Relazione – Gruppo 10

Lucio Ilardi M63001505

Mattia Flagiello M63001540

Sommario

INTRODUZIONE

Il problema che abbiamo dovuto affrontare è quello relativo ai task T2-T3, ossia gestire la registrazione dello studente e la sua autenticazione per l'accesso alle funzionalità del

sistema quali giocare una nuova partita o visionare lo storico delle sessioni di gioco passate. Il processo di sviluppo agile che abbiamo seguito prende ispirazione da SCRUM; ci ha consentito di produrre codice e artefatti, riuscendo a raggiungere gli obiettivi posti nelle diverse iterazioni compatibilmente con le deadline assegnate (circa ogni 2 settimane). Abbiamo adottato l'utilizzo di un product backlog, aggiornato progressivamente durante lo sviluppo.

La prima decisione architetturale che abbiamo preso è stata quella di voler realizzare un componente accessibile via Web mediante API REST, una soluzione più leggera rispetto ad altri standard come SOAP/WSDL. L'interfaccia è stata opportunamente documentata tramite Swagger.

L'architettura del componente è a livelli; in particolare si tratta di una Closed Layered Architecture in cui ogni livello agisce da client per il solo livello sottostante. Abbiamo preferito questa tipologia di architettura per favorire la modificabilità e per suddividere meglio le responsabilità delle diverse parti del sistema.

In base a tali decisioni la scelta della tecnologia è ricaduta sul framework di Spring Boot (e Spring Security per la parte di autenticazione e autorizzazione), dunque in linguaggio Java. Il DBMS utilizzato per garantire la persistenza dei dati è H2, un gestore di basi dati relazionale ed in-memory; i motivi di tale scelta sono stati il fatto che i dati da memorizzare nel database sono relativamente leggeri (unica tabella relativa ai dati dello studente) e il voler garantire una migliore deployability siccome non sono richieste installazioni separate per il DBMS. Altre tecnologie utilizzate sono html e css per la creazione di pagine web, javascript per implementare controlli sul formato dell'input e per mostrare all'utente eventuali messaggi di notifica e di errore provenienti dal backend, Junit per la creazione di casi di test.

La documentazione dell'API è consultabile su https://app.swaggerhub.com/apis-docs/LUCIILARDI/StudentDB/1.0 o comunque su Github https://github.com/Testing-Game-SAD-2023/T23-G10 .

L'interfacciamento è stato realizzato con i gruppi T4-G4 e T5-G30 per i metodi che consentono di avviare una nuova partita e visualizzare lo storico delle partite.

GLOSSARIO DEI TERMINI

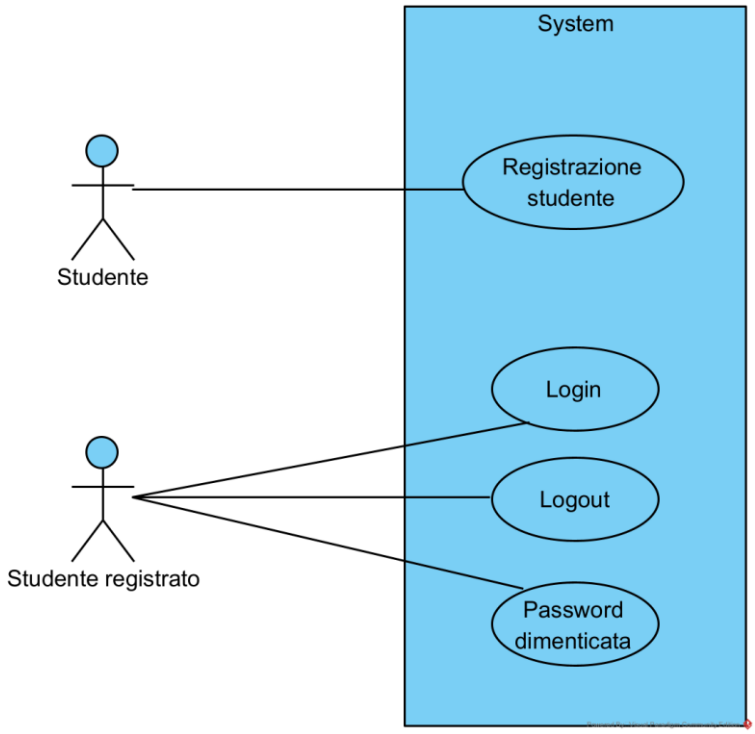
Un chiarimento su alcuni termini che potrebbero causare confusione.

Termine	Descrizione	Sinonimi

Studente	La persona che intende registrarsi e accedere ai servizi dell'applicazione.	Student, User, Utente
Password dimenticata	Il caso d'uso che consente ad uno studente che ha dimenticato la propria password di cambiarla.	Recupero dell'account, forgot password, cambio password
Verificato	Indica il fatto che lo studente ha completato la registrazione effettuando la procedura di conferma via email.	Abilitato
Token	La stringa di caratteri utilizzata per confermare l'account in fase di registrazione o per autorizzare il cambio password nel rispettivo caso d'uso.	Codice, Code

USE CASE DIAGRAM

La prima attività di ingegneria del software che abbiamo affrontato è stata quella di analisi e specifica dei requisiti mediante la creazione di un diagramma dei casi d'uso e la descrizione dettagliata dei singoli scenari considerando anche le possibili eccezioni. Abbiamo individuato due attori, studente e studente registrato, e 4 casi d'uso fondamentali: registrazione, login, logout e password dimenticata. Lo studente deve effettuare la registrazione prima di poter accedere alle altre funzionalità.



USE CASE DESCRIPTION

Caso d'uso : Registrazione studente
Attore primario : Studente
Attori secondari : Nessuno
Descrizione: Lo studente effettua la registrazione nel sistema.
Pre-condizioni: Nessuna
Scenario principale: 1- Lo studente inserisce nome, cognome, corso di studi, email e password. 2- Il sistema controlla che nome e cognome siano stringhe di soli caratteri, che l'email sia valida e che la password contenga almeno 8 caratteri, di cui almeno una maiuscola, un numero e un carattere speciale. 3- Il sistema controlla che lo studente non sia già registrato. 4- Il sistema associa allo studente un identificativo univoco. 5- Il sistema registra lo studente nella repository dei giocatori e invia una email per la verifica dell'account. 6- Lo studente clicca il link ricevuto tramite email. 7- Il sistema abilita l'account dello studente.
Post condizioni: Lo studente è registrato nel sistema.
Scenari alternativi:

2b- Se l'email o la password fornite non sono valide, il sistema chiede nuovamente l'inserimento di esse.
3b- Se l'utente è già registrato il caso d'uso termina.
7b- Se il link non è valido viene restituito un messaggio di errore.

Caso d'uso : Login
Attore primario : Studente registrato
Attori secondari : Nessuno
Descrizione: : Lo studente effettua il login nel sistema per l'accesso alle funzionalità di gioco o di consultazione delle sessioni di gioco passate.
Pre-condizioni: Lo studente è registrato nel sistema.
Scenario principale: 1- Lo studente inserisce indirizzo email e password fornite all'atto della registrazione. 2- Il sistema controlla la validità dei dati forniti. 3- Il sistema autentica il giocatore.
Post condizioni: Lo studente è autenticato nel sistema.
Scenari alternativi: 2b- Se i dati forniti non sono validi, oppure lo studente non ha verificato il proprio account, il sistema restituisce un messaggio di errore e chiede nuovamente l'inserimento di essi.

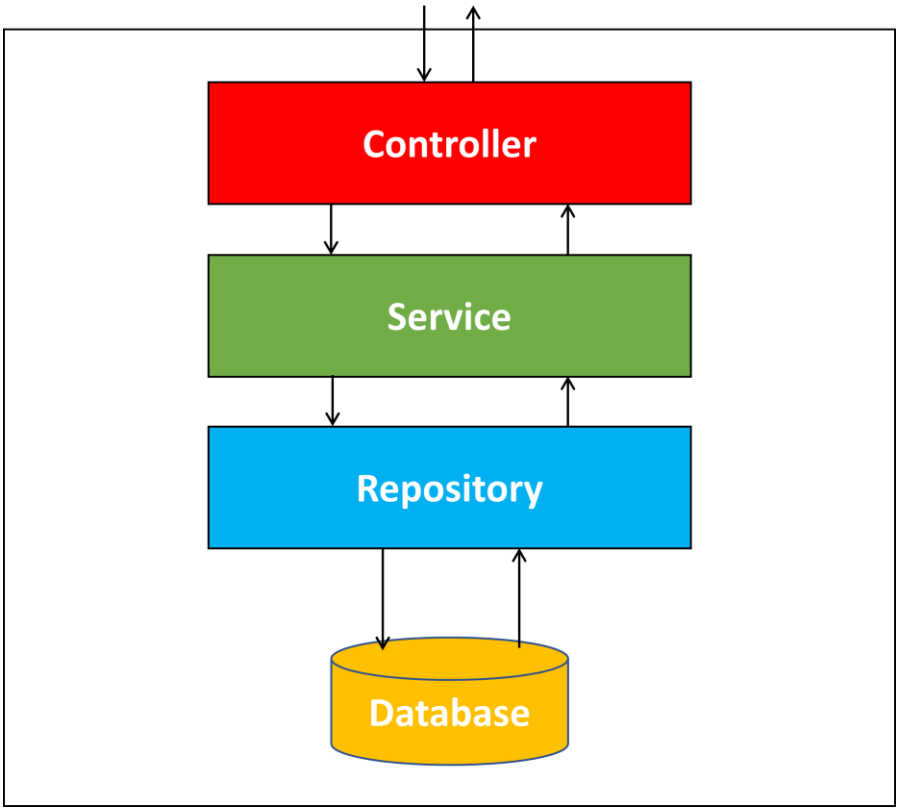
Caso d'uso : Password dimenticata
Attore primario : Studente registrato
Attori secondari : Nessuno
Descrizione: Lo studente richiede il cambio della propria password.
Pre-condizioni: Lo studente è registrato nel sistema.
Scenario principale: 1- Lo studente inserisce il proprio indirizzo email. 2- Il sistema controlla che l'email esista nel database e che l'account sia abilitato. 3- Il sistema invia una email allo studente con un link per effettuare il cambio password. 4- Lo studente clicca sul link ricevuto via email e inserisce la nuova password. 5- Il sistema controlla che la password rispetti i requisiti richiesti. 6- Il sistema aggiorna la password dello studente.

Post condizioni: La password dello studente è stata aggiornata.
Scenari alternativi: 2b- Se l'email inserita non esiste nel database oppure l'account ad esso collegato non è stato verificato il sistema restituisce un messaggio di errore e chiede nuovamente l'inserimento. 5b- Se la password fornita non rispetta i requisiti richiesti il sistema restituisce un messaggio di errore e chiede nuovamente l'inserimento.

Caso d'uso : Logout
Attore primario : Studente registrato
Attori secondari : Nessuno
Descrizione: Lo studente effettua il logout dal sistema.
Pre-condizioni: Lo studente ha effettuato il login.
Scenario principale: 1- Lo studente effettua il logout. 2- Il sistema revoca l'accesso allo studente.
Post condizioni: Lo studente non è più autenticato nel sistema.
Scenari alternativi: Nessuno

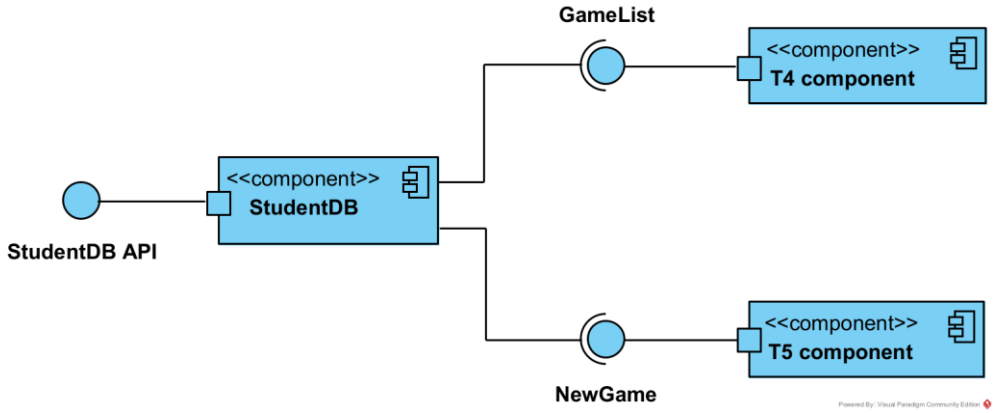
ARCHITECTURE LAYOUT

L'architettura è organizzata secondo i seguenti livelli: controller, service, repository e database. Il controller si occupa di gestire tutte le richieste effettuate tramite l'API; controlla la validità dei dati forniti in input (gli stessi controlli sono implementati anche nella pagine html fornite) e nel caso siano validi invoca i servizi del livello sottostante, altrimenti restituisce messaggi o pagine di errore all'utente. Il livello service è quello che implementa la logica di business, prelevando le richieste fatte dal controller ed aggiornando lo stato del sistema attraverso la chiamata dei metodi forniti dal livello repository; quest'ultimo fornisce infatti le funzioni che permettono di accedere e modificare il database degli studenti.



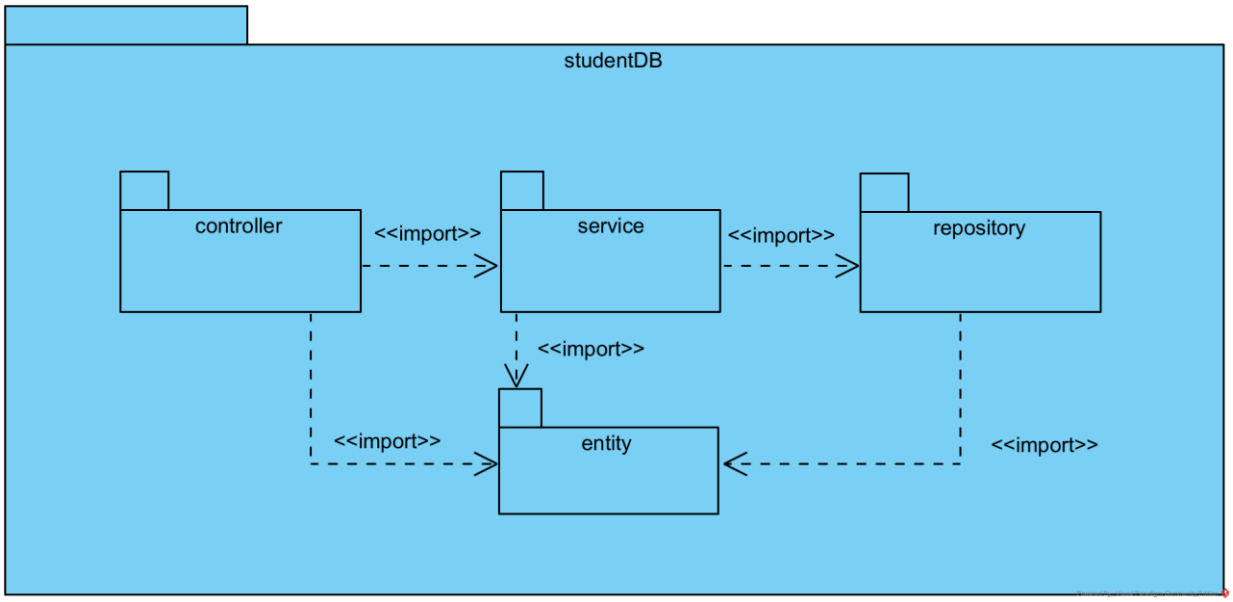
COMPONENT DIAGRAM

Sebbene il nostro sistema sia organizzato come un unico componente, abbiamo ritenuto utile mostrare il seguente diagramma per mettere in mostra interfacce esposte e richieste.



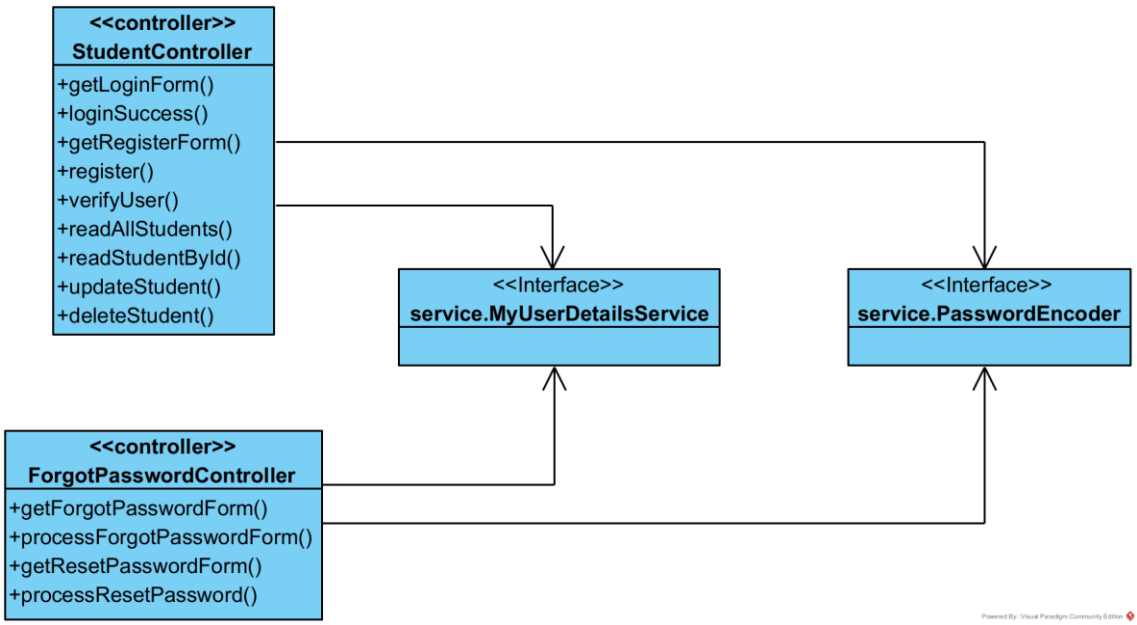
PACKAGE DIAGRAM

I package sono organizzati coerentemente con il disegno dell'architettura riportato sopra, con il package "entity" importato da tutti dato che contiene le classi dati fondamentali.



ANALYSIS CLASS DIAGRAM – CONTROLLER

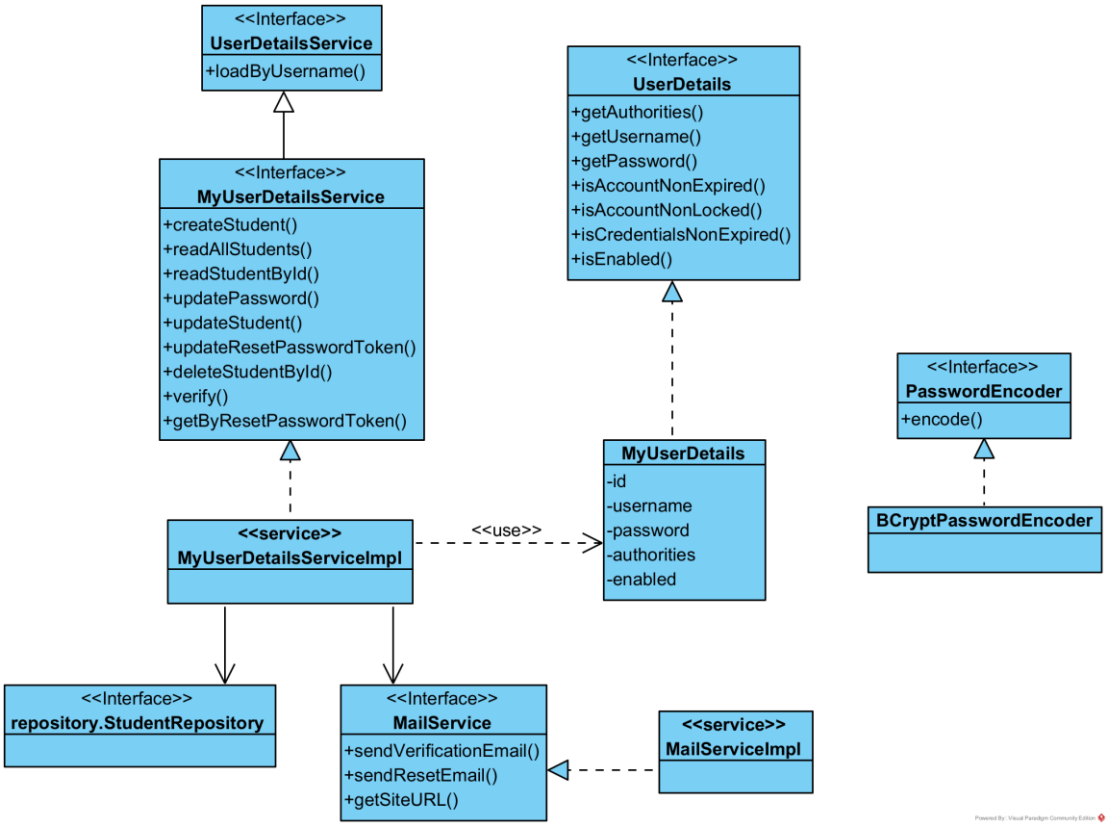
Il livello più alto dell'architettura mostra la presenza di due classi controller: lo StudentController gestisce le richieste relative a registrazione e login/logout, il ForgotPasswordController invece quelle relative al cambio della password. Entrambi i controller invocano i servizi del livello sottostante mediante due interfacce: MyUserDetailsService che contiene tutte le funzioni utili alla realizzazione della logica di business, PasswordEncoder che offre un metodo per la cifratura della password.



ANALYSIS CLASS DIAGRAM – SERVICE

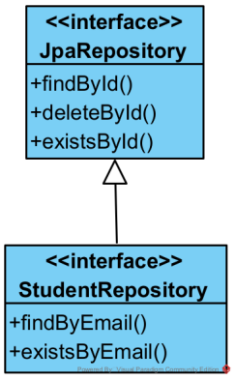
Nel seguente diagramma è mostrata l'organizzazione delle classi a livello service. La classe principale a questo livello è MyUserDetailsService: oltre a contenere una serie di metodi

utili implementa l'interfaccia UserDetailsService, il cui metodo loadByUsername viene invocato in fase di autenticazione per recuperare gli UserDetails (in sostanza i dati dello studente) dal database e confrontarli con le credenziali fornite in input. Si noti come MyUserDetailsService faccia utilizzo di un altro servizio, MailService, ossia quello atto alla gestione delle email da inviare allo studente, e acceda ai metodi del livello inferiore attraverso l'interfaccia StudentRepository. Lo specifico tipo di PasswordEncoder impiegato è il BCryptPasswordEncoder.



ANALYSIS CLASS DIAGRAM – REPOSITORY

L'interfaccia StudentRepository estende la JpaRepository di Spring. L'interfaccia Java Persistence API consente di lavorare con il database utilizzando Java.

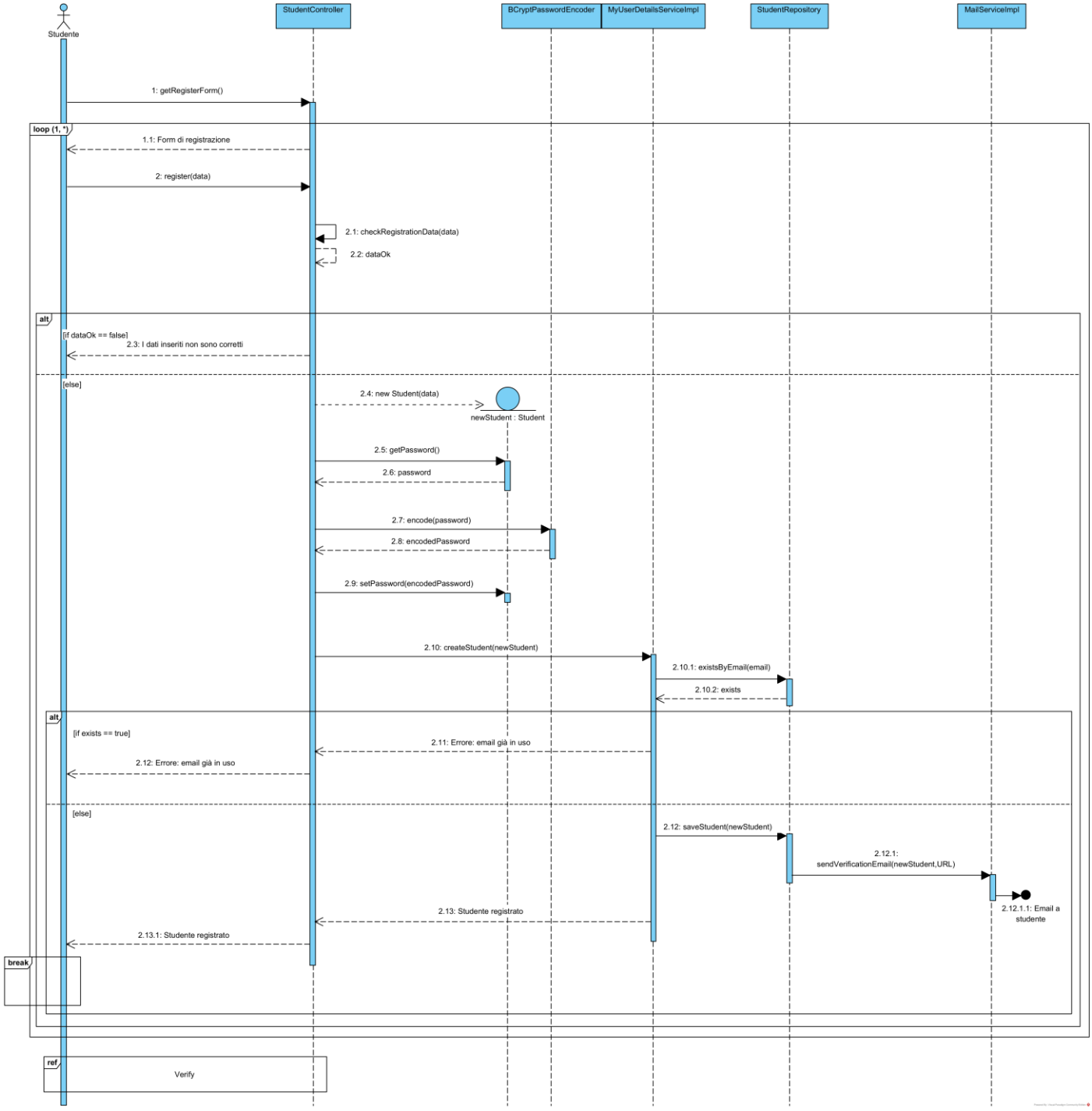


ANALYSIS CLASS DIAGRAM – ENTITY

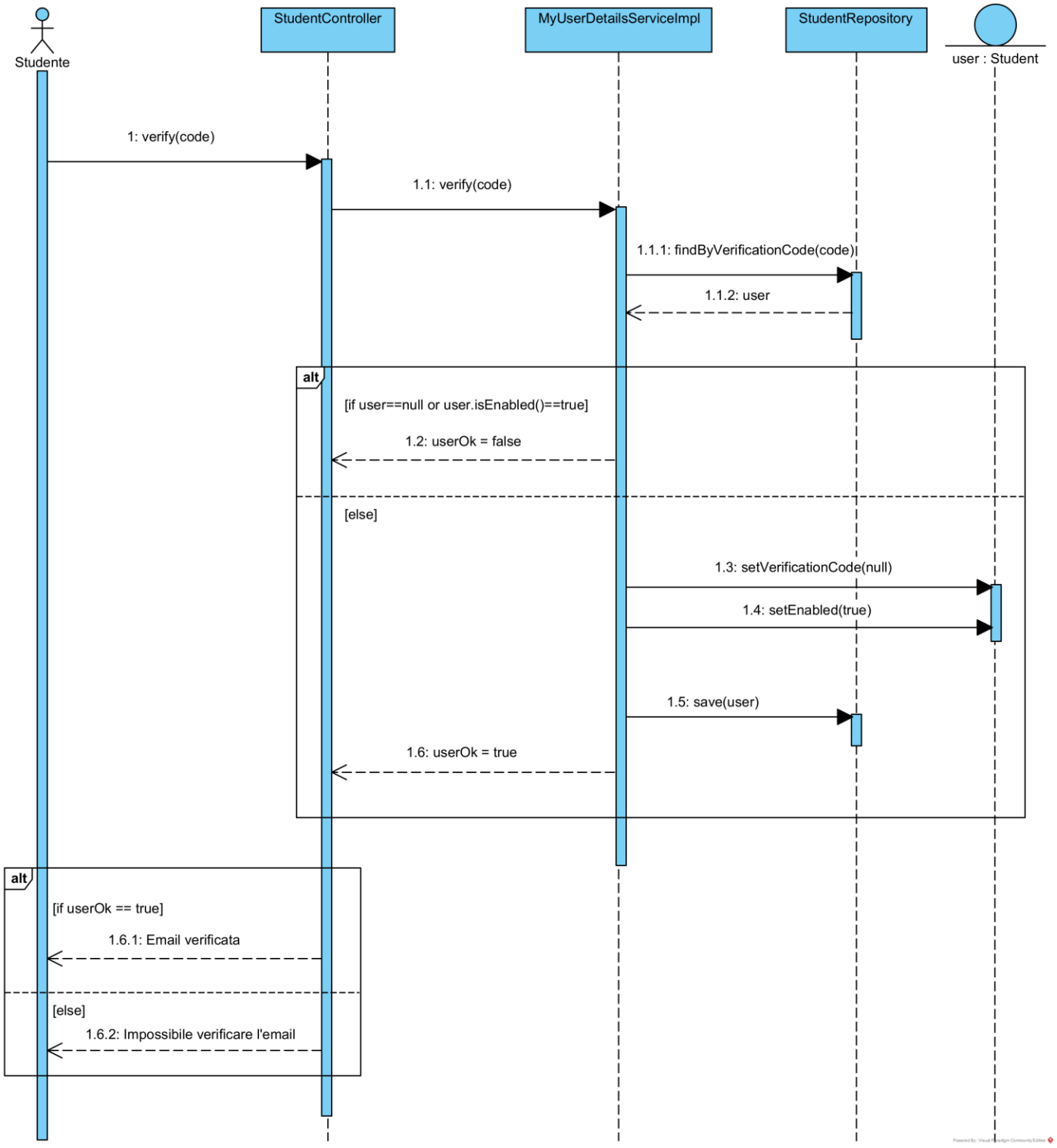
Siccome il corso può assumere un numero finito di valori è stato scelto di creare una classe enum in relazione con lo studente piuttosto che come semplice attributo.

ANALYSIS SEQUENCE DIAGRAM – REGISTRAZIONE

La registrazione è stata gestita nel seguente modo: lo studente fornisce i propri dati e, se questi sono validi, viene registrato nel database ma il suo account non è ancora abilitato. Per abilitarlo deve confermare la registrazione via email: il sistema genera un token randomico per la verifica e lo associa allo studente. Lo stesso token viene inviato via email sottoforma di link. Lo studente che clicca il link effettua una chiamata GET a /verify passando come query parameter il token; a questo punto il sistema controlla se tale token coincide con quello presente nel database, e in caso affermativo abilita l'account, altrimenti restituisce una pagina di errore.

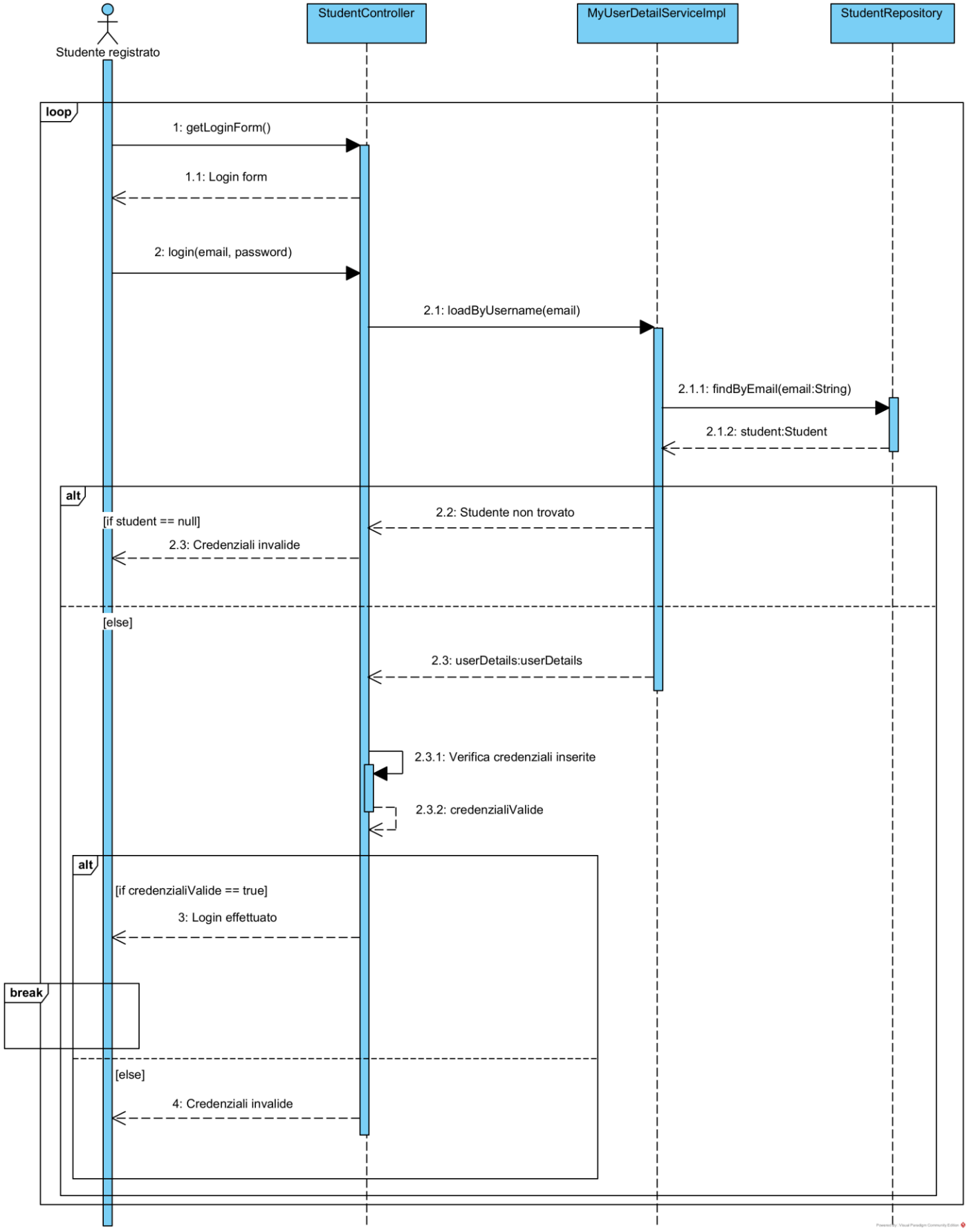

Studente
StudentController
BCryptPasswordEncoder
MyUserDetailsServiceImpl
StudentRepository
MailServiceImpl
1: getRegisterForm()
loop (1, *)
1.1: Form di registrazione
2: register(data)
2.1: checkRegistrationData(data)
2.2: dataOk
alt
[if dataOk == false]
2.3: I dati inseriti non sono corretti
[else]
2.4: new Student(data)
newStudent : Student
2.5: getPassword()
2.6: password
2.7: encode(password)
2.8: encodedPassword
2.9: setPassword(encodedPassword)
2.10: createStudent(newStudent)
2.10.1: existsByEmail(email)
2.10.2: exists
alt
[if exists == true]
2.11: Errore: email già in uso
2.12: Errore: email già in uso
[else]
2.12: saveStudent(newStudent)
2.12.1:
sendVerificationEmail(newStudent,URL)
2.12.1.1: Email a studente
2.13: Studente registrato
2.13.1: Studente registrato
break
ref
Verify

VERIFY:



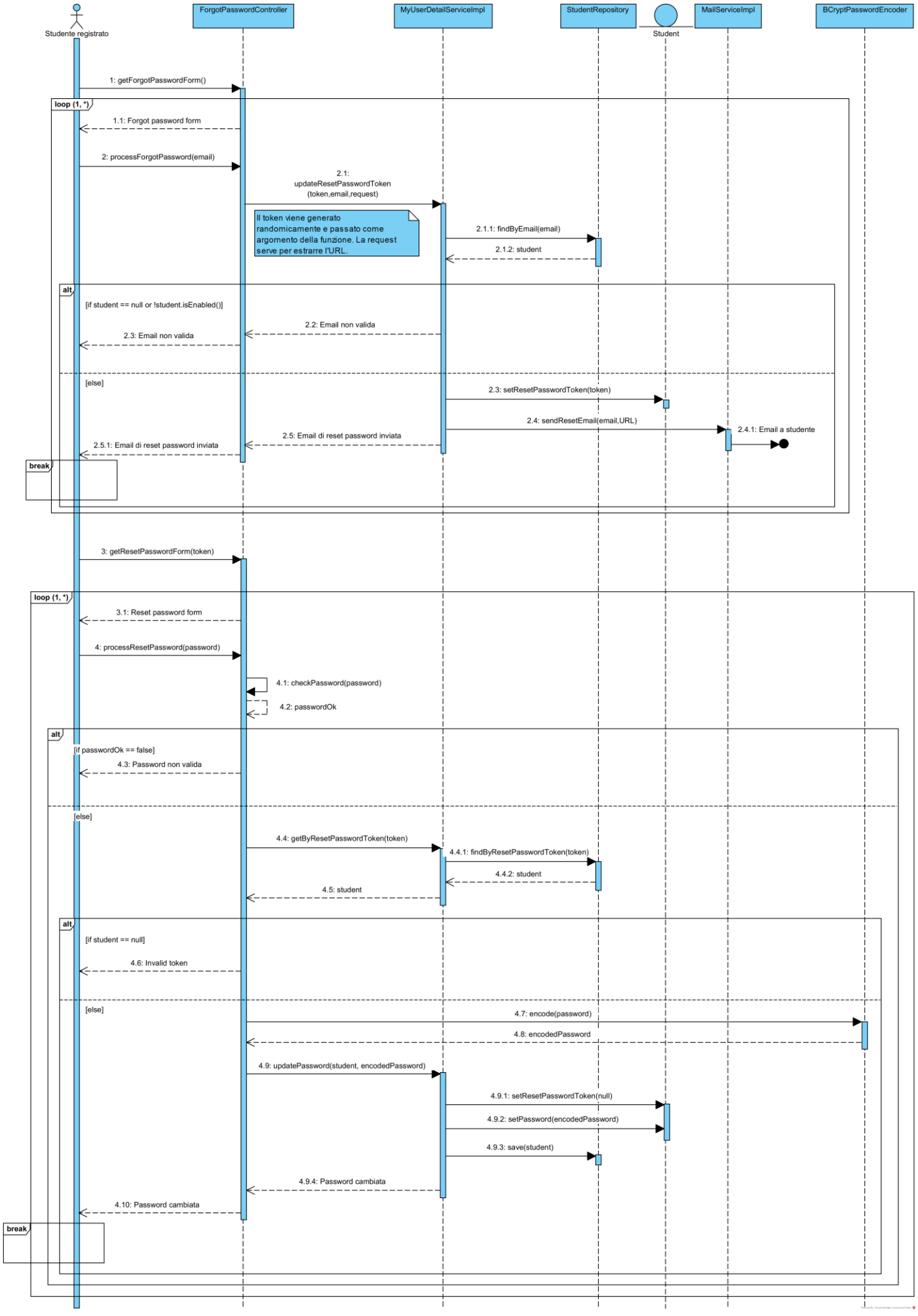
ANALYSIS SEQUENCE DIAGRAM – LOGIN

Come già anticipato, quando uno studente effettua il login il framework di Spring invoca automaticamente il metodo loadByUsername implementato nella classe MyUserDetailsService per recuperare gli UserDetails dal database e confrontarli con email e password fornite per l'autenticazione.


Studente registrato
StudentController
MyUserDetailServiceImpl
StudentRepository
loop
1: getLoginForm()
1.1: Login form
2: login(email, password)
2.1: loadByUsername(email)
2.1.1: findByEmail(email:String)
2.1.2: student:Student
alt
[if student == null]
2.2: Studente non trovato
2.3: Credenziali invalide
[else]
2.3: userDetails:userDetails
2.3.1: Verifica credenziali inserite
2.3.2: credenzialiValide
alt
[if credenzialiValide == true]
3: Login effettuato
break
[else]
4: Credenziali invalide

ANALYSIS SEQUENCE DIAGRAM – PASSWORD DIMENTICATA

Per il cambio password lo studente inserisce la propria email; il sistema, con un meccanismo basato su token simile a quello visto per la registrazione, invia un link che consente di cambiare la password.

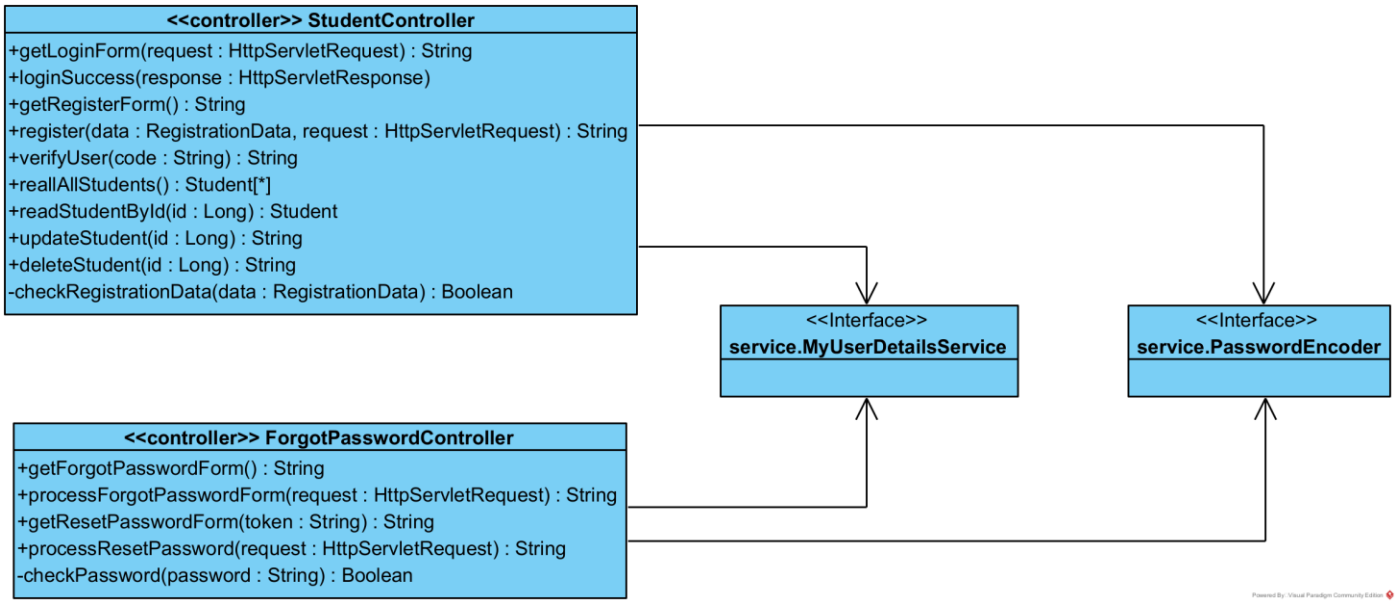

Studente registrato
ForgotPasswordController
MyUserDetailServiceImpl
StudentRepository
Student
MailServiceImpl
BCryptPasswordEncoder
1: getForgotPasswordForm()
loop (1, *)
1.1: Forgot password form
2: processForgotPassword(email)
2.1:
updateResetPasswordToken
(token,email,request)
Il token viene generato randomicamente e passato come argomento della funzione. La request serve per estrarre l'URL.
2.1.1: findByEmail(email)
2.1.2: student
alt
[if student == null or !student.isEnabled()]
2.2: Email non valida
2.3: Email non valida
[else]
2.3: setResetPasswordToken(token)
2.4: sendResetEmail(email,URL)
2.4.1: Email a studente
2.5: Email di reset password inviata
2.5.1: Email di reset password inviata
break
3: getResetPasswordForm(token)
loop (1, *)
3.1: Reset password form
4: processResetPassword(password)
4.1: checkPassword(password)
4.2: passwordOk
alt
[if passwordOk == false]
4.3: Password non valida
[else]
4.4: getByResetPasswordToken(token)
4.4.1: findByResetPasswordToken(token)
4.4.2: student
4.5: student
alt
[if student == null]
4.6: Invalid token
[else]
4.7: encode(password)
4.8: encodedPassword
4.9: updatePassword(student, encodedPassword)
4.9.1: setResetPasswordToken(null)
4.9.2: setPassword(encodedPassword)
4.9.3: save(student)
4.9.4: Password cambiata
4.10: Password cambiata
break

ANALYSIS SEQUENCE DIAGRAM – LOGOUT

Non è stato sviluppato un sequence diagram per il caso d'uso di logout dato che l'interazione è estremamente banale. Infatti ad una chiamata GET a /logout corrisponde la rimozione dalla sessione del Principal (user autenticato), gestita automaticamente da Spring Security, e un redirect a /login?logout. Se la chiamata avviene quando nessuno studente è autenticato semplicemente viene effettuato il reindirizzamento.

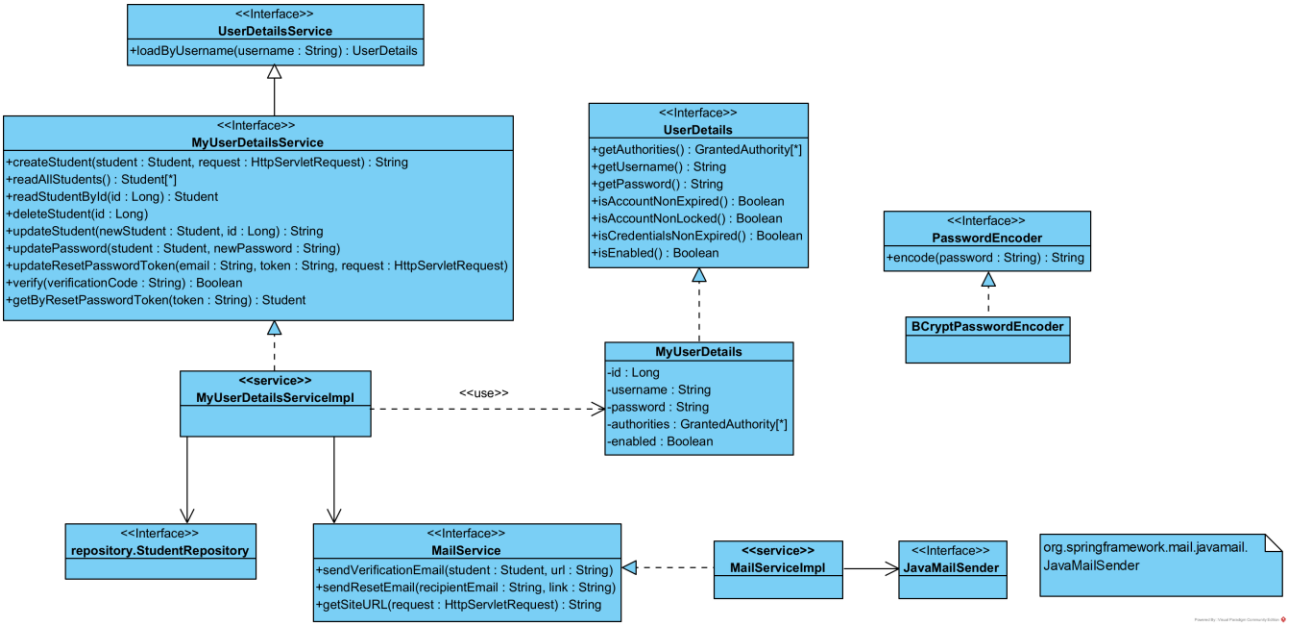
DESIGN CLASS DIAGRAM – CONTROLLER

Il diagramma seguente è specificato con un livello di dettaglio maggiore, con l'aggiunta ai metodi di tutti gli attributi e i rispettivi tipi, nonché i valori di ritorno degli stessi.

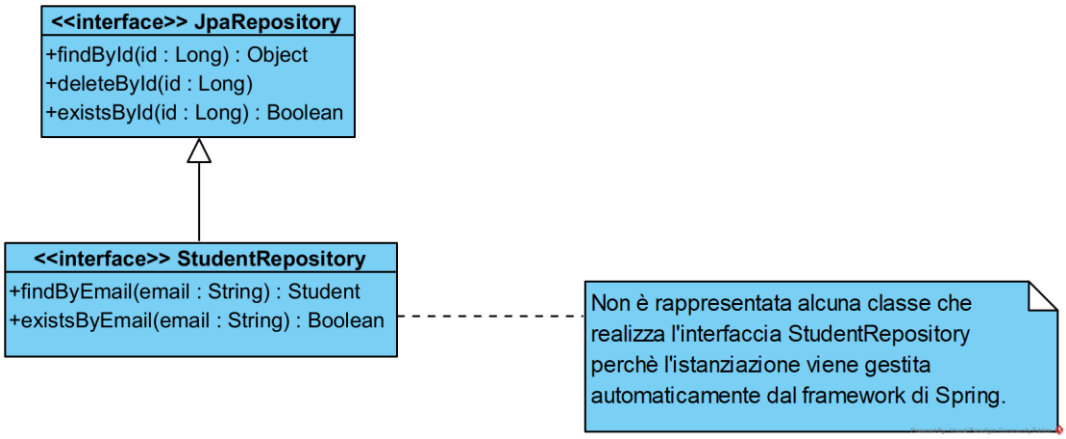


DESIGN CLASS DIAGRAM – SERVICE

Anche in questo caso il diagramma è stato specificato con un superiore livello di dettaglio. Si noti come MailService faccia utilizzo dell'interfaccia JavaMailSender di Spring per l'invio delle email.

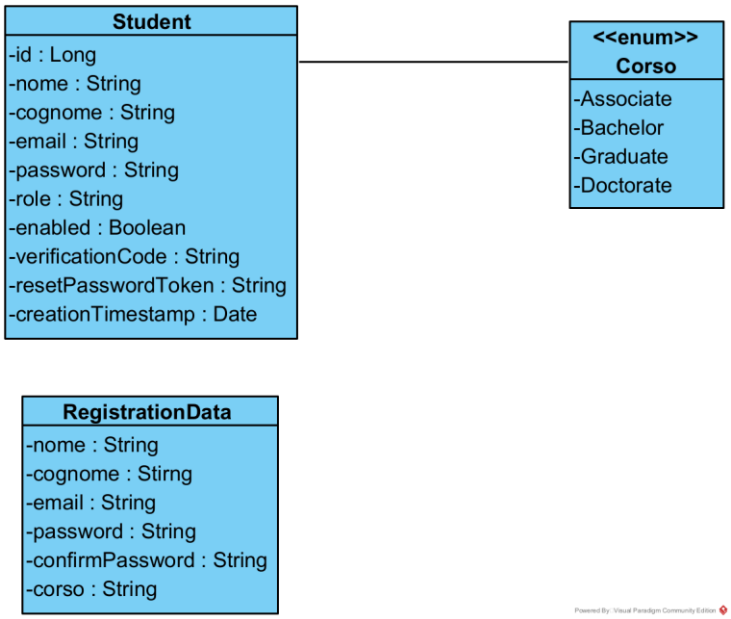


DESIGN CLASS DIAGRAM – REPOSITORY

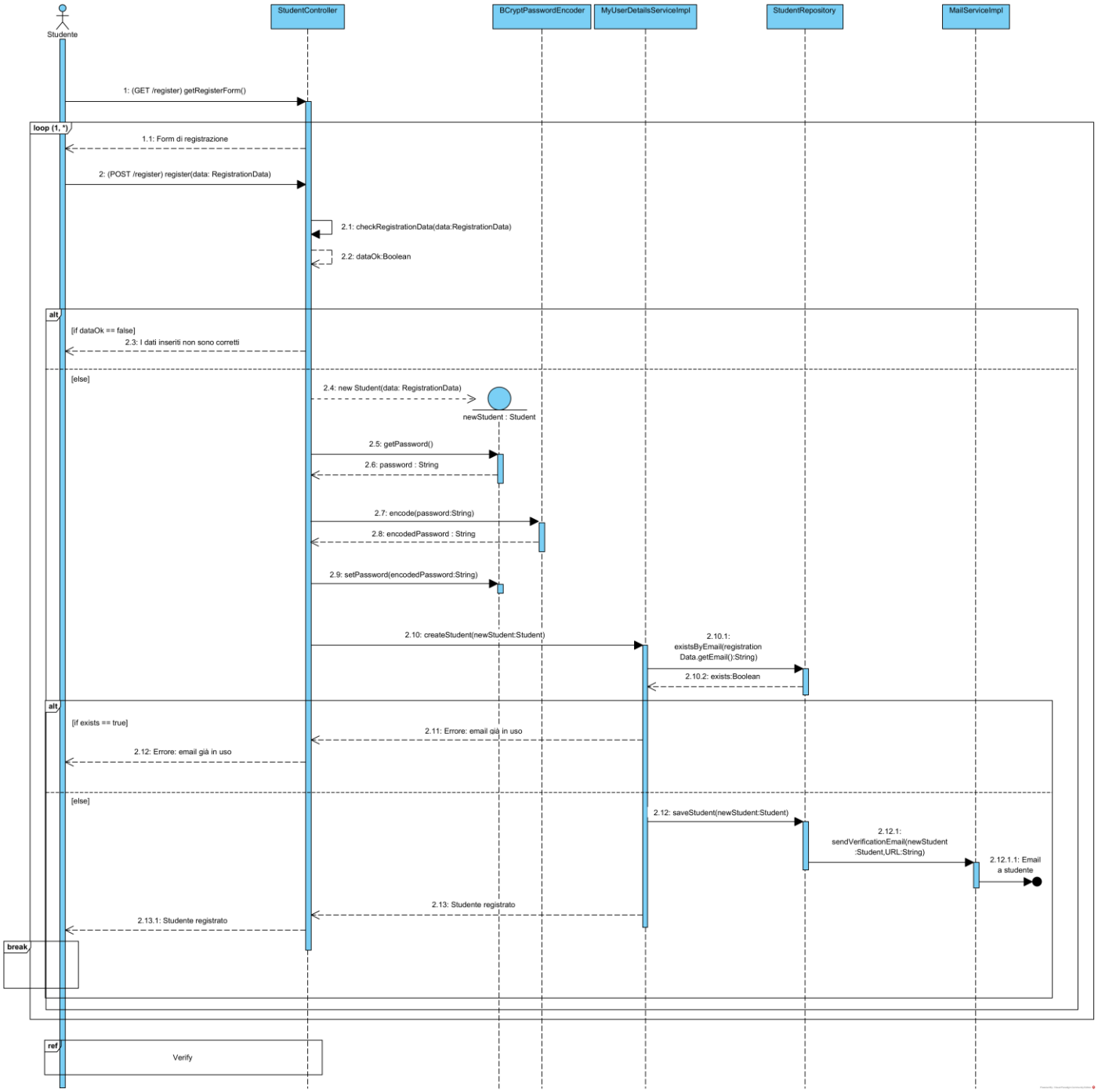


DESIGN CLASS DIAGRAM – ENTITY

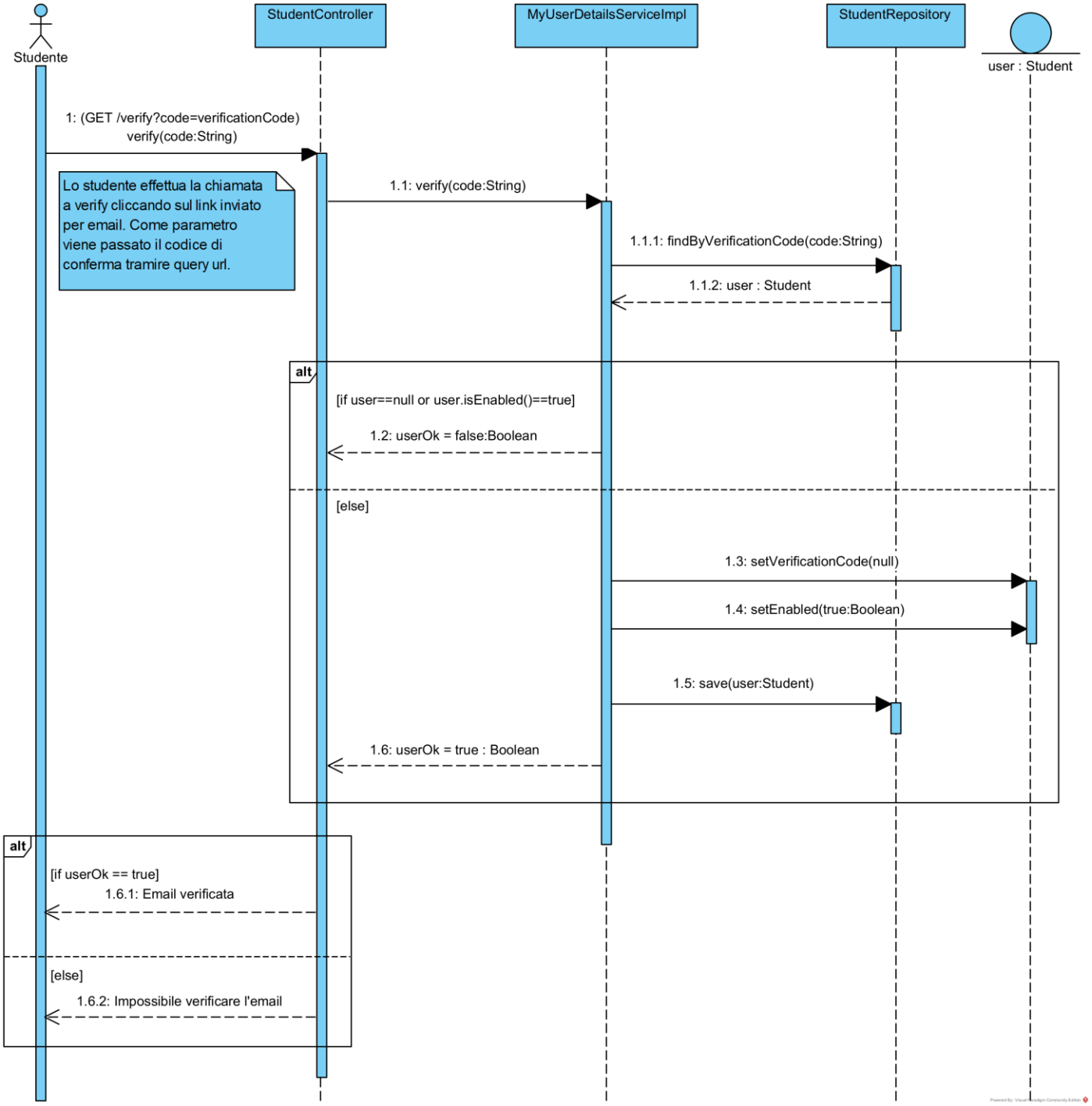
Nel diagramma è stata aggiunta la classe RegistrationData che è utile per raccogliere i dati inviati tramite il form di registrazione.


Student
-id : Long
-nome : String
-cognome : String
-email : String
-password : String
-role : String
-enabled : Boolean
-verificationCode : String
-resetPasswordToken : String
-creationTimestamp : Date
<<enum>>
Corso
-Associate
-Bachelor
-Graduate
-Doctorate
RegistrationData
-nome : String
-cognome : Stirng
-email : String
-password : String
-confirmPassword : String
-corso : String
Powered By: Visual Paradigm Community Edition

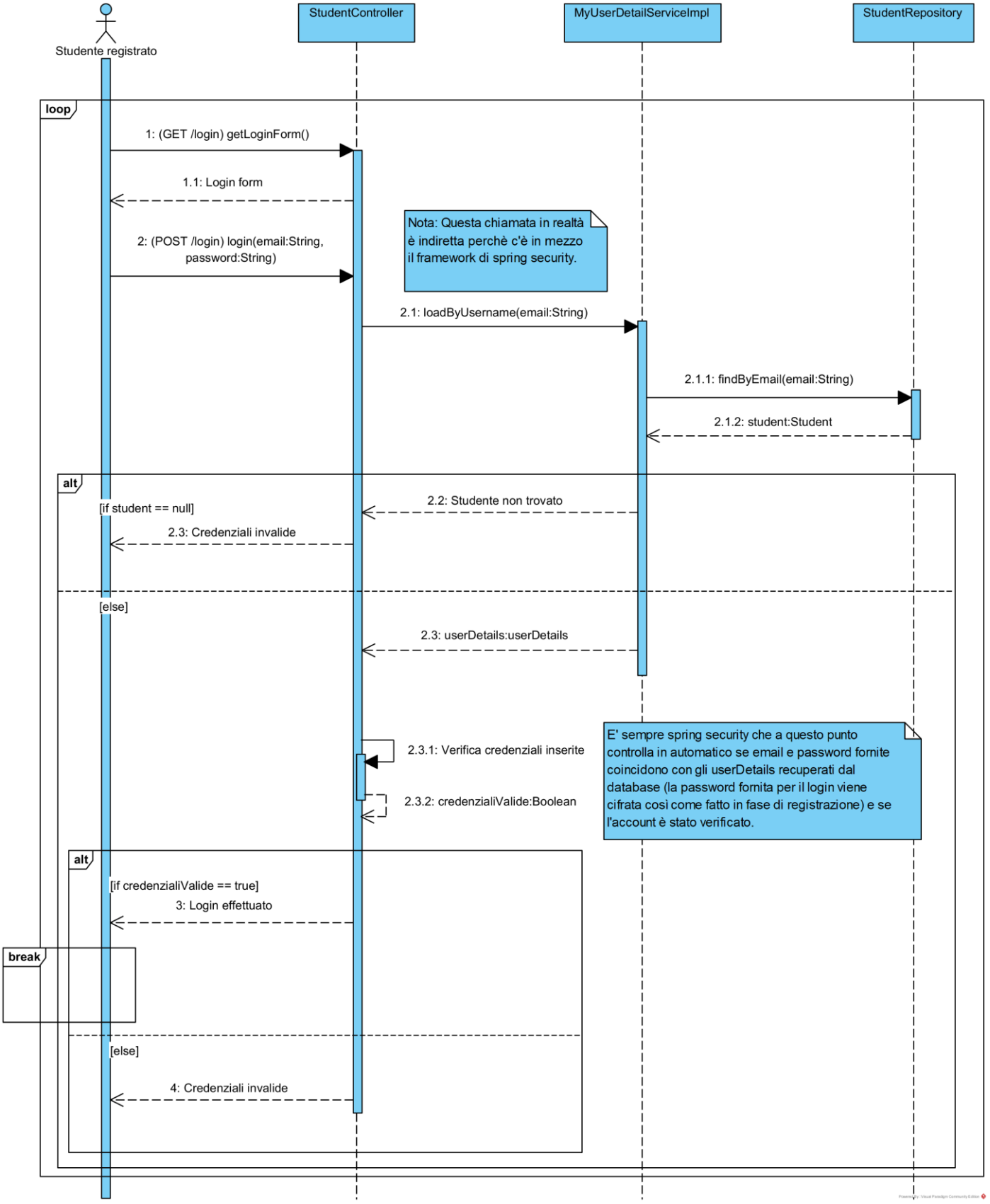
DESIGN SEQUENCE DIAGRAM – REGISTRAZIONE



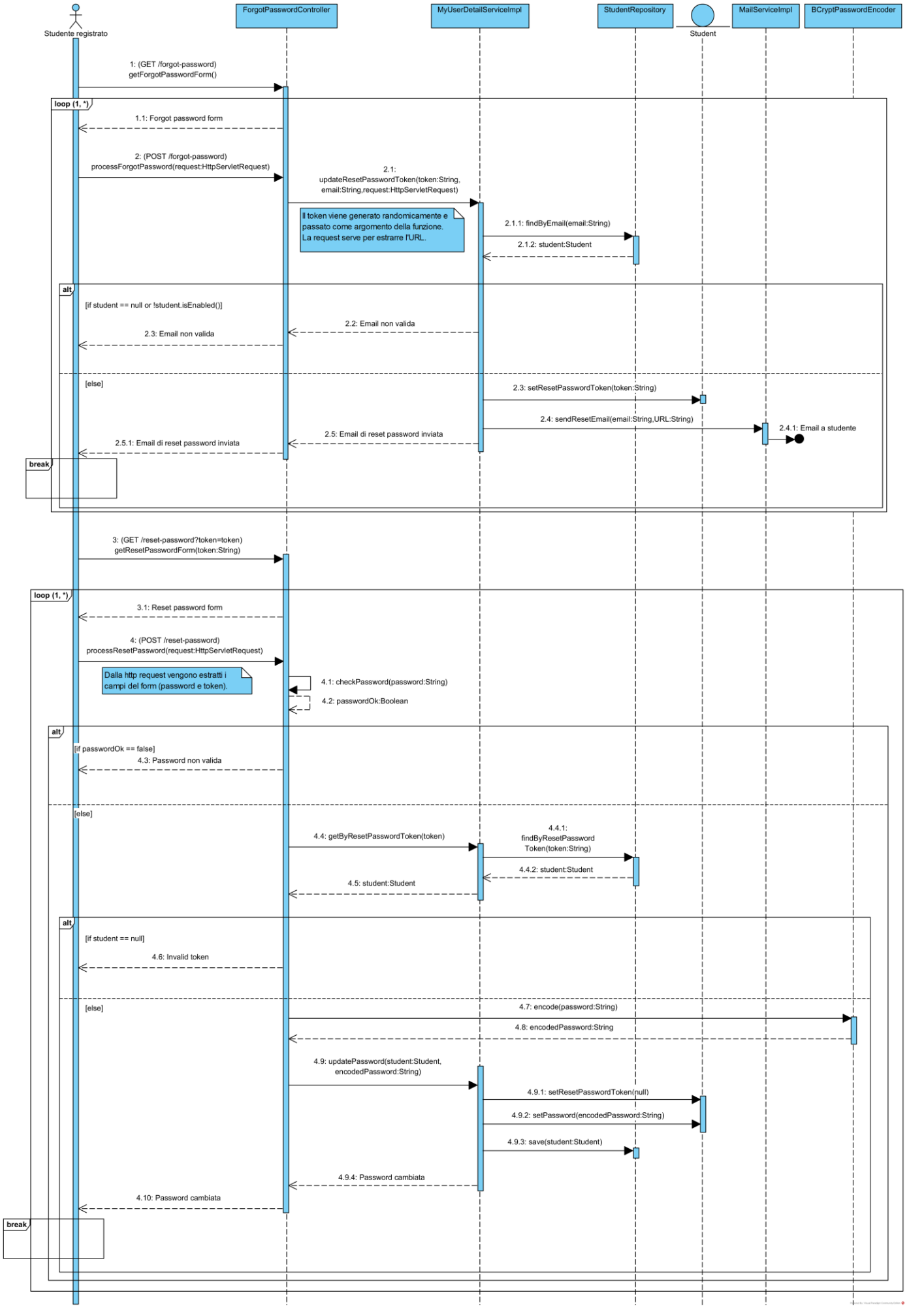
VERIFY:


Studente
StudentController
MyUserDetailsServiceImpl
StudentRepository
user : Student
1: (GET /verify?code=verificationCode)
verify(code:String)
Lo studente effettua la chiamata a verify cliccando sul link inviato per email. Come parametro viene passato il codice di conferma tramire query url.
1.1: verify(code:String)
1.1.1: findByVerificationCode(code:String)
1.1.2: user : Student
alt
[if user==null or user.isEnabled()==true]
1.2: userOk = false:Boolean
[else]
1.3: setVerificationCode(null)
1.4: setEnabled(true:Boolean)
1.5: save(user:Student)
1.6: userOk = true : Boolean
alt
[if userOk == true]
1.6.1: Email verificata
[else]
1.6.2: Impossibile verificare l'email

DESIGN SEQUENCE DIAGRAM- LOGIN



DESIGN SEQUENCE DIAGRAM – PASSWORD DIMENTICATA


Studente registrato
ForgotPasswordController
MyUserDetailServiceImpl
StudentRepository
Student
MailServiceImpl
BCryptPasswordEncoder
1: (GET /forgot-password)
getForgotPasswordForm()
loop (1, *)
1.1: Forgot password form
2: (POST /forgot-password)
processForgotPassword(request:HttpServletRequest)
2.1:
updateResetPasswordToken(token:String,
email:String,request:HttpServletRequest)
Il token viene generato randomicamente e
passato come argomento della funzione.
La request serve per estrarre l'URL.
2.1.1: findByEmail(email:String)
2.1.2: student:Student
alt
[if student == null or !student.isEnabled()]
2.2: Email non valida
2.3: Email non valida
[else]
2.3: setResetPasswordToken(token:String)
2.4: sendResetEmail(email:String,URL:String)
2.4.1: Email a studente
2.5: Email di reset password inviata
2.5.1: Email di reset password inviata
break
3: (GET /reset-password?token=token)
getResetPasswordForm(token:String)
loop (1, *)
3.1: Reset password form
4: (POST /reset-password)
processResetPassword(request:HttpServletRequest)
Dalla http request vengono estratti i
campi del form (password e token).
4.1: checkPassword(password:String)
4.2: passwordOk:Boolean
alt
[if passwordOk == false]
4.3: Password non valida
[else]
4.4: getByResetPasswordToken(token)
4.4.1:
findByResetPassword
Token(token:String)
4.4.2: student:Student
4.5: student:Student
alt
[if student == null]
4.6: Invalid token
[else]
4.7: encode(password:String)
4.8: encodedPassword:String
4.9: updatePassword(student:Student,
encodedPassword:String)
4.9.1: setResetPasswordToken(null)
4.9.2: setPassword(encodedPassword:String)
4.9.3: save(student:Student)
4.9.4: Password cambiata
4.10: Password cambiata
break

FUNCTIONAL TEST

Già prima della scrittura del codice abbiamo individuato le condizioni per l'accettazione delle funzionalità individuate. Questo ci ha permesso di correggere alcuni difetti già nelle fasi iniziali del ciclo di sviluppo e di verificare i requisiti funzionali.

Test per la registrazione dello studente
Input: - Nome : Stringa - Cognome : Stringa - Corso di studi : Stringa - Email : Stringa - Password : Stringa
Test: - Controllare che nome e cognome siano stringhe di soli caratteri. - Controllare che l'email sia valida. - Controllare che la password contenga almeno 8 caratteri, di cui almeno una maiuscola, un numero e un carattere speciale. - Controllare che dopo la registrazione sia presente nella repository uno studente con i dati precedentemente immessi (non ancora abilitato). - Controllare che dopo la verifica dell'account lo studente sia abilitato.
Output: OK, oppure un messaggio di errore che chiede di reinserire i dati.

Test per il login
Input: - Email : Stringa - Password : Stringa
Test: - Controllare che l'email sia registrata nel sistema, che la rispettiva password coincida con quella fornita e che l'account sia abilitato.
Output: OK, oppure un messaggio di errore che chiede di reinserire i dati.

Test per password dimenticata
Input: - Email : Stringa - Nuova Password : Stringa
Test: - Controllare che l'email sia registrata nel sistema, che l'account sia abilitato e che la nuova password rispetti i requisiti richiesti. - Controllare l'autenticità dello studente. - Controllare che nel database sia presente la nuova password inserita.
Output: Password modificata con successo, oppure un messaggio di errore che chiede di reinserire i dati.

Test per logout
Input:
Test: - Controllare che dopo il logout lo studente non sia più autenticato nel sistema.
Output: OK

DETAILED TEST CASES

Ecco riportati i casi di test nello specifico per ogni funzionalità.

- Registrazione:

Test Case ID	**Descrizione**	**Pre-condizioni**	**Input**	**Output Attesi**	**Post-condizioni Attese**
0	Inserimento di dati validi per lo studente.	Nessuna	Nome:Mario Cognome:Rossi Email:mariorossi@gmail.com Password:Password123@ Corso:Associate	Registrazione avvenuta con successo.	Lo studente è registrato nel sistema ed è abilitato.
1	Inserimento di una password malformata.	Nessuna	Nome:Mario Cognome:Rossi Email:mariorossi@gmail.com	Errore: i dati inseriti non sono corretti.	Lo studente non è registrato nel sistema.

			Password:Password Corso:Associate		
2	Inserimento di una email già in uso.	Esiste nel sistema un altro utente con stessa email.	Nome:Mario Cognome:Rossi Email:mariorossi@gmail.com Password:Password123@ Corso:Associate	Errore: email già in uso.	Lo studente non è registrato nel sistema.
3	Inserimento di un nome nullo.	Nessuna	Nome: null Cognome:Rossi Email:mariorossi@gmail.com Password:Password123@ Corso:Associate	Errore: i dati inseriti non sono corretti.	Lo studente non è registrato nel sistema.

- Login

Test Case ID	**Descrizione**	**Pre-condizioni**	**Input**	**Output Attesi**	**Post-condizioni Attese**
0	Inserimento di credenziali valide per lo studente.	Lo studente è registrato ed abilitato.	Email:mariorossi@gmail.com Password:Password123@	Login avvenuto con successo.	Lo studente è autenticato nel sistema.
1	Inserimento di credenziali non valide per lo studente.	Lo studente è registrato ed abilitato.	Email:mariorossi@gmail.com Password:Password	Credenziali invalide.	Lo studente non è autenticato nel sistema.
2	Inserimento di credenziali valide per lo studente.	Lo studente è registrato ma non è abilitato.	Email:mariorossi@gmail.com Password:Password123@	Credenziali invalide.	Lo studente non è autenticato nel sistema.
3	Inserimento di credenziali non valide per lo studente.	Lo studente è registrato ed abilitato.	Email:mario@gmail.com Password:Password123@	Credenziali invalide.	Lo studente non è autenticato nel sistema.
4	Inserimento di credenziali nulle per lo studente.	Lo studente è registrato ed abilitato.	Email:null Password:null	Credenziali invalide.	Lo studente non è autenticato nel sistema.

- Logout

Test Case ID	Descrizione	Pre-condizioni	Input	Output Attesi	Post-condizioni Attese
0	Logout di uno studente autenticato.	Lo studente è autenticato nel sistema.	Nessuno	Logout avvenuto con successo.	Lo studente non è più autenticato nel sistema.
1	Logout di uno studente non autenticato.	Lo studente non è autenticato nel sistema.	Nessuno	Logout avvenuto con successo.	Lo studente è ancora non autenticato nel sistema.

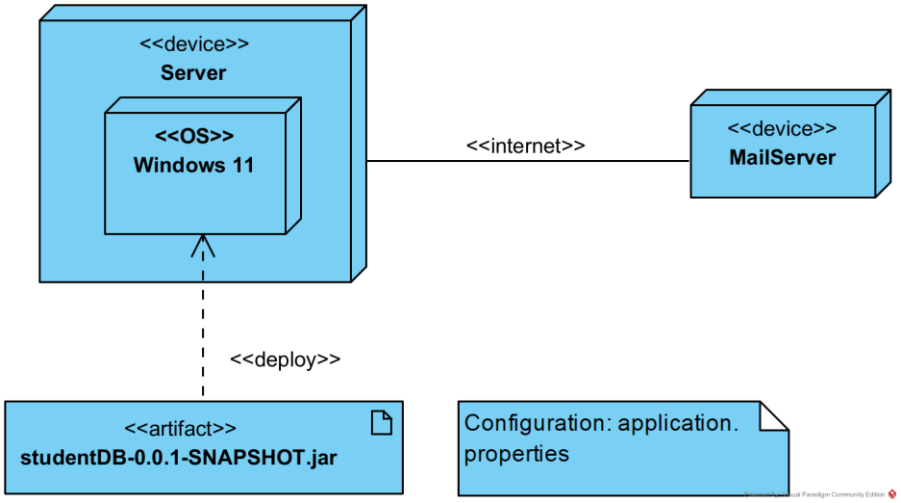
- Password dimenticata

Test Case ID	Descrizione	Pre-condizioni	Input	Output Attesi	Post-condizioni Attese
0	Lo studente inserisce la sua email e una nuova password valida.	Lo studente è registrato nel sistema ed è abilitato.	Email:mariorossi@gmail.com NewPassword:Password321@	Password cambiata con successo.	Lo studente è registrato ed abilitato nel sistema con la nuova password.
1	Lo studente inserisce la sua email ma una nuova password non valida.	Lo studente è registrato nel sistema ed è abilitato.	Email:mariorossi@gmail.com NewPassword:Password	Errore: password non valida.	Lo studente è registrato ed abilitato nel sistema ma la password non è stata cambiata.
2	Lo studente inserisce un'email che non esiste nel sistema.	Nessuna	Email:mario@gmail.com NewPassword:/	Errore: l'email inserita non esiste.	Lo studente non è registrato nel sistema.
3	Lo studente inserisce la sua email ma non ha ancora	Lo studente è registrato nel sistema ma non è abilitato.	Email:mariorossi@gmail.com NewPassword:/	Errore: l'email inserita non è stata ancora verificata.	Lo studente è registrato nel sistema ma non è abilitato.

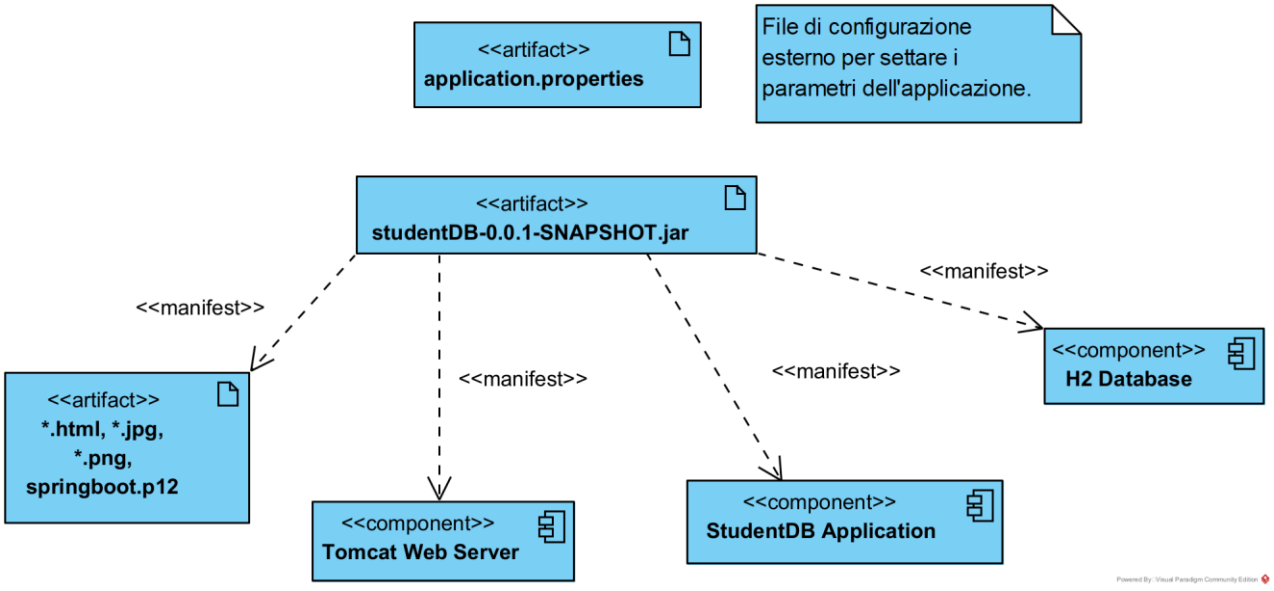
	verificato l'account.				
4	Lo studente inserisce una email nulla.	Nessuna	Email:null NewPassword:/	Errore: l'email inserita non esiste.	Lo studente non è registrato nel sistema.

DEPLOYMENT DIAGRAM & INSTALLATION VIEW

Forniamo il deployment diagram della nostra applicazione:



Questa invece la vista di installazione:



DETTAGLI IMPLEMENTATIVI

- Per gestire l'autorizzazione ai metodi dell'API sono stati definiti due ruoli: ADMIN e USER. Alcuni metodi dell'API sono pubblici, quindi tutti quelli relativi alle procedure di registrazione, login e cambio password. Ogni USER può accedere alla propria area riservata, in base al proprio Id, nella quale può avviare una nuova partita o visionare lo storico delle partite giocate. L'ADMIN, precaricato nel database all'avvio dell'applicazione (email: admin, password: pass), ha accesso ad alcuni metodi CRUD per gestire la repository degli studenti e alla console del database stesso. Le funzionalità fondamentali, ossia quelle di registrazione, login, logout e recupero della password, sono accessibili attraverso chiamate GET a /register, /login, /logout e /forgot-password. Dopo il login di default si accede all'area personale dell'utente che gli consente di avviare una nuova partita o visualizzare lo storico delle partite giocate. Se si volesse richiamare esplicitamente questa pagina si può effettuare una chiamata GET a /user/{userId}; ovviamente ogni utente può accedere solo alla propria area riservata.

- La gestione delle eccezioni è stata non sempre effettuata mediante try/catch perché il framework di Spring consente di dichiarare degli ExceptionHandler che gestiscono automaticamente le eccezioni a cui sono associati. Le classi relative alle eccezioni sono contenute in un package separato (exception).

- Avendo scelto un database in-memory per la persistenza dei dati, abbiamo cercato di evitare consumi di memoria causati da account registrati ma non abilitati. Al fine di risolvere tale problema abbiamo implementato un task periodico che ogni ora (xx:00) elimina dalla repository tutti gli studenti che non hanno verificato il proprio account da più di 8 ore; ciò è realizzato grazie ad un timestamp inizializzato al momento della registrazione dello studente (e settato a null se l'account viene abilitato).

- Per consentire al sistema di inviare le email di conferma dell'account e di cambio password agli studenti, abbiamo utilizzato tramite protocollo SMTP un'email @virgilio. I parametri di configurazione sono contenuti nel file application.properties fornito assieme al file jar. Se si volesse cambiare email ci si accerti di configurare opportunamente questi parametri.

- Per garantire la sicurezza dei dati comunicati via Web abbiamo reso possibile l'accesso al sistema via https. Abbiamo dunque creato un certificato (dal nome springboot.p12); tutti i parametri di configurazione sono sempre contenuti nel file application.properties. In particolare, è possibile specificare il numero di porto sul

quale avviare il server e se utilizzare https o http. Siccome si tratta di un self-signed certificate il browser mostra una pagina di warning ogni qualvolta si tenti di accedere al sistema mediante https, per cui se si volesse procedere in questa direzione servirebbe un certificato valido. Di default il server viene avviato con http sul porto 9000.

- Per l'interfacciamento con gli altri gruppi, sempre nel file di configurazione, è possibile specificare l'indirizzo dei loro componenti in modo da consentire la comunicazione.