipo “ResetPassword”:** Questo vuol dire che l'utente ha chiesto il reset della password senza aver completato la fase di registrazione (ovvero non ha cliccato sul link ricevuto in fase di registrazione).

In questo caso il PasswordResetController lancerà un'eccezione di tipo *TokenException*(“This

*token can't be used for password reset.”).*

L’utente visualizzerà la pagina “password-reset” con il relativo messaggio di errore.

**Scenario di non successo - La nuova password scelta dall’utente coincide con quella precedente:** Prima di criptare la nuova password essa viene confrontata con quella vecchia (alla quale si può accedere dopo aver prelevato l’utente dal TokenRepository tramite il token ad esso associato).

Se le due stringhe matchano allora verrà lanciata l’eccezione *PasswordException* (“*New password can't be equal to the old one*”). Dopodichè l’utente verrà reindirizzato sulla pagina “password-reset” con errore.

### 5.1.6 Logout

In un’applicazione Spring, il processo di logout è gestito dallo Spring Security.

Lo Spring Security fornisce un’implementazione predefinita del logout, che può essere attivata (o anche customizzata) attraverso un’URL dedicata (solitamente nel controller o in file di configurazione dei filtri di Spring).

Nella nostra applicazione, il logout è completamente gestito da Spring.

La richiesta http può essere sia di tipo GET che di tipo POST e l’URL ad esso associato può essere personalizzato nel modulo che gestisce la configurazione di SpringSecurity (*SpringSecurity.java*), come mostrato di seguito:

```
.logout()
    .logoutUrl("/logout")
    .permitAll()
    .invalidateHttpSession(true)
    .clearAuthentication(true)
```

Figura 5.7: *Logout* instruction

Tra le varie opzioni di logout dell’applicazione possono essere specificati:

- *.invalidateHttpSession(true)*: invalida la sessione HTTP associata all’utente al momento del logout.
- *.clearAuthentication(true)*: cancella l’autenticazione dell’utente al momento del logout.
- *.deleteCookies("JSESSIONID")*: utilizzato nella configurazione di Spring Security per eliminare i cookie specificati dal client al momento del logout.

## 5.2 Deployment Diagram

Il deployment diagram fornisce una visione ad alto livello dell'architettura del sistema e aiuta a identificare eventuali problemi legati alla configurazione hardware e di rete. Viene utilizzato per visualizzare come i componenti software e hardware sono interconnessi e come sono distribuiti su diverse macchine.

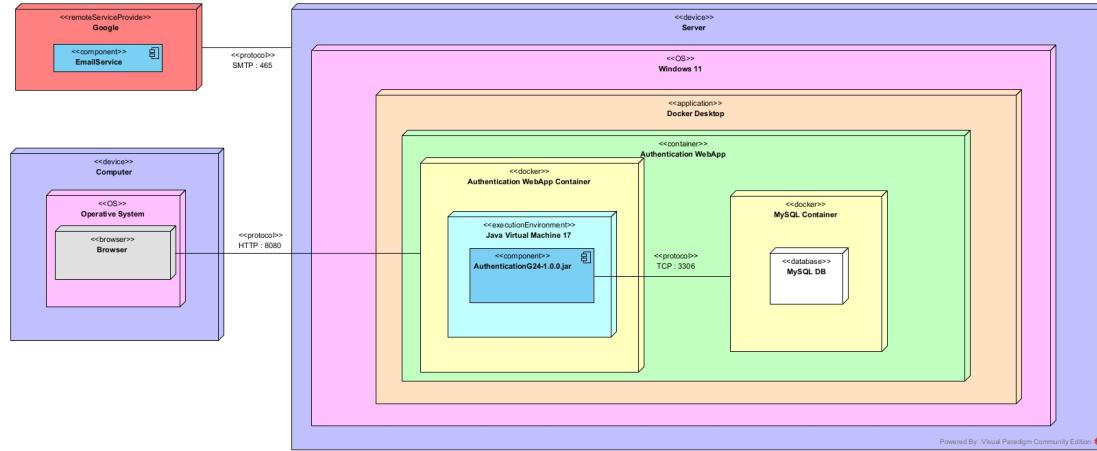


Figura 5.8: Deployment Diagram

Nel nostro caso l'applicazione viene eseguita su un Computer che agisce da Server sul quale è installato il sistema operativo Windows 11. Viene eseguita l'applicazione Docker Desktop che crea un container nel quale vengono istanziati due Docker, uno per l'applicazione ed uno per Database MySQL, interconnessi da una rete privata creata ad hoc da Docker. In particolare, il Database viene connesso all'applicazione attraverso il protocollo TCP alla porta 3306. Per eseguire l'applicazione all'interno di Docker viene creato un ambiente di esecuzione basato su JVM17.

Inoltre, viene resa disponibile all'esterno del Docker la possibilità di connettersi all'applicazione tramite qualsiasi browser e qualsiasi sistema operativo, utilizzando il protocollo HTTP sulla porta 8080. Inoltre, tramite il protocollo SMTP, viene richiamato il servizio Email di Google.

# Capitolo 6

## Integrabilità

### 6.1 API locali

Di seguito verranno documentate le principali funzionalità delle interfacce offerte dai vari controller al Dispatcher.

#### 6.1.1 RegisterController

```
@PostMapping  
public String registration(@Valid @ModelAttribute("user") UserDto userDto,  
                           BindingResult resultUser, @Valid @ModelAttribute("userRegistration")  
                           UserRegistrationDto userRegDto, BindingResult resultReg, Model model)
```

#### Input

*@Valid* → indica al framework di validare l'oggetto annotato.

*@ModelAttribute* → indica che l'oggetto "userDto" deve essere estratto dalla richiesta HTTP e utilizzato come attributo del modello.

*BindingResult* → indica che il framework deve registrare i risultati della validazione dell'oggetto "userDto", in modo da poter gestire eventuali errori di validazione.

*Model* → rappresenta il modello utilizzato dalla vista per visualizzare i dati.

#### Return

Il metodo restituisce una stringa:

- "register": nel caso in cui si sia verificata un'eccezione;
- "redirect:/login?registerSuccess": nel caso in cui la registrazione vada a buon fine.

## Descrizione

Questa funzione riceve in ingresso due oggetti di tipo DTO: userDto, che possiede tutti gli attributi dell'utente tranne i campi password, confirm email e confirm password, i quali sono invece passati al model tramite l'oggetto userRegDto.

In tal modo l'oggetto userDto può essere riutilizzato come parametro per altre funzionalità senza esporre *dati sensibili*.

In particolare il metodo register come prima cosa controlla che i campi password e confirm password coincidano, dopodiché esegue lo stesso controllo sui campi email e confirm email.

Se i controlli vanno a buon fine viene richiamata la funzione del **RegisterService**:

```
public void registerUser(User user, String password) throws UserException
```

Riceve in input l'Utente inserito nel form di registrazione e la password. Effettua un controllo nello UserRepository (cercando se esiste un utente registrato con la stessa email inserita nel form).

Se esiste già un utente con quella email, viene lanciata l'eccezione UserException, altrimenti il metodo provvederà a salvare l'utente nel database, genererà un nuovo token (da validare) ed una email da inviare all'user appena registrato (ma non ancora abilitato).

Se l'utente convaliderà la propria email prima della scadenza del token (30 minuti), il RegisterController richiamerà la funzione del RegisterService:

```
@Override  
@Transactional  
public void enable(String token) throws TokenException
```

La quale provvederà ad abilitare l'utente tramite il database e cancellerà il token creato in precedenza.

Da quanto si evince dalla signature, questo metodo lancia delle eccezioni di tipo TokenException nel caso in cui il token:

1. Non esiste;
2. Esiste ma è scaduto;
3. Esiste ma non è di tipo "Registration".

### 6.1.2 AuthController

La funzionalità più importante che fornisce l'AuthController è sicuramente il login. In questo caso va fatto un discorso a parte, in quanto sarà Spring ad effettuare tutte le operazioni, le quali sono spiegate nel dettaglio nel capitolo "Implementazione" sottoparagrafo "Login".

Questo controller utilizza come service la classe **AuthService**, della quale di seguito riportiamo l'interfaccia.

- La funzione:

```
@Override
public boolean isAdmin()
```

Controlla se l'utente che ha effettuato il login è un user o un admin.

Restituisce *true* se è un admin, *false* altrimenti.

- La funzione:

```
@Override
public boolean isAuthenticated()
```

Controlla se un utente ha effettuato l'autenticazione. In caso positivo ritornerà *true*, altrimenti *false*.

## 6.2 ForgotController

Di seguito si riporta il metodo del controller che viene richiamato nel momento in cui l'utente clicca sul link “Forgot-password” .

```
@PostMapping
public String processForgotPasswordForm(@ModelAttribute("forgotPasswordForm")
                                         @Valid PasswordForgotDto form, BindingResult result, Model model)
```

### Input

*PasswordForgotDto form* → la struttura dati (form) all'interno del quale l'utente inserirà la propria email. Anche per questo motivo la richiesta è di tipo POST. In tal modo l'email dell'utente verrà inserita nel corpo della richiesta http e non nell'URL.

*BindingResult result* → indica che il framework deve registrare i risultati della validazione dell'oggetto "form", in modo da poter gestire eventuali errori di validazione.

*Model model* → rappresenta il modello utilizzato dalla vista per visualizzare i dati.

### Return

I possibili scenari di errore ritornano la stringa “forgot-password”. Essi sono:

- Il token non esiste;
- Il token esiste ma è scaduto;
- Il token non è di tipo “Password Reset”;

- L'utente non è presente nel database.

In caso di successo verrà ritornata la pagina `"redirect:/forgot-password?success"`.

## Descrizione

Se l'utente non commette errori a compilare il form, viene richiamata la seguente funzione del PasswordForgotService:

```
@Override
public void generatePasswordResetToken(String email) throws UserException, TokenException
```

Ha come unico parametro di input la String email. Per comprendere il modo in cui vengono lanciate le eccezioni, riportiamo i possibili scenari di errore:

1. Il token esiste ma è scaduto (TokenException);
2. Il token non è di tipo “Password Reset” (TokenException);
3. L'utente non è presente nel database (UserException).

Se invece il token non esiste, il metodo provvederà a crearne uno nuovo e ad inviare una email all'indirizzo passato come parametro.

L'utente dovrà ora confermare l'identità cliccando sul token per far partire una chiamata di tipo POST al ResetPasswordController (spiegato di seguito).

### 6.2.1 ResetController

Nel momento in cui l'utente clicca sul link ricevuto tramite email (per resettare la password), il PasswordResetController gli mostrerà la pagina “reset- password”.

L'utente dovrà compilare un form con i campi: password, confirm password.

Dopodichè verrà lanciata una richiesta di tipo http POST raccolta e gestita dalla funzione:

```
@PostMapping
public String handlePasswordReset(@ModelAttribute("passwordResetForm")
        @Valid PasswordResetDto form, BindingResult result,
        RedirectAttributes redirectAttribute)
```

## Input

`PasswordEncoder form` → form nel quale l'utente inserirà i campi password e confirm password.

`BindingResult result` → indica che il framework deve registrare i risultati della validazione dell'oggetto “form”, in modo da poter gestire eventuali errori di validazione.

*RedirectAttributes redirectAttributes* → consente di passare dati tra due richieste HTTP successive, in questo caso dalla richiesta POST alla successiva richiesta GET (nel caso di errori infatti, verrà restituita di nuovo la pagina con il form per il reset della password).

### Return

Il metodo ritorna la stringa `"redirect:/reset-password?token=" + form.getToken()` nel caso in cui le due password inserite non corrispondano, altrimenti ritorna la stringa `"redirect:/login?resetSuccess"`.

### Descrizione

Il metodo controlla se le password inserite coincidono e che non ci siano errori nel form.

Se l'esito è positivo, richiama la funzione del PasswordResetService:

`@Transactional`

```
public void passwordReset(String token, String newPassword) throws PasswordException
```

Come prima operazione viene prelevato l'utente associato al token (passato in input) dal database.

In tal modo si può accedere al campo password (dello User) che contiene ancora la vecchia password dell'utente.

L'eccezione PasswordException viene lanciata nel caso in cui la nuova password coincida con quella vecchia.

Nel caso di successo invece, la nuova password inserita viene criptata e aggiornata nel database.

Dopodiché viene cancellato il token.

## 6.3 REST API

Un'API, o *Application Programming Interface* (interfaccia di programmazione delle applicazioni), è un procedura che consente a un'applicazione o a un servizio (client) di accedere a una risorsa all'interno di un altro sistema (server), definendo una serie di regole che determina il modo in cui possono connettersi e comunicare tra loro.

Un'API REST è conforme ai principi di progettazione di stile architetturale REST, o representational state transfer. Possono però essere sviluppate utilizzando qualsiasi linguaggio di programmazione e supportano una varietà di formati di dati.

Affinché un'API sia considerata RESTful, deve rispettare i criteri indicati di seguito:

- Un'architettura *client-server* composta da client, server e risorse, con richieste gestite tramite HTTP.

- Una *comunicazione stateless*, il che vuol dire che ogni richiesta è distinta e non connessa, deve includere tutte le informazioni necessarie per elaborarla. In altre parole, le API REST non richiedono alcuna sessione lato server.
  - Un'*interfaccia uniforme* per i componenti, in modo che le informazioni vengano trasferite in una forma standard. Ciò impone che le risorse richieste siano identificabili e separate dalle rappresentazioni inviate al client.
  - Un *sistema su più livelli* che organizza il server che si occupa di recuperare le informazioni richieste in gerarchie, invisibile al client.

Un'API REST trasferisce le informazioni tra client e server utilizzando diversi formati di rappresentazione dei dati, come **JSON** (Javascript Object Notation) che è indipendente dal linguaggio e facilmente leggibile da persone e macchine, testo semplice o anche altri formati.

Esse utilizzano metodi HTTP per identificare il tipo della richiesta e le operazioni che possono essere eseguite sui dati. Ad esempio, una richiesta GET viene utilizzata per ottenere dati; una richiesta POST per inviare nuovi dati; una richiesta PUT per aggiornare dati esistenti e una richiesta DELETE per cancellarli.

Sono diventate un'opzione popolare per la comunicazione tra diversi sistemi grazie alla loro:

- **scalabilità**, in quanto possono gestire grandi volumi di traffico e richieste senza compromettere le prestazioni del sistema;
  - **flessibilità**, ciò le rende adatte per integrare sistemi diversi e per lo sviluppo di applicazioni multi-piattaforma;
  - **facilità di sviluppo e implementazione**, poiché si basano su standard aperti come HTTP e JSON.

Di seguito si riportano le interfacce delle nostre API.

<b>Metodo</b>	<b>Link</b>	<b>Formato input (JSON)</b>	<b>Output di successo</b>	<b>Output di fail</b>
GET	/userid	{“email”：“...”}	User_id (int)	“NULL”
POST	/verifyauth	{ “email” : “...”, “password” : “...”}	true	false
POST	/forgot-password	{“email”：“...”}	true	false
POST	/reset-password	{“token” : “...”, “password” : “...”}	true	false

Metodo	Link	Formato input (JSON)	Output di successo	Output di fail
POST	/register	{“firstName”:”...”, “lastName”:”...”, “email” : ”...”, “password” : ”...”, “degreeCourse”:”...”, “university”:”...”}	true	false
POST	/confirm-registration	{“token” : ”...”}	true	false

**N.B.** I campi di input *email*, *password*, *token*, *firstName*, *lastName*, *degreeCouse*, *university* sono di tipo String, di dimensione massima pari a 255.

### 6.3.1 Swagger

Per documentare le nostre REST-API abbiamo utilizzato come strumento **Swagger**, uno strumento open source che viene utilizzato per creare, documentare e testare API (Application Programming Interface). In più, offre un’interfaccia utente interattiva per esplorare e testare le API.

É possibile interagire, cliccando *qui*.

# Application G24 API

1.0.0 OAS3

Servers

http://localhost:8080

controller-api

GET /api/userid

POST /api/verifyauth

POST /api/register

POST /api/confirm-registration

POST /api/forgot-password

POST /api/reset-password

Figura 6.1: Interfaccia Swagger

# Capitolo 7

## Testing

Il testing è un processo di valutazione del software o del sistema per verificare se funziona come previsto e per trovare eventuali difetti o problemi. L'obiettivo principale è garantire la qualità del software o del sistema, identificando eventuali errori, difetti o problemi che possono impedire al software di funzionare correttamente o di soddisfare le esigenze degli utenti.

Di seguito, vengono proposte due tipologie di test effettuati:

- Testing basato sulle GUI dell'applicazione;
- Testing basato sui servizi API REST.

### 7.1 Testing basato sulle GUI

#### 7.1.1 Registrazione

Di seguito, si riportano i casi di test effettuati:

##### T1. Inserimento dati corretti

*Preconditions:* Utente non registrato

*Input:*

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcellini@studenti.unina.it

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:* Password12

*Confirm Password:* Password12

**Expected output:** You have registered our app! Check your email to complete your registration.

**Postcondition:** Nel DB l'utente viene salvato e il token viene generato, l'email viene inviata

**Result:** PASS

#### T2. Inserimento non totale dei dati - First Name

**Preconditions:** Utente non registrato

**Input:**

*First name:*

*Last name:* Porcellini

*Email:* p.porcellini@studenti.unina.it

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:* Password12

*Confirm Password:* Password12

**Expected output:** First name should not be empty

**Postcondition:** Utente non registrato

**Result:** PASS

#### T3. Inserimento non totale dei dati - Last Name

**Preconditions:** Utente non registrato

**Input:**

*First name:* Pierpaolo

*Last name:*

*Email:* p.porcellini@studenti.unina.it

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:* Password12

*Confirm Password:* Password12

**Expected output:** Last name should not be empty

**Postcondition:** Utente non registrato

**Result:** PASS

#### T4. Inserimento non totale dei dati - Email

**Preconditions:** Utente non registrato

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:*

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:* Password12

*Confirm Password:* Password12

**Expected output:** Email should be empty

**Postcondition:** Utente non registrato

**Result:** PASS

#### T5. Inserimento non totale dei dati - Password

**Preconditions:** Utente non registrato

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcellini@studenti.unina.it

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:*

*Confirm Password:* Password12

**Expected output:** Password should have at least eight characters, one lowercase letter, one uppercase letter and one number or special character

**Postcondition:** Utente non registrato

**Result:** PASS

#### T6. Inserimento di una mail non corretta

**Preconditions:** Utente non registrato

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcellini

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:* Password12

*Confirm Password:* Password12

**Expected output:** Email has an invalid format

**Postcondition:** Utente non registrato

**Result:** PASS

#### T7. Corrispondenza errata tra Email e Confirm email

**Preconditions:** Utente non registrato

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcelni@studenti.unina.it

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:* Password12

*Confirm Password:* Password12

**Expected output:** The email fields must match

**Postcondition:** Utente non registrato

**Result:** PASS

#### T8. Corrispondenza errata tra Password e Confirm password

**Preconditions:** Utente non registrato

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcellini@studenti.unina.it

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:* Password12

*Confirm Password:* Pafsword12

**Expected output:** The password fields must match

**Postcondition:** Utente non registrato

**Result:** PASS

#### T9. Inserimento dati corretti, utente già registrato

**Preconditions:** Utente già registrato

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcellini@studenti.unina.it

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:* Password12

*Confirm Password:* Password12

**Expected output:** There is already an user registered with the same email

**Postcondition:** Utente non registrato

**Result:** PASS

#### T10. Utente già registrato con token di registrazione scaduto

**Preconditions:** Utente già registrato con token di registrazione scaduto

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcellini@studenti.unina.it

*Confirm Email:* p.porcellini@studenti.unina.it

*Password:* Password12

*Confirm Password:* Password12

**Expected output:** Token has expired, please request a new registration.

**Postcondition:** Utente non abilitato

**Result:** PASS

### 7.1.2 Login

Di seguito, si riportano i casi di test effettuati:

#### T1. Inserimento dati corretti

**Preconditions:** Utente già registrato

**Input:**

*Email:* p.porcellini@studenti.unina.it

*Password:* Password12

**Expected output:** Return page "/home/user.html"

**Postcondition:** Creazione della sessione

**Result:** PASS

#### T2. Inserimento di una password non corretta

**Preconditions:** Utente già registrato

**Input:**

*Email:* p.porcellini@studenti.unina.it

*Password:* Pafsword12

**Expected output:** Bad credentials

**Postcondition:** Accesso negato

**Result:** PASS

#### T3. Utente non abilitato

**Preconditions:** Utente registrato, ma non abilitato

**Input:**

*Email:* andrea.scognamiglio8@studenti.unina.it

*Password:* Pafsword12

**Expected output:** User not enabled

**Postcondition:** Accesso negato

**Result:** PASS

#### T4. Accesso con un utente non registrato

**Preconditions:** Utente non registrato

**Input:**

*Email:* france.bisogno@studenti.unina.it

*Password:* TestPwd1234

**Expected output:** Bad credentials

**Postcondition:** Accesso negato

**Result:** PASS

#### T5. Accesso come admin

**Preconditions:** Utente già registrato

**Input:**

Email: admin@admin

Password: admin

**Expected output:** Return page "/home/admin.html

**Postcondition:** Creazione della sessione

**Result:** PASS

#### 7.1.3 Forgot password

##### T1. Inserimento dati corretti

**Preconditions:** Utente già registrato, token di recupero password inesistente

**Input:**

Email: p.porcellini@studenti.unina.it

**Expected output:** You've successfully requested a new password reset! Check your email.

**Postcondition:** Generazione del token, opportunamente memorizzato nel DB, e invio della mail per il recupero password.

**Result:** PASS

##### T2. Inserimento dati non corretti

**Preconditions:** Utente non registrato

**Input:**

Email: i.galiero@studenti.unina.it

**Expected output:** We could not find an user for that e-mail address.

**Postcondition:** Recupero password negato

**Result:** PASS

##### T3. Utente non abilitato

**Preconditions:** Utente registrato, token di registrazione non confermato

**Input:**

Email: france.bisogno@studenti.unina.it

**Expected output:** You must confirm your registration before you can change your password.

**Postcondition:** Recupero password negato

**Result:** PASS

#### T4. Richiesta di un nuovo token ma il precedente non è scaduto

**Preconditions:** Utente registrato, token già inviato precedentemente ma non è ancora scaduto

**Input:**

Email: andrea.scognamiglio8@studenti.unina.it

**Expected output:** You've successfully requested a new password reset! Check your email.

**Postcondition:** Nessuna generazione di un nuovo token, rinvio della mail con lo stesso token precedente

**Result:** PASS

#### T5. Richiesta di un nuovo token ma il precedente è scaduto

**Preconditions:** Utente registrato, token precedente scaduto

**Input:**

Email: andrea.scognamiglio8@studenti.unina.it

**Expected output:** You've successfully requested a new password reset! Check your email.

**Postcondition:** Generazione di un nuovo token sovrascrivendo il token precedente, invio della mail con il nuovo token

**Result:** PASS

### 7.1.4 Reset Password

#### T1. Token di recupero password scaduto

**Preconditions:** Utente registrato richiede un reset password, il token è scaduto

**Input:**

Link: http://localhost:8080/reset-password?token=fc077faa-fe91-4499-a426-1efa2caf8df

**Expected output:** Token has expired, please request a new password reset.

**Postcondition:** Recupera password negato

**Result:** PASS

#### T2. Token di recupero password non esiste

**Preconditions:** Utente non richiede un reset password

**Input:**

Link: http://localhost:8080/reset-password?token=ciao-va-bene-12-5432

**Expected output:** Could not find password reset token.

**Postcondition:** Recupera password negato

**Result:** PASS

### T3. Token non adatto al recupero password

**Preconditions:** Utente non richiede un reset password

**Input:**

*Link:* http://localhost:8080/reset-password?token=8c02a938-32b9-480f-8a44-3c19b10c96be

**Expected output:** This token can't be used for password reset.

**Postcondition:** Recupera password negato

**Result:** PASS

### T4. Inserimento dei dati corretti

**Preconditions:** Utente registrato richiede un reset password

**Input:**

*Password:* TestPwd1234

*Confirm password:* TestPwd1234

**Expected output:** You have successfully reset your password!

**Postcondition:** Modifica della password effettuata

**Result:** PASS

### T5. Inserimento dei dati non corretti

**Preconditions:** Utente registrato richiede un reset password

**Input:**

*Password:* TesPwd1234

*Confirm password:* TestPwd1234

**Expected output:** The password fields must match

**Result:** PASS

### T6. Inserimento dei dati non corretti

**Preconditions:** Utente registrato richiede un reset password, tenta di inserire come nuova password quella precedente

**Input:**

*Password:* Password12

*Confirm password:* Password12

**Expected output:** New password can't be equal to the old one

**Result:** PASS

### T7. Inserimento dei dati non corretti

**Preconditions:** Utente registrato richiede un reset password

**Input:**

*Password:* Test12

*Confirm password:* Test12

**Expected output:** Password should have at least eight characters, one lowercase letter, one uppercase letter and one number or special character

**Result:** PASS

## 7.2 Testing basato sui servizi API REST

I test dell'API sono stati effettuati tramite l'utilizzo di **Postman**. Quest'ultimo è uno strumento molto utile per testare le API perché consente di testarle in modo efficiente e accurato, verificando i dati inviati e ricevuti e visualizzando facilmente le risposte del server. Nel dettaglio, invia richieste HTTP al server e verifica le risposte ricevute.

### 7.2.1 Registrazione

#### T1. Utente non esiste

**Preconditions:** Utente non registrato

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcellini@studenti.unina.it

*Password:* Password12

**Expected output:** true

**Postcondition:** Nel DB l'utente viene salvato e il token viene generato, l'email viene inviata

**Result:** PASS

The screenshot shows a Postman collection interface. The request is a POST to 'localhost:8080/api/register'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "firstName": "Pierpaolo",  
3   "lastName": "Porcellini",  
4   "email": "p.porcellini@studenti.unina.it",  
5   "password": "Password12",  
6   "degreeCourse": "",  
7   "university": ""  
8 }
```

The response status is 200 OK, time 21.28 s, size 338 B. The response body is 'true'.

Figura 7.1: Test case *registrazione* - Utente non registrato

#### T2. Utente già registrato con token di registrazione scaduto

**Preconditions:** Utente già registrato con token di registrazione scaduto

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcellini@studenti.unina.it

*Password:* Password12

**Expected output:** true

**Postcondition:** Nel DB l'utente viene sovrascritto e il token viene rigenerato, l'email viene inviata

**Result:** PASS

### T3. Inserimento dati corretti, utente già registrato

**Preconditions:** Utente già registrato

**Input:**

*First name:* Pierpaolo

*Last name:* Porcellini

*Email:* p.porcellini@studenti.unina.it

*Password:* Password12

**Expected output:** false

**Postcondition:** Utente non registrato

**Result:** PASS

## 7.2.2 Conferma registrazione

### T1. Utente registrato

**Preconditions:** Utente registrato, non abilitato

**Input:**

*Token:* ba40efd4-2b2e-496b-aae3-81cf10eacd0c

**Expected output:** true

**Postcondition:** Utente abilitato

**Result:** PASS

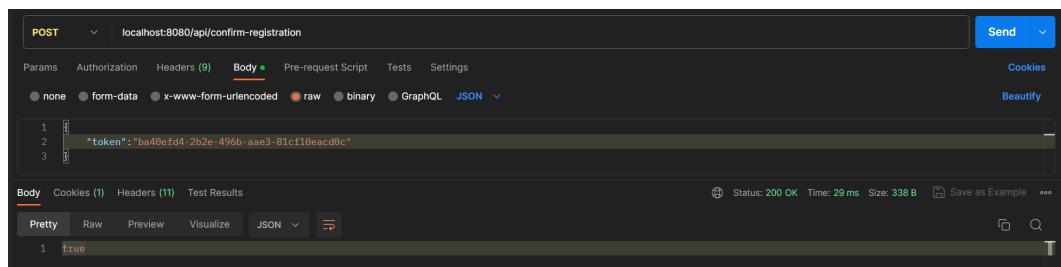


Figura 7.2: Test case *conferma registrazione* - Utente registrato

## T2. Utente non registrato

**Preconditions:** Utente non registrato

**Input:**

*Token:* 077faa-fe91-4499-a426-1efa2cafb8d

**Expected output:** false

**Result:** PASS

### 7.2.3 Verifica credenziali

#### T1. Utente registrato

**Preconditions:** Utente registrato

**Input:**

*Email:* p.porcellini@studenti.unina.it

*Password:* Password12

**Expected output:** true

**Result:** PASS

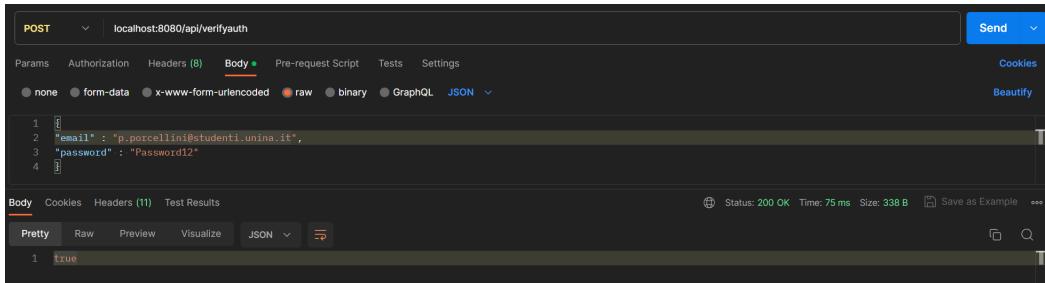


Figura 7.3: Test case *verifica credenziali* - Utente registrato

#### T2. Utente non registrato

**Preconditions:** Utente non registrato

**Input:**

*Email:* i.galiero@studenti.unina.it

*Password:* TestPwd12

**Expected output:** false

**Result:** PASS

### 7.2.4 Identificativo utente

#### T1. Utente registrato

**Preconditions:** Utente registrato

**Input:**

Email: p.porcellini@studenti.unina.it

**Expected output:** 100000003

**Result:** PASS

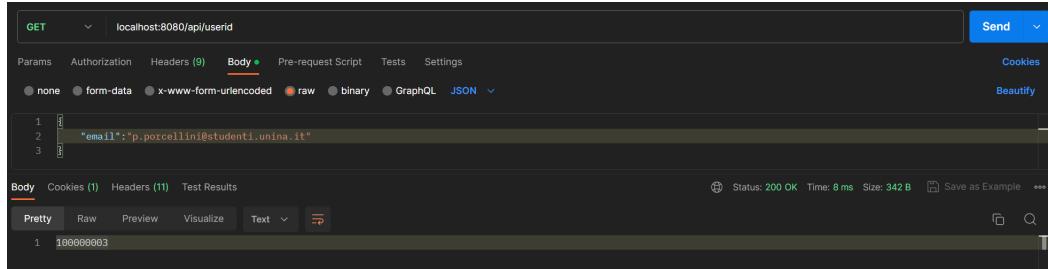


Figura 7.4: Test case *identificativo utente* - Utente registrato

## T2. Utente non registrato

**Preconditions:** Utente non registrato

**Input:**

Email: i.galiero@studenti.unina.it

**Expected output:** NULL

**Result:** PASS

### 7.2.5 Password dimenticata

#### T1. Utente registrato

**Preconditions:** Utente registrato

**Input:**

Email: p.porcellini@studenti.unina.it

**Expected output:** true

**Postcondition:** Nel DB il token viene generato, l'email viene inviata

**Result:** PASS

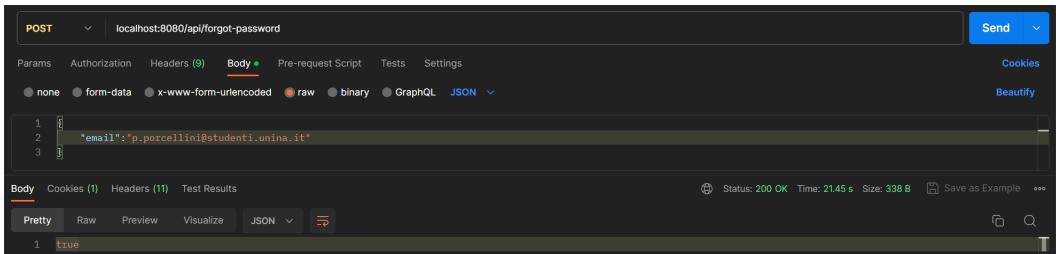


Figura 7.5: Test case *password dimenticata* - Utente registrato

## T2. Utente non registrato

**Preconditions:** Utente non registrato

**Input:**

*Email:* france.bisogno@studenti.unina.it

**Expected output:** false

**Postcondition:** Recupero password negato

**Result:** PASS

### 7.2.6 Reset password

#### T1. Utente registrato

**Preconditions:** Utente registrato

**Input:**

*Token:* 0828bd8e-cb4b-4601-97e4-dc474aceec75

*Password:* NewPassword12

**Expected output:** true

**Postcondition:** Nel DB la password viene correttamente modificata

**Result:** PASS

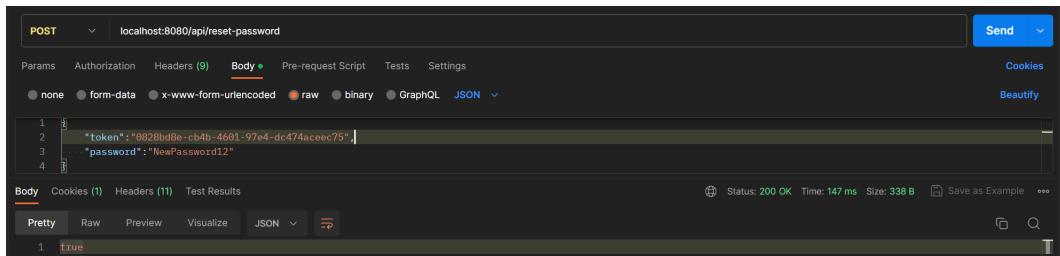


Figura 7.6: Test case *reset password* - Utente registrato

#### T2. Utente non registrato

**Preconditions:** Utente non registrato

**Input:**

*Token:* 077faa-fe91-4499-a426-1efa2cafb8d

*Password:* Ciao123Test

**Expected output:** false

**Postcondition:** Reset password negato

**Result:** PASS

# Capitolo 8

## Installazione

Per quanto riguarda l'installazione in locale del software, l'applicazione viene fornita attraverso un file **JAR** denominato AuthenticationG24-1.0.0.jar.

Per quanto riguarda la **compatibilità**, il software richiede un sistema sul quale è installata la versione di Java 17 (o superiore) e un database relazionale MySQL.

È necessario scaricare la libreria *lombok* per la creazione automatica dei getter/setter e dei costruttori (con e senza argomenti).

In particolare, attraverso il file di configurazione denominato *application.properties* (situato nella stessa cartella in cui si trova il file .jar) è possibile configurare opportunamente diversi parametri tra cui: il mail server, l'URL del database e la connessione a quest'ultimo. Inoltre, è possibile modificare i porti di accesso e uscita dell'applicazione.

```
#spring.datasource.url = jdbc:mysql://localhost:3306/login_system
spring.datasource.url=${DB_URL}
spring.datasource.username=sping
spring.datasource.password=1234

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto:update

#####
# JavaMail Configuration #####
spring.mail.host = smtp.gmail.com
spring.mail.username = noreply.gruppo24@gmail.com
spring.mail.password = hrgxejodrwvsysrfs

spring.mail.properties.mail.smtp.auth = true
spring.mail.properties.mail.smtp.socketFactory.port = 465
spring.mail.properties.mail.smtp.starttls.enable = true
spring.mail.properties.mail.smtp.socketFactory.class = javax.net.ssl.SSLSocketFactory
spring.mail.properties.mail.smtp.socketFactory.fallback = false

spring.mail.test-connection=false

spring.main.allow-bean-definition-overriding=true
#####

server.port=${S_PORT}
#server.port=8081
url = http://localhost:
```

Figura 8.1: File *application.properties*

Come strumento di build automation abbiamo scelto *Maven*. Maven usa un costrutto conosciuto

come Project Object Model (POM), ovvero un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse.

Per importare il progetto Maven nell'IDE Eclipse:

File → Import → Existing Project into Workspace → root directory.

Cliccando su finish, il Progetto verrà importato nel workspace.

A questo punto cliccando con il tasto destro sul *pom.xml*:

run as → maven install

Con il comando *Maven install* viene automaticamente generata la cartella **target** che contiene il file .jar usato per l'esecuzione.

Prima di far partire l'esecuzione in locale dell'applicazione è necessario avviare il database, utilizzando XAMPP. Per l'esecuzione in locale viene avviata la Boot Dashboard di Spring (grazie al tool Spring Tool Suite 4):



Figura 8.2: Spring Tool Suite 4

Dopo aver cliccato sul progetto (“AuthenticationG24”), possiamo avviare l'esecuzione dell'applicazione tramite il pulsante cerchiato in rosso.

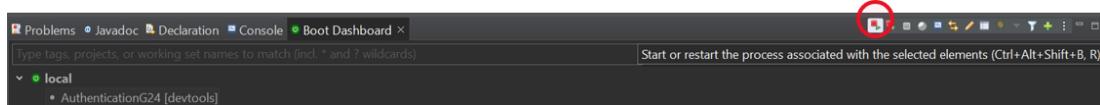


Figura 8.3: Bottone per avviare l'applicazione

## 8.1 Esecuzione con Docker Desktop

Docker Desktop è un'applicazione che permette di istanziare ed eseguire dei *container* in un ambiente virtuale.

Nel nostro caso abbiamo creato due container: il primo contiene l'applicazione, l'altro il database MySQL.

In particolare abbiamo settato i parametri di configurazione in due file:

- **Dockerfile:**

```

FROM eclipse-temurin:17-jdk-alpine
ARG JAR_FILE=target/AuthenticationG24-1.0.0.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]

```

Figura 8.4: File *Dockerfile*

FROM : impostare l'ambiente di esecuzione;  
 ARG\_JAR\_FILE : specificare il percorso e il file eseguibile dell'applicazione;  
 COPY JAR\_FILE: specificare il file eseguibile per il docker;  
 ENTRYPOINT : comando e parametri del comando per eseguire il programma specificato nel COPY.

- **Docker-compose.yml:**

Gestisce la configurazione dei due container e la loro interazione.

```

version: '3.1'

services:
  app:
    build: .
    ports:
      - 8081:8081
    depends_on:
      - db
    environment:
      DB_URL: jdbc:mysql://db:3306/login_system
      S_PORT: 8081
    restart: on-failure

  db:
    image: mysql:latest
    environment:
      MYSQL_USER : spring
      MYSQL_PASSWORD : '1234'
      MYSQL_ROOT_PASSWORD: '1234'
      MYSQL_DATABASE: login_system
    ports:
      - 3306:3306
    healthcheck:
      test: ["CMD", "mysqladmin" , "ping", "-h", "localhost"]
      timeout: 20s
      retries: 10

```

Figura 8.5: File *Docker-compose.yml*

Tramite il prompt dei comandi (eseguito in modalità amministratore), ci spostiamo nella directory contenente il progetto e i file docker.

Digitiamo il comando `docker compose up` ed automaticamente verranno generati ed eseguiti i container.

Nel docker compose sono state inserite alcune variabili d'ambiente che servono a configurare la nostra applicazione, ovvero l'URL del database e il porto del server.

Inoltre, non è possibile utilizzare Maven Install per la creazione del .jar in quanto esso effettua dei test sul database che non è stato ancora generato.

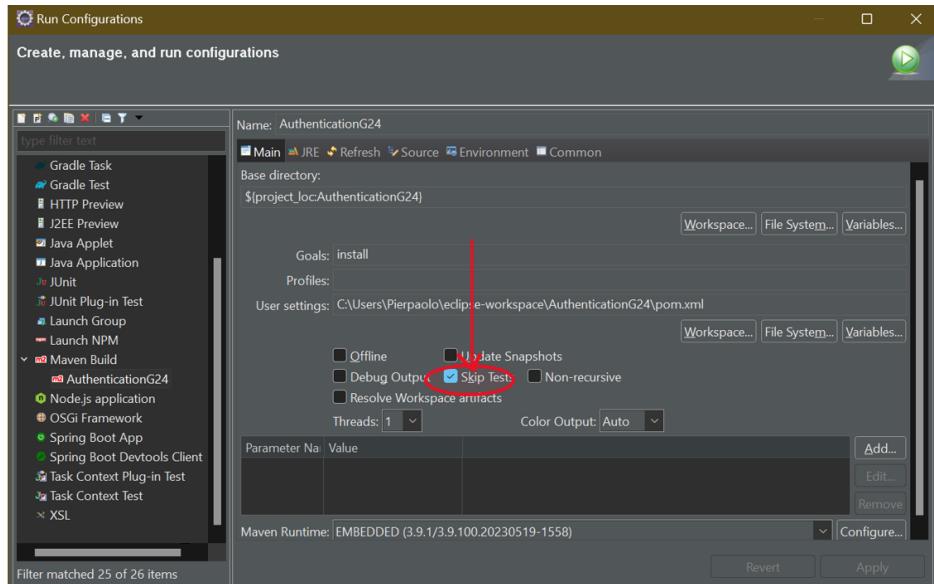


Figura 8.6: Configurazione da effettuare

Si può utilizzare *Maven build* impostando come goal “install” e selezionando l’opzione *Skip test*.

# Capitolo 9

## Manuale d'uso

### 9.1 Pagina index

Questa è la pagina iniziale della nostra applicazione, nel banner in alto troviamo due link che reindirizzano uno alla pagina per il login e uno a quella della registrazione.



Figura 9.1: Schermata principale

### 9.2 Registrazione

Questa è la pagina di registrazione in cui presenta il form per inserimento dei dati e il link per effettuare il login. Di seguito si riportano i possibili errori che si possono verificare durante la registrazione.

The registration form consists of several input fields:

- First Name \* and Last Name \*
- Email \* and Confirm Email \*
- Password \* and Confirm Password \*
- Degree course and University dropdown menus.
- A teal-colored Register button at the bottom.

Below the form, there is a link: Already registered? [Login here.](#)

Figura 9.2: Form di registrazione



Figura 9.3: Campi facoltativi

The validation errors are:

- First Name \* and Last Name \* have red circles with an exclamation mark, indicating they are required.
- First name should not be empty and Last name should not be empty are displayed below the respective fields.
- Email \* and Confirm Email \* have red circles with an exclamation mark, indicating they must match.
- Email should not be empty is displayed below the Email field.
- Password \* has a red circle with an exclamation mark, indicating it must be at least 8 characters long.
- Password should have at least eight characters, one lowercase letter, one uppercase letter and one number or special character is displayed below the Password field.

Figura 9.4: Campi obbligatori

The validation errors are:

- Email \* and Confirm Email \* have red circles with an exclamation mark, indicating they must match.
- The email fields must match is displayed below the Email field.
- Password \* has a red circle with an exclamation mark, indicating it must be at least 8 characters long.
- Confirm Password \* has a red circle with an exclamation mark, indicating it must match the Email field.
- The password fields must match is displayed below the Password field.

Figura 9.5: Matching tra email e password

Di seguito si riporta il template della email che l'utente riceve dopo la richiesta di registrazione, in cui è presente il link per confermare la email e terminare la fase di registrazione, venendo reindirizzati alla pagina di login, con un messaggio di registrazione effettuata.

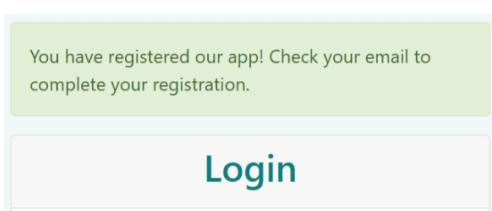


Figura 9.6: Registrazione effettuata

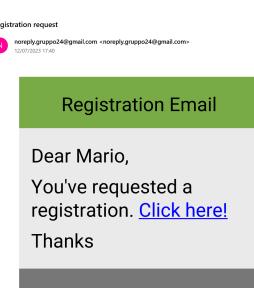


Figura 9.7: Link conferma registrazione

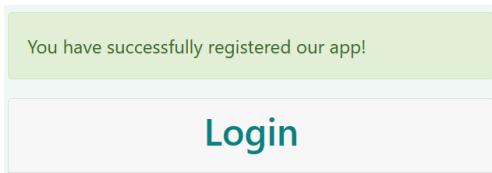


Figura 9.8: Registrazione completata

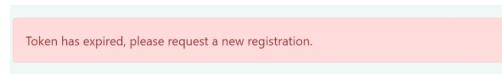


Figura 9.9: Token scaduto

### 9.3 Login

Questa pagina presenta un form per l'inserimento dei dati per effettuare l'autenticazione (email e password), ritroviamo due link: uno per effettuare la registrazione e l'altro per il recupero password. Di seguito si riportano i possibili errori che si possono verificare durante il login.

A screenshot of a login form titled "Login". It contains two input fields: "Email" and "Password", both with placeholder text. Below the inputs is a large teal "Submit" button. At the bottom of the form, there is a link "Not registered? [Register/Sign up here.](#)" and another link "[Forgot your password?](#)".

Figura 9.10: Form di login



#### 9.3.1 Pagina user

Questa è la pagina principale dell'Utente in cui è presente il link per giocare e per lo storico delle partite, infine nel banner in alto è presente il link per il logout. Se l'Admin clicca sul link Home nel banner, può tornare nella sua pagina principale.

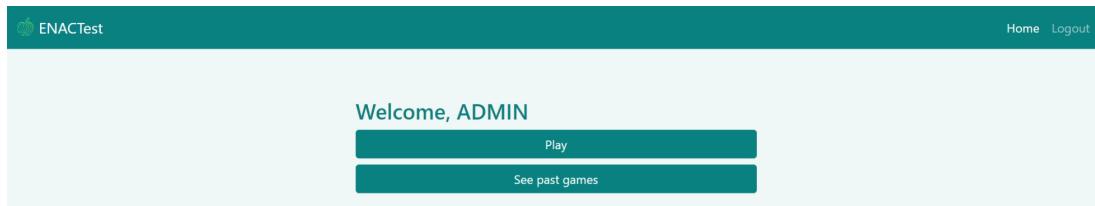


Figura 9.11: Homepage dello user

### 9.3.2 Pagina admin

Questa è la pagina principale dell'Admin in cui è presente l'elenco dei giocatori che hanno completato la procedura di registrazione. Presenta nel banner in alto il collegamento alla pagina homeuser, in quanto l'admin risulta essere anche un giocatore e al logout.

Registered users list					
<b>Id</b>	<b>First name</b>	<b>Last name</b>	<b>Email</b>	<b>Degree Course</b>	<b>University</b>
99999999	ADMIN	ADMIN	admin@admin		

Figura 9.12: Homepage dell'admin

## 9.4 Forgot password

Questa è la pagina per il dimentica password, in cui presenta il form per inserire la email, il link per il login e la registrazione. Di seguito si riportano i possibili errori che si possono verificare durante la richiesta di dimentica password.

A screenshot of a "Forgot Password" form. The title "Forgot Password" is at the top center in a large, bold, teal font. Below it is a rectangular input field with the placeholder "Email". At the bottom of the form is a teal button labeled "Reset Password" in white. Below the button, there is a small block of text with two links: "New user? [Register](#)." and "Already registered? [Login](#)."

Figura 9.13: Form di forgot password



Figura 9.15: Utente non abilitato

New user? [Register](#).  
Already registered? [Login](#).

Figura 9.14: Utente non presente

Di seguito si riporta il template della email che l'utente riceve dopo la richiesta di recupero password, in cui è presente il link per terminare la fase di recupera password venendo reindirizzati alla pagina di reset password.

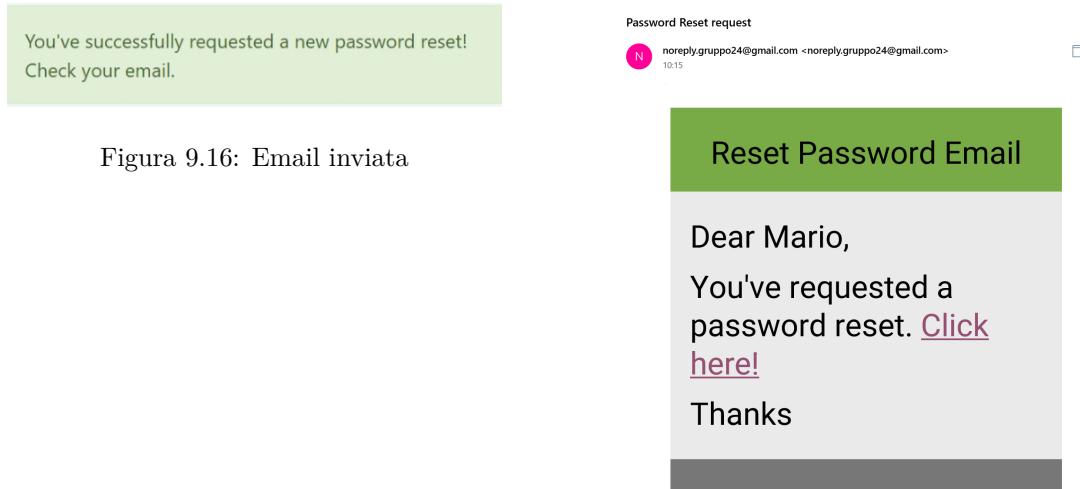
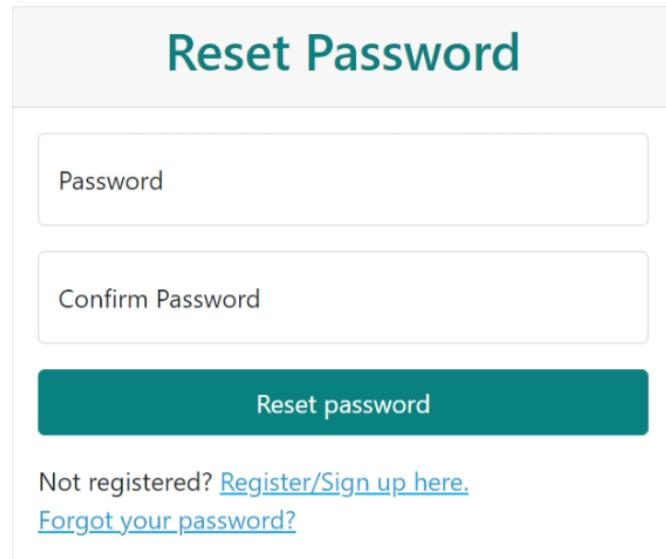


Figura 9.17: Link recupero password

## 9.5 Reset password

Questa è la pagina per il reset della password, in cui presenta il form per l'inserimento della nuova password (con conferma password). Di seguito si riportano i possibili errori che si possono verificare durante la richiesta di reset password.



The image shows a 'Reset Password' form. At the top, the title 'Reset Password' is displayed in a teal header. Below it are two input fields: 'Password' and 'Confirm Password', both in light gray boxes. A large teal button labeled 'Reset password' is centered below the inputs. At the bottom, there are two links: 'Not registered? [Register/Sign up here.](#)' and '[Forgot your password?](#)'.

Figura 9.18: Form di forgot password

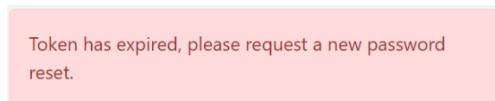


Figura 9.19: Token scaduto

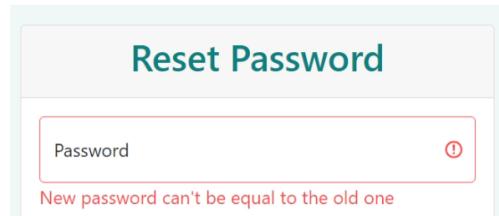


Figura 9.20: Inserimento errato della password

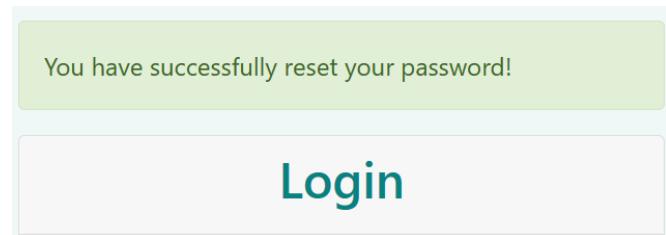


Figura 9.21: Password reimposta