

Università degli Studi di Napoli

Federico II



Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Scuola Politecnica e delle Scienze di Base

Corso di Laurea Magistrale in Ingegneria Informatica

Documentazione Team G7: task T2-T3

Studenti:

Luca Comentale M63001520
Carmine De Monaco M63001414
Emanuele Di Maio M63001427
Roberta Di Marino M63001523
Giovanni Dioguardi M63001418

Docenti:

Prof. Anna Rita Fasolino

Anno Accademico
2022/2023

Indice

1	Introduzione	2
1.1	Requisiti Specifici del task	2
2	Specifiche di progetto	3
2.1	Utenti del sistema	3
2.2	Requisiti Funzionali	3
2.3	Storie utente	4
2.4	Requisiti Non Funzionali	4
2.4.1	Sicurezza	4
2.4.2	Usabilità	5
2.5	Stima dei Costi	6
2.6	Visione ad alto livello del sistema	7
3	Processo di Sviluppo	10
3.1	Tool per la condivisione del lavoro	10
3.1.1	Trello	11
3.2	Tool e tecnologie per lo sviluppo	13
4	Fase di Analisi	14
4.1	Use Case Diagram	14
4.2	Sequence Diagram	16
4.2.1	Sequence Diagram: Registration	16
4.2.2	Sequence Diagram: Login	17
4.2.3	Sequence Diagram: Change Password	18
4.3	Activity Diagram	19
4.3.1	Activity Diagram: Registration	19
4.3.2	Activity Diagram: Login	20
4.3.3	Activity Diagram: Change Password	21
4.4	System Domain Model	22

5 Fase di Progettazione	23
5.1 Pattern Architetturale MVC	23
5.1.1 Spring MVC	23
5.2 Component Diagram	25
5.3 Sequence Diagram: Progettazione	27
5.3.1 Sequence Diagram: Registration	27
5.3.2 Sequence Diagram: Login	29
5.3.3 Sequence Diagram: Change Password	30
5.4 Glossario	32
5.5 Package Diagram	32
5.6 WebApp Class Diagram	34
5.6.1 View	34
5.6.2 Controller	35
5.6.3 Model	37
5.6.4 Utils	38
5.7 Class Diagram con dipendenze di ciascun Controller	40
5.7.1 RegistrationController	40
5.7.2 ConfirmationController	41
5.7.3 LoginController	42
5.7.4 ChangePasswordController	43
5.7.5 ChangePasswordProcessController	44
5.7.6 LogoutController	45
5.8 Deployment	45
5.9 Integrabilità	47
5.9.1 API Locali	47
5.9.2 Strutture Dati	47
5.9.2.1 Student	47
5.9.2.2 Role	48
5.9.3 Services	48
5.9.3.1 Registrazione	48
5.9.3.2 Login	49
5.9.3.3 Change Password	50
5.9.4 Utils	51
5.9.4.1 ConfirmationService	51
5.9.4.2 EmailService	51
5.9.4.3 FindInfoService	53
5.9.4.4 StudentDetailService	55
5.9.4.5 UtilityService	55
5.9.5 REST API	56
5.9.5.1 REST API: Registration (POST)	56
5.9.5.2 REST API: Student Info (GET)	58

5.9.5.3	REST API: ChangePassword (POST)	59
5.9.5.4	REST API: Student ID (GET)	60
5.9.5.5	REST API: ConfirmToken (POST)	61
5.9.5.6	REST API: ChangePasswordProcess (POST)	62
5.9.5.7	REST API: Login (POST)	63
6	Implementazione	65
6.1	Introduzione	65
6.2	Spring Security	65
6.3	Sequence Diagram di Dettaglio	66
6.3.1	Sequence Diagram: Registration	66
6.3.2	Sequence Diagram: Confirm Registration	69
6.3.3	Sequence Diagram: Login	70
6.3.4	Sequence Diagram: Change Password	71
6.3.5	Sequence di Dettaglio: Note nel caso di REST API	73
7	Testing	75
7.1	Testing basato sulle GUI dell'applicazione	75
7.1.1	Testing GUI: registration	75
7.1.2	Testing GUI: login	77
7.1.3	Testing GUI: changePassword	79
7.1.4	Testing GUI: confirmation email	81
7.1.5	Testing GUI: changePasswordProcess	82
7.2	Testing delle API REST con Postman	83
7.3	Testing Registrazione	83
7.3.1	Test Case 1	83
7.3.2	Test Case 2	84
7.3.3	Test Case 3	84
7.4	Testing sui dati dello studente	86
7.4.1	Test Case 4	86
7.4.2	Test case 5	86
7.4.3	Test Case 6	87
7.4.4	Test Case 7	87
7.5	Testing Change Password	88
7.5.1	Test Case 8	88
7.5.2	Test Case 9	88
7.5.3	Test Case 10	88
7.5.4	Test Case 11	89
7.5.5	Test Case 12	89
7.6	Testing Conferma Email	89
7.6.1	Test Case 13	89

7.6.2	Test Case 14	90
7.6.3	Test Case 15	90
7.7	Testing Autenticazione	90
7.7.1	Test Case 16	90
7.7.2	Test Case 17	91
7.8	Testing con Postman	91
8	Installazione	94
8.1	Installazione software in locale	94
8.2	Esecuzione tramite SpringToolSuite4	96

Capitolo 1

Introduzione

Il lavoro svolto dal Team consiste nello sviluppo di parte di un applicativo web finalizzato alla realizzazione di un gioco di Testing, dove gli studenti, sviluppando dei test da applicare a delle classi Java, competono contro dei bot o altri giocatori, guadagnando punti vincendo. In particolare, si tratta della registrazione e l'autenticazione degli studenti all'applicativo.

Lo studente avrà la possibilità di potersi registrare al sistema e una volta salvati i suoi dati all'interno di una base dati, potrà effettuare il login con la propria email ed una password scelta in fase di registrazione. Lo studente avrà la possibilità di poter cambiare la propria password in caso quest'ultima fosse stata dimenticata.

1.1 Requisiti Specifici del task

L'applicazione deve consentire agli studenti di registrarsi per poter conservare la storia delle attività svolte, oppure per accedere a requisiti di gioco più complessi. All'atto della registrazione, lo studente fornirà nome, cognome, un indirizzo email valido ed una password, il sistema dopo aver controllato la validità dei dati forniti, aggiungerà il giocatore all'elenco dei giocatori registrati e gli assocerà un Id univoco. Sarebbe desiderabile raccogliere anche altre informazioni sugli studenti, come il corso di studi a cui sono iscritti (Bachelor, Master Degree, o altro). All'atto della autenticazione, lo studente fornirà l'indirizzo email fornito per la registrazione e la relativa password, il sistema dopo aver controllato la validità dei dati forniti, autenticherà il giocatore e gli fornirà una schermata per l'accesso alle funzionalità di gioco o di consultazione delle sessioni di gioco passate.

Capitolo 2

Specifiche di progetto

2.1 Utenti del sistema

Gli utilizzatori dell'applicazione si dividono in due tipologie principali di utenti:

- **Studente non registrato:** non ha accesso ai servizi del sistema. Lo studente avrà la possibilità di effettuare la registrazione inserendo i propri dati personali, in particolare l'email e la password che saranno necessarie per autenticarsi successivamente.
- **Studente registrato:** ha accesso alle funzionalità del sistema. Dopo aver effettuato il login, lo studente accederà ad una schermata per l'accesso alle funzionalità di gioco o di consultazione delle sessioni di gioco passate.

2.2 Requisiti Funzionali

I requisiti funzionali previsti per la web app sono i seguenti:

- L'applicazione deve consentire agli studenti di potersi registrare inserendo i propri dati personali e le credenziali di accesso. In particolare, per i dati personali sono richiesti nome, cognome, data di nascita, sesso, nazionalità e istruzione; per le credenziali di accesso, sono richieste un'email ed una password. Per confermare la registrazione, allo studente è richiesto di confermare la validità della propria email tramite email inviata all'atto della registrazione. In seguito alla conferma dell'email, lo studente risulterà registrato ed eventualmente potrà autenticarsi.

- L'applicazione deve consentire ad uno studente di autenticarsi inserendo le proprie credenziali di accesso, ossia email e password.
- L'applicazione, deve consentire ad uno studente autenticato, di poter accedere ad una schermata per l'accesso alle funzionalità di gioco o di consultazione delle sessioni di gioco passate. Inoltre, lo studente deve avere la possibilità di effettuare il logout.
- L'applicazione deve consentire ad uno studente di poter effettuare un cambio della propria password. Lo studente è tenuto a inserire l'email associata all'account di cui vuole cambiare la password. All'atto della richiesta, dopo aver validato la propria email, lo studente proseguirà con l'operazione di ripristino della password; in particolar modo, allo studente viene richiesta la nuova password e quest'ultima verrà aggiornata nel sistema.

2.3 Storie utente

Storie utente
Come studente non registrato, voglio accedere alla pagina di registrazione dell'applicazione, in modo da potermi registrare.
Come studente registrato, voglio accedere alla pagina di autenticazione dell'applicazione, in modo da potermi autenticare.
Come studente registrato, voglio accedere alla pagina di cambio password dell'applicazione, in modo da poter cambiare la password.

Tabella 2.1: Storie Utente dell'applicazione

2.4 Requisiti Non Funzionali

2.4.1 Sicurezza

Per soddisfare i requisiti di sicurezza si richiede all'applicazione che:

- Uno studente non autenticato non deve poter accedere alla schermata per l'accesso alle funzionalità di gioco, di consultazione delle sessioni di gioco passate e di logout.
- Uno studente autenticato non deve poter accedere alla schermata di login

- Uno studente, per poter concludere l'operazione di registrazione, deve confermare la propria email.
- Uno studente registrato che intende cambiare la propria password non può terminare l'operazione senza aver effettuato la conferma tramite la propria email.
- Il sistema deve garantire che la password non sia visibile in chiaro all'interno del sistema stesso.

2.4.2 Usabilità

Per soddisfare i requisiti di usabilità si richiede all'applicazione che:

- l'applicazione deve poter essere comprensibile agli studenti che intendono utilizzarla.

2.5 Stima dei Costi

Si riporta una stima dei costi del processo di sviluppo del software:

Termine	Descrizione
Stima	complessità espressa con un punteggio da 0 a 5
Priorità	importanza di ciascuna storia utente o attività (Bassa, Media, Alta)
Tempo	tempo espresso in ore, impiegato per svolgere un'attività
Risorse Necessarie	membri impegnati nello svolgimento dell'attività

Storie Utente	Descrizione	Stima	Priorità
US-1	Come studente non registrato, voglio accedere alla pagina di registrazione dell'applicazione, in modo da potermi registrare	3	Alta
US-2	Come studente registrato, voglio accedere alla pagina di autenticazione dell'applicazione, in modo da potermi autenticare.	5	Alta
US-3	Come studente registrato, voglio accedere alla pagina di cambio password dell'applicazione, in modo da poter cambiare la password.	2	Media

Attività	Risorse necessarie	Tempo	Stima	Priorità
Analisi dei requisiti	Product Owner, Team di sviluppo	30	4	Alta
Sprint Planning	Team di sviluppo	16	2	Alta
Sprint Review	Team di sviluppo	5	1	Media
Sviluppo delle storie utente	Team di sviluppo	60	5	Alta
Documentazione del progetto	Team di sviluppo	40	3	Alta
Studio delle tecnologie	Team di sviluppo	20	4	Alta

2.6 Visione ad alto livello del sistema

In figura (2.1) si presenta una vista semplice e chiara dall'esterno del sistema software, mostrando le **User Interface** con le quali le entità esterne interagiscono e i servizi esterni che vengono utilizzati.

Nel sistema è previsto che tramite il browser si possa accedere alle seguenti interfacce utente:

- **RegistrationUI:** dedicata all'inserimento dei dati personali dello studente che intende registrarsi.
- **LoginUI:** dedicata all'autenticazione dello studente. Risulta essere l'interfaccia utente principale poiché da quest'ultima uno studente non registrato, mediante la funzionalità "Create New Account", verrà indirizzato all'interfaccia RegistrationUI e potrà registrarsi; sarà possibile cambiare la password per uno studente registrato mediante la funzionalità "Forgot Password?" attraverso la quale verrà indirizzato alla pagina ChangePasswordUI; per uno studente autenticato sarà possibile accedere alla dashboard dedicata alle funzionalità di gioco.
- **ChangePasswordUI:** dedicata all'inserimento dell'email per avviare l'operazione di cambio password.
- **ChangePasswordProcessUI:** dedicata all'operazione effettiva di cambio password.
- **DashboardUI:** dedicata all'accesso alle funzionalità di gioco, sessione delle partite passate e al logout.

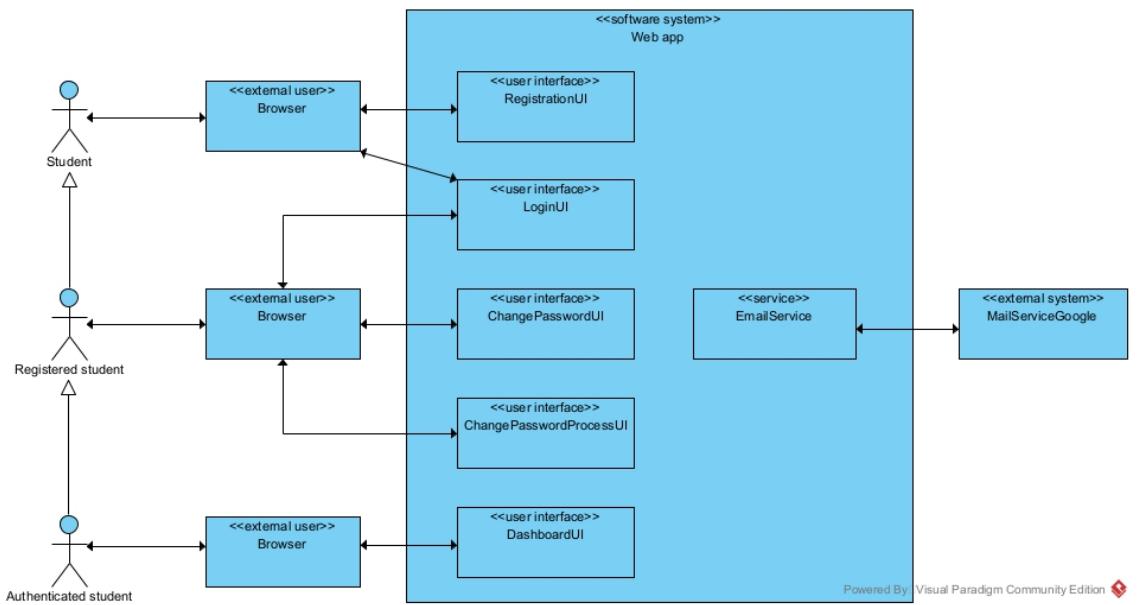


Figura 2.1: Context Diagram

Nel sistema è previsto che tramite l'external system **MailServiceGoogle** è possibile usufruire del servizio email per l'invio della conferma.

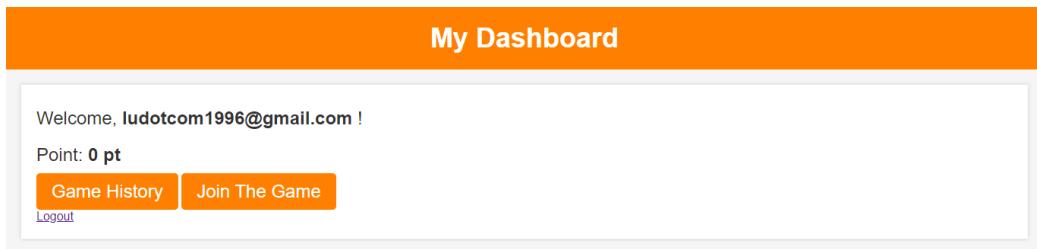


Figura 2.2: DashboardUI

Student Login

Email

Password

LOGIN

[Forgot Password?](#)

[Create New Account](#)

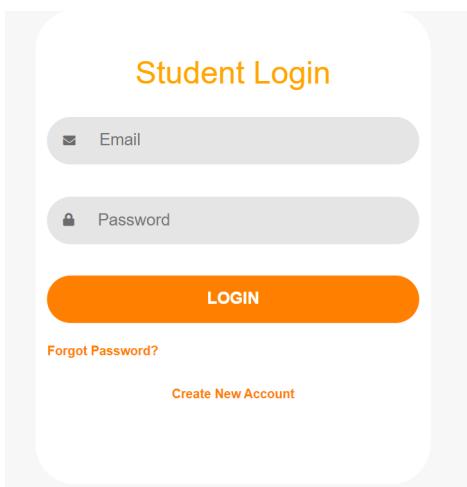


Figura 2.3: LoginUI

Student Registration Form

Name

Surname

Date Of Birth gg/mm/aaaa

Gender -- Select Gender --

Nationality -- Select Nationality --

Degree -- Select Degree --

Email

Password

Repeat Password

[Annulla](#) **Register**

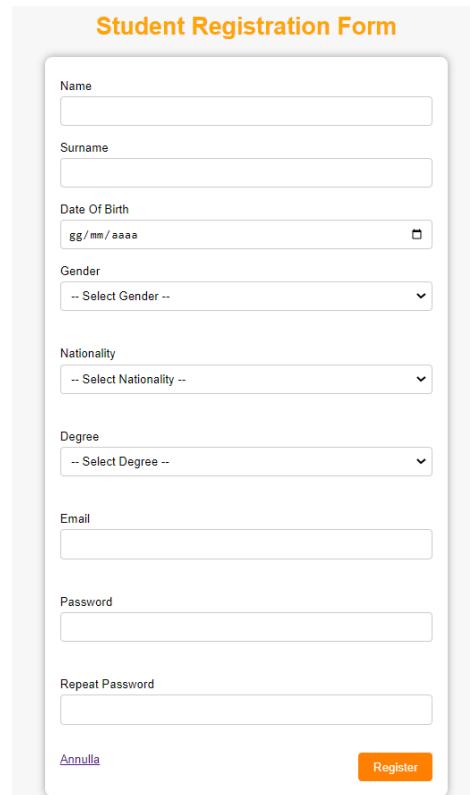


Figura 2.4: RegistrationUI

Change Password

Email iudotcom1996@gmail.com

SEND EMAIL

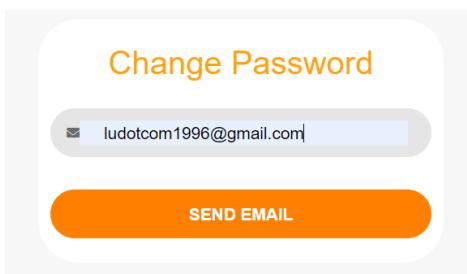


Figura 2.5: ChangePasswordUI

Reset Password

.....

.....

Passwords Match

RESET

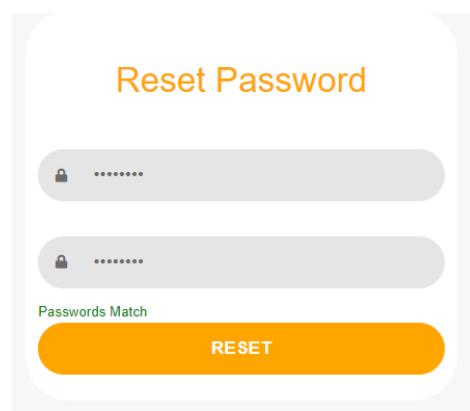


Figura 2.6: ChangePasswordProcessUI

Capitolo 3

Processo di Sviluppo



Figura 3.1: QR per accedere al GitHub del team G7

3.1 Tool per la condivisione del lavoro

Il lavoro del team è stato svolto principalmente con riunioni organizzate in presenza, usufruendo dello strumento **Microsoft Teams**¹ per funzionalità di condivisione schermo, condivisione di file, pianificazioni di eventuali meeting online e pianificazione del lavoro giornaliero. **GitHub**² è stato utilizzato per la gestione del codice, in particolare per la condivisione dello stesso tra i diversi membri del team, ottenendo un repository chiaro e ordinato di tutta l'implementazione del sistema. Ogni volta che un membro effettua una modifica dei file memorizzati nel repository, esegue una push sul repository

¹<https://www.microsoft.com/it-it/microsoft-teams/log-in>

²<https://github.com/>

condiviso. In questo modo, gli altri membri possono eseguire una pull per ottenere, in locale, le modifiche effettuate. Risulta possibile visualizzare tutti i commit effettuati e ottenere tutte le versioni del sistema, dal primo prototipo alla release finale. In figura (3.1) viene riportato il codice QR che reindirizza direttamente alla home del repository del Team.

3.1.1 Trello

Trello³ è uno strumento di gestione delle attività basato su bacheche che offre una modalità visuale per organizzare e tenere traccia delle attività. È ampiamente utilizzato per la gestione dei progetti, compresi quelli che seguono il framework SCRUM. Trello può essere utilizzato nel contesto di SCRUM⁴ utilizzando delle bacheche ripartite in liste che rappresentano le fasi del processo. In particolare, la bacheca viene ripartita in liste di cose "da fare", "in esecuzione", "fatte". Le liste possono essere personalizzate in base alle specifiche esigenze del progetto. Per il lavoro del team sono state utilizzate tre bacheche differenti, organizzate settimanalmente:

- **RIUNIONI TEAM G7:** bacheca utilizzata per la gestione delle riunioni del team, effettuate in presenza o a distanza.
- **COMUNICAZIONI DEL TEAM G7:** bacheca utilizzata per riportare tutte le comunicazioni del team, sia inerenti agli aspetti organizzativi, sia per comunicazioni inerenti alle dinamiche del progetto.
- **SVILUPPO DELLA WEB-APP:** bacheca utilizzata per riportare la dinamica del progetto, seguendo un approccio Agile.

All'interno della bacheca **SVILUPPO DELLA WEB-APP** ad ogni attività, seguendo un approccio Agile, è stata assegnata una label di diverso colore (verde,giallo,rosso) per indicare il grado di difficoltà riscontrato: semplice,medio,difficile rispettivamente.

³<https://trello.com/>

⁴<https://www.scrum.org/>

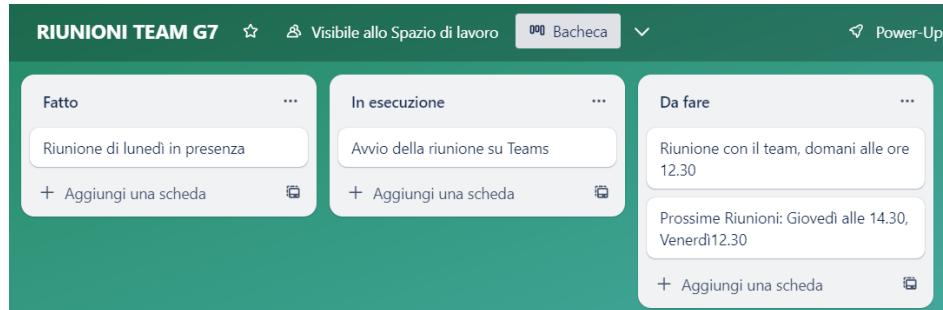


Figura 3.2: Bacheca RIUNIONI TEAM G7



Figura 3.3: Bacheca COMUNICAZIONI DEL TEAM G7

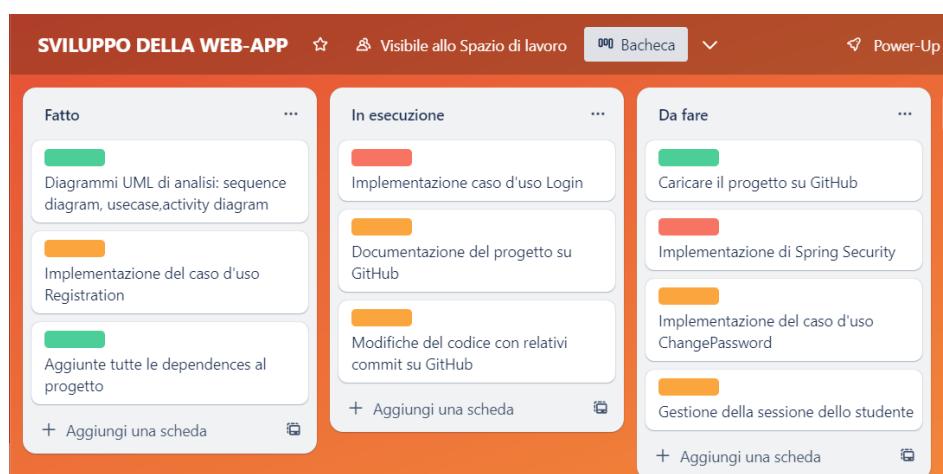


Figura 3.4: Bacheca SVILUPPO WEB-APP

3.2 Tool e tecnologie per lo sviluppo

Di seguito si riportano le tecnologie utilizzate per la progettazione lo sviluppo del sistema software:

- **IDE e Linguaggio Implementativo:** Spring Tool Suite 4⁵, Java 17
- **Database:** Database documentale MongoDB⁶;
- **Build Automation:** Maven⁷;
- **Supporto allo sviluppo e Servlet:** Spring Boot DevTools 3.10, Tomcat Apache⁸;
- **Pattern Architetturali:** Spring Web MVC⁹;
- **Autenticazione:** Spring Security 6.1.0¹⁰;
- **Accesso al Database:** Spring Data MongoDB¹¹;
- **Template Engine:** Thymeleaf¹²;
- **Template:** HTML 5.2¹³, JavaScript, CSS;
- **Servizio Email:** Spring Java Mail Sender¹⁴;
- **Diagramma UML:** Visual Paradigm Community Edition 17.1¹⁵;
- **Account Email:** Google Account¹⁶;
- **Container:** Docker¹⁷;
- **Strumento di Testing:** Postman¹⁸

⁵<https://spring.io/tools>

⁶<https://www.mongodb.com/>

⁷<https://maven.apache.org/>

⁸<https://tomcat.apache.org/>

⁹<https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>

¹⁰<https://spring.io/projects/spring-security>

¹¹<https://spring.io/projects/spring-data-mongodb>

¹²<https://www.thymeleaf.org/>

¹³<https://www.html.it/>

¹⁴<https://www.baeldung.com/spring-email>

¹⁵<https://www.visual-paradigm.com/>

¹⁶<https://www.google.com/intl/it/account/about/>

¹⁷<https://www.docker.com/>

¹⁸<https://www.postman.com/>

Capitolo 4

Fase di Analisi

Il seguente capitolo affronta la documentazione di analisi che è stata prodotta dal team al fine di chiarire le funzionalità e i servizi contenuti all'interno del sistema. I diagrammi riportati all'interno di questo capitolo sono stati realizzati tramite il tool Visual Paridgm che adotta la notazione UML. La modellazione di tali diagrammi è stata effettuata considerando esclusivamente soltanto lo scenario di successo. Per dettagli di scenari alternativi, si faccia riferimento al Capitolo 5 di questo documento.

4.1 Use Case Diagram

Gli attori che interagiscono con il sistema sono:

- **Student**: studente che intende registrarsi all'applicazione.
- **Registered Student**: studente registrato che intende autenticarsi per accedere alla dashboard, oppure, che ha la possibilità di cambiare la propria password.

Sono stati analizzati tre casi d'uso differenti, riportati in figura (4.1) :

- **Registration**: lo studente non registrato inserisce i propri dati personali al fine della registrazione; lo studente validerà la propria email personale, confermando l'intenzione di proseguire nella registrazione effettiva.
- **Login**: lo studente che ha già effettuato la registrazione potrà autenticarsi ed accedere alle funzionalità di gioco dell'applicazione e alle sessione delle partite giocate.

- **Change Password:** lo studente che ha già effettuato la registrazione potrà cambiare la propria password, solo dopo aver validato la propria email personale.

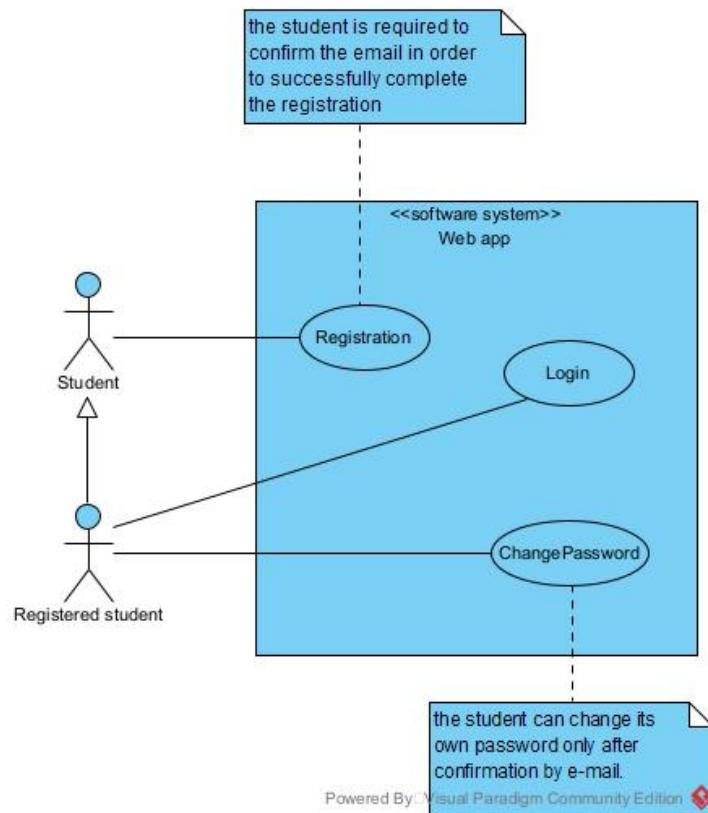


Figura 4.1: Diagramma dei casi d'uso

4.2 Sequence Diagram

Si riportano i sequence diagram applicati in fase di analisi utilizzati per descrivere il comportamento del sistema durante un'interazione con gli attori esterni. Tali diagrammi si concentrano sulla descrizione del comportamento dinamico di un sistema, mostrando come gli attori del sistema interagiscono nel tempo per ottenere un determinato risultato.

4.2.1 Sequence Diagram: Registration

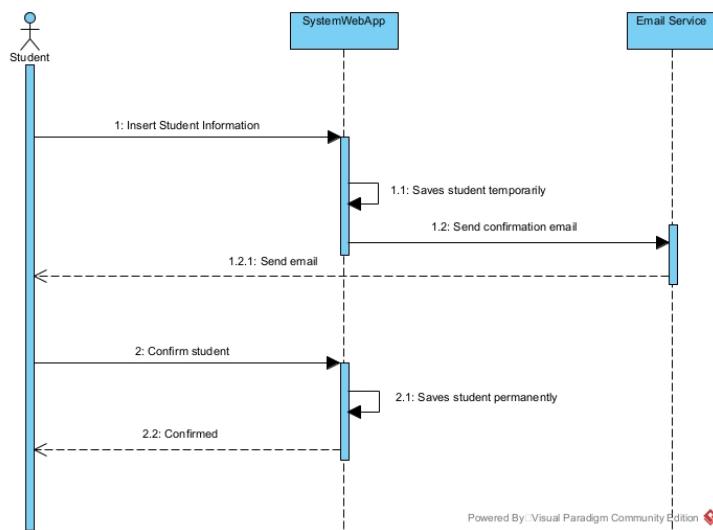


Figura 4.2: Sequence Diagram: Registration

In figura (4.2) viene riportato il sequence diagram relativo al caso d'uso "Registration". L'attore è lo studente che intende registrarsi all'applicazione. L'interazione tra lo studente e il sistema avviene tramite i seguenti passi:

- "Insert Student Information": lo studente inserisce i propri dati personali sull'applicazione.
- "Saves Student temporarily": il sistema salva temporaneamente i dati dello studente all'interno della base dati, in attesa della conferma dell'email.
- "Send confirmation email": il sistema avvia la funzionalità di invio email tramite un servizio esterno per la gestione dell'invio dell'email; attraverso l'email di conferma, lo studente validerà la propria email personale, confermando l'intenzione di proseguire nella registrazione effettiva.

- "Confirm student": lo studente dopo aver ricevuto l'email, effettua la conferma.
- "Saves student permanently": il sistema effettua il salvataggio permanente dei dati dello studente all'interno della base dati.

4.2.2 Sequence Diagram: Login

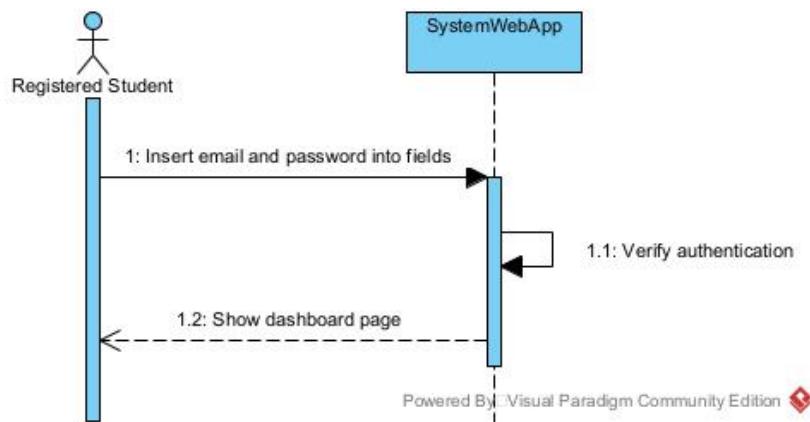


Figura 4.3: Sequence Diagram: Login

In figura(4.3) viene riportato il sequence diagram relativo al caso d'uso "Login". L'attore è lo studente registrato che intende autenticarsi. L'interazione tra lo studente registrato e il sistema avviene tramite i seguenti passi:

- "Insert email and password into fields": lo studente registrato inserisce le proprie credenziali di accesso: email e password.
- "Verify authentication": il sistema avvia le operazioni di verifica delle credenziali dello studente.
- "Show dashboard page": lo studente verrà indirizzato sulla pagina dashboard dedicata alle funzionalità di gioco e alle sessione delle partite giocate dallo studente.

4.2.3 Sequence Diagram: Change Password

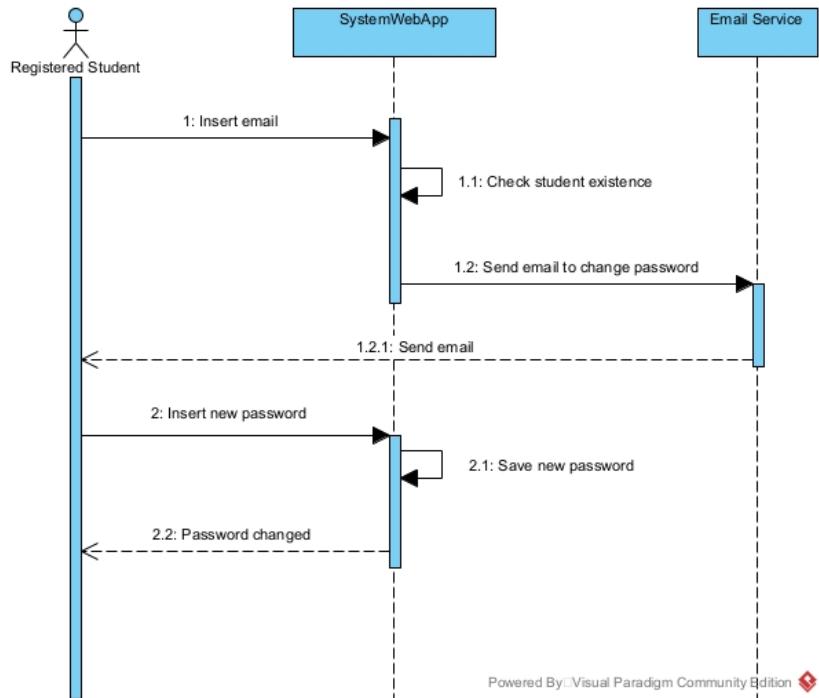


Figura 4.4: Sequence Diagram: ChangePassword

In figura(4.4) viene riportato il sequence diagram relativo al caso d'uso "Change Password". L'attore è lo studente registrato che intende autenticarsi. L'interazione tra lo studente registrato e il sistema avviene tramite i seguenti passi:

- "Insert email": lo studente già registrato inserirà l'email utilizzata in fase di registrazione prima di poter effettuare l'operazione di cambio password.
- "Check student existence": il sistema controllerà la validità dell'email, verificando se quest'ultima è associata ad un account esistente nella base dati.
- "Send email to change password": se l'email è valida, il sistema invierà una email di conferma che consentirà allo studente di poter effettuare il cambio password.
- "Confirm student": lo studente dopo aver ricevuto l'email, effettua la conferma.

- "Insert new password": lo studente inserisce la nuova password;
- "Save new password": il sistema avvia le operazioni di salvataggio della nuova password sulla base dati;

4.3 Activity Diagram

Per fornire una vista sul flusso di attività di ciascun caso d'uso e sulla sincronizzazione tra attore-sistema, si riportano i seguenti activity diagram:

4.3.1 Activity Diagram: Registration

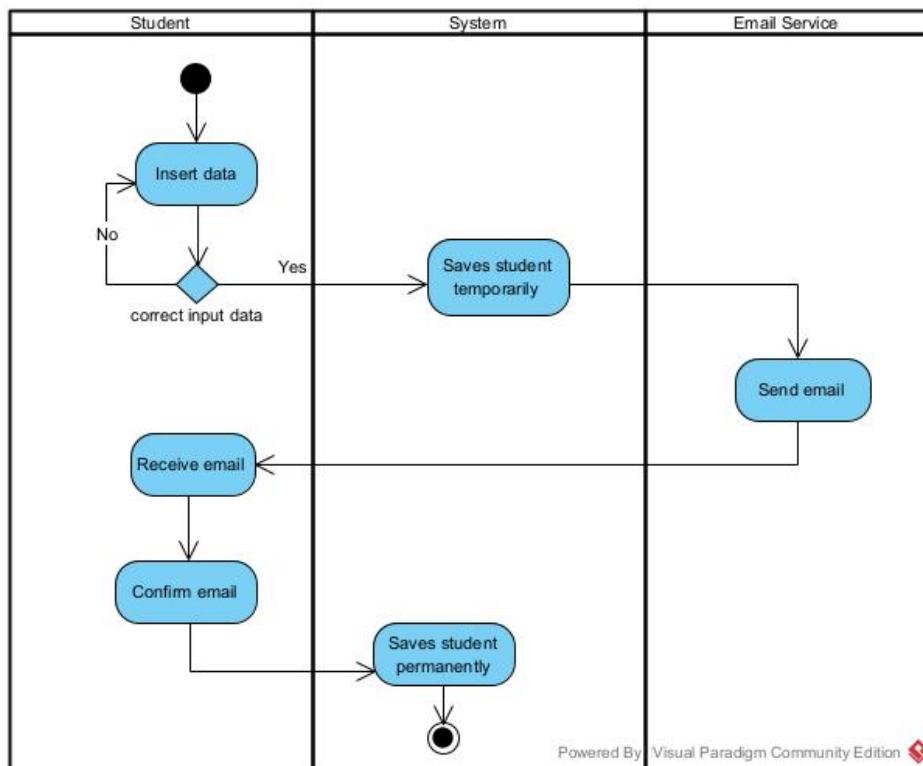


Figura 4.5: Activity Diagram: Registration

In figura (4.5), viene riportato l'activity diagram relativo al caso d'uso "Registration". Sono state definite tre partizioni: la prima rappresenta lo studente che vuole effettuare la registrazione, la seconda il sistema e la terza l'utilizzo del servizio esterno per l'invio della email di conferma. Lo studente

inserisce i propri dati personali, se questi non sono stati inseriti correttamente, potrà inserirli nuovamente, altrimenti, il sistema effettua un salvataggio temporaneo dei dati dello studente nella base dati e invia una richiesta di conferma mediante il servizio email dedicato. Dopo che l'email è stata inviata, lo studente può confermare la propria email ed infine, il sistema può concludere le operazioni di registrazione e salvataggio dei dati dello studente all'interno della base dati in modo permanente.

4.3.2 Activity Diagram: Login

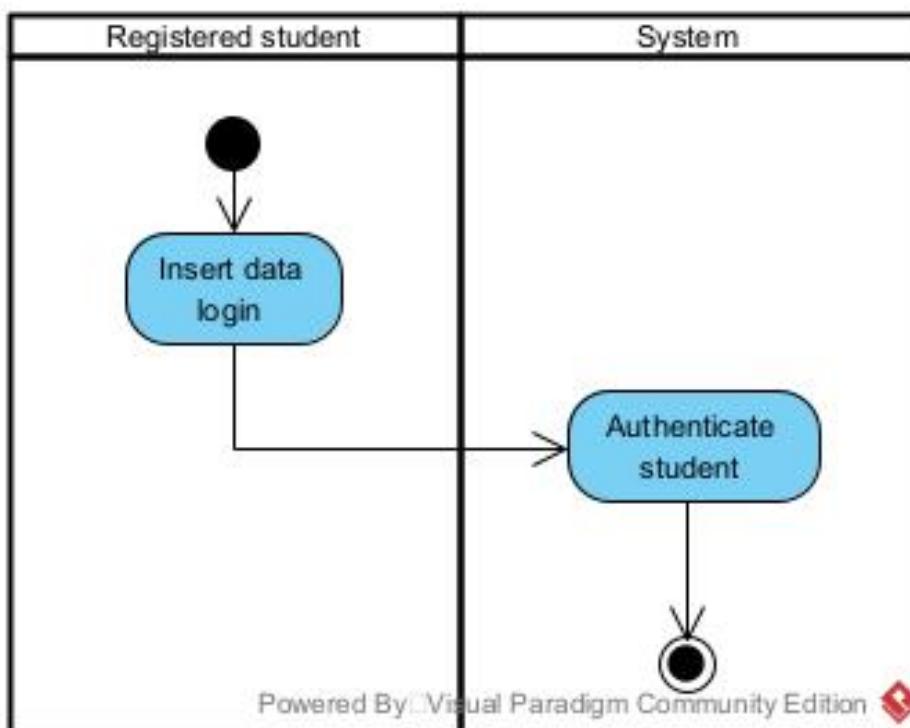


Figura 4.6: Activity Diagram: Login

In figura (4.6), viene riportato l'activity diagram relativo al caso d'uso "Login". Sono state definite due partizioni: la prima rappresenta lo studente registrato che vuole effettuare l'autenticazione e la seconda rappresenta il sistema. Lo studente inserisce le proprie credenziali di accesso la cui precondizione è che queste siano valide. In seguito, il sistema provvede ad autenticare lo studente registrato, che verrà indirizzato alla pagina contenente le funzionalità di gioco.

4.3.3 Activity Diagram: Change Password

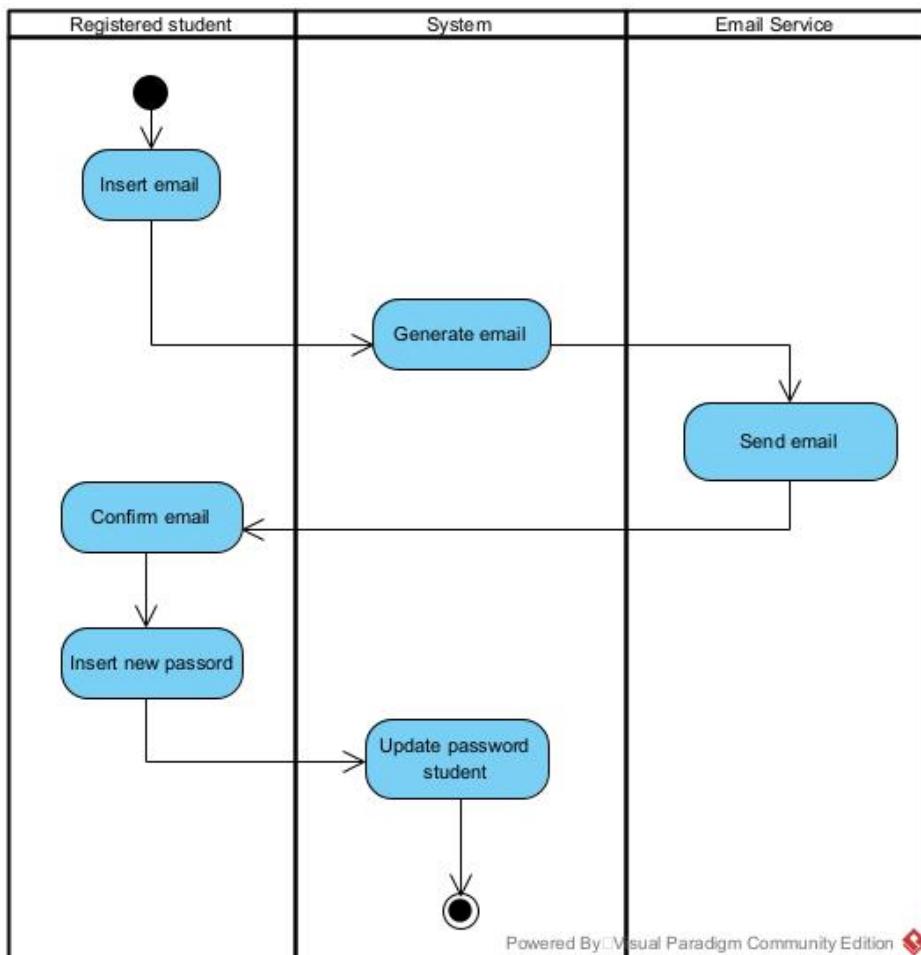


Figura 4.7: Activity Diagram: Change Password

In figura(4.7), viene riportato l'activity diagram relativo al caso d'uso "Change Password". Sono state definite tre partizioni: la prima rappresenta lo studente registrato che vuole effettuare il cambio della password, la seconda rappresenta il sistema e la terza il servizio di email esterno. Lo studente registrato inserisce la propria email ed attende una email che gli consentirà di per poter continuare l'operazione di cambio password. Lo studente, tramite la conferma, inserirà la nuova password. A valle dell'operazione, il sistema aggiornerà la password dello studente all'interno della base dati.

4.4 System Domain Model

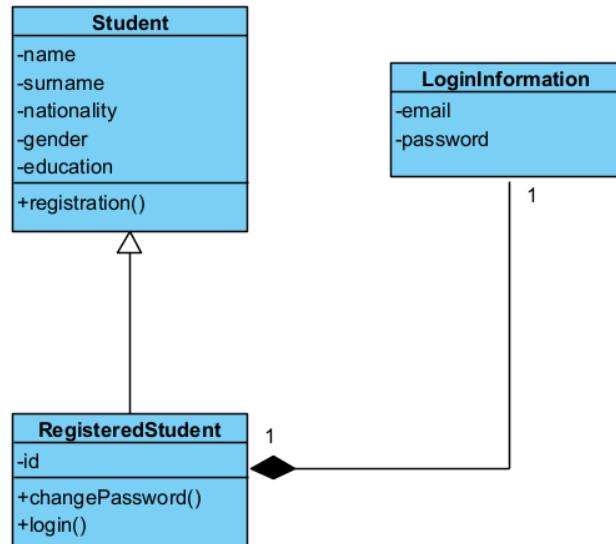


Figura 4.8: System Domain Model

Il system domain model descrive gli oggetti, i concetti e le interazioni all'interno del dominio del sistema, senza specificare ancora come questi saranno implementati come classi e oggetti nel sistema software.

In figura (4.8) viene riportato il system domain model che presenta una classe **Student** contenente come attributi, i dati personali dello studente, mentre la *registration()* è definita come responsabilità della classe. La classe **RegisteredStudent** eredita gli attributi e le responsabilità dalla classe **Student** aggiungendo l'attributo *id*, che rappresenta l'*id* dello studente. Le responsabilità associate allo studente registrato sono il *login()* per l'autenticazione e il *changePassword()* per il cambio password. Allo studente registrato vengono associati gli attributi "email" e "password" che sono contenuti nella classe **LoginInformation**.

Capitolo 5

Fase di Progettazione

Il seguente capitolo specifica la fase di progettazione che è stata eseguita dal team, partendo dai diagrammi di analisi. La documentazione di progettazione fornirà una descrizione dettagliata dei requisiti, delle scelte architettoniche, della struttura dei componenti, dei flussi di dati e delle interazioni tra le parti del sistema.

5.1 Pattern Architetturale MVC

Il modello MVC (Model-View-Controller) è un pattern architetturale ampiamente utilizzato nel processo di sviluppo del software. Suddivide il sistema in tre componenti principali: Model, View e Controller. Il Pattern architettonico MVC separa in modo chiaro la logica di business, i dati e i dettagli dell’interfaccia utente. Ciò consente una maggiore modularità, facilità di manutenzione e riutilizzo del codice. Inoltre, permette lo sviluppo parallelo di diverse componenti, poiché ognuna ha una responsabilità specifica.

5.1.1 Spring MVC

Spring MVC è un framework utilizzato per realizzare applicazioni web basate sul modello MVC. Il framework Web MVC di Spring¹, è guidato dalle *request*, progettato attorno a un Servlet centrale che invia le richieste ai controller e offre alte funzionalità che facilitano lo sviluppo di applicazioni Web. In figura (5.1) viene mostrato il flusso di elaborazione delle richieste di Spring Web MVC DispatcherServlet.

I vantaggi principali nell’utilizzo di Spring MVC sono stati:

¹<https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>

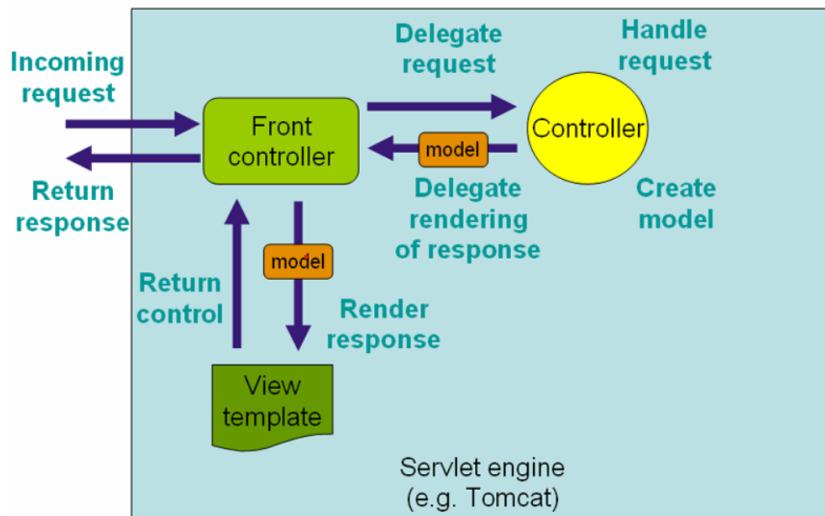


Figura 5.1: Spring MVC

- **Separazione dei compiti**: adattamento al pattern architetturale MVC che consente di separare le diverse responsabilità dell'applicazione.
- **Ampia flessibilità**: eterogeneità nella scelta degli strumenti e delle tecnologie da utilizzare nell'applicazione; facile integrazione con altri framework e librerie.
- **Gestione delle richieste e delle risposte**: fornisce un sistema di gestione delle richieste e delle risposte che consente di mappare richieste HTTP a metodi specifici del controller utilizzando annotazioni come `@RequestMapping`.

5.2 Component Diagram

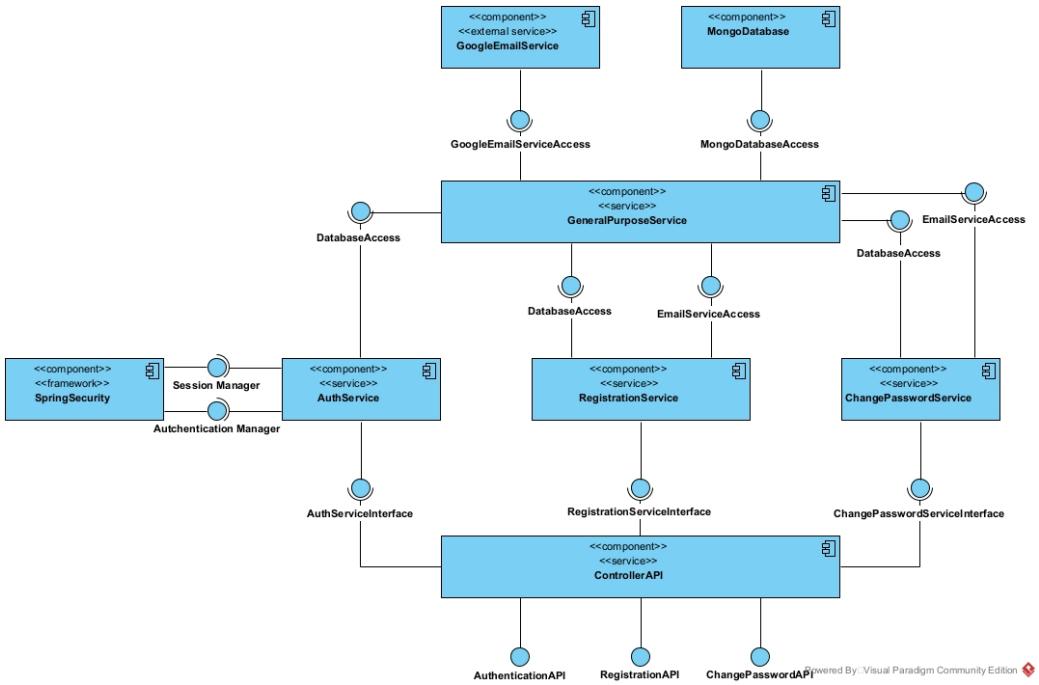


Figura 5.2: Component diagram

Nella figura (5.2) viene mostrato il component diagram che fornisce una panoramica ad alto livello dell’architettura del sistema e mostra come i vari componenti interagiscono tra loro per fornire funzionalità complessive. Il sistema complessivo espone all’esterno i servizi per la registrazione, l’autenticazione e il cambio password. I componenti di cui il sistema si compone sono:

- **Controller API:** componente che funge da *Façade* esponendo all'esterno i servizi: *AuthenticationAPI*, *RegistrationAPI*, *ChangePasswordAPI* e utilizza i servizi: *AuthServiceInterface* fornito dal component *AuthService*; *RegistrationServiceInterface* fornito dal component *RegistrationService*; *ChangePasswordServiceInterface* fornito da *ChangePasswordService*.
- **AuthService:** componente che permette l’acquisizione dei dati dello studente ed effettuare il processo di autenticazione. Per fare ciò, utilizza i servizi offerti da *GeneralPurposeService* nel caso in cui si utilizzassero le REST API. Diversamente, quando si intende utilizzare la versione in

locale viene sfruttato il component *SpringSecurity* che offre funzionalità di gestione della sessione utente e dell'autenticazione nativa in Spring.

- **RegistrationService:** componente che offre servizi di gestione della registrazione dei dati verso il database. Anch'esso, si avvale dei servizi offerti da GeneralPurposeService.
- **ChangePasswordService:** componente che offre i servizi di gestione del cambio password. Anch'esso, si avvale dei servizi offerti da GeneralPurposeService.
- **GeneralPurposeService:** componente che offre diversi servizi utilizzabili nel sistema. I servizi offerti sono: DatabaseAccess, EmailServiceAccess e altri servizi di utilità.
- **GoogleEmailService:** componente esterno che offre il servizio di invio email per la convalida del profilo e per il cambio password.
- **MongoDatabase:** componente esterno che indica la base dati reale.

5.3 Sequence Diagram: Progettazione

Il team ha lavorato al raffinamento dei sequence diagram di analisi seguendo il pattern architetturale MVC ed introducendo vari scenari alternativi.

5.3.1 Sequence Diagram: Registration

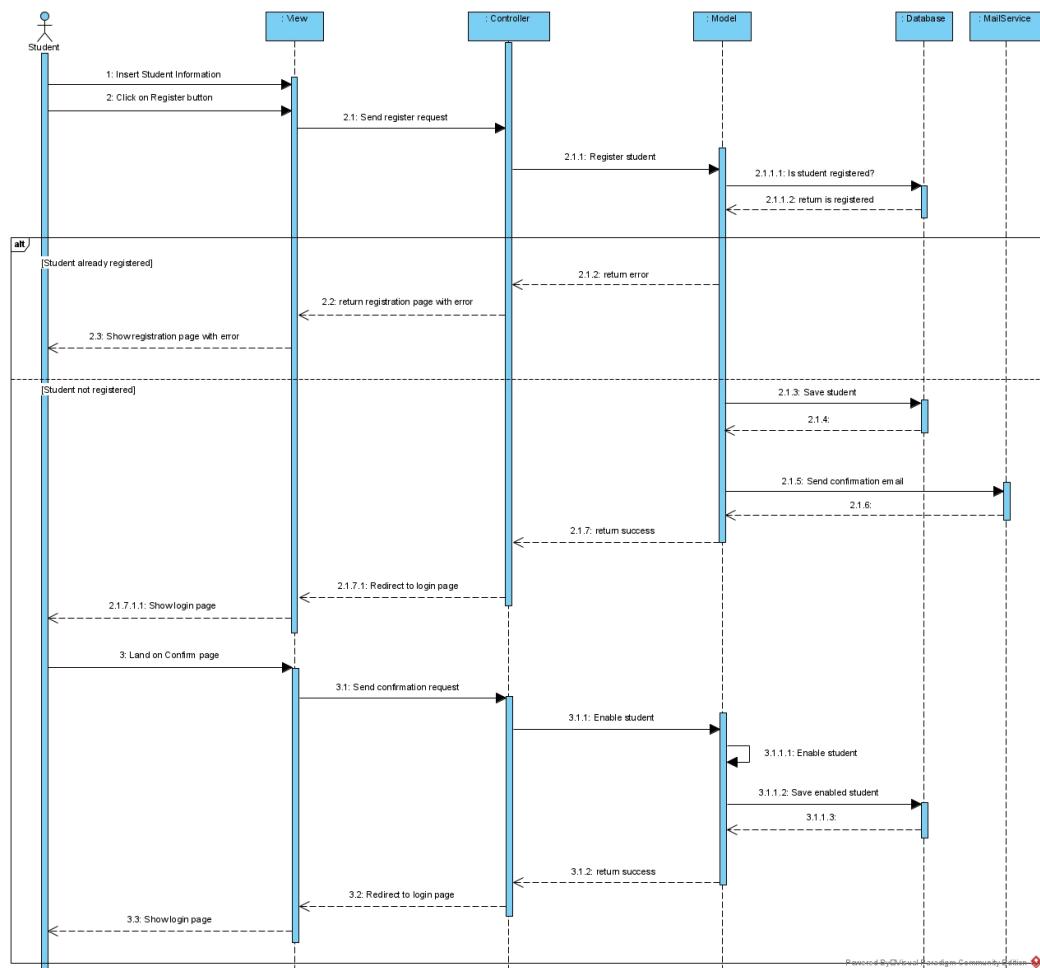


Figura 5.3: Sequence Diagram: Registration

In figura (5.3), viene riportato il sequence diagram del caso d'uso "Registration". Lo studente inserisce le proprie informazioni e clicca il pulsante di registrazione sulla pagina di registrazione fornita da una **View**; viene inviata una richiesta di registrazione al **Controller**, il quale, avviverà le operazioni di registrazione dello studente tramite il **Model**. Quest'ultimo, effettua una

verifica sulla base dati, per controllare se lo studente non sia già registrato all'applicazione. Se lo studente è già registrato, gli verrà mostrata una pagina di errore, altrimenti, i dati dello studente saranno salvati sulla base dati. Il Model contiene tutti i servizi offerti ai Controller per l'accesso ai dati e le funzionalità di gestione della loro persistenza. In particolare, dal Model, verrà richiesto il servizio di invio di conferma email per validare l'email dello studente. In attesa di validazione della propria email, lo studente verrà indirizzato sulla pagina di login. Tramite URL, lo studente confermerà la propria email e il Controller darà il permesso di abilitazione alla registrazione. A quel punto, il Model abiliterà lo studente e salverà l'abilitazione sulla base dati. A registrazione completata, lo studente verrà indirizzato sulla pagina di login.

5.3.2 Sequence Diagram: Login

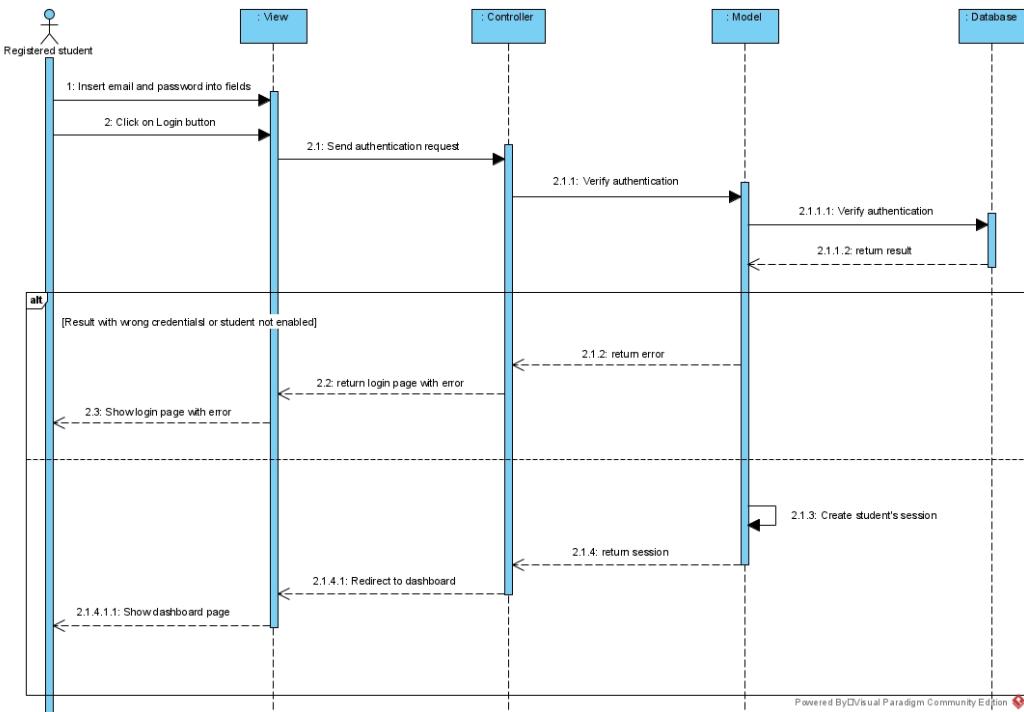


Figura 5.4: Sequence Diagram: Login

In figura (5.4) viene riportato il sequence diagram del caso d'uso "Login". Lo studente registrato, inserisce la propria email e la password e preme il bottone di login che è visibile sulla pagina Login fornita dalla **View**. Al **Controller** arriva una richiesta di autenticazione, quindi, si avvierà la verifica dell'autenticazione. In particolare, tramite **Model** si ricercherà se l'email dello studente è presente all'interno della base dati. Se lo studente non è registrato, oppure, se le credenziali inserite non sono associate a nessuno studente abilitato nella base dati, quest'ultimo verrà indirizzato su pagina di login che mostrerà una notifica di errore di credenziali non valide. Se la verifica va a buon fine, verrà creata una sessione per quale lo studente risulta essere autenticato e gli verrà mostrata la dashboard contenente le funzionalità di gioco e lo storico delle partite giocate.

5.3.3 Sequence Diagram: Change Password

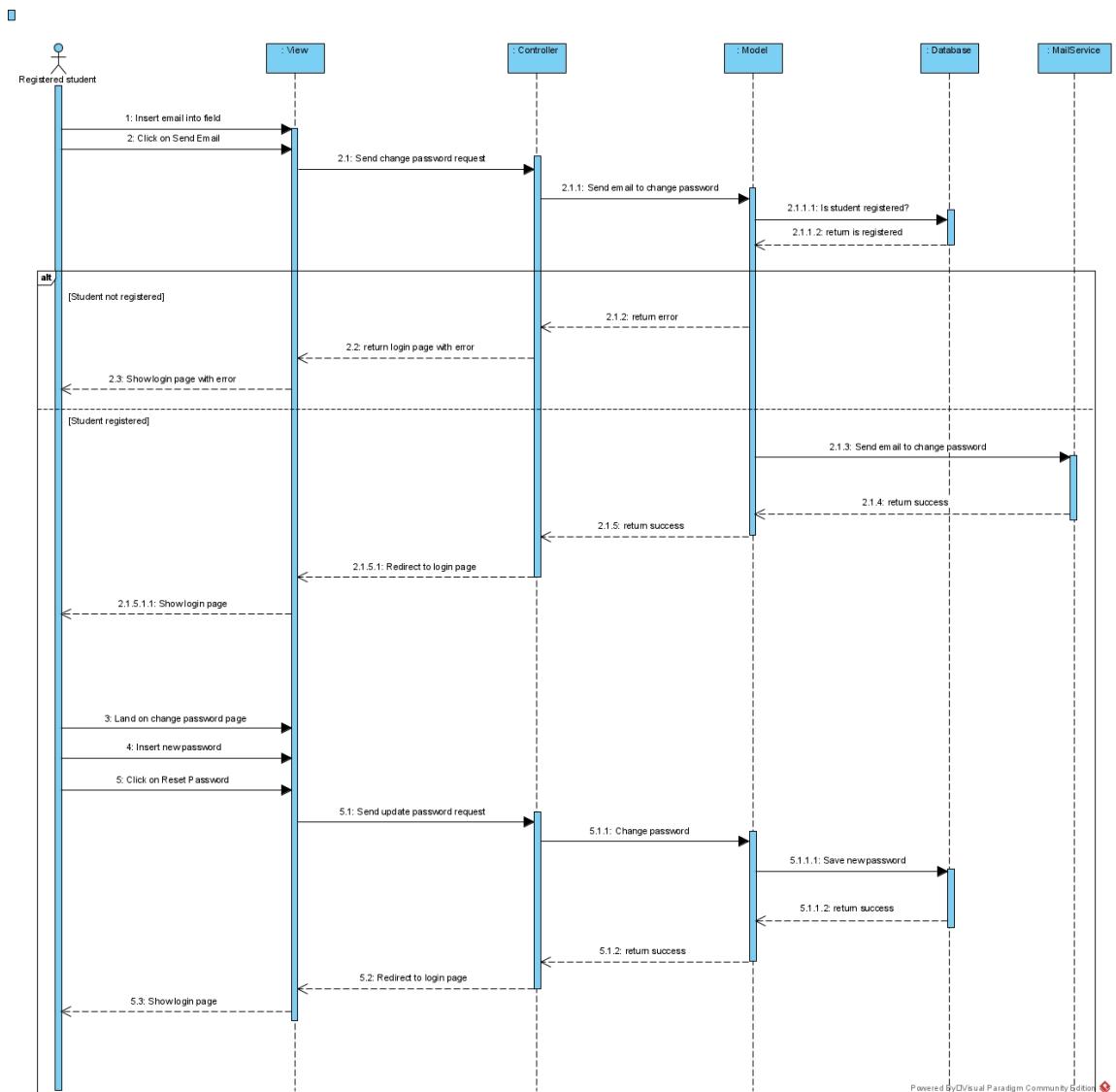


Figura 5.5: Sequence Diagram: Change Password

In figura (5.5) viene riportato il sequence diagram del caso d'uso "Change Password". Lo studente registrato, inserisce l'email su una pagina dedicata alla verifica dell'email; una volta premuto il bottone "Send Email", attende una email contenente un URL che consentirà di avviare la procedura di cambio password. Una volta che lo studente ha inserito la propria email, la richiesta di cambio password arriva al **Controller**, il quale, avvierà l'operazione di cambio password. Tramite **Model**, si controlla se lo studente che

intende cambiare la propria password, sia uno studente già registrato all'applicazione. Se lo studente non è registrato, visualizzerà una notifica di errore e verrà indirizzato sulla pagina di login. Viceversa, se lo studente è registrato, gli verrà inviata tramite il servizio **MailService**, una email contenente un URL. Nell'attesa, lo studente visualizzerà la pagina di login. Lo studente tramite URL, verrà indirizzato ad una pagina fornita dalla **View** dedicata al cambio password, in cui inserirà la nuova password che intende salvare. La View, mediante il click originato sul pulsante di "Reset Password", inoltrerà al **Controller**, una richiesta di update della password. Il Controller avvierà le operazioni di aggiornamento mediante il Model. La password verrà aggiornata e salvata sul database; infine, lo studente verrà indirizzato sulla pagina login.

5.4 Glossario

Si riporta il seguente glossario contenente la terminologia utilizzata in fase di progettazione e in fase di implementazione (Capitolo 6):

Termine	Descrizione
web app	tipologia di applicazione sviluppata
confirmationToken	codice utilizzato in fase di registrazione per la verifica del profilo dello studente
passwordToken	codice utilizzato in fase di cambio password per indirizzare su una pagina dedicata all'inserimento di una password
URLweb	identifica il dominio del sito web
URLpath	identifica il percorso di una pagina del sito web
studentRepository	oggetto che consente di effettuare operazioni sulla base dati

5.5 Package Diagram

Il package diagram mostra l'organizzazione e le dipendenze tra i package in un sistema o un'applicazione software. In particolare, esso fornisce una panoramica sul raggruppamento dei diversi package nel sistema e sulle loro interazioni. Il package diagram in figura (5.6), mostra come i moduli sono stati ripartiti all'interno del progetto rispettando il pattern MVC.

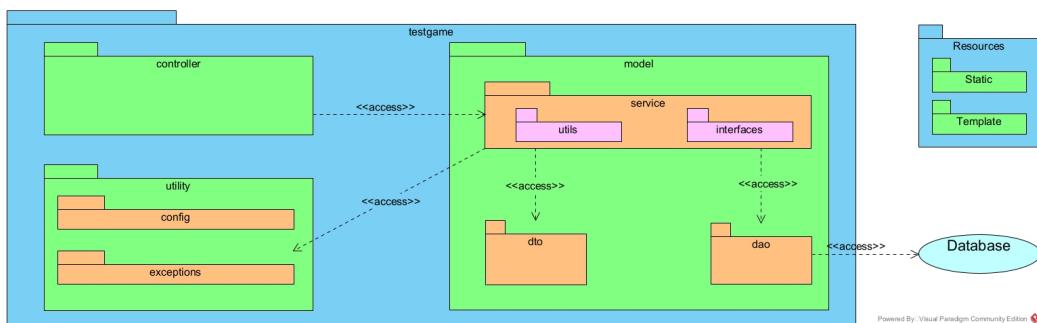


Figura 5.6: Package Diagram

Nel caso specifico, le *View* sono contenute nel package "*Resources*", che contiene tutte le User Interfaces con le quali lo studente registrato/non registrato interagisce. In definitiva, tale pattern permette una separazione netta tra i componenti di controllo, presentazione e gestione dei dati. In riferimento alla figura (5.6), si distinguono quattro pattern differenti:

- **Model:** Nel contesto del pattern MVC il model si occupa di effettuare le operazioni sui dati tramite i servizi che offre ai Controller. In questo caso, il package *Model* include diversi componenti correlati al Model:
 1. *Servizi (Services)*: Gestiscono i dati, le operazioni richieste dai Controller e si occupano della comunicazione con altre componenti del sistema. In particolare, tali servizi forniscono funzionalità specifiche per l'applicazione, come ad esempio la gestione degli studenti, la gestione dei servizi di mail ecc..
 2. *DTO (Data Transfer Objects)*: I 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.
 3. *DAO (Data Access Objects)*: I DAO 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, i DAO presenti nel package "model" consentono al Model di interagire con il database o altre fonti di dati per effettuare operazioni di lettura o scrittura.
- **Controller:** Il suo compito principale è quello di ricevere le richieste dell'utente provenienti dalla View, elaborarle e prendere le decisioni adeguate per soddisfare tali richieste. In particolare, si occupa di coordinare il flusso dei dati tra il Model e la View. Quando l'utente interagisce con l'interfaccia utente, ad esempio inviando un modulo o cliccando su un pulsante, la View cattura questa azione e la comunica al Controller. Il Controller, che analizza la richiesta, determina quale operazione deve essere eseguita sul Model, infatti, una volta che il Controller ha elaborato la richiesta, comunica con il Model per ottenere o aggiornare i dati necessari.
- **View:** Nel contesto del pattern MVC, la View rappresenta la componente responsabile della presentazione dell'interfaccia utente e della visualizzazione dei dati provenienti dal Model. La sua funzione principale è quella di mostrare le informazioni in un formato comprensibile e interattivo. Tipicamente, la View si occupa di rendere visibili gli elementi dell'interfaccia utente, come pulsanti, caselle di testo, tavole e grafici, consentendo all'utente di interagire con essi. Per questo motivo, essa può essere implementata utilizzando tecnologie come HTML, CSS e JavaScript per creare pagine web.

- **Utility:** Il package "Utility" è stato aggiunto per fornire funzioni di utilità all'interno dell'applicazione, in particolare per la configurazione di aspetti di sicurezza e di definizione di eventuali errori che si possono presentare.

5.6 WebApp Class Diagram

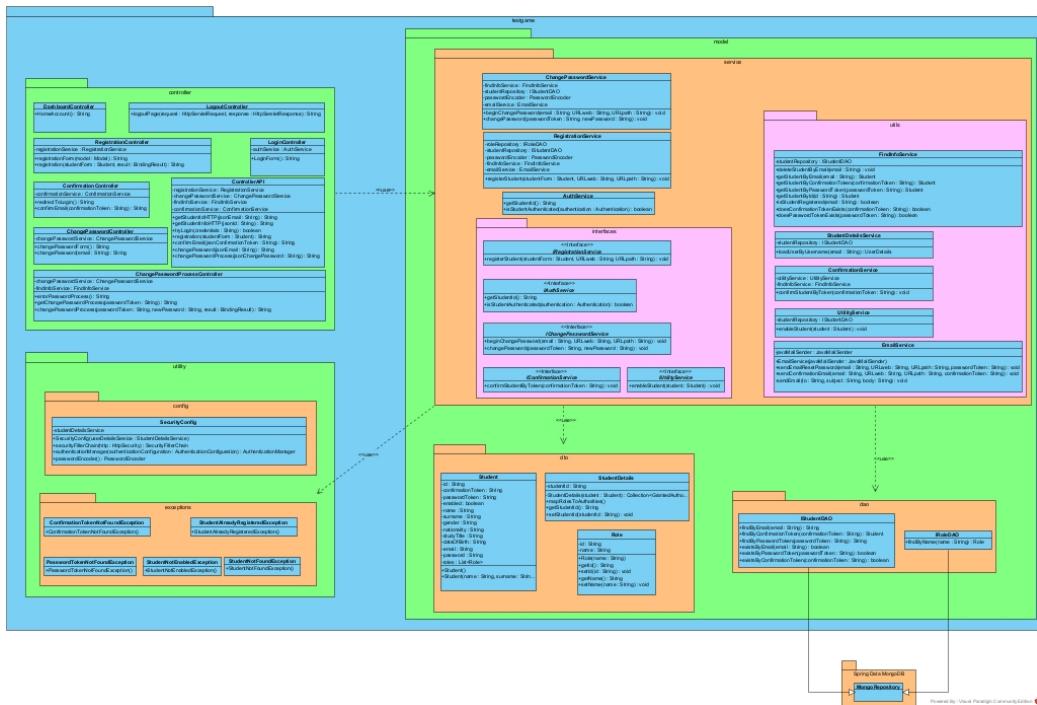


Figura 5.7: Class Diagram

Il diagramma in figura (5.7) mostra, insieme ai Package relativi al Modello MVC, il package "Utility", il quale contiene le classi di utilità relative alla gestione delle eccezioni e alla sicurezza dell'intera applicazione offrendo vari servizi, tra cui quelli di crittografia (per la password utente). In particolare, questo package verrà utilizzato da Model in accordo con le *dipendenze d'uso* in figura (5.7). Di seguito, si analizza nel dettaglio ognuno dei package, evidenziando la relazione d'uso con gli altri.

5.6.1 View

All'interno della View, sono presenti le diverse pagine con cui lo studente può interagire mentre naviga nella nostra WebApp. Tali pagine sono caratte-

rizzate da diversi elementi come *bottoni*, *text field*. Sono state utilizzate diverse UI per le varie finestre di navigazione in cui, ad ogni bottone, è associato un handler specifico per la gestione dei click su di esso. In particolare:

- **Registration Page:** La pagina di registrazione presenta un Form in cui lo studente inserisce tutti i dati personali, la propria mail e la password con cui intende registrarsi al sito. È presente un bottone "Register" che, una volta cliccato, permette di avviare il processo di registrazione al sistema. In più è presente anche un collegamento "Annulla" che permette di annullare l'operazione di registrazione e tornare alla schermata principale di login. [Figura(2.4)]
- **Login Page:** La pagina di login presenta due "text field" adibiti all'inserimento rispettivamente della mail e della password. In particolare è presente: un tasto Login che, una volta cliccato, avvia il processo di autenticazione per mostrare la dashboard personale dello studente; un link "Forgot Password", che permette di essere reindirizzati alla pagina di cambio password; un link "create new Account" che permette di essere reindirizzati alla pagina di registrazione di cui sopra. [Figura(2.3)]
- **Dashboard Page** Questa pagina rappresenta il "profilo" dello studente nel quale esso può visualizzare la propria mail, il punteggio accumulato nel gioco e due buttoni che permettono rispettivamente di *unirsi ad un nuovo gioco* o *visualizzare lo storico delle partite*. Infine, è presente anche un collegamento "*logout*" che permette allo studente di disconnettersi dal sistema. [Figura(2.2)]
- **ChangePassword Page:** La pagina di cambio password prevede inizialmente un text field nel quale lo studente inserisce la propria mail per il recupero password e un bottone "*Send Mail*" che avvia la procedura di invio mail contenente il link per il cambio password. [Figura(2.5)]
- **ChangePasswordProcess Page** Questa pagina viene mostrata in risposta ad un click sul link di cambio password. In particolare, essa è formata da due text field adibiti all'inserimento della password e alla sua conferma. Infine, è presente un bottone "Reset" che provvede ad avviare la procedura di aggiornamento della password sulla base dati. [Figura(2.6)]

5.6.2 Controller

Il controller viene chiamato in causa dalla View. Per semplificare le chiamate e rendere indipendente la gestione di ogni caso d'uso nel package Con-

troller, sono stati inseriti diversi controller ognuno dei quali adibito ad una specifica funzione. In questo modo, quindi, è stato ridotto l'accoppiamento tra le varie classi. In particolare, le classi deputate al ruolo di controller sono le seguenti:

- **RegistrationController** intercetta le richieste di tipo GET e POST sulla pagina di registrazione dello studente. In particolare, la GET restituisce il Form adibito alla registrazione mentre la POST richiama il servizio di registrazione dello stesso sulla base dati e gestendo eventuali errori;
- **LoginSpringController** Intercetta richieste di tipo POST, ed è implementato direttamente in Spring Security. Verifica l'autenticità dello studente che sta effettuando il login;
- **ConfirmationController** acquisisce mediante URL il *confirmation-Token* inviato allo studente tramite email ed effettua una ricerca sul database dello studente con quel token e lo abilita,in caso di token errato, restituisce errore;
- **ChangePasswordController** restituisce il Form per l'immissione della mail alla quale inviare il token per il cambio password. Dopodiché il controller effettua una ricerca sul database dello studente con quella mail e, in caso di successo, restituisce la schermata di login, inviando una email con il *passwordToken*, che servirà a generare una view in cui immettere la nuova password; mentre in caso di ricerca fallita, restituisce errore;
- **ChangePasswordProcessController** viene attivato da una GET sulla pagina *changePasswordProcess*. Acquisice il *passwordToken* presente nell'URL e fornisce il form per poter cambiare la password dello studente. Il token serve a identificare univocamente lo studente che ha effettuato la richiesta del cambio password, inviata tramite POST;
- **DashboardController** restituisce la View del profilo dello studente una volta effettuato il login;
- **LogoutController** che si occupa di disconnettere lo studente dal sistema all'atto della pressione sul link "logout";

5.6.3 Model

Il package Model, come accennato in precedenza, contiene al suo interno tre tipi di package: *Service*, *DTO* e *DAO*.

Service Il package Service contiene le classi che forniscono i servizi principali che vengono utilizzati dai controller. Come mostra la figura (5.7) i controller utilizzano questi servizi che, a loro volta, utilizzano i package DTO, DAO e utils. In particolare:

- **RegistrationService**, consente di gestire l'operazione di salvataggio sulla base dati dello studente che sta effettuando la registrazione.
- **ChangePasswordService**, consente di gestire l'operazione iniziale di cambio password attraverso l'invio di una mail contenente un link per il cambio password all'indirizzo di posta inserito nell'apposito form. Nel caso in cui la mail inserita non sia presente nel database o non sia valida, restituisce una schermata di errore.
- **AuthService**, consente di gestire l'autenticazione dello studente e fornire attraverso un'apposita funzione l'ID dello studente autenticato.

Le rispettive interfacce dei servizi presentati implementano le rispettive interfacce contenute all'interno del rispettivo package "Interfaces". Un ulteriore package presente all'interno del Service è il package "utils" che contiene le seguenti classi:

- **FindInfoService** contiene tutti metodi CRUD per l'accesso al database.
- **StudentDetailsService** permette di caricare lo studente dalla base dati a partire dalla sua mail utilizzando gli appositi metodi messi a disposizione dallo StudentDAO.
- **ConfirmationService** permette di gestire ed effettuare la conferma dell'account attraverso l'apposita funzione che riceve in ingresso il token per la conferma della registrazione.
- **UtilityService** permette di gestire l'abilitazione dello studente quando egli clicca sul link di verifica account ricevuto tramite mail;
- **EmailService** permette di gestire l'invio della mail ad uno specifico account allegando al suo interno un link specifico a seconda che questo servizio venga chiamato per la conferma della registrazione o per il cambio password.

DTO Il package DTO contiene le classi adibite al trasferimento dei dati ai moduli dell'applicazione. In particolare:

- **Role**: classe che consente di creare, gestire e utilizzare i ruoli all'interno del sistema.
- **Student**: classe che consente di gestire e passare le informazioni dello studente tra le classi dell'applicazione.
- **StudentDetails**: classe che contiene diversi metodi che permettono di mappare i livelli di autorità dello studente, ritornare/settare l'ID dello studente.

DAO Il package DAO contiene le classi adibite alla persistenza dei dati, ovvero l'accesso al database e il trasferimento delle informazioni alle classi DTO in particolare dello studente. Nel nostro caso, Spring Data MongoDB mette a disposizione delle classi che implementano automaticamente le operazioni sulla base dati. Per fare ciò bisogna creare delle interfacce che ereditano le classi native di MongoDB. In particolare:

- **IRoleDAO** classe che gestisce l'accesso alle informazioni dei ruoli contenute nella base dati.
- **IStudentDAO** classe che gestisce l'accesso alle informazioni degli studenti contenute nella base dati.

5.6.4 Utils

Il package Utils è stato introdotto per contenere le classi relative alla gestione delle eccezioni e alla configurazione della sicurezza. In particolare sono stati definiti ulteriori due package: "config" ed "exceptions". Per quanto riguarda il package "config":

- **SecurityConfig** implementa le impostazioni di sicurezza attraverso il metodo FilterChain che permette di filtrare le richieste e gli accessi a specifici indirizzi in base alle autorizzazioni messe a disposizione da SpringSecurity. Inoltre implementa anche la logica per gestire il login e il logout.

Per quanto riguarda il package "exceptions":

- **ConfirmationTokenNotFoundException** è un'eccezione lanciata quando il token di conferma non viene trovato.

- ***StudentAlreadyRegisteredException*** è un’eccezione lanciata quando uno studente già registrato con quella specifica mail cerca di registrarsi nuovamente.
- ***PasswordTokenNotFoundException*** è un’eccezione lanciata quando il token relativo al cambio password non viene trovato all’interno della base dati.
- ***StudentNotEnabledException*** è un’eccezione lanciata quando uno studente, che non ha provveduto a confermare la propria mail, tenta di accedere al sistema mediante login.
- ***StudentNotFoundException*** è un’eccezione lanciata quando lo studente non viene trovato all’interno della base dati.

5.7 Class Diagram con dipendenze di ciascun Controller

Si riportano nei seguenti sottocapitoli le dipendenze di ciascun Controller verso ciascuna classe.

5.7.1 RegistrationController

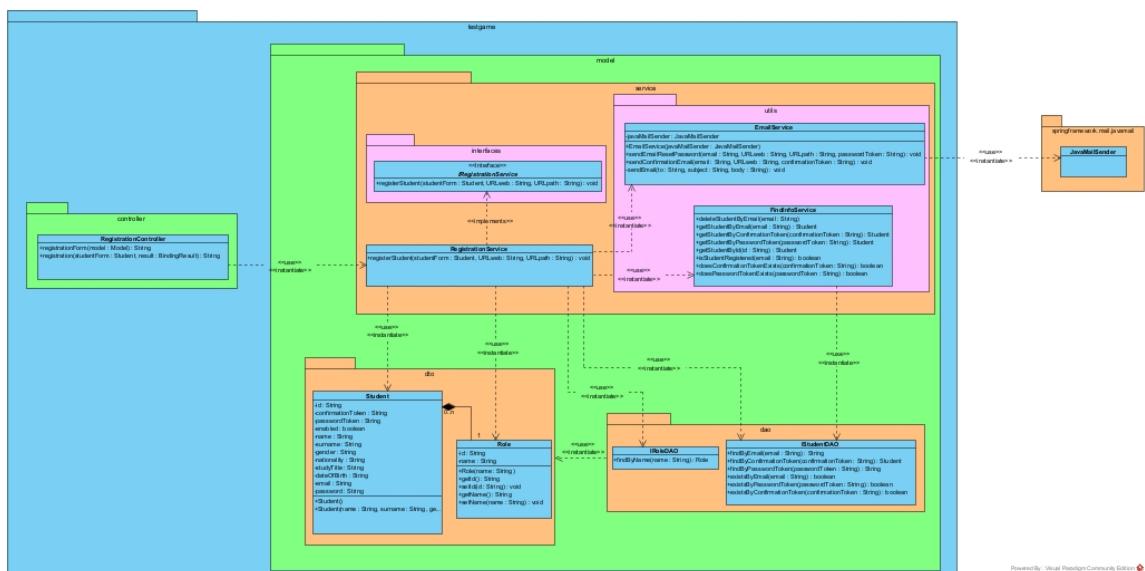


Figura 5.8: Class diagram con dipendenze: RegistrationController

5.7.2 ConfirmationController

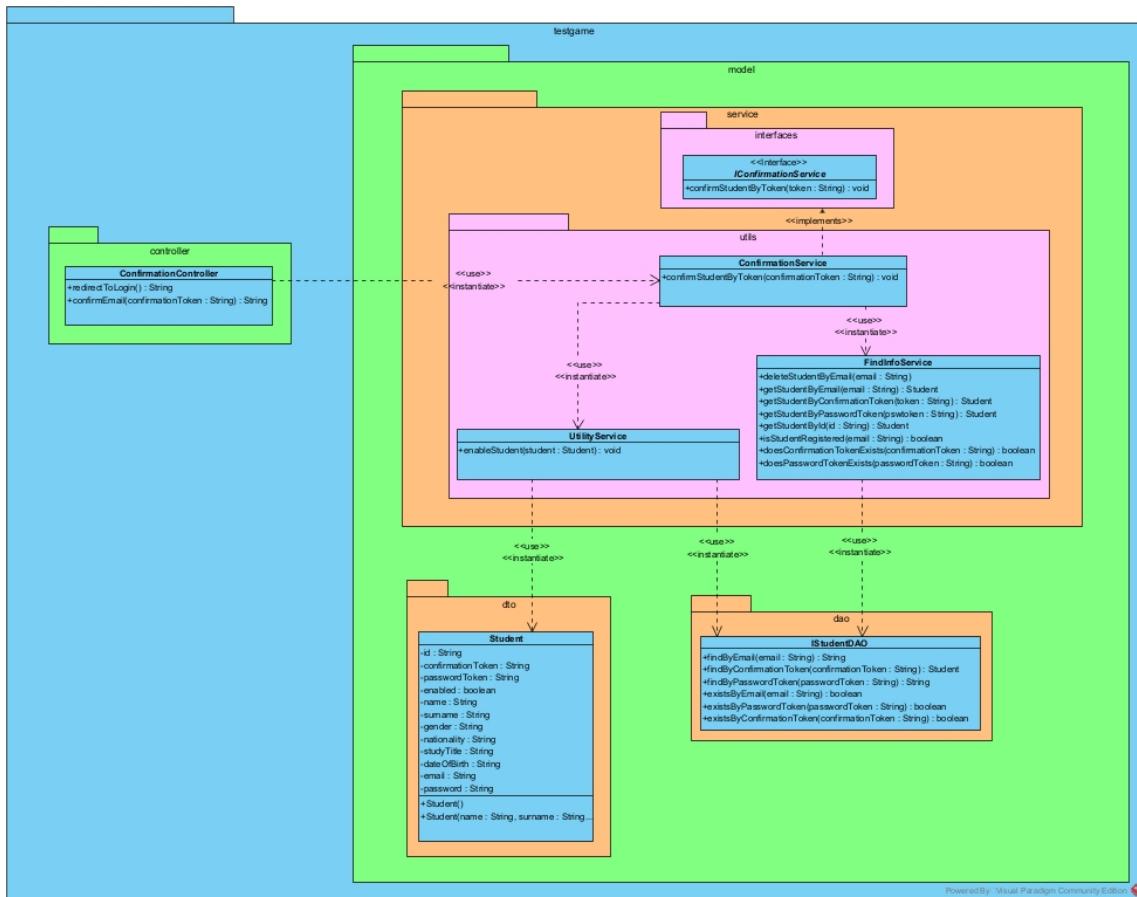


Figura 5.9: Class diagram con dipendenze: ConfirmationController

5.7.3 LoginController

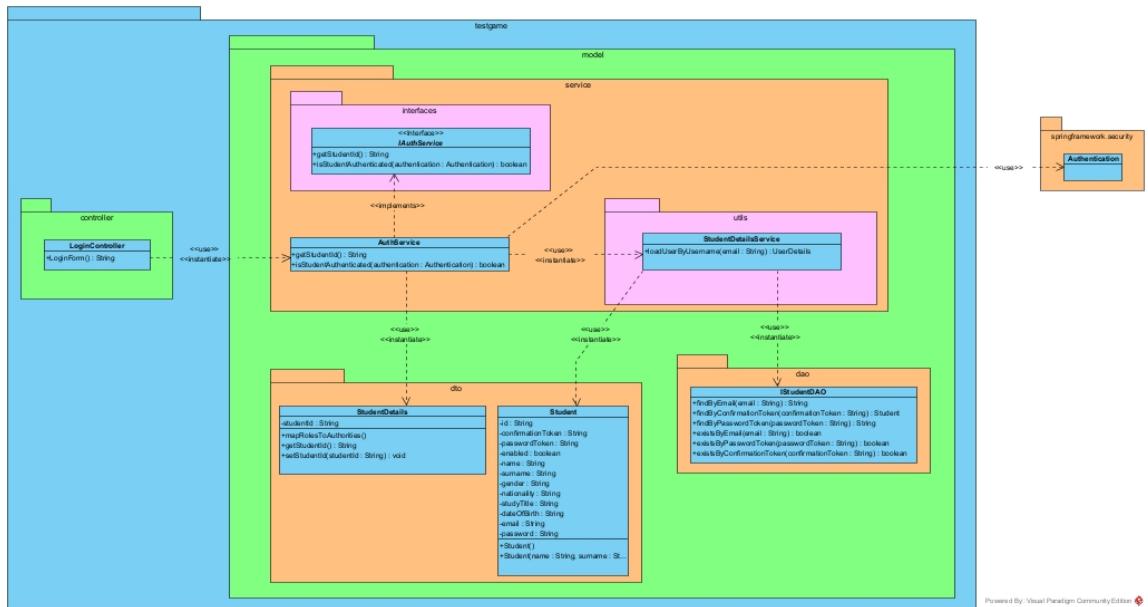


Figura 5.10: Class diagram con dipendenze: LoginController

5.7.4 ChangePasswordController

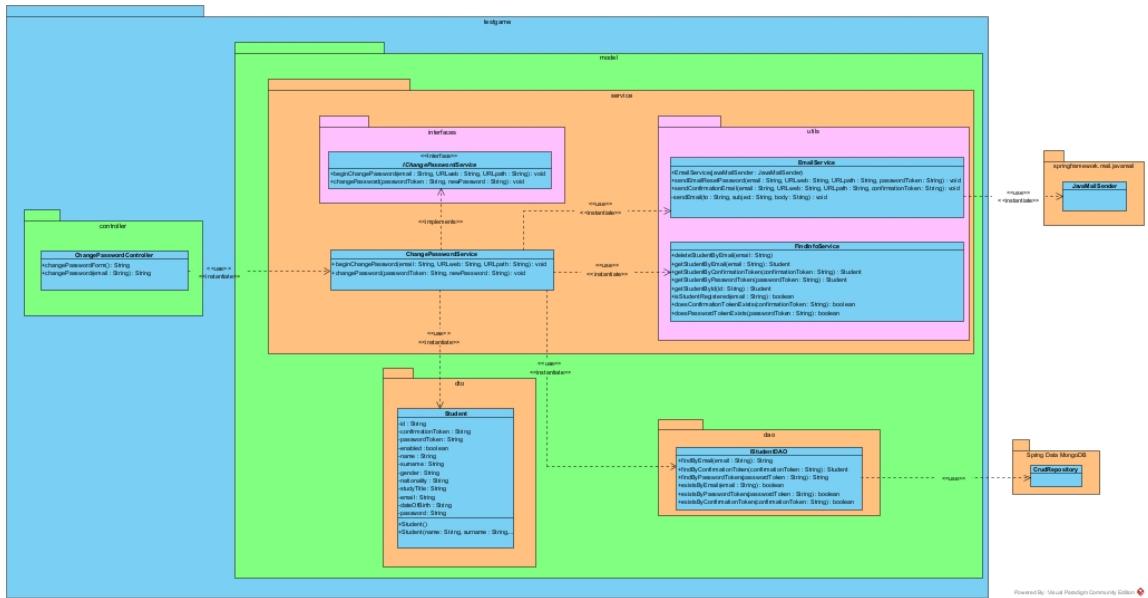


Figura 5.11: Class diagram con dipendenze: ChangePasswordController

5.7.5 ChangePasswordProcessController

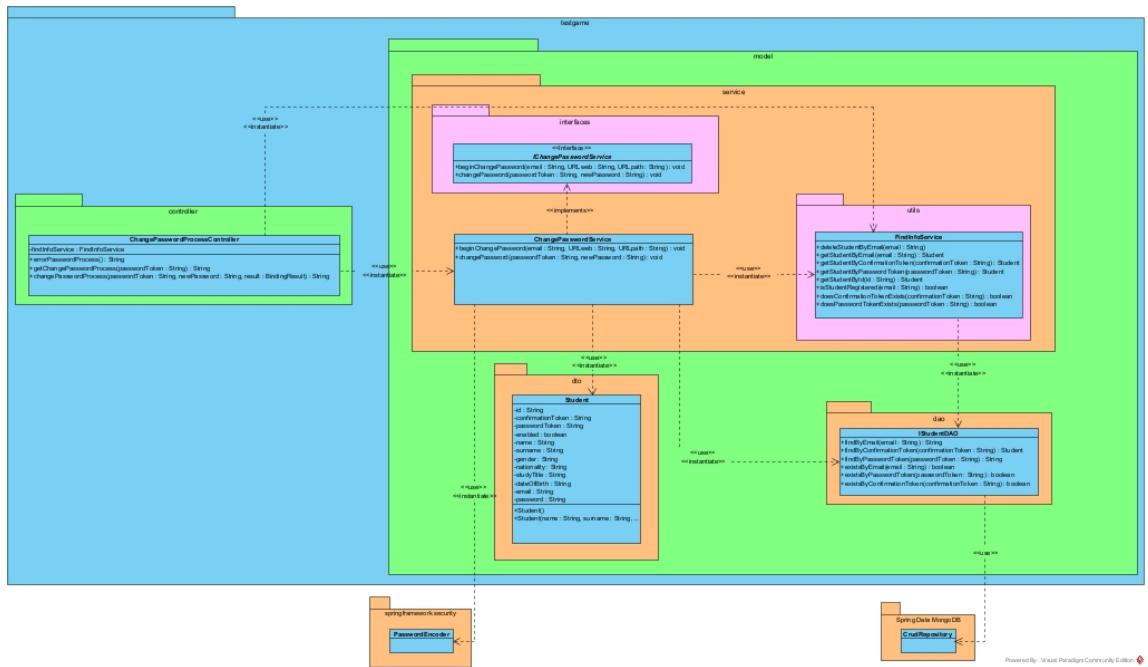


Figura 5.12: Class diagram con dipendenze: ChangePasswordProcessController

5.7.6 LogoutController

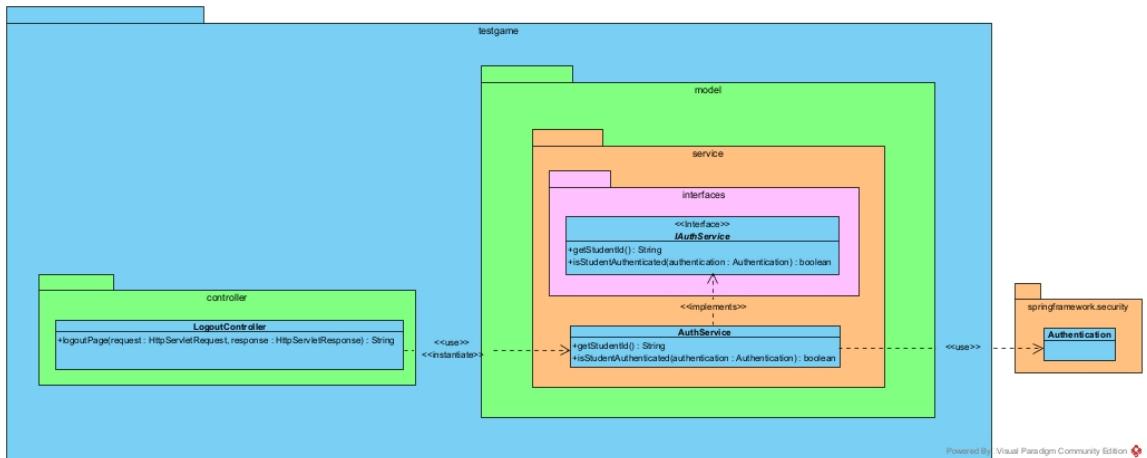


Figura 5.13: Class diagram con dipendenze:LogoutController

5.8 Deployment

Il deployment, o distribuzione, rappresenta un passaggio cruciale nel ciclo di vita di un'app, in cui l'applicazione viene resa accessibile agli utenti finali tramite un ambiente di produzione online. Questo processo coinvolge una serie di attività complesse, che vanno dalla preparazione dell'ambiente di hosting alla configurazione dei server, all'installazione dei software necessari e alla gestione delle dipendenze. Un deployment ben eseguito è fondamentale per garantire l'affidabilità, la scalabilità e le prestazioni ottimali dell'applicazione, assicurando un'esperienza utente positiva.

Di seguito è riportato il deployment diagram, il quale specifica come l'applicazione viene distribuita e allocata all'interno dei componenti hardware e Software. In questo caso, l'applicazione sviluppata viene eseguita su un Computer che funge da Server sul quale è installato il sistema operativo Windows 11. Nel sistema operativo viene eseguita l'applicazione Docker Desktop, la quale si occupa della gestione delle istanze di tipo Docker. Per fare ciò Docker Desktop crea un container nel quale vengono istanziati due Docker, uno per l'applicazione ed uno per Database. Si noti che per permettere l'esecuzione dell'applicazione del Docker viene creato un ambiente di esecuzione dato da JVM17. Per far comunicare i due Docker, questi vengono connessi ad una rete privata creata ad hoc da Docker Desktop. In particolare, il Database viene connesso all'applicazione attraverso il protocollo TCP alla porta 27017.

Inoltre viene esposta all'esterno del Docker la possibilità di potersi connettere tramite un qualsiasi browser, da un qualsiasi sistema operativo, all'applicazione attraverso il protocollo HTTP alla porta 8080. Inoltre, attraverso il protocollo HTTP, viene richiamato il servizio Email di Google.

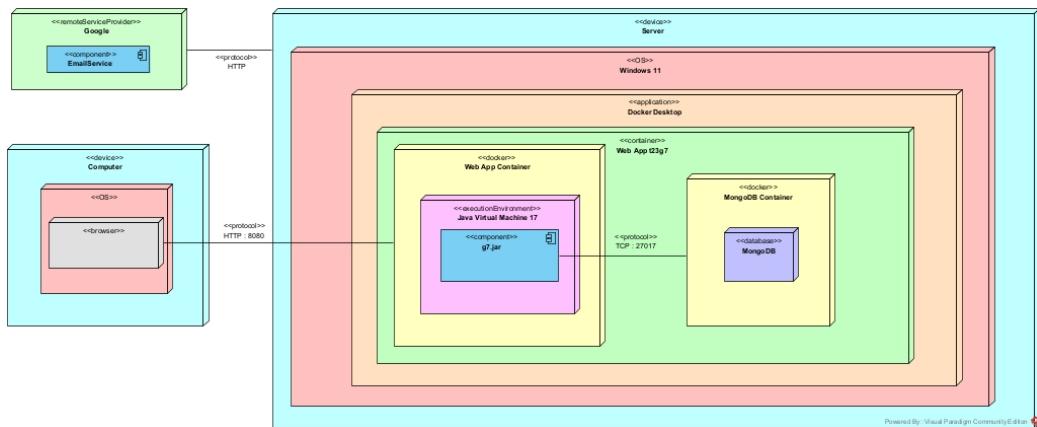


Figura 5.14: Deployment Diagram

5.9 Integrabilità

5.9.1 API Locali

Le seguenti API sono state pensate per essere utilizzate da altri moduli nello stesso ambiente utilizzato durante lo sviluppo. Per sapere i requisiti tecnici e le tecnologie implementate si faccia riferimento a Tecnologie Utilizzate

Le API offrono l'accesso ai seguenti servizi:

- **Registrazione**
- **Login**
- **Cambio Password**

5.9.2 Strutture Dati

5.9.2.1 Student

La struttura dati utilizzata per memorizzare le informazioni personali e del profilo dello studente è la seguente:

```
// Informazioni personali
private String name;
private String surname;
private String gender;
private String nationality;
private String studyTitle;
private String dateOfBirth;

// Informazioni del profilo utente
private String email; //email fornita in fase di
                      registrazione
private String password; //password per accedere al
                        profilo
private String confirmToken; //token di conferma generato
                            all'atto della registrazione per verificare il
                            profilo. Usato con l'URL del sito. Cancellato una
                            volta effettuato il processo. Invalidato dopo un certo
                            tempo.
private String passwordToken; //token generato all'atto
                            del cambio password per creare un URL unico che
                            permette di accedere alla pagina di cambio. Cancellato
```

```

una volta effettuato il processo. Invalidato dopo un
certo tempo.
private boolean enabled;
private List<Role> roles = new ArrayList<>(); //lista di
ruoli associati allo studente (in questo caso "STUDENT"
")

```

5.9.2.2 Role

La struttura dati utilizzata per memorizzare le informazioni per la gestione del ruolo dello studente sull'applicazione è la seguente:

```

private long int id
private String name; //nome del ruolo

```

5.9.3 Services

Di seguito sono riportati i metodi esposti dai Servizi in base al loro package di appartenenza.

5.9.3.1 Registrazione

Salvataggio dello studente nella base dati :

```

void registerStudent(model.dto.Student studentForm,
String URLweb, String URLpath)

```

Metodo che permette di salvare lo studente sulla base dati. Come parametro di ingresso deve essere passato un oggetto di tipo *Student* con tutti i campi popolati escluso l'id, che verrà creato e associato allo studente automaticamente. Inoltre viene generato un token di conferma dell'identità, che viene inviato tramite email allo studente appena registrato al fine di verificare la sua identità. L'oggetto *studentForm* può essere popolato da un apposito form.

Input:

- *studentForm* oggetto di tipo *Student* che deve contenere i campi pieni, al fine di salvare lo studente sulla base dati.
- *URLweb* parametro che verrà inserito nel corpo dell'email, e funge da link di conferma per confermare l'identità dello studente.

- *URLpath* parametro che verrà inserito nel corpo dell'email, concatenato con *URLweb*, che indica un percorso del sito che effettuerà l'abilitazione dello studente.

Return: Non ritorna valori. Eventuali errori che possono presentarsi, sono eccezioni che devono essere opportunamente gestite con try-catch.

Exception : *StudentAlreadyRegisteredException* si sta tentato di salvare nella base dati uno studente con una email già registrata.

5.9.3.2 Login

L'autenticazione dello studente viene gestita dal framework Spring Security, il quale fornisce un' interface chiamata *UserDetailsService*, che mette a disposizione tutte le funzioni per la gestione del login. In particolare, nel nostro caso, è stata utilizzata quest' interfaccia per la creazione di una classe ad hoc per l' autenticazione della Web App.

```
public class StudentDetailsService implements
    UserDetailsService
```

Questa classe utilizza il metodo *loadUserByUsername(String email)* la quale cerca per email lo studente nel database. In caso la ricerca abbia successo, carica le informazioni dello studente nella sessione della Web App, altrimenti lancia l'eccezione *UsernameNotFoundException*.

```
public String getStudentId()
```

Metodo che permette di ricavare l'ID dello studente registrato dalla sessione.

Return: Ritorna un valore di tipo String che rappresenta l'ID univoco associato all'utente autenticato nella sessione corrente, NULL se non è autenticato.

```
public boolean isStudentAuthenticated(org.springframework
    .security.core.Authentication authentication)
```

Metodo che restituisce *true* se nella sessione corrente uno studente è autenticato. Se non è autenticato restituisce *false*.

Input: *authentication* istanza di tipo Authentication di Spring Security che permette di ricavare le informazioni della sessione corrente.

Return: Se nella sessione corrente, uno studente è loggato, allora ritorna true, altrimenti false.

5.9.3.3 Change Password

Richiesta del Cambio Password

```
public void beginChangePassword(String email, String  
URLweb, String URLpath)
```

Metodo che permette di cominciare la procedura di cambio password. In particolare, controlla prima che lo studente sia registrato e che sia abilitato, e se vengono passati con successo genera un token di una pagina univoca, chiamato token password. Il token viene salvato sulla base dati e inviata tramite email allo studente. Se i controlli falliscono, restituisce le eccezioni descritte.

Input:

- *email* parametro di tipo String che permette di cercare sulla base dati lo studente e inviare il token da cliccare.
- *URLweb* url del sito su cui utilizzare il servizio
- *URLpath* url del percorso che verrà concatenato con *URLsite* per creare la pagina di procedura di cambio password. (es. */changePasswordProcess/*).

Return: Non ritorna valori.

Exception:

- *StudentNotFoundException* eccezione restituita quando l'email fornita in ingresso non corrisponde a nessuno studente nella base dati.
- *StudentNotEnabledException* eccezione restituita quando lo studente trovato con l'email fornita in ingresso non è abilitato.

```
public void changePassword(String passwordToken, String  
newPassword)
```

Metodo che permette di effettuare il salvataggio della nuova password sulla base dati. In particolare, viene fornito il token della password, preso dall'URL della pagina univoca e la nuova password in chiaro (NON cifrata). Alla fine del cambio, il token viene cancellato. Se il token della password non viene trovato, verrà restituita una eccezione.

Input:

- *passwordToken*: parametro di tipo String che funge da chiave di ricerca per selezionare lo studente che ha richiesto il cambio.
- *newPassword*: parametro di tipo String. È la nuova password scritta in chiaro da salvare nella base dati allo studente che ha richiesto il cambio.

Return: Non ritorna valori. In caso di errori, restituisce le eccezioni che devono essere opportunamente gestite con try-catch.

Exception: PasswordTokenNotFoundException ,eccezione restituita quando il token della password fornito in input non viene trovato nella base dati.

5.9.4 Utils

Di seguito vengono riportate le API dei servizi di Utilità.

5.9.4.1 ConfirmationService

```
public void confirmStudentByToken(String  
confirmationToken)
```

Cerca nella base dati lo studente con il token fornito in input e in caso di studente lo abilita, altrimenti ritorna un'eccezione.

Input: confirmationToken chiave di ricerca dello studente nella base dati.

Return: Non restituisce valori.

Exception: ConfirmationTokenNotFoundException, Eccezione restituita nel caso in cui non viene trovato il token nella base dati

5.9.4.2 EmailService

```
public void sendEmailResetPassword(String email, String  
URLweb, String URLpath, String passwordToken)
```

Metodo che sfrutta la SendMail per inviare una mail con uno specifico link attivo al cambio della password.

Input:

- *email* Parametro di tipo String che indica l'indirizzo email dell'utente a cui mandare la mail.

- *URLweb* Parametro di tipo String che indica l'URL del sito web sul quale collegarsi per ricevere il servizio.
- *URLpath* Parametro di tipo String che specifica il percorso "URL-web:8080/URLpath" al quale fare riferimento.
- *passwordToken* Parametro di tipo String che indica il token randomico per poter verificare la email dello studente in maniera univoca.

Return: Il metodo non ritorna nessun parametro.

```
public void sendConfirmationEmail(String email, String URLweb, String URLpath, String confirmationToken)
```

Metodo che sfrutta la SendMail per inviare una mail all'indirizzo di posta elettronica inserito dall'utente per verificare la sua validità.

Input:

- *email* Parametro di tipo String che indica l'indirizzo email dell'utente a cui mandare la mail.
- *URLweb* Parametro di tipo String che indica l'URL del sito web sul quale collegarsi per ricevere il servizio.
- *URLpath* Parametro di tipo String che specifica la pagina web dove avviare la conferma del servizio.
- *confirmationToken* Parametro di tipo String che indica il token randomico per poter verificare la email dello studente in maniera univoca.

Return: Il metodo non ritorna nessun parametro.

```
private void sendEmail(String to, String subject, String body)
```

Metodo che viene utilizzato per inviare effettivamente una mail all'indirizzo di posta elettronica passato in ingresso.

Input:

- *email* Parametro di tipo String che indica l'indirizzo email dell'utente a cui mandare la mail.

- *subject* Parametro di tipo String che indica il nome dell'oggetto della mail inviata dal servizio.
- *body* Parametro di tipo code String che rappresenta il body, ossia il corpo, del messaggio inviato tramite posta elettronica.

Return: Il metodo non ritorna nessun parametro.

5.9.4.3 FindInfoService

```
deleteStudentByEmail(String email);
```

Cerca nella base dati lo studente con l'email fornita in ingresso e lo cancella dalla base dati.

Input: *email*, chiave di ricerca dello studente nella base dati.

Return: Nessun valore.

```
public Student getStudentByEmail(String email)
```

Cerca nella base dati lo studente con l'email fornita in ingresso e restituisce un oggetto Student popolato dalle informazioni trovate nella base dati. Restituisce null se l'email non corrisponde a nessuno studente.

Input: *email*, chiave di ricerca dello studente nella base dati.

Return: Restituisce un oggetto Student con i campi popolati se viene trovato oppure NULL se non viene trovato.

```
public Student getStudentByConfirmationToken(String confirmationToken)
```

Cerca nella base dati lo studente con il token di conferma fornito in ingresso e restituisce un oggetto Student popolato dalle informazioni trovate nella base dati.

Input: *confirmationToken* ,che funge da chiave di ricerca dello studente nella base dati.

Return: Restituisce un oggetto Student con i campi popolati se viene trovato oppure NULL se non viene trovato.

Exception: *ConfirmationTokenNotFoundException*, restituita quando non viene trovato lo studente sulla base dati cercandolo tramite il confirmationToken

```
public Student getStudentByPasswordToken(String  
passwordToken)
```

Cerca nella base dati lo studente con il token della password fornito in ingresso e restituisce un oggetto Student popolato dalle informazioni trovate nella base dati.

Input: *passwordToken* , che funge da chiave di ricerca dello studente nella base dati.

Return: Restituisce un oggetto Student con i campi popolati se viene trovato oppure NULL se non viene trovato.

```
public Student getStudentById(String id)
```

Cerca nella base dati lo studente usando l'id fornito in ingresso e restituisce un oggetto Student popolato dalle informazioni trovate nella base dati.

Input: *id* che funge da chiave di ricerca dello studente nella base dati.

Return: Restituisce un oggetto Student con i campi popolati se viene trovato oppure NULL se non viene trovato.

```
public boolean isStudentRegistered(String email)
```

Cerca nella base dati lo studente con l'email fornita in ingresso e restituisce true se viene trovato, altrimenti false. **Input:** *email* che funge da chiave di ricerca dello studente nella base dati.

Return: *true*, se viene trovato uno studente con questa email, false se non viene trovato.

```
public boolean doesConfirmationTokenExists(String  
confirmationToken)
```

Cerca nella base dati il token di conferma dell'identità associato a un qualsiasi studente, e se viene trovato restituisce true, altrimenti false.

Input: *confirmationToken* che funge da chiave di ricerca dello studente nella base dati.

Return: *true*, se il token esiste, false altrimenti.

```
public boolean doesPasswordTokenExists(String  
passwordToken)
```

Cerca nella base dati il token della password associato a un qualsiasi studente, e se viene trovato restituisce true, altrimenti false. Il token della password associato a uno studente registrato serve a generare una pagina web univoca per poter cambiare la password dello studente.

Input: *PasswordToken* che funge da chiave di ricerca dello studente nella base dati.

Return: *true*, se il token esiste, false altrimenti.

5.9.4.4 StudentDetailsService

Questi metodi servono per creare la sessione che dovrà poi essere gestita da Spring Security.

```
public UserDetails loadUserByUsername(String email)
```

Metodo che carica lo StudentDetails a partire dalla sua email, che viene passata come parametro di ingresso.

Input: *email*, parametro di tipo String che permette di cercare sulla base dati lo studente attraverso la sua email.

Return: Ritorna l'oggetto StudentDetails, ereditata da UserDetails.

Exception: *UsernameNotFoundException*, eccezione restituita quando l'email fornita in ingresso non corrisponde a nessuno studente nella base dati.

5.9.4.5 UtilityService

Questo metodo serve per abilitare lo studente registrato dopo aver confermato l'email.

```
public void enableStudent(Student student)
```

Metodo che permette di abilitare il nuovo studente registrato. L'abilitazione viene effettuata passando lo studente da abilitare come parametro @code String. Esso verrà abilitato settando a code true il campo enabled, e dopodiché salvato sulla base dati. Uso tipico: dal controller, per ottenere lo studente si può chiamare `getStudentByConfirmationToken()` e dopodiché passarlo come parametro.

Input: *student*. parametro di tipo Student e rappresenta lo studente che deve essere abilitato. **Return:** Non ritorna valori.

5.9.5 REST API

La REST API (*Representational State Transfer*) è un'architettura software che definisce un insieme di principi per la creazione di servizi web. È un approccio comune per la progettazione di interfacce per l'integrazione tra sistemi, consentendo loro di comunicare e scambiare dati in modo efficiente. Tra le varie caratteristiche associate alle API REST troviamo:

- ***Stateless***: Le API REST sono stateless, il che significa che ogni richiesta del client deve contenere tutte le informazioni per elaborare tale richiesta.
- ***Metodi HTTP***: Le API REST utilizzano metodi HTTP per definire 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.
- ***URI (Uniform Resource Identifier)***: Ogni risorsa in un'API REST è identificata da un URI univoco.
- ***Rappresentazione dei dati***: Le API REST utilizzano formati di rappresentazione dei dati come JSON (JavaScript Object Notation), che è il formato più utilizzato per la sua semplicità e leggibilità, o XML (eXtensible Markup Language) per trasmettere i dati tra client e server.

In definitiva, le API REST sono ampiamente utilizzate nello sviluppo di applicazioni web per consentire l'*integrazione* tra diversi sistemi e piattaforme. Infatti, essi sono flessibili, scalabili e seguono principi chiari che semplificano la progettazione, lo sviluppo e la manutenzione delle interfacce di programmazione. Di seguito verranno introdotte le REST API relative al progetto per ogni caso d'uso implementato.

5.9.5.1 REST API: Registration (POST)

POST /*api/registration*

Figura 5.15: POST on Registration

Questa API [figura (5.15)] permette di inoltrare una richiesta HTTP di tipo POST che avvia il processo di registrazione dello studente.

Parametri di input Il payload di input deve essere un oggetto JSON contenente i campi mostrati in figura (5.16)

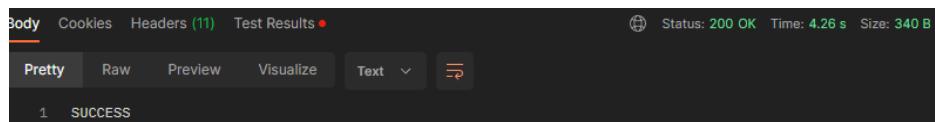
```
1  {
2    "name": "Paolo",
3    "surname": "Rossi",
4    "gender": "Male",
5    "nationality": "Italian",
6    "studyTitle": "Bachelor Degree",
7    "dateOfBirth": "20-11-1991",
8    "email": "ludotcom1996@gmail.com",
9    "password": "1234"
10 }
```

Figura 5.16: Body Registration JSON

Codici di risposta Inoltrando una richiesta di questo tipo, si possono avere diversi tipi di codici di risposta:

- **200 OK:** La registrazione dello studente è avvenuta con successo.
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno durante la registrazione dello studente.

Esempio di risposta Se la richiesta di registrazione va a buon fine sarà possibile visualizzare una risposta come mostra la figura (5.17).



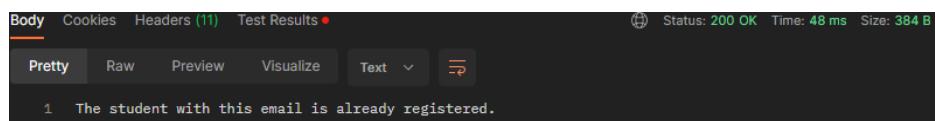
The screenshot shows a REST API response in a tool like Postman or cURL. The 'Body' tab is selected, displaying the following JSON:

```
1   SUCCESS
```

At the top right, the status is shown as "Status: 200 OK Time: 4.26 s Size: 340 B".

Figura 5.17: Success Reply Registration

Al contrario, nel caso in cui si effettui una post con una mail già presente all'interno del database otteniamo la risposta mostrata in figura (5.18)



The screenshot shows a REST API response in a tool like Postman or cURL. The 'Body' tab is selected, displaying the following JSON:

```
1   The student with this email is already registered.
```

At the top right, the status is shown as "Status: 200 OK Time: 48 ms Size: 384 B".

Figura 5.18: Reply Registration Failed

5.9.5.2 REST API: Student Info (GET)

GET /api/studentinfo

Figura 5.19: GET on StudentInfo

Questa API [figura (5.19)] permette di inoltrare una richiesta HTTP di tipo GET che a partire dall'ID ritorna le info dello studente salvato.

Parametri di input Il payload di input deve essere un oggetto JSON contenente i campi mostrati in figura (5.20)

```
1  {
2    ...
3    "id": "648aa07a4c04160246a12175"
4  }
```

Figura 5.20: Body StudentInfo JSON

Codici di risposta Inoltrando una richiesta di questo tipo, si possono avere diversi tipi di codici di risposta:

- **200 OK:** La ricerca dello studente è avvenuta con successo.
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno durante la registrazione dello studente.

Esempio di risposta Se la richiesta di informazioni dello studente a partire dall'ID va a buon fine si osserva la risposta mostrata in figura (5.21).

The screenshot shows a REST API response in a browser-like interface. At the top, it says "Status: 200 OK Time: 19 ms Size: 727 B". Below that, there are tabs for "Pretty", "Raw", "Preview", "Visualize", and "JSON". The "Pretty" tab is selected, displaying the JSON response in a hierarchical, readable format. The JSON object represents a student with various fields like id, name, surname, gender, nationality, studyTitle, dateOfBirth, email, password, and roles.

```

1  {
2      "id": "648aa07a4c04160246a12175",
3      "confirmationToken": null,
4      "passwordToken": null,
5      "enabled": true,
6      "name": "Mario",
7      "surname": "Rossi",
8      "gender": "Male",
9      "nationality": "Italian",
10     "studyTitle": "Bachelor Degree",
11     "dateOfBirth": "20-11-1991",
12     "email": "ludotcom1996@gmail.com",
13     "password": "$2a$10$IUg0gTKzVRa98YhhPqHadu3usEXFkaqVhj/GISbxdM1tR3mG1ey2u",
14     "roles": [
15         {
16             "id": "648a9ed84c04160246a1216c",
17             "name": "STUDENT"
18         }
19     ]

```

Figura 5.21: Success Reply StudentInfo

Al contrario, nel caso in cui viene passato un ID non corretto viene restituito un oggetto vuoto.

5.9.5.3 REST API: ChangePassword (POST)

POST /api/changepassword

Figura 5.22: POST on ChangePassword

Questa API [figura (5.22)] permette di inoltrare una richiesta HTTP di tipo POST che avvia il processo di cambio password. al fine di inviare una mail all'indirizzo di posta passato come input.

Parametri di input Il payload di input deve essere un oggetto JSON contenente i campi mostrati in figura (5.23).

```

1  {
2      "email": "ludotcom1996@gmail.com"
3  }
4

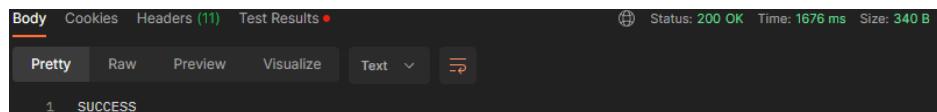
```

Figura 5.23: Body ChangePassword JSON

Codici di risposta Inoltrando una richiesta di questo tipo, si possono avere diversi tipi di codici di risposta:

- **200 OK:** L'invio della mail è avvenuta con successo.
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno durante la registrazione dello studente.

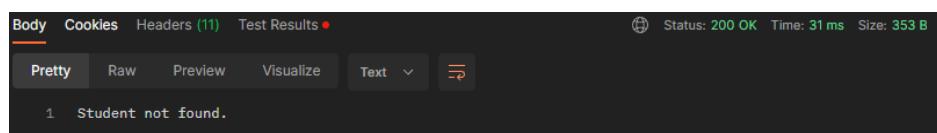
Esempio di risposta Se la richiesta di cambio password va a buon fine sarà possibile visualizzare la risposta mostrata in figura (5.24).



The screenshot shows a REST API response in a browser-like interface. The top bar includes tabs for 'Body', 'Cookies', 'Headers (11)', and 'Test Results'. On the right, it displays 'Status: 200 OK', 'Time: 1676 ms', and 'Size: 340 B'. Below the tabs, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', 'Text', and a copy icon. The main content area shows the number '1' followed by the word 'SUCCESS'.

Figura 5.24: Success Reply ChangePassword

Al contrario, nel caso in cui si effettui una post con una mail che non è registrata o non abilitata all'interno del sistema si ottiene la risposta mostrata in figura (5.25).



The screenshot shows a REST API response in a browser-like interface. The top bar includes tabs for 'Body', 'Cookies', 'Headers (11)', and 'Test Results'. On the right, it displays 'Status: 200 OK', 'Time: 31 ms', and 'Size: 353 B'. Below the tabs, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', 'Text', and a copy icon. The main content area shows the number '1' followed by the message 'Student not found.'

Figura 5.25: Reply ChangePassword Failed

5.9.5.4 REST API: Student ID (GET)

GET /*api/studentid*

Figura 5.26: GET on StudentID

Questa API [figura (5.26)] permette di inoltrare una richiesta HTTP di tipo GET che a partire dalla mail dello studente ritorna il suo ID.

Parametri di input Il payload di input deve essere un oggetto JSON contenente i campi mostrati in figura (5.27).

```
1  {
2    ...
3    "email": "ludotcom1996@gmail.com"
4 }
```

Figura 5.27: Body StudentID JSON

Codici di risposta Inoltrando una richiesta di questo tipo, si possono avere diversi tipi di codici di risposta:

- **200 OK:** La ricerca dell'ID è avvenuta con successo.
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno durante la registrazione dello studente.

Esempio di risposta Se la richiesta di informazioni dello studente a partire dall'ID va a buon fine si osserva la risposta mostrata in figura (5.28)

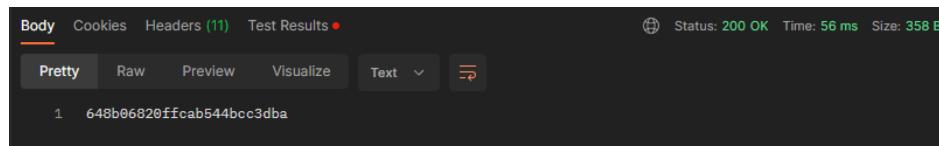


Figura 5.28: Success Reply ConfirmToken

Al contrario, nel caso in cui viene passato un ID non corretto viene restituito un messaggio di errore.

5.9.5.5 REST API: ConfirmToken (POST)

POST /api/confirm/token

Figura 5.29: POST on ConfirmToken

Questa API [figura (5.29)] permette di inoltrare una richiesta HTTP di tipo POST che avvia il processo di validazione della mail attraverso il link ricevuto tramite mail.

Parametri di input Il payload di input deve essere un oggetto JSON contenente i campi mostrati in figura(5.30).

```
1  {
2    | ... "confirmationToken": "668442c4-1fd5-4ef4-bcce-454d0c3f7dd7"
3  }
```

Figura 5.30: Body ConfirmToken JSON

Esempio di risposta Se la richiesta di cambio password va a buon fine sarà possibile visualizzare la risposta mostrata in figura (5.31)

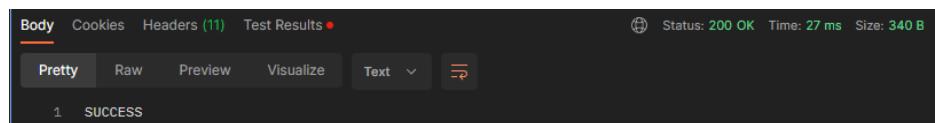


Figura 5.31: Success Reply ConfirmToken

Al contrario, nel caso in cui si effettui una post con un *confirmationToken* non valido si ottiene la risposta mostrata in figura (5.32).

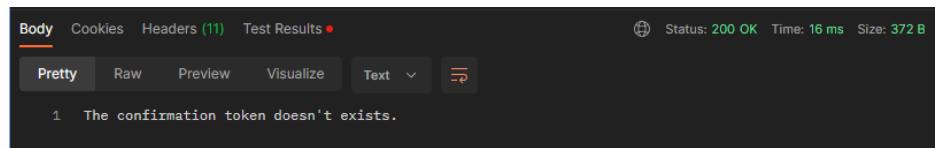


Figura 5.32: Reply ConfirmToken Failed

5.9.5.6 REST API: ChangePasswordProcess (POST)

POST /api/changepassword/token

Figura 5.33: POST on ChangePasswordToken

Questa API [figura (5.33)] permette di inoltrare una richiesta HTTP di tipo POST che avvia il processo di reset della password.

Parametri di input Il payload di input deve essere un oggetto JSON contenente i campi mostrati in figura(5.34).

```

1  {
2    ...
3    "passwordToken": "e2c5ef08-e660-4925-936f-14a2b2991bfb",
4    "newPassword": "Prova"
}

```

Figura 5.34: Body ChangePasswordProcess JSON

Esempio di risposta Se la richiesta di cambio password va a buon fine sarà possibile visualizzare la risposta mostrata in figura (5.35)

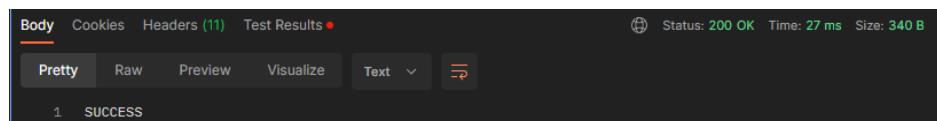


Figura 5.35: Success Reply ChangePasswordProcess

Al contrario, nel caso in cui si effettui una post con un *passwordToken* non valido si ottiene la risposta mostrata in figura (5.36).

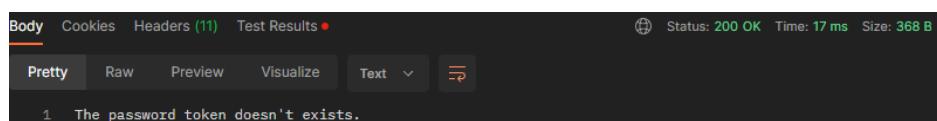


Figura 5.36: Reply ChangePasswordProcess Failed

5.9.5.7 REST API: Login (POST)

POST */api/login*

Figura 5.37: POST on Login

Questa API [figura (5.37)] permette di inoltrare una richiesta HTTP di tipo POST che avvia il processo di autenticazione dello studente.

Parametri di input Il payload di input deve essere un oggetto JSON contenente i campi mostrati in figura(5.38).

```
1  {
2    "email": "ludotcom1996@gmail.com",
3    "password": "123"
4 }
```

Figura 5.38: Body Login JSON

Esempio di risposta Se la richiesta di login va a buon fine, ossia le credenziali sono corrette, allora sarà possibile visualizzare la risposta mostrata in figura (5.39)

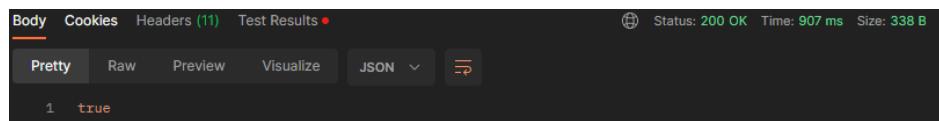


Figura 5.39: Success Reply Login

Al contrario, nel caso in cui si effettui una post con una mail/password sbagliata o con una mail ancora non verificata si ottiene la risposta mostrata in figura(5.40).

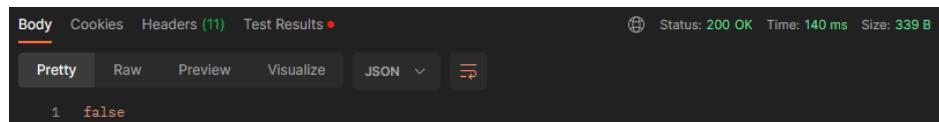


Figura 5.40: Reply Login Failed

Capitolo 6

Implementazione

6.1 Introduzione

In questo capitolo, viene presentata l'implementazione del nostro progetto introducendo i relativi diagrammi di sequenza di dettaglio. Tuttavia, prima di vedere nello specifico i diagrammi di sequenza è importante fornire un contesto adeguato. A tale scopo, è doveroso introdurre il framework principale per gestire la sicurezza all'interno dell'applicazione, ossia **Spring Security**.

6.2 Spring Security

Spring Security è un framework di sicurezza potente e altamente personalizzabile che fornisce funzionalità di autenticazione, autorizzazione e protezione delle risorse per le applicazioni Spring. In particolare, Spring Security fornisce i seguenti servizi:

Autenticazione : Spring Security supporta diversi meccanismi di autenticazione tra cui autenticazione *basata su form*, *su token* e altri tipi di tecniche.

Autorizzazione : Spring Security consente di definire regole di autorizzazione per proteggere le risorse dell'applicazione. Tali regole possono essere definite utilizzando espressioni basate su ruoli, basate su ACL (Access Control List) o personalizzate. Questo permette di limitare l'accesso alle funzionalità dell'applicazione solo agli utenti autorizzati.

Protezione da attacchi : Spring Security permette di proteggere l'applicazione da attacchi di cross-site scripting (XSS), di injection SQL e attacchi di falsificazione di richieste tra siti (CSRF).

Infine, è importante sottolineare la **gestione delle sessioni** in quanto Spring Security offre funzionalità per gestire le sessioni degli utenti attraverso cookies e timeout.

6.3 Sequence Diagram di Dettaglio

I Sequence Diagram di dettaglio sono un tipo di diagramma dinamico che forniscono una visione dettagliata dell'interazione tra gli oggetti all'interno di un sistema software. Questi diagrammi mostrano le chiamate di metodo tra gli oggetti nel corso del tempo e illustrano l'ordine e il flusso delle operazioni. Nei prossimi paragrafi verranno presentati i Sequence Diagram di dettaglio per illustrare l'interazione tra gli oggetti nel nostro sistema.

6.3.1 Sequence Diagram: Registration

In figura (6.1) viene riportato il primo Sequence Diagram di dettaglio relativo al caso d'uso “Registration”. Uno studente non registrato inserisce le proprie informazioni personali (nome, cognome, data di nascita, sesso ecc.) all'interno di un apposito *form di registrazione* presentato attraverso la relativa **View** "registration". A questo punto, quando lo studente clicca sul tasto **“Register”** viene inviata una richiesta HTTP di tipo POST contenente le informazioni precedentemente inserite nel Form, la quale, viene gestita inizialmente da un **FrontController**, in accordo con l'architettura SpringMVC, e successivamente da uno specifico controller, chiamato **RegistrationController** mappato tramite una *PostMapping* sull'URL “/registration”. Tale controller, invocando il metodo **registerStudent** del servizio **RegistrationService**, avvia la procedura per l'effettiva registrazione dello studente sulla base dati. In particolare, attraverso la funzione **isStudentRegistered(email)**, che prende in input la mail inserita dallo studente nel form di registrazione, il sistema controlla se lo studente che sta effettuando la registrazione è già registrato tramite il metodo **existsByEmail(email)** dello **StudentDAO**:

- **Scenario di successo (student==null):** Nel caso in cui lo studente stia effettuando la registrazione per la prima volta, la procedura restituisce un valore *null* e il **RegistrationService** provvede a creare un nuovo studente i cui campi vengono settati attraverso i valori provenienti dallo *StudentForm*. Successivamente, **RegistrationService** genera

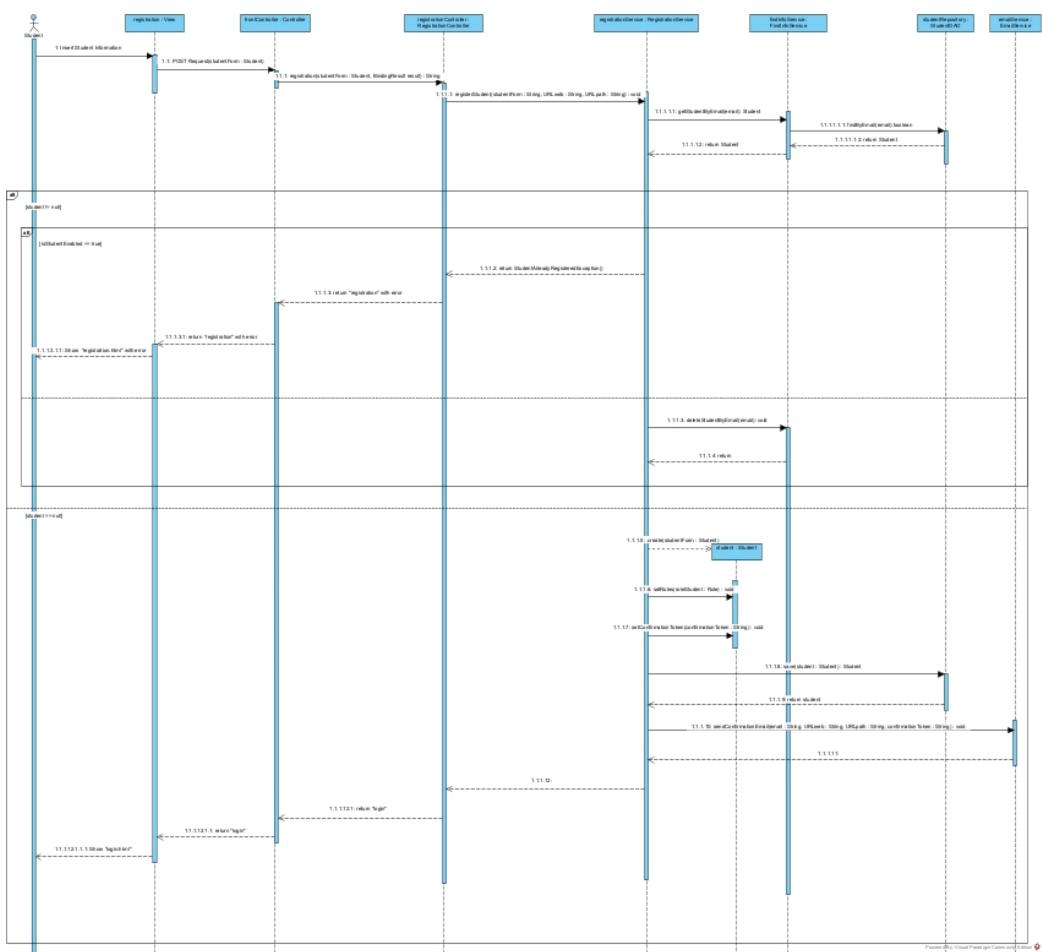


Figura 6.1: I° Sequence Diagram Dettaglio Registrazione

un nuovo token di conferma (*new Confirmation Token*) e, in più, essendo lo studente uno “*user*” nel sistema, gli verrà associato anche un *ruolo* che sarà fondamentale per l’autorizzazione nel sistema. A questo punto, dopo aver creato lo studente, il ***RegistrationService*** invoca il metodo **save(student)** dello ***StudentDAO***. Questo metodo provvede a salvare temporaneamente lo studente sulla base dati impostando a **false** una variabile chiamata **enable** che indica la validità o meno dell’account. Dopo aver effettuato questo salvataggio temporaneo, il ***RegistrationService***, invoca il metodo **sendConfirmationEmail**, del servizio ***EmailService*** con lo scopo di inviare una mail contenente un link di validazione alla casella postale dello studente. Una volta inviata la mail alla casella postale dello studente, si ritorna alla schermata di Login.

- **Scenario Alternativo (student != null):** Nel caso in cui lo studente sia già registrato, ovvero la mail sia già presente all’interno del database e nel caso in cui **isStudentEnable() == true**, viene lanciata una eccezione **StudentAlreadyRegisteredException()** che viene catturata dal ***RegistrationController***, il quale fa sì che venga mostrata una ***View*** con il relativo messaggio di errore; al contrario, nel caso **student != null** ma **isStudentEnable() == false** vuol dire che lo studente è salvato sulla base dati ma non ha confermato la mail. Pertanto, nel caso in cui lo stesso studente provasse a registrarsi nuovamente con la stessa mail esso viene prima cancellato dal database e poi nuovamente salvato con i nuovi parametri di registrazione. In definitiva, solamente in questo ultim caso si esce da tale condizione e si ricade nel caso di successo

6.3.2 Sequence Diagram: Confirm Registration

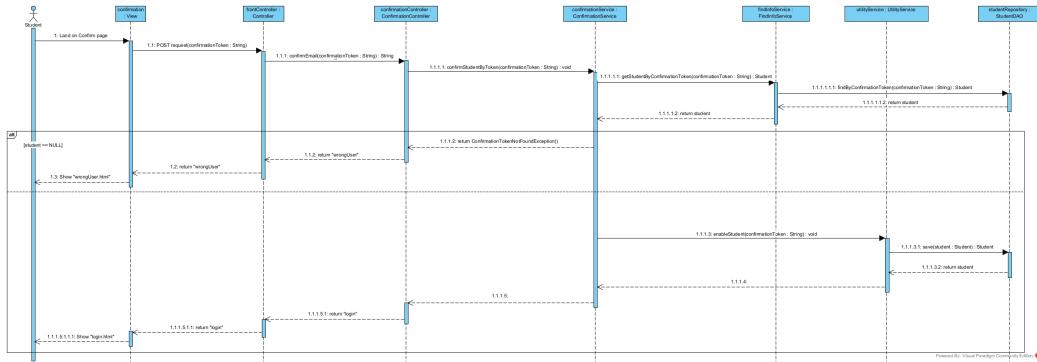


Figura 6.2: II° Sequence Diagram Dettaglio Registrazione

In figura (6.2) viene riportato il secondo Sequence Diagram di dettaglio relativo al caso d'uso “Registration”. In particolare, in questo scenario, dopo che lo studente clicca sul link di conferma ricevuto tramite email, viene inviata una richiesta HTTP di tipo POST che viene intercettata inizialmente dal **FrontController** e successivamente dal controller **ConfirmationController** attraverso una *PostMapping* sull'URL "/confirm/token=..." il quale invoca il metodo **confirmStudentByToken(confirmationToken)** del servizio **ConfirmationService**. A questo punto, tale metodo invoca **getStudentByConfirmationToken(confirmationToken)** del servizio **FindInfoService**. Successivamente, attraverso la funzione **findByConfirmationToken(confirmationToken)** dello **StudentDAO** si effettua una ricerca sul database dello studente che possiede quel *confirmationToken* specifico. Allora:

- **Scenario di Successo (student != NULL):** In caso di successo tale metodo restituisce lo studente avente quel determinato token, e, tramite la funzione **enableStudent(confirmationToken)** dell'**UtilityService**, viene messa a *true* la variabile *enable* per indicare l'avvenuta convalescenza. Dopo questa operazione, lo studente viene salvato sul database attraverso il metodo **save(student)** dello **StudentDAO** e viene restituita la View con la schermata di login.
- **Scenario Alternativo (student==NULL):** Nel caso in cui lo studente non dovesse essere presente sul database, viene restituito un valore *NULL* e il **ConfirmationController** aggiorna la View con una pagina di errore, nello specifico "wrongUser.html".

6.3.3 Sequence Diagram: Login

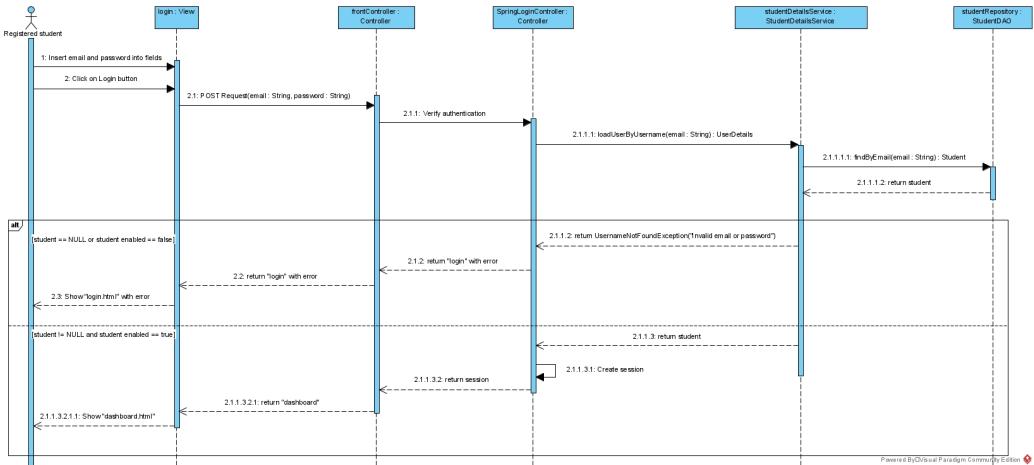


Figura 6.3: Sequence Diagram Dettaglio Login

In figura (6.3) viene riportato il Sequence Diagram di dettaglio relativo al caso d'uso “Login”. In questo scenario, uno studente registrato interagisce con una *View* la quale mostra un Form di login dove inserire email e password. Una volta che lo studente registrato clicca sul tasto “*Login*” viene inviata una richiesta HTTP di POST che viene gestita inizialmente da un *Front-Controller* e successivamente da uno specifico controller chiamato *SpringLoginController* tramite una *PostMapping* sull’URL “/login”. A questo punto, il controller *SpringLoginController*, fornito dal framework *Spring-Security*, effettua un’operazione di autenticazione verificando l’autenticità della mail, della password e del **ruolo** dello studente. Infatti, invocando il metodo **loadUserByUsername(email)** del servizio *StudentDetailsService* e il metodo **findByEmail(email)** dello *StudentDAO* si effettua una ricerca sulla base dati dello studente che sta effettuando il Login con quella specifica “email”.

Allora:

- **Scenario di Successo (student!=NULL):** Nel caso in cui lo studente viene trovato all’interno della base dati, il metodo restituisce un oggetto di tipo *Studente*. A questo punto, lo *SpringLoginController* provvede a creare una sessione che sarà mantenuta fin quando lo studente non effettuerà il logout o finché non verrà chiuso il browser. Una volta creata la sessione, lo studente verrà reindirizzato alla dashboard, dove potrà accedere allo storico delle partite o giocarne delle nuove attraverso appositi pulsanti.

- **Scenario di Successo (student==NULL):** Nel caso in cui lo studente con quella determinata email non viene trovato sulla base dati, il metodo restituisce un valore NULL e lo *StudentDetailService* lancia un' eccezione **UsernameNotFoundException** che viene "catturata" dal controller ,il quale aggiorna la *View* di login con un apposito messaggio di errore: "Invalid Username or Password".

6.3.4 Sequence Diagram: Change Password

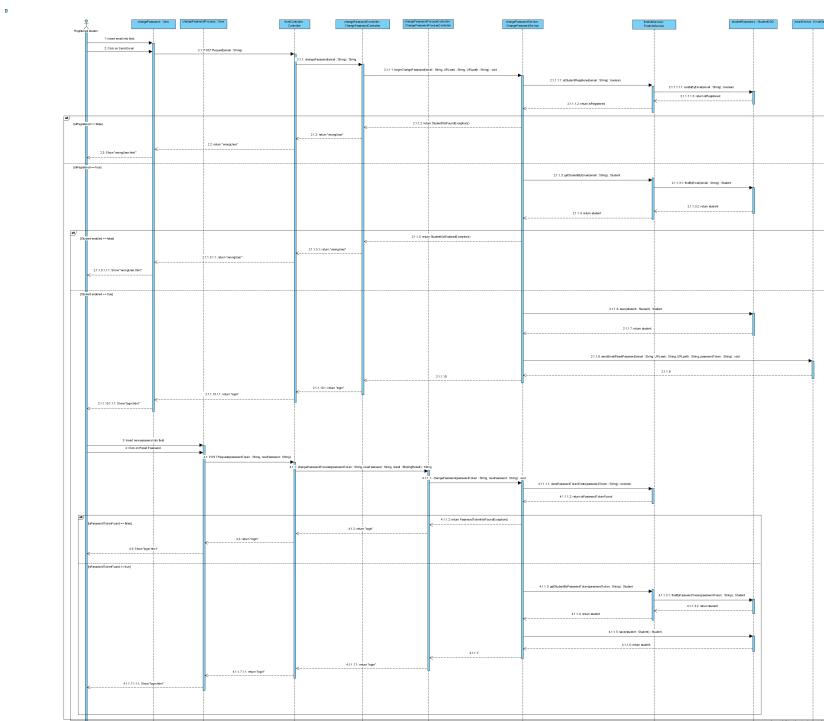


Figura 6.4: Sequence Diagram Dettaglio Cambio Password

In figura (6.4) viene riportato il Sequence Diagram di dettaglio relativo al caso d'uso “Change Password”. In questo scenario, lo studente registrato, cliccando sul link "*Forgot Password?*" presente sulla schermata di Login, viene reindirizzato su una nuova View "*changePassword*". In particolare, dopo aver inserito nell'apposito campo la propria mail per il cambio password ed aver cliccato sul tasto "*Send Mail*" viene inviata una richiesta HTTP di tipo POST che viene gestita inizialmente dal ***FrontController*** e successivamente dal ***ChangePasswordController*** attraverso il metodo ***changePassword(email)*** con una *PostMapping* sull'URL "/*changePassword*". A questo punto, il ***ChangePasswordController*** invoca il metodo ***beginChangePassword(email, URLweb, URLpath)*** del servizio ***ChangePasswordService*** il quale richiama il metodo ***isStudentRegistered(email)*** del servizio ***FindInfoService*** in modo tale che quest ultimo, attraverso la funzione ***existsByEmail(email)*** dello ***StudentDAO*** effettua una ricerca della mail passata come input all'interno del database. Allora:

- **Scenario Alternativo (*isRegistered==false*):** Se la mail non viene trovata all'interno del database, il metodo restituisce un valore tale per cui il ***ChangePasswordService*** lancia un' eccezione ***StudentNotFoundExceptio*** che viene gestita dal ***ChangePasswordController***, il quale ritorna una pagina di errore ***wrongUser.html***
- **Scenario di successo(*isRegistered==true*):** Se la mail viene trovata all'interno del database vuol dire che lo studente è registrato all'interno del sistema. Pertanto, il ***ChangePasswordService*** invoca il metodo ***getStudentByEmail(email)*** del ***FindInfoService*** che, attraverso la ***findByEmail(email)*** della classe ***StudentDAO*** effettua la ricerca dello studente sul database avente quella mail. Allora:
 - **Scenario alternativo(*Student Enabled==false*):** Se lo studente è registrato nel sistema ma non ha ancora confermato la propria mail, il ***ChangePasswordService*** lancia una eccezione ***StudentNotEnabledException*** che viene gestita dal ***ChangePasswordController*** lanciando una schermata di errore "***wrongUser.html***".
 - **Scenario di successo (*Student Enabled==true*):** Se lo studente è registrato e abilitato, viene *salvato* sulla base dati con un nuovo campo ***passwordToken***, che servirà per l'invio successivo della mail. Successivamente, attraverso il metodo ***sendEmailResetPassword*** del servizio ***EmailService***, viene inviata una mail all'indirizzo specificato contenente il link per il cambio pas-

sword. Una volta effettuato l'invio della mail e avvisato l'utente attraverso un popup, si ritorna alla schermata di login.

ProceduraCambioPassword Per una questione di leggibilità descriviamo la seconda parte del sequence diagram in questo nuovo paragrafo. In questo nuovo scenario, lo studente registrato, dopo aver cliccato sul link di "cambio password" ricevuto tramite email interagisce con una nuova view, "*ChangePasswordProcess*", nella quale può inserire e confermare la nuova password scelta. Dopo aver cliccato sul tasto "*Reset Password*" viene inviata una richiesta HTTP di tipo POST che viene gestita prima dal **FrontController** e successivamente dal **ChangePasswordProcessController** attraverso una *PostMapping* sull'URL "/passwordToken". A questo punto, viene invocato il metodo **changePassword** del servizio **ChangePasswordService**, il quale, attraverso il metodo **doesPasswordTokenExists(passwordToken)** controlla se il token passato tramite l'URL è quello associato allo studente. Allora:

- **Scenario di successo (isPasswordTokenFound==true):** In caso di successo il metodo restituisce un valore *true* e viene invocata la funzione **getStudentByPasswordToken(passwordToken)** del servizio **FindInfoService** che, attraverso la **findByPassworkToken** dello **StudentDAO** cerca sulla base dati lo studente avente quello specifico token passato come input. Dopodichè viene aggiornato il campo password e lo studente viene salvato sulla base dati attraverso il metodo **save(student)** dello **StudentDAO**. Una volta finito il salvataggio, si ritorna alla schermata principale di Login.
- **Scenario Alternativo (isPasswordTokenFound==false):** Nel caso in cui il token non viene trovato all'interno della base dati il metodo restituisce un valore *false* per cui il **ChangePasswordService** lancia un'eccezione **PasswordTokenNotFoundException** la quale viene gestita dal **ChangePasswordProcessController** reindirizzando lo studente sulla schermata di Login.

6.3.5 Sequence di Dettaglio: Note nel caso di REST API

Nel caso delle REST API, dal momento che i servizi di backend restano essenzialmente gli stessi, le uniche differenze con i casi presentati fino ad ora, sono sostanzialmente due:

1. Nel caso delle REST API, le richieste vengono gestite da un ***RestController*** e non più da un controller per ogni servizio locale.
2. Nel caso delle REST API il payload non viene rappresentato più dal Form HTML ma da un oggetto JSON che viene passato al momento delle richieste di tipo POST e GET.

Capitolo 7

Testing

Il Testing ha l'obiettivo di trovare difetti e dimostrare che un sistema soddisfa i suoi requisiti. Questa fase, è stata eseguita con continuità durante il ciclo di sviluppo. Nel seguente capitolo si è propongono due tipologie di test effettuati:

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

7.1 Testing basato sulle GUI dell'applicazione

Per effettuare i test basati sulle interfacce utente, è stato utilizzato un approccio basato sul criterio di copertura dei cammini indipendenti. In particolar modo, sono stati definiti i cinque CFG (Control Flow Graph) che definiscono tutti gli scenari implementati.

7.1.1 Testing GUI: registration

Il CFG riportato in figura (7.1), mostra il flusso di controllo che è possibile ottenere a partire dalla pagina "login.html" se viene cliccato il bottone "Create New Account". Il numero di cammini linearmente indipendenti, ossia il numero ciclomatico di McCabe è pari a due; da tale risultato, si derivano due casi di test.

Precondizione TC1: L'email inserita per la registrazione non è associata a nessuno studente registrato.

Precondizione TC2: L'email inserita per la registrazione è associata ad uno studente già registrato.

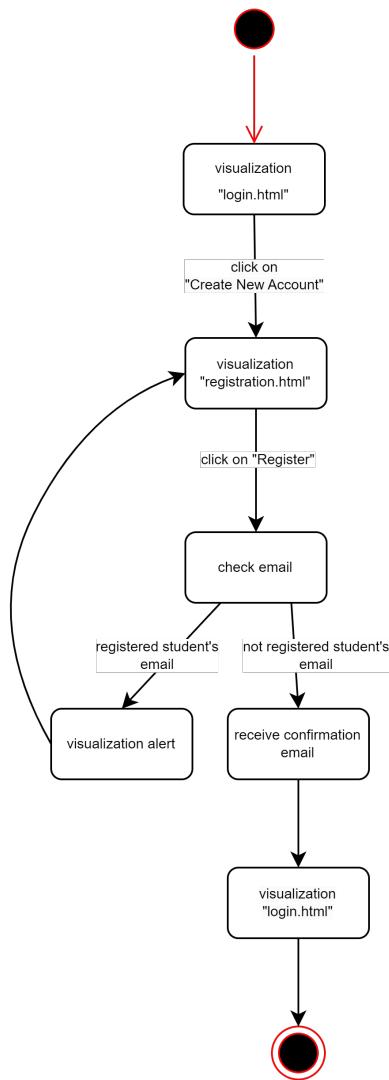


Figura 7.1: Registration CFG

ID	Input	Expected Output	RESULT
TC1	click on "Create New Account", click on "Register"	"registration.html", "login.html"	SUCCESS
TC2	click on "Create New Account", click on "Register"	"registration.html", "registration.html" with alert	SUCCESS

7.1.2 Testing GUI: login

Il CFG riportato in figura (7.2), mostra il flusso di controllo che è possibile ottenere a partire dalla pagina "login.html" se viene cliccato il bottone "LOGIN". Il numero di cammini linearmente indipendenti, ossia il numero ciclomatico di McCabe è pari a due; da tale risultato, si derivano due casi di test.

Precondizione TC1: Le credenziali inserite sono valide.

Precondizione TC2: Le credenziali inserite non sono valide.

ID	Input	Expected Output	RESULT
TC1	click on "LOGIN", click on "Logout"	"dashboard.html", "login.html"	SUCCESS
TC2	click on "LOGIN"	"login.html" with alert	SUCCESS

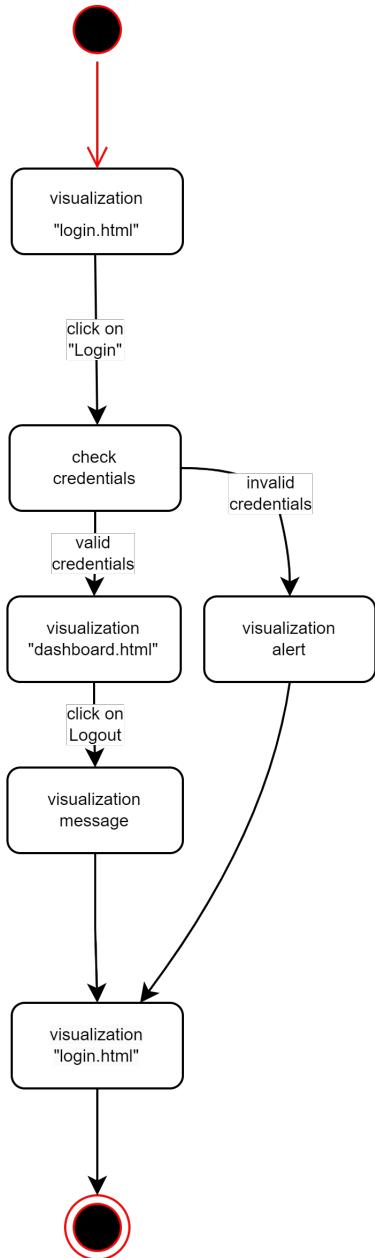


Figura 7.2: Login and Log Out CFG

7.1.3 Testing GUI: changePassword

Il CFG riportato in figura (7.3), mostra il flusso di controllo che è possibile ottenere a partire dalla pagina "login.html" se viene cliccato il bottone "Forgot Password". Il numero di cammini linearmente indipendenti, ossia il numero ciclomatico di McCabe è pari a due; da tale risultato, si derivano due casi di test.

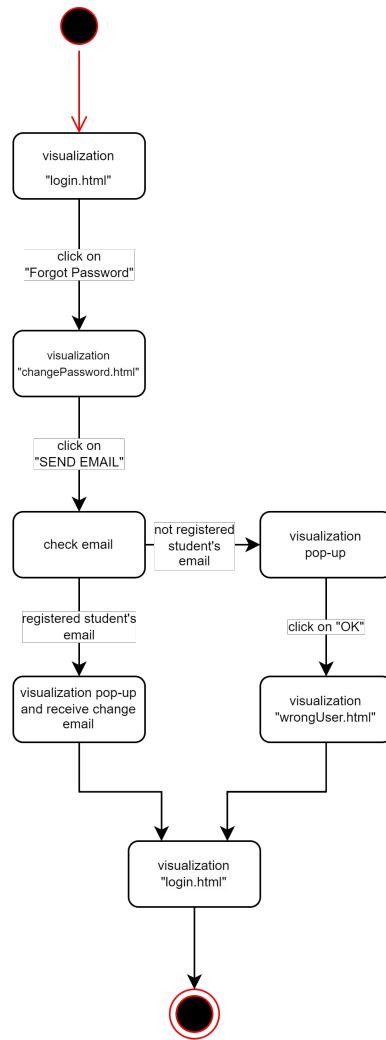


Figura 7.3: Change Password CFG

Precondizione TC1: L'email inserita per cambiare la password è associata ad uno studente registrato.

Precondizione TC2: L'email inserita per cambiare la password non è associata ad uno studente registrato.

ID	Input	Expected Output	RESULT
TC1	click on "Forgot Password", click on "SEND EMAIL", click on "OK"	"changePassword.html", "changePassword.html" with pop-up, "login.html"	SUCCESS
TC2	click on "Forgot Password", click on "SEND EMAIL", click on "OK"	"changePassword.html", "changePassword.html" with pop-up, "wrongUSER.html", "login.html"	SUCCESS

7.1.4 Testing GUI: confirmation email

Il CFG riportato in figura (7.4), mostra il flusso di controllo che è possibile ottenere a partire dalla pagina inviata per email per confermare la password. Il numero di cammini linearmente indipendenti, ossia il numero ciclomatico di McCabe è pari a due; da tale risultato, si derivano due casi di test.

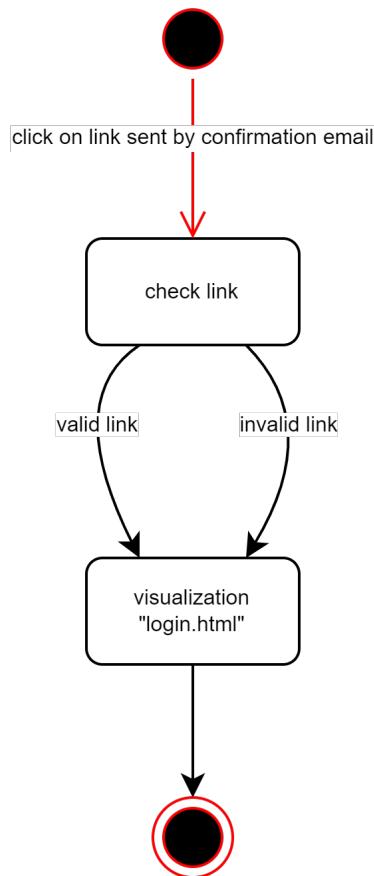


Figura 7.4: Link Confirmation CFG

Precondizione TC1: Il link a cui si cerca di accedere è valido.

Precondizione TC2: Il link a cui si cerca di accedere è valido.

ID	Input	Expected Output	RESULT
TC1	click on link sent by confirmation email	"login.html"	SUCCESS
TC2	click on link sent by confirmation email	"login.html"	SUCCESS

7.1.5 Testing GUI: changePasswordProcess

Il CFG riportato in figura (7.5), mostra il flusso di controllo che è possibile ottenere a partire dalla pagina inviata per email per confermare la password. Il numero di cammini linearmente indipendenti, ossia il numero ciclomatico di McCabe è pari a due; da tale risultato, si derivano due casi di test.

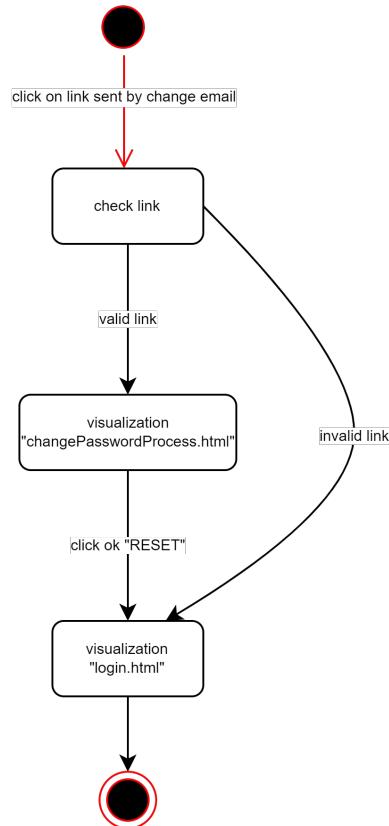


Figura 7.5: Link for Password changing CFG

Precondizione TC1: Il link a cui si cerca di accedere è valido.

Precondizione TC2: Il link a cui si cerca di accedere è valido.

ID	Input	Expected Output	RESULT
TC1	click on link sent by change email, click on "RESET"	"changePasswordProcess.html", "login.html"	SUCCESS
TC2	click on link sent by change email	"changePasswordProcess.html", "login.html"	SUCCESS

7.2 Testing delle API REST con Postman

I test dell'API sono stati resi automatizzati mediante l'utilizzo di **Postman**, ossia un'applicazione che consente di inviare richieste HTTP e visualizzare le risposte in modo interattivo. In particolare, consente di verificare se le risposte sono conformi alle aspettative, semplificando il processo di controllo delle qualità dell'API.

7.3 Testing Registrazione

7.3.1 Test Case 1

Descrizione: Lo studente non registrato si registra sull'applicazione.

Condizioni di validità dell'input:

1. L'email dello studente deve essere valida.

Precondizione:

1. Lo studente non deve essere presente nella base dati.

Input:

```

"name": "Mario",
"surname": "Rossi",
"dateOfBirth": "16-07-1999",
"gender": "M",
"nationality": "Italian",
"studyTitle": "Diploma",

```

```
"email": "grupposette7@gmail.com",
```

```
"password": "Gruppo7sette"
```

ID	Expected Output	Output	Result
TC1	Student registered	Student registered	SUCCESS

7.3.2 Test Case 2

Descrizione: Lo studente che effettua la registrazione ha già effettuato la registrazione in precedenza ma non ha confermato la propria email, quindi è uno studente salvato sulla base dati ma non abilitato.

Precondizione:

1. Lo studente presente sulla base dati con profilo non abilitato.

Input:

```
"name": "Mario",
```

```
"surname": "Rossi",
```

```
"dateOfBirth": "16-07-1999",
```

```
"gender": "M",
```

```
"nationality": "Italian",
```

```
"studyTitle": "Diploma",
```

```
"email": "grupposette7@gmail.com",
```

```
"password": "Gruppo7sette"
```

ID	Expected Output	Output	Result
TC2	Student registered	Student registered	SUCCESS

7.3.3 Test Case 3

Descrizione: Lo studente già registrato effettua una registrazione con le stesse credenziali.

Precondizione:

1. I dati dello studente sono salvati e abilitati sulla base dati.

Input:

```
"name": "Mario",  
"surname": "Rossi",  
"dateOfBirth": "16-07-1999",  
"gender": "M",  
"nationality": "Italian",  
"studyTitle": "Diploma",  
"email": "grupppo7sette@gmail.com",  
"password": "Gruppo7sette"
```

ID	Expected Output	Output	Result
TC3	Student already registered	Student already registered	SUCCESS

7.4 Testing sui dati dello studente

7.4.1 Test Case 4

Descrizione: Otttenere ID di uno studente non registrato sulla base dati.

Precondizione:

1. Lo studente non è registrato sulla base dati.

Input:

"email" : "marioverdiaaabb@gmail.com"

ID	Expected Output	Output	Result
TC4	NULL	NULL	SUCCESS

7.4.2 Test case 5

Descrizione: Otttenere ID di uno studente registrato sulla base dati tramite la sua email.

Precondizione:

1. Lo studente è registrato sulla base dati.

Input:

"email" : "grupposette7@gmail.com"

ID	Expected Output	Output	Result
TC5	Retreive student ID	Retreive student ID	SUCCESS

7.4.3 Test Case 6

Descrizione: Otttenere i dati dello studente con un ID che non è presente nella base dati.

Precondizione:

1. L'ID non è presente sulla base dati.

Input:

"id ":"u8d4jf49jf9j0jd"

ID	Expected Output	Output	Result
TC6	Student non found	Student not found	SUCCESS

7.4.4 Test Case 7

Descrizione: Otttenere i dati dello studente a partire da un ID che è presente sulla base dati.

Precondizione:

1. L' ID è presente sulla base dati.

Input:

"id ":"648b6d9cd9fb1b164af08def"

ID	Expected Output	Output	Result
TC7	Retreive student data	Retreive student data	SUCCESS

7.5 Testing Change Password

7.5.1 Test Case 8

Descrizione: Lo studente non registrato è intento a cambiare la propria password.

Precondizione:

1. Lo studente non è registrato sulla base dati.

Input:

"email": "marioverdi@gmail.com"

ID	Expected Output	Output	Result
TC8	Student not found	Student not found	SUCCESS

7.5.2 Test Case 9

Descrizione: Lo studente che è salvato sulla base dati ma non ha confermato la propria email, intende cambiare la propria password.

Precondizione:

1. Lo studente è salvato sulla base dati ma non è abilitato.

Input:

"email": "grupposette7@gmail.com"

ID	Expected Output	Output	Result
TC9	Student not enable	Student not enable	SUCCESS

7.5.3 Test Case 10

Descrizione: Lo studente registrato sulla base dati intende cambiare la propria password.

Precondizione:

1. Lo studente è registrato sulla base dati.

Input:

"email": "grupposette7@gmail.com"

ID	Expected Output	Output	Result
TC10	Started change password	Started change password	SUCCESS

7.5.4 Test Case 11

Descrizione: Lo studente è intento a cambiare la propria password con un token inesistente.

Precondizione:

1. Token non presente sulla base dati.

Input:

```
"passwordToken": "dofej0c3dfsfsdf",  
"newPassword": "123"
```

ID	Expected Output	Output	Result
TC11	Password token not found	Password token not found	SUCCESS

7.5.5 Test Case 12

Descrizione: Lo studente è intento a cambiare la propria password con un token vuoto.

Precondizione:

1. IL token sulla base dati è vuoto.

Input:

```
"passwordToken": "",  
"newPassword": "123"
```

ID	Expected Output	Output	Result
TC12	Password token empty	Password token empty	SUCCESS

7.6 Testing Conferma Email

7.6.1 Test Case 13

Descrizione: Lo studente che intende confermare la propria email per terminare la registrazione, utilizza un token non valido.

Precondizione:

1. IL token inserito non è valido.

Input:

```
"confirmationToken": "adfadjovfevj30vm5"
```

ID	Expected Output	Output	Result
TC13	Confirmation token not found	Confirmation token not found	SUCCESS

7.6.2 Test Case 14

Descrizione: Lo studente che intende confermare la propria email per terminare la registrazione.

Precondizione:

1. IL token inserito è già stato utilizzato.

Input:

"confirmationToken" : "0c8db4a7-8ac9-4673-a41f-c6c74f046dba"

ID	Expected Output	Output	Result
TC14	Confirmation token not found	Confirmation token not found	SUCCESS

7.6.3 Test Case 15

Descrizione: Lo studente che intende confermare la propria email per terminare la registrazione, inserisce un token di conferma valido.

Precondizione:

1. IL token inserito è presente sulla base dati.

Input:

"confirmationToken" : "0c8db4a7-8ac9-4673-a41f-c6c74f046dba"

ID	Expected Output	Output	Result
TC15	Student confermed	Student confermed	SUCCESS

7.7 Testing Autenticazione

7.7.1 Test Case 16

Descrizione: Lo studente che intende autenticarsi inserisce le credenziali correttamente.

Precondizione:

1. Lo studente è registrato sulla base dati.

Input:

"email" : "grupposette7@gmail.com" ,
"password" : "123"

ID	Expected Output	Output	Result
TC16	Login successful	Login successful	SUCCESS

7.7.2 Test Case 17

Descrizione: Lo studente che intende autenticarsi inserisce la password errata.

Precondizione:

1. Lo studente è registrato sulla base dati.

Input:

"email": "grupposette7@gmail.com",
"password": "aaabbbb"

ID	Expected Output	Output	Result
TC17	Login failed	Login failed	SUCCESS

7.8 Testing con Postman

Si riportano i test case effettuati mediante l'utilizzo di Postman:

In figura(7.6), viene riportata la ripartizione dei diversi in base allo stato dello studente: registrato sulla base dati, non registrato e lo studente che non ha ancora confermato la propria email.

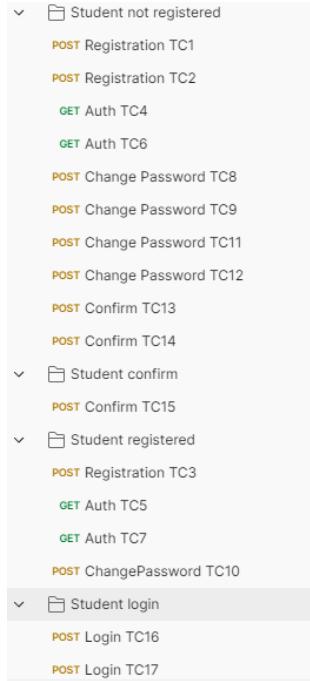


Figura 7.6: Ripartizione dei test case

POST Change Password TC9 http://localhost:8080/api/changepassword	200 OK 6 ms 361 B
PASS Status code is 200	
PASS Student not enabled	
POST Change Password TC11 http://localhost:8080/api/changepassword/token	200 OK 5 ms 368 B
PASS Status code is 200	
PASS Password token not found	
POST Change Password TC12 http://localhost:8080/api/changepassword/token	200 OK 4 ms 358 B
PASS Password token empty	
PASS Status code is 200	
POST Confirm TC13 http://localhost:8080/api/confirm/token	200 OK 5 ms 372 B
PASS Confirmation token not found	
PASS Status code is 200	
POST Confirm TC14 http://localhost:8080/api/confirm/token	200 OK 5 ms 372 B
PASS Confirmation token not found	
PASS Status code is 200	

Figura 7.7: Test Case

```

POST Registration TC1
http://localhost:8080/api/registration
| PASS Status code is 200
| PASS Student registered
200 OK 10776 ms 340 B

POST Registration TC2
http://localhost:8080/api/registration
| PASS Student registered
| PASS Status code is 200
200 OK 10637 ms 340 B

GET Auth TC4
http://localhost:8080/api/studentid
| PASS Status code is 200
| PASS NULL
200 OK 5 ms 337 B

GET Auth TC6
http://localhost:8080/api/studentinfo
| PASS Status code is 200
| PASS Student not found
200 OK 5 ms 293 B

POST Change Password TC8
http://localhost:8080/api/changepassword
| PASS Status code is 200
| PASS Student not found
200 OK 5 ms 352 B

```

Figura 7.8: Test Case

```

POST Registration TC3
http://localhost:8080/api/registration
| PASS Status code is 200
| PASS Student already registered
200 OK 7 ms 384 B

GET Auth TC5
http://localhost:8080/api/studentid
| PASS Status code is 200
| PASS Retrieve student id
200 OK 6 ms 358 B

GET Auth TC7
http://localhost:8080/api/studentinfo
| PASS Status code is 200
| PASS Retrieve student data
200 OK 5 ms 721 B

POST ChangePassword TC10
http://localhost:8080/api/changepassword
| PASS Status code is 200
| PASS Started change password process
200 OK 10431 ms 340 B

```

Figura 7.9: Test Case

```

POST Confirm TC15
http://localhost:8080/api/confirm/token
| PASS Status code is 200
| PASS Student confirmed
200 OK 14 ms 340 B

```

Figura 7.10: Test Case

```

POST Login TC16
http://localhost:8080/api/login
| PASS Status code is 200
| PASS Login successful
200 OK 2556 ms 338 B

POST Login TC17
http://localhost:8080/api/login
| PASS Status code is 200
| PASS Login failed
200 OK 1044 ms 339 B

```

Figura 7.11: Test Case

Capitolo 8

Installazione

Il presente capitolo si concentra sull'importanza e sul processo di installazione del software esplorando i passaggi necessari per l'installazione e l'esecuzione della web Application.

8.1 Installazione software in locale

Per quanto riguarda l'installazione in locale del software, l'applicazione viene fornita attraverso un file JAR denominato testgame-1.0-alpha.jar. In particolare, attraverso il file di configurazione denominato *application.properties* è possibile configurare opportunamente diversi parametri tra cui: il mail server, il database e la connessione a quest'ultimo. Ovviamente, bisogna assicurarsi che tale file si trovi nella stessa cartella in cui si trova il file *.jar* per poter garantire un corretto funzionamento. Dal punto di vista della compatibilità, il sistema sul quale il software viene installato deve avere **necessariamente** una versione di *Java 17* o superiore e un database documentale *MongoDB*. **È importante sottolineare che eventuali modifiche al progetto iniziale comportano necessariamente la rigenerazione del file jar.**

Generazione del file jar Utilizzando un IDE come SpringToolSuite4 fornito dalla stessa Spring e reperibile al sito <https://spring.io/tools>, dal momento che all'interno del progetto è stato utilizzato *Maven* come tool di build automation, per la generazione del file Jar bisogna seguire i seguenti passi:

1. Importare il progetto all'interno dell'IDE. A tale scopo: File -> Import -> Maven -> Existing Maven Project e selezionare la cartella definita come Root.

2. Una volta che il progetto viene caricato e le dipendenze risolte bisogna cliccare sulla root principale con il tasto destro. Sul menù a tendina che si apre: Run As -> Maven Clean;
3. Infine, nuovamente: Tasto destro -> Run As -> Maven Install e attendere il messaggio ***Build Success*** in console.

Esecuzione con Docker Desktop Docker Desktop è un'applicazione che permette di istanziare ed eseguire degli oggetti in un ambiente virtuale. In questo caso, tali oggetti, chiamati Docker, sono l'applicazione web e il Database MongoDB. Per la loro creazione, è necessario definire in un apposito file, chiamato *docker-compose.yml*, i parametri di configurazione. Questi parametri definiscono le dipendenze necessarie per permettere l'esecuzione dell'applicazione. In un ulteriore file chiamato *Dockerfile* vengono specificati i parametri del container che si vuole generare, questi parametri riguardano le porte esposte all'esterno, il file eseguibile, l'ambiente in cui il file testgame-1.0-alpha.jar deve essere eseguito e l'entry point, ovvero un file che specifica al docker cosa eseguire come file principale. Il *docker-compose.yml* possiede due sezioni: la prima contiene la configurazione dell'applicazione principale, la seconda specifica la configurazione del Database.

Per l'installazione nel Docker è necessario creare i file *docker-compose.yml* e *Dockerfile*.

Esecuzione con parametri di Default In questo caso l'applicazione è già fornita di file *docker-compose.yml* e *Dockerfile* pronti all'esecuzione. Per poter generare il container e i suoi Docker bisogna eseguire nella cartella che contiene la directory chiamata target, che contiene l'eseguibile, il comando
`docker-compose up`
con privilegi di amministratore.

Esecuzione con parametri personalizzati Nel caso si volessero modificare i parametri di default bisogna modificare il *dockerfile* in questo modo:

- **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;
- **EXPOSE** : porte esposte all'esterno dell'applicazione;

- **ENTRYPOINT** : comando e parametri del comando per eseguire il programma specificato nel **COPY** .

e poi specificare le porte esposte alla rete interna al container nel docker-compose.yml.

Ricompilare il progetto ed eseguire il comando docker-compose come specificato in precedenza.

8.2 Esecuzione tramite SpringToolSuite4

Per eseguire la web Application direttamente dall'IDE SpringToolSuite4 si seguano i seguenti passi:

1. File -> Import -> Maven -> Existing Maven Project
2. Dal Project Explorer espandere il progetto e la cartella src/main/java
3. Espandere il package com.testgame.gseven
4. Tasto destro sul file GsevenApplication.java -> Run As -> Java Application
5. Attendere che sulla console si visualizzi il seguente messaggio: **Application availability**
6. Aprire il browser e lanciare "localhost:8080/login" per la schermata iniziale.

