

Indice

1	Premesse	2
1.1	Punto di partenza	2
1.1.1	Comprensione dell'architettura di riferimento	3
1.2	Checkpoint	21
1.2.1	Attori e ruoli	21
1.2.2	Revisione sommaria del task T1	23
1.2.3	Revisione sommaria del task T23	25
2	Analisi dei requisiti	26
2.1	Analisi dei casi d'uso	28
2.2	Analisi degli scenari dei casi d'uso	31
2.2.1	Scenari dei casi d'uso relativi il task T1	31
2.2.2	Scenari dei casi d'uso relativi i task T23	40
3	Progettazione	41
3.1	Sequence Diagrams relativi il task T1	41
3.2	Sequence Diagrams relativi i task T23	45
4	Implementazione	47
4.1	Extras	50
4.1.1	Aggiunta degli API endpoints	50
4.1.2	Automazione del processo di compilazione e building dei container mediante task in VSCode	50
5	Testing	55
6	Guida per sviluppatori in erba	60

Documentazione Testing-Game-SAD-2023/A13

Caterina Maria Accetto - M63/1117

1 Premesse

Il lavoro che andremo a presentare, si inserisce in un quadro più ampio rappresentato dal progetto ERASMUS, della durata triennale, denominato: **ENACTEST (European iNnovative AllianCe for TESTing educaTion)** [1] e sviluppato, in parte, dagli studenti della Federico II del corso di Software Architecture Design, dall'obiettivo estremamente ambizioso: valorizzare l'importanza del testing, disciplina spesso bistrattata e poco approfondita nell'ambito dei corsi universitari, attraverso l'innovativa strategia della *gamification* che, come suggerisce il termine, consiste nell'utilizzare elementi mutuati dai giochi ma in contesti non ludici [2]. Il risultato dell'applicazione di tale meccanismo, è stato la progettazione e conseguente sviluppo del gioco interattivo: **"Man vs Automated Testing Tools challenges"** che vede gli studenti, da qui in poi chiamati *players*, competere, a colpi di test progettati mediante il framework JUnit, contro dei robot (Randoop oppure EvoSuite) capaci di generare automaticamente tali test; la sfida può essere considerata vinta dal partecipante capace di portare a termine un certo obiettivo di copertura.

1.1 Punto di partenza

Trattandosi di un progetto complesso, stratificato, sviluppato da numerose persone, in costante aggiornamento ed evoluzione, è stato indispensabile poter far affidamento sulla popolare piattaforma *GitHub* [3] per garantire: **(a)** collaborazione tra diversi team di sviluppatori grazie alla possibilità di visionare in tempo reale le modifiche apportate al codice risolvendo, eventualmente, in modo efficace i conflitti, e **(b)** gestire in completa serenità la propria parte di lavoro assegnato.

Il lavoro che andremo ad esporre all'interno di tale documentazione, muove i suoi primi passi a partire dal progetto: A10-2024 [4] sviluppato appositamente per integrare i due seguenti requisiti:

ID	Descrizione
R5	Si vuole esporre l'applicativo su di un indirizzo IP configurabile, in modo che non funzioni solo su localhost.
R11	Si vuole prevedere per l'amministratore un meccanismo che dia accesso diretto a diverse funzionalità di interrogazione che prevedono la visualizzazione dell'elenco dei giocatori iscritti e delle classi disponibili.

Tabella 1: Requisiti assegnati nell'ambito del progetto A10-2024

Per una migliore comprensione, si rimanda alla lettura della documentazione reperibile al seguente indirizzo: <https://github.com/Testing-Game-SAD-2023/A10-2024/blob/main/Documentazione/Documentazione%20A10%20R5-R11.pdf>

1.1.1 Comprensione dell'architettura di riferimento

Solamente quando si è certi di aver compreso profondamente gli obiettivi del progetto e dell'applicazione che si sta tentando di sviluppare: un gioco interattivo che permetta a dei *players* di sfidare dei software di generazione automatica di test, si può pensare di analizzarne l'architettura, che, in questo caso specifico, è a **microservizi**, una scelta dettata dalla necessità di: **(a)** garantire una elevata agilità dei vari team di sviluppo che, in questo modo, hanno la possibilità di lavorare in completa autonomia e di concentrarsi sulla progettazione, sviluppo, testing e distribuzione di un piccolo servizio¹ alla volta, senza doversi interfacciare, o dover collaborare, con team che si occupano dello sviluppo di tutt'altri servizi in quanto, ciascuno di questi ultimi, completamente indipendente gli uni dagli altri, **(b)** aumentare la scalabilità complessiva del sistema grazie alla possibilità di scalare istanze dei singoli servizi e non istanze multiple dell'intera applicazione, **(c)** semplificare il rilascio e la manutenzione dei servizi in quanto indipendentemente "rilasciabili" ed aventi dimensioni ridotte ed in ultima battuta **(d)** permettere di sperimentare ed integrare facilmente nuove tecnologie all'interno del sistema.

Arrivati a questo punto, non ci resta che scoprire quali siano questi famigerati servizi che popolano l'applicazione sviluppata fino ad ora:

¹i servizi sono per definizione dei componenti software autonomi ed indipendenti che implementano alcune funzionalità utili, strettamente focalizzate e coese e presentano le seguenti caratteristiche: **(a)** sono dotati di APIs, che potrebbe venir definite come dei confini impermeabili estremamente difficili da sorpassare che consentono di comunicare con l'esterno nascondendo i dettagli implementativi, **(b)** sono lasciamente accoppiati (loosely coupled) ed una richiesta di questo tipo può venire soddisfatta facendo in modo che ciascun di questi servizi sia dotato di un proprio database, **(c)** dovrebbero venir sviluppati da team costituiti da pochi elementi garantendo, in questo modo, rapidi tempi di consegna ed una minima collaborazione ed interazione con tutti gli altri team, **(d)** dovrebbero essere progettati in maniera tale da essere facilmente assemblati, **(e)** dovrebbero garantire la proprietà di indipendenza dallo stato, nel senso che dovrebbero funzionare senza ricordare nulla del passato di tutti i client che li usano ed infine **(f)** dovrebbero essere caratterizzati da una propria architettura di visualizzazione logica, di forma esagonale, capace di disaccoppiare la logica di business, posizionata al centro, dal livello di presentazione e dal modulo di gestione dei dati e delle risorse.

ID	Descrizione sintetica	Descrizione dettagliata
T1	Servizio di gestione delle classi da testare e visualizzazione della classifica dei <i>players</i>	<p>Il servizio permette agli <i>amministratori</i> di:</p> <p>(a) registrarsi inserendo: nome, cognome, username e password, (b) effettuare la procedura di login.</p> <p>Il servizio permette agli <i>amministratori</i>, interagendo con un opportuno cruscotto, di:</p> <p>(a) caricare, modificare, ordinare, filtrare, ricercare, scaricare ed eliminare classi di programmazione da testare all'interno del catalogo, (b) visualizzare una classifica rudimentale dei <i>players</i>.</p> <p>Il servizio permette ai <i>players</i> correttamente autenticati ed in procinto di giocare, di: visualizzare tutte le classi precedentemente caricate dagli <i>amministratori</i>.</p>
T23	Servizio di autenticazione e registrazione dei <i>players</i>	<p>Il servizio permette ai <i>players</i> di: (a) registrarsi inserendo: nome, cognome, e-mail, password e specificando il corso di studi ('BSc', 'MSc', 'ALTRO'), (b) autenticarsi inserendo le proprie credenziali.</p> <p>Il servizio permette ai <i>players</i> correttamente registrati di: impostare una nuova password nel caso in cui l'avessero dimenticata.</p> <p>Il servizio permette ai <i>players</i> correttamente registrati ed autenticati di: (a) accedere all'area riservata di selezione dei parametri di gioco, (b) effettuare il logout.</p>

T4	Servizio di repository dei dati di gioco	Il servizio permette al <i>game engine</i> di: (a) creare una partita memorizzando una serie di informazioni, (b) aggiornarla per recuperare la data e l'ora di inizio e fine, (c) eliminarla, (d) generare, contestualmente la partita appena creata, i round, (e) aggiornarli, (f) creare, nell'ambito dei round, diversi turni associati a ciascun partecipante della partita, (g) aggiornarli (h) recuperare, durante ciascun round, i punteggi prodotti dai robot relativi la classe di test in gioco.
T5	Front-end avvio partita	Il servizio permette ai <i>players correttamente autenticati</i> di: (a) accedere all'area riservata di selezione dei parametri di gioco dove poter visualizzare le classi ed i robot disponibili precedentemente caricati dagli <i>amministratori</i> , (b) avviare una partita accedendo all'arena di gioco vero e proprio.
T6	Front-end per giocare una partita	Il servizio mette a disposizione un editor di test case in grado di: (a) fornire una finestra di editing testuale Java all'interno della quale il <i>player</i> correttamente autenticato avrà l'opportunità di scrivere codice ed eseguire le classiche operazioni di editing, richiedere la compilazione, visualizzare i risultati di copertura, (b) confrontare i risultati prodotti dal giocatore con quelli ottenuti dal robot e decretare un vincitore.
T7	Servizio di compilazione ed esecuzione	Il servizio permette di compilare ed eseguire i casi di test prodotti dal <i>player</i> in partita.
T8	Servizio robot EvoSuite	Il servizio permette di: (a) eseguire il robot EvoSuite su di una data classe Java e (b) restituire in output le informazioni relative l'esito dei test prodotti dal robot (informazioni di copertura, decisioni, ...).

T9	Servizio robot Randoop	Il servizio permette di: (a) eseguire il robot Randoop su di una data classe Java e (b) restituire in output le informazioni relative l'esito dei test prodotti dal robot (informazioni di copertura, decisioni, ...).
----	------------------------	--

Tabella 2: Descrizione dei servizi sviluppati fino a questo momento

Per quanto riguarda l'integrazione di tali servizi eterogenei, è stato deciso di fare affidamento sul **Gateway Pattern**, il cui scopo principale è proprio quello di fornire, non solo agli utilizzatori finali ma anche al sistema stesso, una interfaccia semplificata ed unificata per poter accedere a tutti i servizi messi a disposizione dal sistema nascondendo, all'esterno, i dettagli implementativi e la complessità costitutiva degli elementi sottostanti, in questo modo si fornisce un singolo punto d'accesso attraverso il quale comunicare e ricevere risposte e risultati.

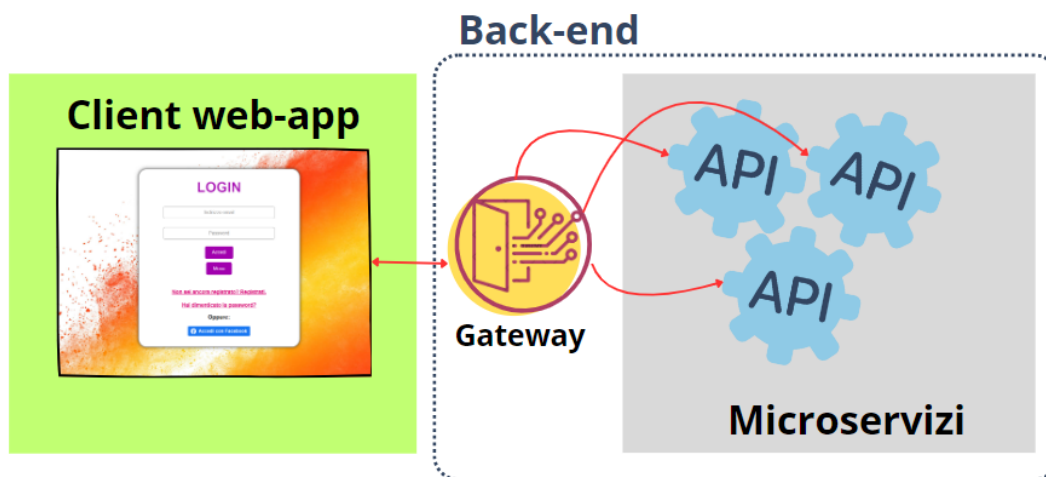


Figura 1: Gateway pattern

Come possiamo osservare in figura, il pattern si compone dei seguenti tre elementi fondamentali:

1. **Gateway:** "elemento di unione" il cui ruolo è quello di: **(a)** fornire una interfaccia unificata per poter accedere ai servizi messi a disposizione dal sistema che si intende incapsulare, **(b)** gestire le richieste provenienti dal *Client* ed infine **(c)** eseguire tutte le operazioni necessarie a restituire i risultati al *Client*.
2. **Client:** entità che interagisce sfruttando i servizi messi a disposizione dal *Gateway* rimanendo sempre inconsapevole della complessità, della struttura e dei dettagli implementativi del sistema sottostante.
3. **Sistema sottostante:** un sistema oppure una collezione di servizi eterogenei che si intende isolare agli "occhi" del *Client* mediante incapsulazione tramite il *Gateway*.

/home_admin					
GET	description	operationID	parameters	RequestBody	response
	Restituisce il cruscotto dell' admin	showHomeAdmin			"200": pagina di registrazione visualizzata correttamente
/registraAdmin					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la pagina di registrazione dell' amministratore	showRegistraAdmin			"200": pagina di registrazione visualizzata correttamente
/modificaClasse					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la pagina di modifica delle classi da testare	showModificaClasse			"200": pagina di modifica delle classi visualizzata correttamente.
/uploadClasse					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la pagina di caricamento delle classi da testare	showUploadClasse			"200": pagina di caricamento delle classi visualizzata correttamente.
/uploadClasseAndTest					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la pagina di caricamento delle classi da testare e dei relativi test prodotti dai robot EvoSuite e Randoop	showUploadClasseAndTest			"200": pagina di caricamento delle classi e dei test prodotti dai robot visualizzata correttamente
/reportClasse					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la pagina di visualizzazione dei report riguardanti le varie classi caricate dagli amministratori	showReportClasse			"200": pagina di reportistica delle classi visualizzata correttamente
/Reports					
GET	description	operationID	parameters	RequestBody	response
	Restituisce una pagina generale di reportistica	showReports			"200": pagina di reportistica visualizzata correttamente
/interaction					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la lista di tutte le interazioni	elencaInt			"200": lista delle interazioni restituita correttamente
/findreport					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la lista di tutti i report	elencaReport			"200": lista dei report restituita correttamente
/getLikes/{name}					
GET	description	operationID	parameters	RequestBody	response
	Restituisce il numero di "likes" guadagnati da una classe specifica	likes	name		"200": numero di likes ottenuti correttamente per la classe specificata
/newinteraction					
POST	description	operationID	parameters	RequestBody	response
	Carica una nuova interazione all' interno del sistema	UploadInteraction		required: true content: interazione	"200": interazione caricata correttamente
/newlike/{name}					
POST	description	operationID	parameters	RequestBody	response
	Aggiunge un nuovo like per la classe specificata	newLike	name		"200": nuovo like aggiunto correttamente per la classe specificata
/newReport/{name}					
POST	description	operationID	parameters	RequestBody	response
	Aggiunge un nuovo report per la classe specificata	newReport	name	required: true content: commento	"200": nuovo report aggiunto correttamente per la classe specificata
/deleteint/{id_i}					
POST	description	operationID	parameters	RequestBody	response
	Elimina una specifica interazione dal sistema	eliminaInteraction	id_i		"200": interazione eliminata correttamente dal database
/home					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la lista di tutte le classi caricate precedentemente dagli amministratori	elencaClassi			"200": lista delle classi ottenuta correttamente
/orderbydate					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la lista delle classi caricate ordinate in base alla data	ordinaClassi			"200": lista delle classi ordinate per data ottenuta correttamente
/orderbyname					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la lista delle classi caricate ordinate per nome	ordinaClassiNome			"200": lista delle classi ordinate per nome ottenuta correttamente

/Cfilterby/{category}					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la lista delle classi filtrate in base alla categoria	filtraClassi	category		"200": lista delle classi filtrate per categoria ottenuta correttamente
/Cfilterby/{text}/{category}					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la lista delle classi filtrate per nome e categoria	filtraClassi	text, category		"200": lista delle classi filtrate per nome e categoria ottenuta correttamente
/Dfilterby/{difficulty}					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la lista delle classi filtrate per difficoltà	elencaClassiID	difficulty		"200": lista delle classi filtrate per difficoltà ottenuta correttamente
/Dfilterby/{text}/{difficulty}					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la lista delle classi filtrate per nome e difficoltà	elencaClassiID	text, difficulty		"200": lista delle classi filtrate per nome e difficoltà ottenuta correttamente
/insert					
	description	operationID	parameters	RequestBody	response
POST	Carica una nuova classe	uploadClasse		required: true content: ClassUT	"200": classe caricata correttamente
/uploadFile					
	description	operationID	parameters	RequestBody	response
POST	Carica una nuova classe nel filesystem condiviso	uploadFile		required: true content: classFile, (file della classe da caricare), model (stringa JSON che rappresenta i metadati della classe)	"200": classe caricata correttamente
/uploadTest					
	description	operationID	parameters	RequestBody	response
POST	Carica una nuova classe e genera e salva i test prodotti dai robot nel filesystem condiviso	uploadTest		required: true content: classFile, model, testFile (file relativo i test generati da Randoop), testFileEvo (file relativo i test generati da EvoSuite)	"200": classe e relativi test caricati correttamente
/delete/{name}					
	description	operationID	parameters	RequestBody	response
POST	Elimina una specifica classe	eliminaClasse	name		"200": classe eliminata correttamente
/deleteFile/{fileName}					
	description	operationID	parameters	RequestBody	response
POST	Elimina una specifica cartella	eliminaFile	fileName		"200": cartella eliminata correttamente
/home/{text}					
	description	operationID	parameters	RequestBody	response
GET	Ricerca classi per nome	ricercaClasse	text		"200": lista delle classi richieste ottenute correttamente
/downloadFile/{name}					
	description	operationID	parameters	RequestBody	response
GET	Scarica una specifica classe	downloadClasse	name		"200": classe scaricata correttamente
/update/{name}					
	description	operationID	parameters	RequestBody	response
POST	Aggiorna i dati associati ad una specifica classe (nome, data di caricamento, difficoltà, descrizione, categoria)	modificaClasse	name	required: true content: ClassUT	"200": classe modificata correttamente
/registraAdmin					
	description	operationID	parameters	RequestBody	response
POST	Registra un nuovo amministratore specificando: nome, cognome, username e password	registraAdmin		required: true content: Admin	"200": amministratore registrato correttamente
/loginAdmin					
	description	operationID	parameters	RequestBody	response
POST	Autentica un amministratore precedentemente registrato	loginAdmin		required: true content: Admin	"200": amministratore "loggato" correttamente
GET	Restituisce la pagina di login riservata agli amministratori	showLoginForm			"200": pagina di login riservata agli amministratori visualizzata correttamente
/player					
	description	operationID	parameters	RequestBody	response
GET	Restituisce una classifica rudimentale di tutti i players che si sono cimentati a sfidare i robot	showplayer			"200": classifica dei players visualizzata correttamente

/class					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la pagina di gestione delle classi da testare	showclass			"200": pagina di gestione delle classi visualizzata correttamente

Tabella 4: Descrizione degli **API endpoints** relativi il servizio T1

/register					
	description	operationID	parameters	RequestBody	response
POST	Registra un nuovo player specificando: nome, cognome, e-mail, password e corso di studi inviando una mail di riepilogo se il token JWT è invalido	register		required: true content: name, surname, e-mail, password, check_password, studies	"302": re-indirizzamento alla pagina di successo (/login.success) "500": "Already logged in" (token JWT valido) "500": "Failed to confirm your registration"
GET	Restituisce la pagina di registrazione dei players se il token JWT è invalido	showRegistrationForm			"302": re-indirizzamento alla pagina di registrazione dei players (/register) "302": re-indirizzamento all'arena di gioco (/main) (token JWT valido)
/login					
	description	operationID	parameters	RequestBody	response
POST	Autentica un player precedentemente registrato se il token JWT è invalido	login		required: true content: email, password	"302": re-indirizzamento all'arena di gioco (/main) "500": "Already logged in" (token JWT valido) "401": "Email not found" (e-mail non trovata nel database) oppure "401": "Incorrect password" (password sbagliata)
GET	Restituisce la pagina di login del player se il token JWT è invalido	showLoginForm			"302": re-indirizzamento alla pagina di login dei players (/login) "302": re-indirizzamento all'arena di gioco (/main) (token JWT valido)
/logout					
	description	operationID	parameters	RequestBody	response
POST	Cancella i cookies associati alla sessione di quello specifico utente autenticato se il token di autenticazione è valido	logout		required: true content: authToken	"200": "Logout successful" "401": "User not authenticated"
GET	Rimozione token di autenticazione	logout			"302": re-indirizzamento alla pagina di login del player (/login)
/password_reset					
	description	operationID	parameters	RequestBody	response
POST	Richiesta recupero password da parte del player precedentemente registrato se il token JWT è invalido	resetPassword		required: true content: password	"200": "Password reset mail sent successfully" (token JWT invalido + player registrato) "500": "Already logged in" (token JWT valido) "500": "Failed to send password reset mail" "400": "Email not found" (e-mail non trovata nel database)
GET	Restituisce la pagina di recupero password se il token JWT è invalido	showResetForm			"302": re-indirizzamento alla pagina recupero della password (/password_reset) "302": re-indirizzamento all'arena di gioco (/main) (token JWT valido)
/password_change					
	description	operationID	parameters	RequestBody	response
POST	Inserimento di una nuova password da parte del player precedentemente registrato se il token di reset è valido	changePassword		required: true content: email, resetToken, newPassword, confirmPassword	"200": "Password change successful" "500": "Already logged in" (token JWT valido) "401": "Email not found" (e-mail non trovata nel database) "401": "Invalid reset token" (resetToken invalido) "400": "Password not valid" (password non rispetta i requisiti) "400": "Check_password not valid" (le password non combaciano)
GET	Restituisce la pagina di modifica della password se il token JWT è invalido	showChangeForm			"302": re-indirizzamento alla pagina di modifica della password (/password_change) "302": re-indirizzamento all'arena di gioco (/main) (token JWT valido)

/validateToken					
	description	operationID	parameters	RequestBody	response
POST	Verifica token JWT	checkValidityToken		required: true content: jwt	isJwtValid(jwt) = true, se il token è valido altrimenti isJwtValid(jwt) = false in caso contrario
/students_list					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la lista di tutti i players registrati all'interno del sistema	getAllStudents			"200": lista di tutti i players precedentemente registrati
/mail_register					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la pagina per richiedere l'invio di una nuova mail di conferma di avvenuta registrazione se il token JWT è invalido	showMailForm			"302": re-indirizzamento all'arena di gioco (/main) (token JWT valido) "302": re-indirizzamento alla pagina di richiesta di invio di una nuova mail di conferma di avvenuta registrazione

Tabella 5: Descrizione degli **API endpoints** relativi il servizio T23

/games/{id}					
	description	operationID	parameters	RequestBody	response
GET	Restituisce una specifica partita	FindById	id (identificativo della partita)		"200": specifica partita richiesta "400": "Bad request" "404": "No game found for the provided 'Id'" (identificativo non trovato nel database) "429": "Too many requests" "500": "Internal server error"
PUT	Aggiorna le informazioni riguardanti una specifica partita attraverso la possibilità di modificare: nome, round corrente, descrizione, inizio e fine	Update	id (identificativo della partita)	required: true name, currentRound, description, startedAt, closedAt	"200": specifica partita aggiornata correttamente "400": "Bad request" "404": "No game found for the provided 'Id'" (identificativo non trovato nel database) "413": "Request body too large" "429": "Too mant requests" "500": "Internal server error"
DELETE	Elimina una specifica partita	Delete	id (identificativo della partita)		"204": "Game deleted" "404": "No game found for the provided 'Id'" (identificativo non trovato nel database) "429": "Too many requests" "500": "Internal server error"
/games					
	description	operationID	parameters	RequestBody	response
POST	Creazione di una nuova partita specificando tutte le informazioni necessarie a descriverla (nome, difficoltà, descrizione, ...) ma specificando anche tutti i partecipanti coinvolti	Create		required: true content: players, name, description, difficulty, startedAt, closedAt	"201": partita creata correttamente "400": "Bad request" "413": "Request body too large" "429": "Too many requests" "500": "Internal server error"
GET	Recupera una partita giocata in un certo intervallo temporale	FindByInterval	required: false startDate, endDate, page (numero della pagina da recuperare) pageSize (elementi per pagina) accountId (id del player)		"200": partita ricercata in un certo intervallo temporale correttamente trovata e restituita "400": "Bad request" "404": "No game found for the provided 'Id'" "429": "Too many requests" "500": "Internal server error"
/rounds/{id}					
	description	operationID	parameters	RequestBody	response
GET	Restituisce uno specifico round	FindById	id (identificativo del round)		"200": "The round corresponding to the provided 'Id'" "400": "Bad request" "404": "No round found for the provided 'Id'" "429": "Too many requests" "500": "Internal server error"
PUT	Aggiorna le informazioni riguardanti la date e l'ora di inizio e fine di uno specifico round	Update	id (identificativo del round)	required: true content: startedAt, closedAt	"200": round aggiornato correttamente "400": "Bad request" "404": "No round found for the provided 'Id'" "413": "Request body too large" "429": "Too many requests" "500": "Internal server error"
DELETE	Elimina uno specifico round	Delete	id (identificativo del round)		"204": "Round deleted" "400": "Bad request" "404": "No round found for the provided 'Id'" "429": "Too many requests" "500": "Internal server error"

/rounds					
	description	operationID	parameters	RequestBody	response
POST	Creazione di un nuovo round specificando tutte le informazioni necessarie a descriverlo (identificativo della partita di riferimento, nome della classe sotto test, ...)	Create		required: true content: gameId, testClassId, order, startedAt, closedAt,	"201": "Created" "400": "Bad request" "413": "Request body too large" "429": "Too many requests" "500": "Internal server error"
GET	Restituisce tutti i round associati ad una specifica partita	FindById	gameId (identificativo della partita)		"200": lista tutti i round associati ad una specifica partita restituita correttamente "404": "No game found for the provided 'Id'" "400": "Invalid game id" "429": "Too many requests" "500": "Internal server error"
/turns/{id}					
	description	operationID	parameters	RequestBody	response
GET	Restituisce uno specifico turno	FindById	id (identificativo del turno)		"200": "The turn corresponding to the provided 'Id'" "400": "Bad request" "404": "No turn found for the provided 'Id'" "429": "Too many requests" "500": "Internal server error"
PUT	Aggiorna le informazioni riguardanti i punteggi, il vincitore e la data e l'ora di inizio e fine turno	Update	id (identificativo del turno)	required: true content: scores, isWinner, startedAt, closedAt	"200": specifico turno aggiornato correttamente "400": "Bad request" "404": "Not found" "413": "Request body too large" "429": "Too many requests" "500": "Internal server error"
DELETE	Elimina uno specifico turno	Delete	id (identificativo del turno)		"200": specifico turno eliminato correttamente "400": "Bad request" "404": "No turn found for the provided 'Id'" "429": "Too many requests" "500": "Internal server error"
/turn/{id}/files					
	description	operationID	parameters	RequestBody	response
PUT	Carica un turno a partire da uno zip file	SaveFile	id (identificativo del turno)	required: true content; file	"200": file caricato correttamente relativo uno specifico turno "400": "Bad request" "404": "No turn found for the provided 'Id'" "413": "Request body too large" "429": "Too many requests" "500": "Internal server error"
GET	Scarica uno specifico turno come uno zip file	GetFile	id (identificativo del turno)		"200": file scaricato correttamente relativo uno specifico turno "400": "Bad request" "404": "No turn found for the provided 'Id'" "429": "Too many requests" "500": "Internal server error"
/turns					
	description	operationID	parameters	RequestBody	response
GET	Restituisce tutti i turni associati ad uno specifico round	FindByRound	roundId (identificativo del round)		"200": lista di tutti i turni associati ad uno specifico round restituita correttamente "400": "Invalid round id" "404": "No round found for the provided 'Id'" "413": "Request body too large" "429": "Too many requests" "500": "Internal server error"
POST	Creazione nuovo turno associato ad uno specifico round ed ad uno specifico giocatore	CreateBulk		required: true content: players, roundId, startedAt, closedAt	"201": creazione turno avvenuta correttamente "400": "Bad request" "404": "No round or player found for the provided 'Id'" "409": "User turn already exists in round" "413": "Request body too large" "429": "Too many requests" "500": "Internal server error"

/robots					
	description	operationID	parameters	RequestBody	response
GET	Restituisce i risultati prodotti filtrandoli in base al nome della classe, alla sua difficoltà ed al robot prescelto contro cui si è scelto di giocare	FindByFilter	required: true testClassId (identificativo della classe da testare), difficoltà, type (tipologia del test engine: Randoop oppure EvoSuite)		"200": lista dei risultati prodotti dallo specifico test engine richiesto restituita correttamente "400": "Invalid parameters" "404": "No results found for the provided 'Id'" "429": "Too many requests" "500": "Internal server error"
DELETE	Elimina i risultati prodotti dai robot rispetto una specifica classe da testare	DeleteByTestClass	required: true testClassId (identificativo della classe da testare)		"200": risultati prodotti per quella specifica classe eliminati correttamente "400": "Invalid parameters" "404": "No results found for the provided 'Id'" "429": "Too many requests" "500": "Internal server error"
POST	Creazione dei risultati di test prodotti dal robot fornendo in batch tutte le informazioni necessarie	CreateBulk		required: true, content; robots, difficulty, type, scores, testClassId	"201": creazione dei risultati prodotti dal robot avvenuta correttamente "400": "Invalid parameters" "413": "Request body too large" "429": "Too many requests" "500": "Invalid server error"

Tabella 6: Descrizione degli **API endpoints** relativi il servizio T4

/main					
	description	operationID	parameters	RequestBody	response
GET	Restituisce l'arena di gioco se il token JWT associato al player è valido	GUIController			"302": re-indirizzamento all'arena di gioco "302": re-indirizzamento alla pagina di login del player (/login) (token JWT invalido)
/report					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la pagina riepilogativa delle scelte di gioco (classe che si è deciso di testare e robot che si è deciso di sfidare) se il token JWT associato al player è valido	reportPage			"200": pagina riepilogativa delle scelte di gioco visualizzata correttamente "302": re-indirizzamento alla pagina di login del player (/login) (token JWT invalido)
/save-data					
	description	operationID	parameters	RequestBody	response
POST	Salva le informazioni di selezione riguardanti la partita che si intende giocare (classe da testare, difficoltà e robot da sfidare) impostate dal player se il token JWT è valido	saveGame		required: true, content; playerId, robot, classe (classe da testare), difficulty	"200": dati salvati correttamente "400": "Unauthorized" (token JWT invalido) "400": "Bad request" (parametri invalidi)
/editor					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la pagina di editing dove il player ha la possibilità di scrivere i casi di test rispetto la specifica classe selezionata e confrontare i risultati ottenuti dal robot prescelto se il token JWT è valido	editorPage			"200": pagina di editing dei casi di test visualizzata correttamente (/editor) "302": re-indirizzamento alla pagina di login del player (/login) (token JWT invalido)

Tabella 7: Descrizione degli **API endpoints** relativi il servizio T5

GET	description	operationID	parameters	RequestBody	response
	Restituisce la pagina di indice	indexPath			"200": pagina di indice visualizzata correttamente
/receiveClassUnderTest					
GET	description	operationID	parameters	RequestBody	response
	Restituisce la classe che il player attualmente in partita intende testare [interazione con l' API endpoint: /downloadFile del servizio T1]	receiveClassUnderTest	idUsernte, idPartita, idTurno, nomeCUT, robotScelto, difficoltà		"200": classe da testare ricevuta correttamente "500": "Errore durante la ricezione del file ClassUnderTest.java"
/sendInfo					
POST	description	operationID	parameters	RequestBody	response
	Invio del codice prodotto dal player e dei test generati dal robot al servizio di compilazione [interazione con l'API endpoint: /compile-and-codecoverage del servizio T7]	handleSendInfoRequest		required: true, content:testingClassName, testingClassCode, underTestClassName, underTestClassCode	"200": dati inviati correttamente "500": "Internal server error"
/run					
POST	description	operationID	parameters	RequestBody	response
	Restituzione al player in partita dei risultati di copertura e del punteggio ottenuto dal robot [interazione con l' API endpoint: /compile-and-codecoverage del servizio T7 per ottenere i risultati di copertura e di compilazione ed interazione con l'API ednpoint: /robots del servizio T4 per ottenere punteggi dei robot e con l'endpoint: /turns, /rounds e /games, dello stesso servizio, per aggiornare e chiudere la partita decretando un vincitore]	runner		required: true, content:testingClassName, testingClassCode, underTestClassName, underTestClassCode, testClassId, type, difficulty, turnId, roundId, gameId	"200": esecuzione avvenuta con successo "500": "Errore in compilecodecoverage" (se lo status code è diverso da 2xx) "500": "Errore in robots" (se lo status code è diverso da 2xx) "500": "Errore in put turn" (se lo status code è diverso da 2xx) "500": "Errore in put round" (se lo status code è diverso da 2xx) "500": "Errore in put games" (se lo status code è diverso da 2xx)
/getJaCoCoReport					
POST	description	operationID	parameters	RequestBody	response
	Restituzione del report di copertura del codice generato da JaCoCo [interazione con l'API endpoint: /compile-and-codecoverage del servizio T7 per ottenere i risultati di copertura]	getJaCoCoReport		required: true, content:testingClassName, testingClassCode, underTestClassName, underTestClassCode	"200": report di copertura prodotto da JaCoCo restituito correttamente "500": "Errore compilazione" (se lo status code è diverso da 2xx)

Tabella 8: Descrizione degli **API endpoints** relativi il servizio T6

/compile-and-codecoverage					
POST	description	operationID	parameters	RequestBody	response
	Compilazione dei due files Java ricevuti in ingresso (codice prodotto dal giocatore e test prodotti dal robot), e restituzione dei risultati di copertura prodotti da JaCoCo	compileAndTest		required: true content: RequestDTO (Data Transfer Object)	"200": risultati ottenuti correttamente

Tabella 9: Descrizione degli **API endpoints** relativi il servizio T7

Conclusa questa rapida descrizione di tutti gli endpoints presenti all'interno dell'applicazione fino a questo momento, potrebbe essere interessante domandarsi perchè questi ultimi ricoprano un ruolo così rilevante e da non sottovalutare: **(a)** innanzitutto consentono, molto banalmente, di stabilire la posizione esatta delle risorse all'interno delle API, **(b)** aiutano gli sviluppatori non solo ad organizzare tutte queste risorse, ma anche **(c)** a controllarne gli accessi da parte dei consumatori incrementando, in questo modo, il livello di sicurezza complessivo del sistema; poichè la sicurezza non è mai abbastanza, e tenuto conto del fatto che l'applicazione è stata progettata per poter essere distribuita sulla rete e quindi venir impiegata, in un prossimo futuro, da studenti volenterosi di cimentarsi nella scrittura di test d'unità migliorando, di fatto, le proprie abilità e conoscenze in fatto di testing, è lecito domandarsi **come provare a proteggere gli API endpoints**:

1. Hashing delle password One-Way

Sappiamo bene, che uno dei metodi per garantire la sicurezza degli account e fornire funzionalità di autenticazione degli accessi, è quello di impiegare una **password** ma questo non basta: è indispensabile **proteggere la memorizzazione delle password di tutti gli utilizzatori dell'applicazione, all'interno dei sistemi di archiviazione**; una buona strategia potrebbe essere quella di crittare tutte queste password prima di memorizzarle, in questo modo, all'interno del database, verranno salvate delle versioni criptate inviolabili persino agli stessi amministratori dei server, i quali non avranno la possibilità di decifrarle mentre, per quanto riguarda l'autenticazione degli utenti, si andranno a codificare le password appena inserite con la stessa funzione di hashing impiegata in fase di registrazione così da poterle confrontare con le versioni memorizzate all'interno del database e di conseguenza verificare la correttezza degli inserimenti. Da un punto di vista pratico, il meccanismo impiegato, durante la fase di registrazione dei *players*, per garantire che il livello di sicurezza precedentemente descritto venga raggiunto, è quello di fare affidamento alla classe: `BCryptPasswordEncoder` fornita dal framework Spring Security.



Attenzione!

Un identico trattamento di protezione e controllo degli accessi mediante procedura di **password encoding** dovrebbe venir riservato, in fase di registrazione, anche agli stessi *amministratori* del sistema che però, in questa attuale versione dell'applicazione, ne risultano essere sguarniti.

2. Protocollo HTTPS

Per garantire: **(a) sicurezza**, nel senso che HTTPS crittografa i dati scambiati tra il browser degli utenti ed il web server in modo da proteggere le informazioni sensibili dei clients, si pensi a password e chiavi segrete, per scongiurare possibili attacchi man-in-the-middle o per proteggersi dai metodi di sniffing dei pacchetti, **(b) affidabilità ed autenticità** delle comunicazioni in rete, bisognerebbe prendere fortemente in considerazione la possibilità di migrare verso il protocollo HTTPS.



Attenzione!

Attualmente, la comunicazione e distribuzione in rete dell'applicazione, così come tutte le chiamate ai vari microservizi, avvengono per mezzo del protocollo HTTP.

3. Misure di autenticazione delle API

Tutte le volte che si progettano, sviluppano e distribuiscono applicazioni rivolte ad un pubblico, è necessario implementare delle adeguate misure di autenticazione ed autorizzazione per prevenire, non solo l'uso improprio, ma anche l'abuso dei servizi messi a disposizione. La soluzione che si è deciso di adottare, nell'ambito dello sviluppo di questa applicazione web, è quella che consiste nel fare affidamento sui **JSON Web Token**, informalmente chiamati: token JWT, un metodo standardizzato [6] per rappresentare in modo sicuro l'identità dei *players* quando si comunica in rete senza la necessità di dover interrogare il database o memorizzare lo stato dei giocatori all'interno del server (**autenticazione stateless**).



Attenzione!

Una identica misura di autenticazione ed autorizzazione mediante **token JWT** dovrebbe venir riservata per rappresentare l'identità degli *amministratori*.

4. Filtro degli indirizzi IP

Un ulteriore livello di sicurezza che si potrebbe pensare di introdurre, soprattutto in questa fase delicata di sviluppo, potrebbe essere quello di limitare gli indirizzi IP in grado di accedere alle API in base alla posizione, operando in questo modo si scongiura, soprattutto, il rischio del verificarsi di eventuali cyber-attacchi, i quali, nella maggior parte dei casi, provengono da un numero limitato di paesi.



Suggerimento per future integrazioni

Trattandosi di un progetto ERASMUS che vede il coinvolgimento di nove partner provenienti da diversi paesi europei (Italia, Spagna, Portogallo, Svezia e Belgio), potrebbe essere utile limitare, almeno momentaneamente, gli accessi ai soli indirizzi IP appartenenti a queste "sedi consentite".

Lo step successivo, dopo aver capito come proteggere i propri endpoints, è quello di monitorarli attraverso:

1. Documentazione degli API endpoints

Documentare gli API endpoints significa descrivere, in maniera più o meno completa ed accurata, quali servizi offre una Application Programming Interface e come utilizzarli.

Produrre una documentazione di qualità consente di:

- **Facilitare l'uso dell'API:** grazie a documenti di questo tipo, gli sviluppatori hanno la possibilità di comprendere in maniera adeguata e dettagliata come utilizzare tutti gli endpoints messi a disposizione, semplificando notevolmente la loro integrazione all'interno dell'applicazione.
- **Ridurre il tempo di sviluppo:** una documentazione chiara e completa consente agli sviluppatori di comprendere rapidamente come interagire con l'API e scrivere, di conseguenza, codice più rapidamente.
- **Facilitare la manutenzione:** una documentazione ben mantenuta garantisce che tutti gli sviluppatori (in generale tutte le persone coinvolte nel progetto) siano aggiornati circa le modifiche apportate sugli API endpoints e sui requisiti di utilizzo.
- **Supportare la scalabilità:** una buona documentazione semplifica il coinvolgimento di nuovi sviluppatori all'interno del progetto, i quali non avranno difficoltà ad individuare le informazioni necessarie per comprendere quali sono le funzionalità e le capacità delle API sulle quali dovranno lavorare.
- **Facilitare la risoluzione dei problemi:** solitamente, documentazioni di questo tipo, dovrebbero prevedere, per ciascuna chiamata, una sezione appositamente dedicata a descriverne i risultati e relativi codici errori, in questo modo, non solo si semplifica il processo di diagnostica, ma si velocizza anche il processo di risoluzione di eventuali problemi.

Suggerimento per future integrazioni

Trattandosi di una applicazione complessa, di grandi dimensioni, fortemente eterogenea, sviluppata da numerose persone e che in un prossimo futuro necessiterà di venir rilasciata presso gli altri partner europei coinvolti nel progetto, potrebbe essere una buona idea procedere alla documentazione sistematica, facendo magari affidamento allo standard **OpenAPI**, di tutti gli **API endpoints** associati ai servizi attualmente forniti dall'applicazione.

Attenzione!

Allo stato attuale, solamente le REST API relative il task T4 (servizio di repository dei dati di gioco), sono state documentate approfonditamente mediante il framework open-source noto come: **Swagger**:

Listing 1: Snippet di codice YAML (index.yaml) della documentazione delle REST API relative il task T4

```
openapi: "3.0.0"
info:
  version: "0.0.1"
  title: "Game Repository API"
  description: Game Repository REST API

servers:
  ...

paths:
  /games/{id}:
    parameters:
      - name: id
        description: Game identifier
        in: path
        required: true
        schema:
          type: integer
          format: int64
    get:
      summary: Retrieve a game by id
      description: Retrieve a game by id
      tags:
        - games
      responses:
        "200":
          description: The game corresponding to the 'Id'
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Game"
        "400":
          ...
```

2. Implementazione di test di regressione

Il compito dei **test di regressione** è quello di verificare se le modifiche apportate al codice sorgente di una applicazione abbiano o meno introdotto nuovi errori o alterato il normale e corretto funzionamento delle funzionalità preesistenti, infatti, nel contesto dello sviluppo software, il termine "regressione" si riferisce proprio a quella spiacevole situazione per la quale errori o problemi precedentemente risolti si ripresenteranno ugualmente a seguito delle recenti modifiche apportate al software.

Come era lecito aspettarsi, è possibile eseguire manualmente test di questo tipo oppure procedere alla loro automazione, con il vantaggio, in questo ultimo caso, non solo di velocizzare l'intero processo di esecuzione dei test ma soprattutto di abilitare con successo, all'interno del proprio processo di sviluppo software, le pipeline di CI/CD (Continuous Integration/Continuous Deployment) consentendo, in questo modo, un rilascio più rapido, affidabile e di alta qualità del software prodotto.

Suggerimento per future integrazioni

Avendo mostrato, da un lato: l'importanza ricoperta dagli **API endpoints**, e dall'altro: il ruolo cruciale svolto dai **test di regressione**, appare evidente che per garantire un corretto rilascio dell'applicativo ed evitare che le modifiche apportate ai vari servizi introducano problemi inaspettati, potrebbe essere vantaggioso procedere allo sviluppo di appositi test automatici di regressione rivolti a tutti gli API endpoints (ricordiamo che **Swagger** non è uno strumento di test ma offre degli utili meccanismi per validare le richieste inviate alle API e le risposte ricevute per verificare che queste ultime siano conformi alle specifiche) e solo successivamente valutare se introdurre, all'interno del processo di sviluppo software, delle pipeline strutturate di CI/CD sfruttando, ad esempio, programmi quali **Jenkins**, **GitLab** oppure **Azure DevOps**.

3. Monitoraggio delle dipendenze

Sebbene in una architettura a microservizi uno dei vincoli cardine della progettazione prescriva che i servizi siano completamente indipendenti gli uni con gli altri, è possibile, tuttavia, che le API in-

teragiscano tra di loro oppure abbiano bisogno di consumare API sviluppate da terze parti, per cui è indispensabile procedere al monitoraggio delle loro dipendenze, se presenti; un passaggio di questo tipo è fondamentale per essere in grado di **gestire adeguatamente i cambiamenti all'infrastruttura ed alle stesse API**, mitigando eventuali rischi associati alle modifiche e valutare anche il loro impatto all'interno del sistema.



Attenzione!

Alcuni API endpoints appartenenti al task T1, ne consumeranno altri provenienti dal task T5 i quali, a loro volta, verranno utilizzati dal servizio di autenticazione e registrazione dei *players* (task T23) creando, oltre a molta confusione, una serie di conflitti; la situazione si andrà a complicare ulteriormente a seguito dell'aggiunta dei token JWT di autenticazione ed autorizzazione dell'identità degli *amministratori*.

L'effetto più evidente dell'introduzione dei token JWT, lato *amministratore*, è quello di realizzare implicitamente, all'interno dell'architettura, un ulteriore "confine di sicurezza" allo scopo di distinguere i giocatori dagli admins, quindi, per garantire la validità dei token ed autorizzare l'attore corretto nell'utilizzo di un certo servizio, è indispensabile mantenere sempre separati queste regioni onde evitare conflitti (**una trattazione più approfondita verrà affrontata nei capitoli successivi**).

1.2 Checkpoint

Questa sezione vuole fungere da "punto di controllo", un luogo dove poter tirare le somme, schiarirsi le idee ed analizzare in maniera critica il lavoro che è stato svolto sull'applicazione fino a questo momento, non solo per evidenziarne le criticità ma soprattutto per **(a)** comprenderlo profondamente e di conseguenza sollevare dubbi sulle scelte progettuali o implementative, accennando in questo modo, per i futuri sviluppatori, percorsi da intraprendere e completare oppure scartare del tutto una volta fatte delle attente valutazioni, e **(b)** suggerire una serie di azioni migliorative, da sviluppare, magari, nelle future iterazioni.

Un altro motivo altrettanto importante e che giustifica la presenza di un sotto-paragrafo di questo tipo, è quello di permettere al lettore di questa documentazione di capire in che modo si andrà ad inserire il lavoro di progettazione, sviluppo ed implementazione assegnatomi e che verrà descritto nei capitoli successivi; ricordiamo, per chi avesse saltato qualche passaggio, che l'attività svolta di cui andremo a discutere, prende l'avvio dalla versione dell'applicazione relativa il progetto: A10-2024 e ne rappresenta una versione migliorativa.

1.2.1 Attori e ruoli

Dall'analisi della [Tabella 2](#), appare evidente come i principali consumatori dei servizi forniti dall'applicazione, i nostri attori primari², siano essenzialmente due:

1. Giocatori (Players)

Il compito dei *players*, una volta essersi correttamente autenticati inserendo, in un apposito form (**\login**), le proprie credenziali (nome, cognome, e-mail, password e corso di studi) definite in fase di registrazione (**\register**), è quello di accedere all'area riservata di selezione dei parametri di gioco (**\main**), dove poter visualizzare i robot da sfidare e tutte le classi caricate in precedenza dagli amministratori (**\home**); una volta aver selezionato la classe da testare ed il robot che si intende sfidare, specificando il livello di difficoltà (**\report**), il giocatore non dovrà far altro che confermare le proprie scelte ed iniziare a scrivere test (**\editor**).

²un attore è per definizione qualcosa o qualcuno dotato di un comportamento ed è possibile distinguere tra i seguenti quattro ruoli fondamentali: **(a) attore primario**: colui o qualcosa che sfrutta i servizi forniti dal sistema per raggiungere i propri obiettivi-utente, **(b) attore finale**: colui o qualcosa che ha interesse affinché il sistema venga utilizzato, **(c) attore di supporto**: colui o qualcosa che offre e fornisce servizi al sistema ed infine **(d) attore fuori scena**: colui o qualcosa che ha interesse per quanto riguarda il comportamento del caso d'uso.

I giocatori correttamente registrati e che avranno ricevuto il token di reset (`\password_reset`), avranno la possibilità di impostare una nuova password nel caso in cui l'avessero dimenticata (`\password_change`).

2. Amministratori (Admins)

Gli *amministratori* sono degli utenti privilegiati ed aventi accesso esclusivo ad una interfaccia dedicata grazie alla quale poter gestire una serie di operazioni amministrative (`\home_admin`), si occuperanno, da un lato: **(a)** della gestione di un catalogo di classi di programmazione, da mantenere sempre aggiornato, dove poter, ad esempio, visualizzare tutte le classi caricate in precedenza dai vari *amministratori* (`\class`), caricarne di nuove, modificarle, eventualmente cancellarle o eseguire delle operazioni di ricerca e filtraggio, dall'altro lato: **(b)** avranno la possibilità di visualizzare una classifica rudimentale dei *players* (`\player`).



Attenzione!

La classifica è considerata "rudimentale" perché nella versione attuale dell'applicazione, non è stato ancora implementato alcun meccanismo di valutazione dei punteggi, tuttavia, è possibile visualizzare l'elenco di tutti i giocatori iscritti, in particolare, in corrispondenza di ciascun *player*, oltre alle informazioni "di contatto", verrà visualizzato: **(a)** il numero complessivo di partite giocate ed **(b)** il tempo totale, espresso in minuti, di utilizzo dell'applicazione.

/player

Elenco Studenti							
Show	10	entries	Search:				
Posizione	Punti	Name	Surname	Email	Studies	Partite Giocate	Tempo Totale (min)
1	null	Pippetto	da Padova	pippo60@superio.com	ALTRO	0	0.00
2	null	apaciuli	ciuputi	apaciupi@ciupi.com	BSc	0	0.00
3	null	apaci	ciupli	apaciupi@ciupini.com	BSc	0	0.00
4	null	adcd	ecdce	ecedcd@gece.com	BSc	0	0.00
5	null	avdc	adcd	adcd@ciupini.com	BSc	0	0.00
6	null	Caterina Maria	Accetto	caterina.acce@gmail.com	BSc	0	0.00
7	null	Caterina Accetto		caterina.acce@gmail.com	ALTRO	26	26.40
8	null	Cate	Rina	rina@gmail.com	BSc	0	0.00
9	null	anna	fas	arfasolino@unina.it	BSc	2	0.00



La colonna per memorizzare i **punteggi** e poter visualizzare una classifica è stata predisposta ma **non** è stato ancora implementato alcun meccanismo di valutazione vero e proprio di questi punteggi.

Figura 3: Classifica provvisoria e rudimentale non ancora completamente implementata

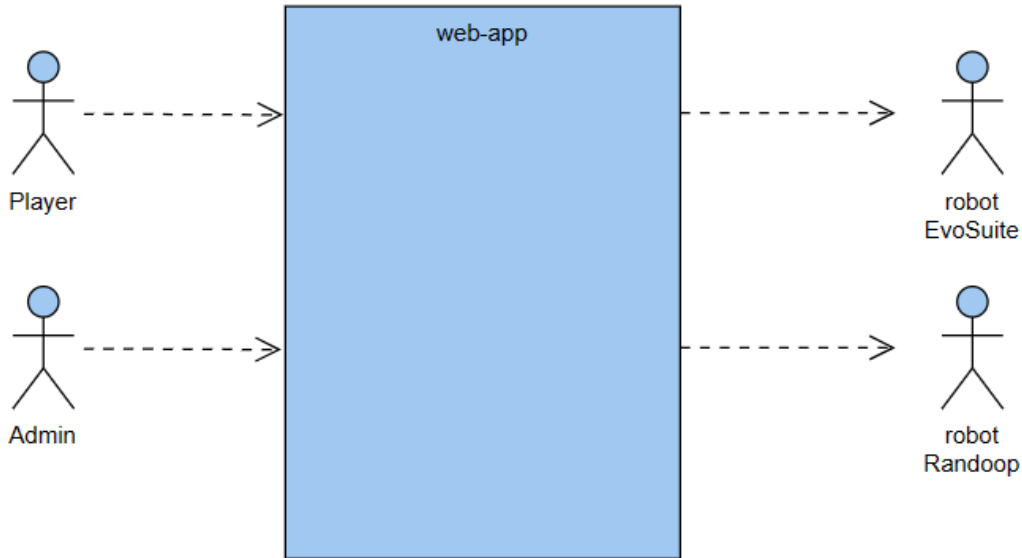


Figura 4: Diagramma di contesto di alto livello

1.2.2 Revisione sommaria del task T1




L'obiettivo di tale sezione è quello di revisionare tutti i requisiti funzionali prefissati per il task T1 (Servizio di gestione delle classi da testare e visualizzazione della classifica dei *players*) [7] allo scopo di: **(a)** cogliere lo stato d'avanzamento complessivo dei lavori, comprendendo quali sono i requisiti mancanti ancora da implementare o eventualmente migliorare ed integrare, **(b)** valutare se è il caso di introdurne di nuovi.

Requisiti funzionali	Stato d'avanzamento
L'applicazione deve mostrare agli admin un elenco di tutte le classi di programmazione	
L'applicazione deve consentire all'admin di scaricare il codice di una classe selezionata	
L'applicazione deve consentire all'admin di inserire una nuova classe con nome, descrizione e codice	
L'applicazione deve consentire all'admin di cancellare una classe esistente selezionata	
L'applicazione deve consentire all'admin di cercare una classe per nome, attraverso un campo di ricerca	
L'applicazione deve consentire all'admin di scegliere e cambiare i criteri di ordinamento delle classi visualizzate	

L'applicazione deve consentire all'admin di selezionare una o più etichette per filtrare le classi visualizzate	
L'applicazione deve consentire all'admin di modificare una classe esistente, inclusi nome, descrizione e codice	
L'applicazione deve consentire all'admin di registrarsi inserendo le proprie informazioni personali (nome, cognome, username e password)	
L'applicazione deve consentire all'admin di effettuare il login al sistema inserendo il proprio username e password	
L'applicazione deve consentire al player di inserire un nuovo "like" ad una classe	Responsabilità T23
L'applicazione deve consentire al player di inserire un nuovo report ad una classe	Responsabilità T23
L'applicazione deve consentire agli admins di visualizzare il numero di "likes" per ogni classe disponibile	
L'applicazione deve consentire all'admin di visualizzare l'elenco dei report effettuati dai players sulle classi	
L'applicazione deve consentire all'admin di visualizzare la classifica di tutti i giocatori correttamente iscritti	Responsabilità R11

Tabella 10: Tabella riassuntiva dello stato d'avanzamento complessivo dei requisiti funzionali definiti per il task T1

La legenda di colori mostrata in [Tabella 10](#) è così da intendersi:

-  : requisiti correttamente implementati e funzionanti
-  : requisiti predisposti ma non ancora integrati/non correttamente funzionanti.
Consideriamo un esempio e prendiamo in esame il requisito: "L'applicazione deve consentire all'admin di modificare una classe esistente, inclusi nome, descrizione e codice", nella versione attuale dell'applicazione, non è possibile modificare il codice di una classe già caricata ma solamente aggiornare le seguenti informazioni: **(a)** nome, **(b)** data, **(c)** difficoltà e **(d)** descrizione.
-  : requisiti non ancora implementati/non assegnati al giusto servizio

-  : requisiti modificati

1.2.3 Revisione sommaria del task T23

Procediamo alla revisione dei requisiti funzionali assegnati al task T23 (Servizio di autenticazione e registrazione dei *players*)[8]:




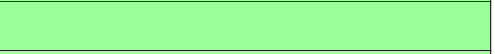




Requisiti funzionali	Stato d'avanzamento
Il sistema deve consentire ad ogni giocatore di potersi registrare nel sistema inserendo: nome, cognome, e-mail, password e corso di studi	
Il sistema, in caso di avvenuta registrazione, deve inviare al giocatore registrato un'e-mail contenente un identificativo univoco (ID)	
Il sistema deve permettere al player registrato di effettuare il login	
In caso di login errato, il sistema deve restituire un errore	
Il sistema deve permettere al player registrato di impostare una nuova password nel caso in cui avesse dimenticato la propria	
Il sistema deve permettere al player registrato di accedere all'area riservata di selezione dei parametri di gioco solo dopo aver effettuato il login	
Il sistema deve permettere al player registrato di effettuare il logout	
Il sistema, per ogni sessione di autenticazione correttamente effettuata, deve assegnare al giocatore un token di autenticazione	

Tabella 11: Tabella riassuntiva dello stato d'avanzamento complessivo dei requisiti funzionali definiti per il task T23

2 Analisi dei requisiti

Dopo questa breve panoramica dello stato d'avanzamento dello sviluppo dei requisiti assegnati per i primi tre task, siamo finalmente in grado di descrivere i requisiti assegnati, i quali, lo anticipiamo, si focalizzeranno esclusivamente sui task T1 e T23 (questo spiega anche perché non sono stati analizzati nel dettaglio tutti gli altri task), nel tentativo di introdurre una serie di funzionalità migliorative per quanto riguarda la gestione dei *players* e degli *amministratori*.

Task di appartenenza	Descrizione
T1	Il sistema, per ogni sessione di autenticazione correttamente effettuata, deve assegnare all'amministratore un token di autenticazione
T1	Il sistema deve implementare un meccanismo che permetta di distinguere gli amministratori, aventi particolari privilegi, dai semplici giocatori (vedi qui)
T1	Il sistema deve consentire ai soli utenti aventi specifiche caratteristiche di potersi registrare inserendo: nome, cognome, username, e-mail e password
T1 - T23	Il sistema deve prevedere la possibilità, in fase di registrazione, di ammettere utenti con doppi nomi o, più in generale, con più nomi separati da uno spazio, ad esempio: "Anna Maria"
T1 - T23	Il sistema deve prevedere la possibilità, in fase di registrazione, di ammettere utenti con cognomi costituiti da più parole separate da spazi oppure da apostrofi, come nel caso: "Degl'Antoni"
T1 - T23	Il sistema deve, accettare, in fase di registrazione, solamente password lunghe tra gli 8 e i 16 caratteri e che contengano: almeno una lettera maiuscola, una minuscola, un numero ed un carattere speciale
T1	Il sistema, in fase di registrazione, deve implementare una apposita misura di sicurezza che consenta di proteggere le password archiviate all'interno del database le quali dovranno venir crittate mediante un algoritmo di hashing

T1	Il sistema deve controllare che tutti i campi, negli appositi form di login e di registrazione, vengano compilati e che siano validi, in caso di errore, deve essere visualizzato a video un messaggio di errore
T1	Il sistema deve permettere all'amministratore registrato di effettuare il logout
T1	Il sistema deve consentire all'amministratore correttamente registrato ed autenticato, di visualizzare, in una apposita sezione dell'area riservata, la lista di tutti gli admin del sistema
T1	Il sistema deve permettere all'amministratore correttamente registrato, di impostare una nuova password nel caso in cui avesse dimenticato la propria
T1	Il sistema deve consentire a tutti gli amministratori correttamente registrati ed autenticati, di invitarne di nuovi, e che non necessariamente rispettano i vincoli richiesti, tramite l'invio, per posta elettronica, di un apposito token
T1	Il sistema deve permettere agli utenti che hanno ricevuto il token di invito per posta, di potersi registrare nel sistema come admin
T1	Il sistema deve implementare un meccanismo che permetta di distinguere gli amministratori del dominio da quelli che sono stati invitati (vedi qui)
T23	Il sistema, in caso di avvenuta registrazione, deve mostrare a video un feedback, dopodichè, indirizzare l'utente alla pagina di login
T23	Il sistema deve consentire agli utenti di decidere se registrarsi tramite il classico form (e-mail + password) oppure effettuare il social login tramite Facebook e completare in questo modo la procedura di registrazione
T1-T23	Il sistema deve essere provvisto di un entry-point, una pagina dove poter smistare gli utenti dell' applicazione in base al ruolo ricoperto ed indirizzarli verso le sezioni dedicate

Tabella 12: Requisiti funzionali e non funzionali assegnati; ricordiamo che i **requisiti funzionali** definiscono le funzionalità e le operazioni specifiche che il sistema deve essere in grado di eseguire, il "cosa" deve essere fatto, mentre i **requisiti non funzionali** specificano gli attributi di qualità oppure le caratteristiche del sistema affinché quest'ultimo possa svolgere tutte le sue mansioni nella maniera più soddisfacente possibile, il "come" il sistema debba essere.

2.1 Analisi dei casi d'uso

I soli requisiti funzionali descritti informalmente in [Tabella 12](#), dovranno venir "tradotti" in una notazione più sintetica che permetta: **(a)** non solo di catturare e comprendere le esigenze degli utenti finali, garantendo che tutti i loro bisogni vengano soddisfatti, ma soprattutto **(b)** evidenziare le specifiche azioni che ciascun utente è in grado di eseguire interagendo con il sistema e **(c)** come quest'ultimo risponda di conseguenza; per riuscire nell'intento faremo affidamento ai **Diagrammi dei Casi d'Uso (Use Case Diagrams)**, definibili come delle storie scritte, una collezione di scenari correlati, sia di successo che di fallimento, che descrivono come un attore, utilizzando il sistema, sia in grado di raggiungere i propri obiettivi-utente.

Per non complicare la rappresentazione e chiarificare ancora meglio cosa fa effettivamente il sistema, quali sono i servizi forniti e chi sono gli attori interagenti, realizzeremo due casi d'uso per ciascun dei servizi presi in esame:

1. Casi d'Uso relativo il task T1

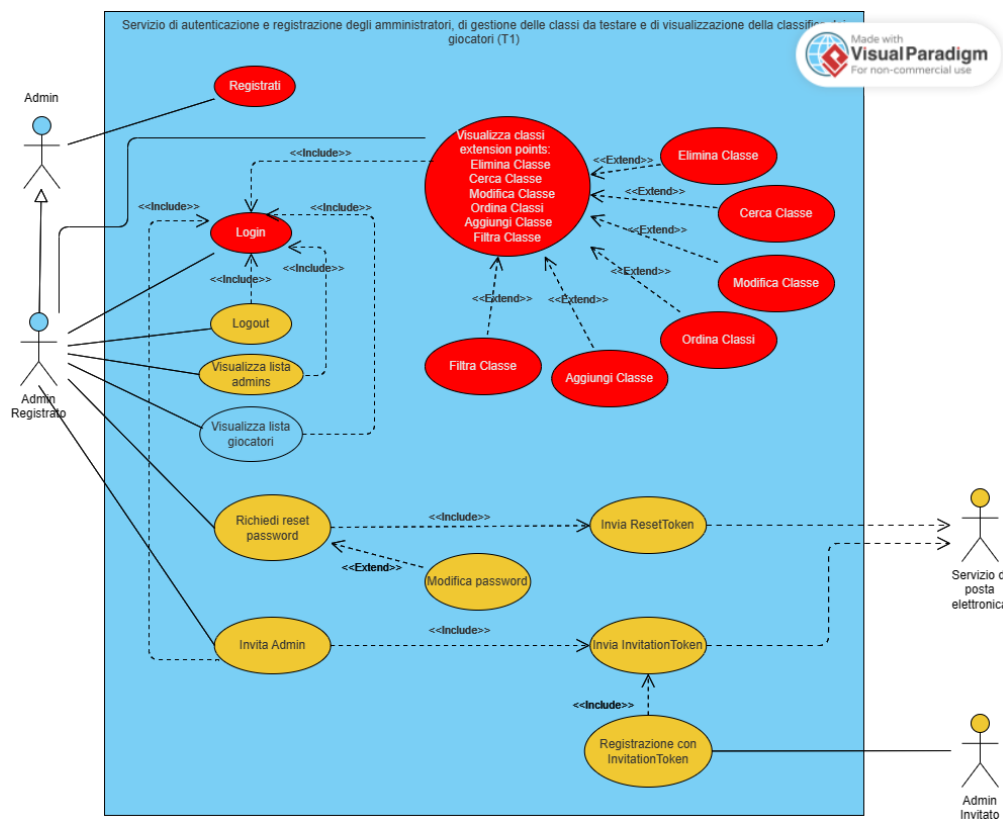




Figura 5: Diagramma dei Casi d'Uso relativo il task T1

L'utilizzo dei colori in [Figura 5](#), risponde alla seguente legenda:

-  : casi d'uso non modificati.
-  : casi d'uso modificati.

”Registrati”: in fase di registrazione, il sistema prevede: **(a)** la possibilità di inserire utenti con più nomi separati, eventualmente, da uno spazio, **(b)** la possibilità di inserire utenti con cognomi costituiti da più parole separate da spazi oppure apostrofi, **(c)** un meccanismo per distinguere gli *amministratori* dai normali *giocatori* accettando esclusivamente username del seguente formato: [username di lunghezza tra 2-30 caratteri]_unina.

Sarà possibile registrarsi inserendo: nome, cognome, username, e-mail istituzionale (@unina.it oppure @studenti.unina.it) e password.

L'applicazione accetterà solamente password lunghe tra gli 8 e i 16 caratteri e contenenti: almeno una lettera maiuscola, una minuscola, un numero ed un carattere speciale e se la password inserita rispetterà tali criteri, non verrà memorizzata in chiaro all'interno del database, bensì crittata mediante un algoritmo di hashing.


Il sistema controllerà che tutti i campi del form di registrazione vengano compilati e che gli input siano validi, in caso di errore, visualizzerà a video un messaggio di errore.

L'applicazione, prima di consentire la registrazione, controllerà la validità del token JWT assegnato all'utente, in caso di invalidità del token, si potrà procedere con l'operazione.

”Login”: in fase di log-in, il sistema controllerà che tutti i campi del form vengano compilati ed in caso di mancata compilazione, verrà visualizzato a video un messaggio d'errore.

All'*amministratore* correttamente ”loggato”, il sistema assegnerà un token JWT di autenticazione ed autorizzazione per la gestione della sessione.

”Visualizza Classi: l'*amministratore* correttamente ”loggato” e con un token JWT valido, potrà accedere, mediante un apposito cruscotto, a tutte le funzionalità di gestione delle classi di programmazione.

-  : casi d'uso aggiunti

2. Casi d'Uso relativo i task T23

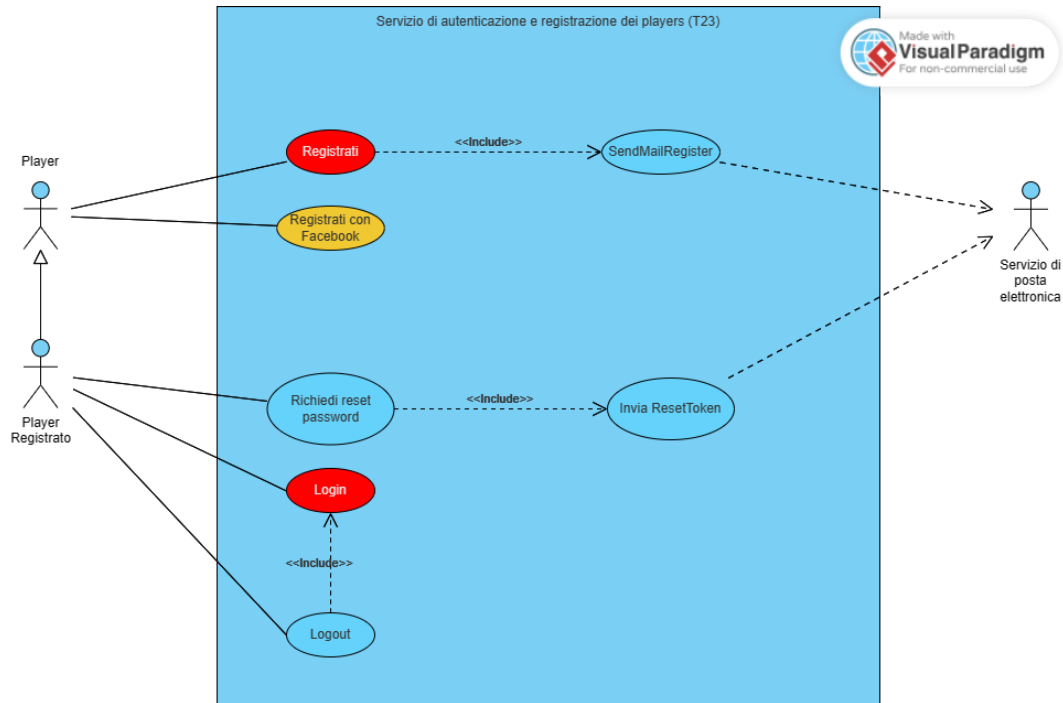


Figura 6: Diagramma dei Casi d'Uso relativo i task T23

- ■ : casi d'uso non modificati.
- ■ : casi d'uso modificati.

”**Login**”: in fase di log-in, il sistema controllerà che l’utente non si sia già precedentemente registrato tramite Facebook, in qual caso restituirà un messaggio d’errore.

”**Registrati**”: in fase di registrazione, il sistema prevede: **(a)** la possibilità di inserire utenti con più nomi separati, eventualmente, da uno spazio, **(b)** la possibilità di inserire utenti con cognomi costituiti da più parole separate da spazi oppure apostrofi.

L’applicazione accetterà solamente password lunghe tra gli 8 e i 16 caratteri e contenenti: almeno una lettera maiuscola, una minuscola, un numero ed un carattere speciale; in caso di input errato, verrà visualizzato un messaggio d’errore.

Durante questa fase, settando un apposito parametro, si specifica che la registrazione sia avvenuta mediante classico form (e-mail + password) e non tramite Facebook.



In caso di avvenuta registrazione, il sistema mostrerà a video un feedback, dopodichè, indirizzerà l’utente alla pagina di login.

- ■ : casi d'uso aggiunti.

2.2 Analisi degli scenari dei casi d'uso

Gli **scenari dei casi d'uso** (istanze di casi d'uso), potrebbero venir definiti come una sequenza specifica di azioni e risposte intercorrenti fra il sistema ed alcuni attori.

2.2.1 Scenari dei casi d'uso relativi il task T1

In questa sezione andremo ad analizzare i casi d'uso che sono stati aggiunti (), per quanto riguarda quelli modificati (), le modifiche apportate sono state già descritte nel paragrafo precedente; una trattazione più approfondita verrà riservata al caso d'uso: **"Visualizza classi"** per descrivere come questo insieme di operazioni amministrative venga gestito mediante il token JWT di autenticazione ed autorizzazione.

1. Casi d'uso aggiunti

	Richiesta reset password
Attore primario	Admin registrato
Attore secondario	Servizio di posta elettronica
Descrizione	Permette ad un amministratore correttamente registrato, di richiedere di impostare una nuova password nel caso in cui avesse dimenticato la propria
Pre-condizioni	L' amministratore deve essere registrato nel sistema (token JWT invalido) e si deve trovare nella sua pagina dedicata di login
Sequenza di eventi principale	1) L' admin registrato clicca sul link: "Hai dimenticato la password?" 2) L'admin registrato verrà indirizzato alla pagina "Reimposta password" dove inserisce la propria e-mail e clicca sul pulsante: "Invia e-mail" 3) L'admin registrato viene indirizzato alla pagina di: "Modifica password" 4) Il servizio di posta elettronica invia una e-mail contenente un ResetToken
Post-condizioni	La password viene resettata
Casi d'uso correlati	Viene esteso dal caso d'uso: "Modifica password"

Sequenza di eventi alternativi	<p>1.1) Se il token JWT associato all'utente risulta valido, significa che l'amministratore si è già loggato e l'operazione non viene completata</p> <p>2.2) Se l'e-mail inserita non viene trovata nel database, viene restituito un messaggio d'errore e l'operazione non viene completata</p>
---------------------------------------	--

Tabella 13: Scenario del caso d'uso: **"Richiesta reset password"**

	Invia ResetToken
Attore primario	Servizio di posta elettronica
Attore secondario	Admin registrato
Descrizione	Il servizio di posta elettronica invia una e-mail per consentire il reset della password
Pre-condizioni	L'admin registrato deve aver richiesto di impostare una nuova password ed inserito, nell'apposito form, una e-mail valida
Sequenza di eventi principale	1) Il servizio di posta elettronica invia una e-mail all'indirizzo specificato contenente un ResetToken
Post-condizioni	L'admin registrato può procedere alla modifica della password
Casi d'uso correlati	-
Sequenza di eventi alternativi	1.1) Se la mail inserita non fosse valida oppure, se si fossero verificati altri problemi, l'operazione non viene conclusa e si mostra a video un messaggio d'errore

Tabella 14: Scenario del caso d'uso: **"Invia ResetToken"**

	Modifica password
Attore primario	Admin registrato
Attore secondario	Servizio di posta elettronica
Descrizione	Permette ad un admin registrato di poter modificare la propria password, inserendone una nuova
Pre-condizioni	L'admin registrato deve aver richiesto di impostare una nuova password ed il suo token JWT deve essere invalido
Sequenza di eventi principale	1) L'admin registrato controlla la propria casella di posta elettronica ed apre l'e-mail ricevuta dal sistema contenente il ResetToken 2) Copia il ResetToken ricevuto 3) L'admin registrato, nella pagina: "Modifica password", inserisce nel form il proprio indirizzo di posta elettronica, il ResetToken ricevuto ed inserisce una nuova password 4) L'admin registrato clicca sul pulsante: "Reimposta" 5) L'admin registrato viene indirizzato alla pagina di login
Post-condizioni	La password viene modificata
Casi d'uso correlati	Estende il caso d'uso: "Richiesta reset password"
Sequenza di eventi alternativi	1.1) Se l'admin registrato non avesse ricevuto l'e-mail di reset, deve ripetere l'intera procedura di richiesta 3.1) Se il token JWT associato all'utente risulta valido, significa che l'admin registrato si è già autenticato e l'operazione non viene conclusa 4.1) Se la mail inserita non dovesse risultare valida, viene mostrato a video un messaggio d'errore e l'operazione non viene conclusa 4.2) Se il ResetToken inserito non fosse valido, viene mostrato a video un messaggio d'errore e l'operazione non viene conclusa 4.3) Se la nuova password inserita non rispetta il formato prescritto, viene mostrato a video un messaggio d'errore e l'operazione non viene conclusa

Tabella 15: Scenario del caso d'uso: "**Modifica password**"

	Logout
Attore primario	Admin registrato
Attore secondario	-
Descrizione	Permette ad un admin registrato di effettuare il logout
Pre-condizioni	L'amministratore deve essersi precedentemente "loggato" e si deve trovare nella sua area riservata
Sequenza di eventi principale	1) L'admin registrato clicca sul bottone: "Logout" 2) L'admin registrato viene re-indirizzato alla pagina di login
Post-condizioni	I cookies associati alla sessione-utente vengono rimossi
Casi d'uso correlati	Include il caso d'uso: "Login"
Sequenza di eventi alternativi	-

Tabella 16: Scenario del caso d'uso: **"Logout"**

	Visualizza lista admins
Attore primario	Admin registrato
Attore secondario	-
Descrizione	Permette ad un admin registrato di visualizzare la lista di tutti gli amministratori del sistema e le relative informazioni di contatto
Pre-condizioni	L'amministratore deve essersi precedentemente "loggato" (token JWT valido) e si deve trovare nella sua area riservata
Sequenza di eventi principale	1) L'admin registrato clicca, all' interno del cruscotto, sul tasto: "Info"
Post-condizioni	L'admin registrato visualizza l' elenco di tutti gli amministratori del sistema
Casi d'uso correlati	Include il caso d'uso: "Login"
Sequenza di eventi alternativi	1.1) Se il token JWT associato all'utente risulta invalido, l'utente verrà re-indirizzato alla pagina di login

Tabella 17: Scenario del caso d'uso: **"Visualizza lista admins"**



How to

Il meccanismo che è stato implementato per consentire la distinzione tra gli *amministratori* del sistema ed i semplici *players*, consiste nel "pretendere" l'inserimento, in fase di registrazione, di uno specifico dominio di posta: quella istituzionale nel caso degli *admin* (@studenti.unina.it oppure @unina.it), ed un qualsiasi dominio (@gmail, @libero, @outlook, ...) per i *giocatori*.

	Invita admin
Attore primario	Admin registrato
Attore secondario	Servizio di posta elettronica
Descrizione	Permette ad un amministratore correttamente registrato, di invitarne di nuovi e non necessariamente appartenenti al dominio
Pre-condizioni	L'amministratore deve essersi precedentemente "loggato" (token JWT valido) e si deve trovare nella sua area riservata
Sequenza di eventi principale	1) L'admin registrato clicca sul tasto: "Invita" 2) L'admin registrato viene indirizzato alla pagina: "Invita amministratori" dove specifica l'indirizzo e-mail dell'utente che intende invitare e clicca sul pulsante: "Invia e-mail" 3) L'admin registrato viene indirizzato alla sua area riservata 4) Il servizio di posta elettronica invia una e-mail contenente un InvitationToken
Post-condizioni	L'admin registrato ha invitato un nuovo utente come amministratore del sistema
Casi d'uso correlati	Include il caso d'uso: " Login " e viene esteso dal caso d'uso: " Invia InvitationToken "

Sequenza di eventi alternativi	<p>1.1) Se il token JWT associato all'utente risulta invalido, verrà mostrato a video un messaggio d'errore e la procedura non viene completata</p> <p>2.1) Se l'e-mail inserita viene trovata all'interno del database, verrà mostrato a video un messaggio d'errore e la procedura non viene completata</p>
---------------------------------------	---

Tabella 18: Scenario del caso d'uso: **Invita admin**

	Invia InvitationToken
Attore primario	Servizio di posta elettronica
Attore secondario	Admin registrato, Admin invitato
Descrizione	Il servizio di posta elettronica invia una e-mail per consentire l'invito di nuovi amministratori
Pre-condizioni	L'admin registrato e correttamente autenticato, deve aver richiesto di invitare un nuovo utente come amministratore del sistema, inserendo, nell'apposito form, una e-mail valida associata all'admin invitato
Sequenza di eventi principale	1) Il servizio di posta elettronica invia una e-mail all'indirizzo specificato contenente un InvitationToken
Post-condizioni	L'admin invitato ha i privilegi per completare la registrazione all'interno del sistema
Casi d'uso correlati	-
Sequenza di eventi alternativi	1.1) Se la mail inserita non fosse valida oppure, se si fossero verificati altri problemi, l'operazione non viene conclusa e si mostra a video un messaggio di errore

Tabella 19: Scenario del caso d'uso: **"Invia InvitationToken"**

	Registrati con InvitationToken
Attore primario	Admin invitato
Attore secondario	Servizio di posta elettronica
Descrizione	Permette all'admin invitato di registrarsi all' interno del sistema
Pre-condizioni	Il servizio di posta elettronica deve aver inviato una e-mail all'indirizzo dell' admin invitato contenente un InvitationToken. L'admin invitato deve essere caratterizzato da un token JWT invalido
Sequenza di eventi principale	1) L'admin invitato controlla la propria casella di posta elettronica ed apre l'e-mail ricevuta dal sistema contenente l'InvitationToken 2) L'admin invitato copia l'InvitationToken 3) L'admin invitato, nella pagina: "Registrazione Amministratore tramite Token di invito", inserisce nel form l'InvitationToken ricevuto, il proprio nome, cognome, username e password 4) L'admin invitato clicca sul pulsante: "Registrati" 5) L'admin invitato viene re-indirizzato alla pagina di login
Post-condizioni	L'admin invitato si è registrato correttamente nel sistema
Casi d'uso correlati	Include il caso d'uso: " Invia InvitationToken "

<p>Sequenza di eventi alternativi</p>	<p>1.1) Se l'admin invitato non avesse ricevuto l'e-mail di invito, la procedura di registrazione non può essere completata</p> <p>3.1) Se il token JWT associato all'utente risulta valido, verrà visualizzato a video un messaggio d'errore e l'operazione non può essere completata</p> <p>4.1) Se l'InvitationToken inserito non fosse valido, viene mostrato a video un messaggio d'errore e l'operazione non viene conclusa</p> <p>4.2) Se l'e- mail inserita non viene trovata nel database, viene mostrato a video un messaggio d'errore e l'operazione non viene conclusa</p> <p>4.3) Se il nome inserito non rispettasse il formato prescritto, viene mostrato a video un messaggio d'errore e l'operazione non viene conclusa</p> <p>4.4) Se il cognome inserito non rispettasse il formato prescritto, viene mostrato a video un messaggio d'errore e l'operazione non viene conclusa</p> <p>4.5) Se l'username inserito non rispettasse il formato prescritto, viene mostrato a video un messaggio d'errore e l'operazione non viene conclusa</p> <p>4.6) Se la password inserita non rispettasse il formato prescritto, viene mostrato a video un messaggio d'errore e l'operazione non viene conclusa</p>
--	--

Tabella 20: Scenario del caso d'uso: "Registrati con InvitatioToken"



How to


Il meccanismo che è stato implementato per consentire la distinzione tra gli amministratori del dominio (domini di posta della Federico II) e quelli che sono stati invitati (un qualsiasi dominio di posta), consiste nell'accettare, in fase di registrazione, un particolare formato di username: [username]_unina, per rappresentare gli admin del dominio ed [username]_invited per rappresentare, di contro, quelli che sono stati invitati.

2. Casi d'uso modificati

	Visualizza classi
Attore primario	Admin registrato
Attore secondario	-
Descrizione	Permette all'admin registrato di visualizzare le informazioni relative le classi di programmazione
Pre-condizioni	L'admin registrato deve essersi precedentemente "loggato" (token JWT valido) e si deve trovare nella sua area riservata
Sequenza di eventi principale	1) L'admin registrato clicca sul tasto "Classes"
Post-condizioni	L'admin registrato visualizza l'elenco delle classi di programmazione
Casi d'uso correlati	Include il caso d'uso: "Login" ed estende i casi d'uso: "Elimina Classe", "Aggiungi Classe", "Cerca Classe", "Modifica Classe", "Ordina Classi" e "Filtra Classi"
Sequenza di eventi alternativi	1.1) Se il token JWT associato all'utente risulta invalido, l'utente verrà re-indirizzato alla pagina di login

Tabella 21: Scenari del caso d'uso: **"Visualizza classi"**

2.2.2 Scenari dei casi d'uso relativi i task T23

In questa sezione analizzeremo esclusivamente l'unico caso d'uso che è stato aggiunto (): **"Registrati con Facebook"**:

	Registrati con Facebook
Attore primario	Player
Attore secondario	-
Descrizione	Permette ad un player di effettuare la registrazione all'applicazione tramite Facebook
Pre-condizioni	Il player deve possedere un account Facebook, non deve essere ancora registrato e si deve trovare nella pagina di login dedicata
Sequenza di eventi principale	1) Il player clicca sul bottone: "Accedi con Facebook" 2) Il player compila il form di autenticazione di Facebook inserendo: e-mail (o numero di telefono) , password e clicca sul pulsante: "Accedi" 3) Il player viene re-indirizzato all'area riservata di gioco
Post-condizioni	Il player ha completato la registrazione al sistema e si è autenticato con Facebook (social login)
Casi d'uso correlati	-
Sequenza di eventi alternativi	2.1) Se il tokenFb è invalido, verrà visualizzato a video un messaggio d'errore e la procedura di registrazione non può essere completata

Tabella 22: Scenari del caso d'uso: **"Registrati con Facebook"**

3 Progettazione

Per poter catturare in maniera chiara e dettagliata le interazioni intercorrenti tra i vari oggetti del sistema, faremo affidamento ad un prezioso strumento di progettazione: i **Diagrammi di Sequenza (Sequence Diagrams)**, in grado, in definitiva, di descrivere il comportamento dell'applicazione.

3.1 Sequence Diagrams relativi il task T1

All'interno di tale sezione, procederemo a dettagliare, mediante **Diagrammi di sequenza**, i principali casi d'uso, e relativi scenari, individuati e descritti all'interno del paragrafo: [2.1 \(Analisi dei casi d'uso\)](#).

1. Sequence Diagram: Richiesta reset password

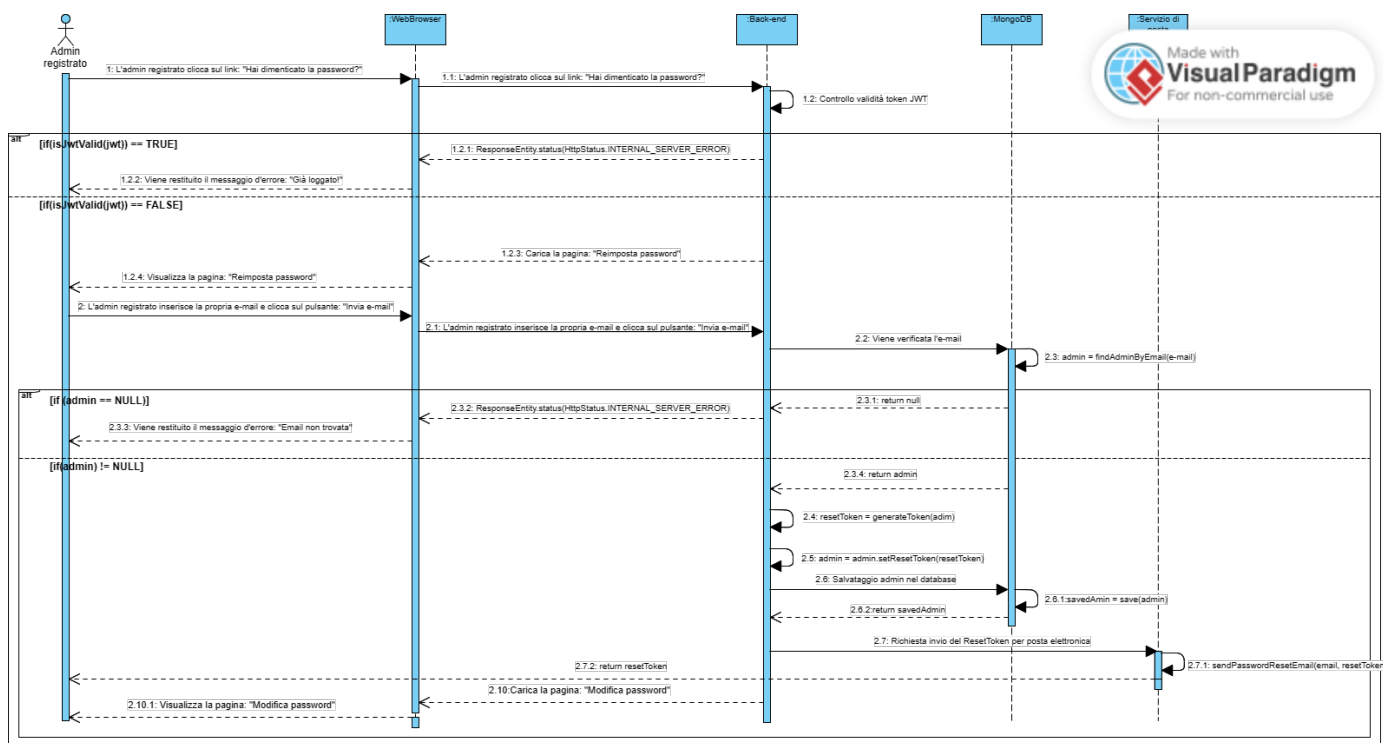


Figura 7: Sequence Diagram relativo il caso d'uso: **"Richiesta reset password"**; per non appesantire la schematizzazione, si è evitato di eseguire il controllo nel caso in cui l'invio dell'e-mail per posta elettronica non fosse andato a buon fine.



Attenzione!

| Per una visione migliore del diagramma clicca [qui!](#)

2. Sequence Diagram: Modifica password

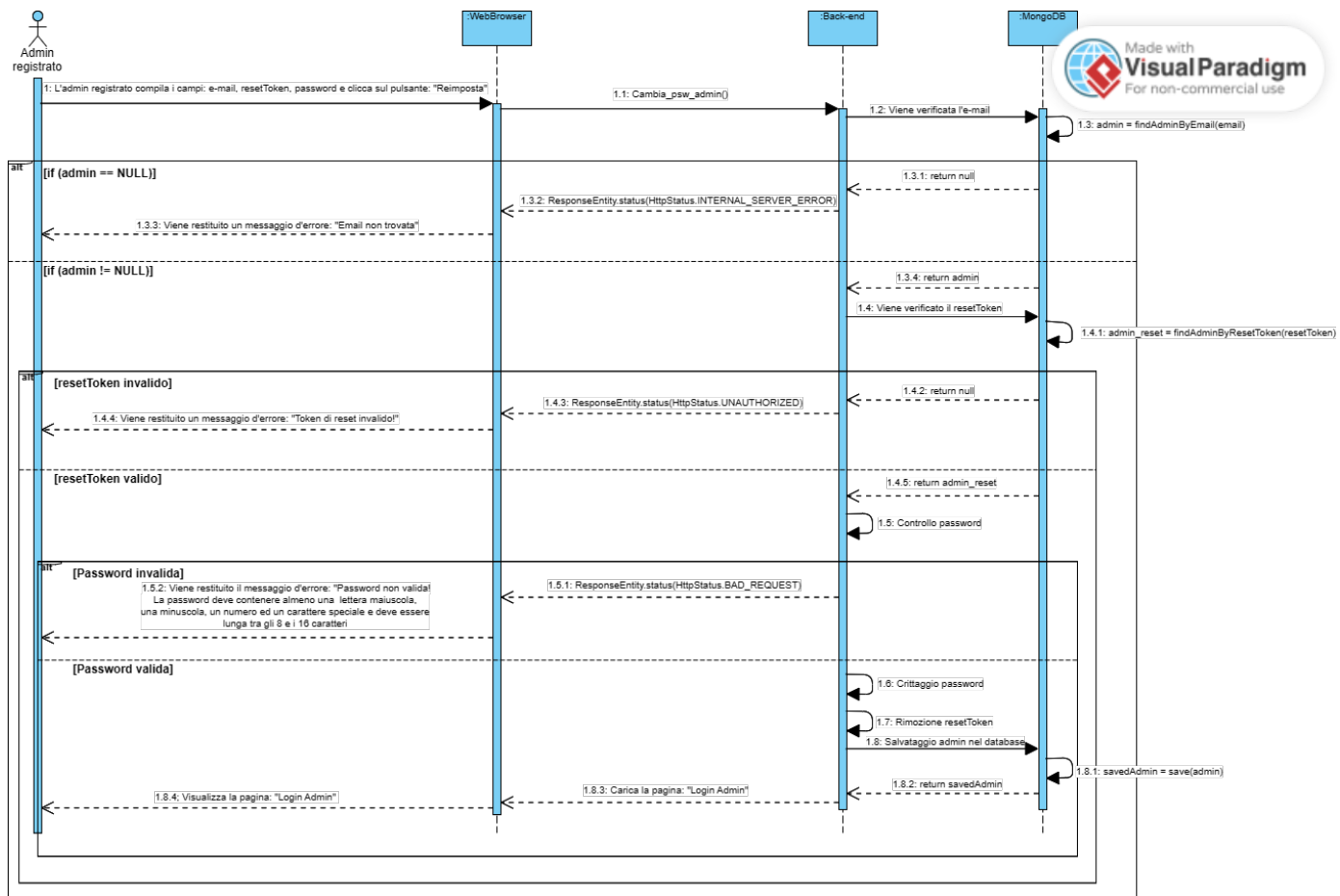


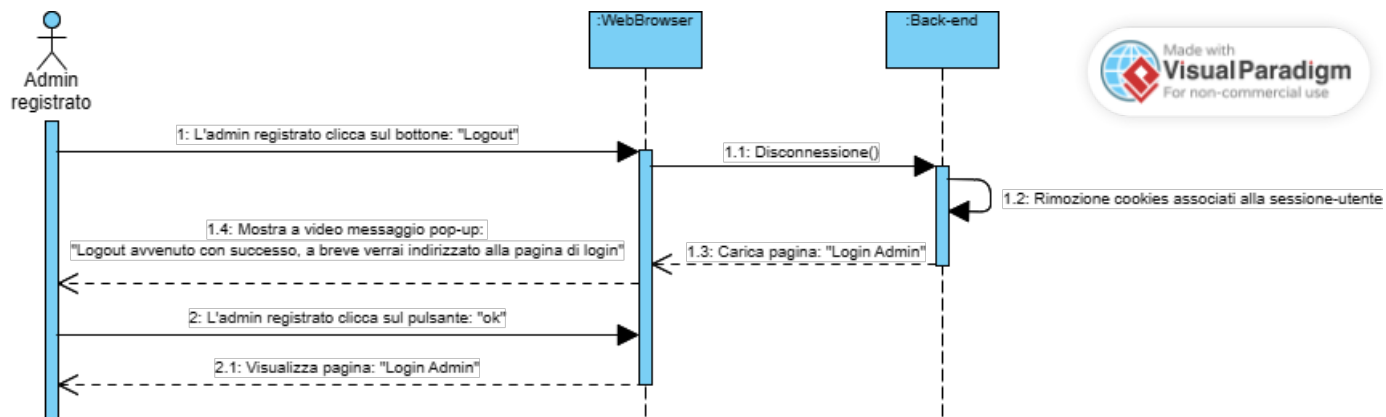
Figura 8: Sequence Diagram relativo il caso d'uso: "Modifica password".



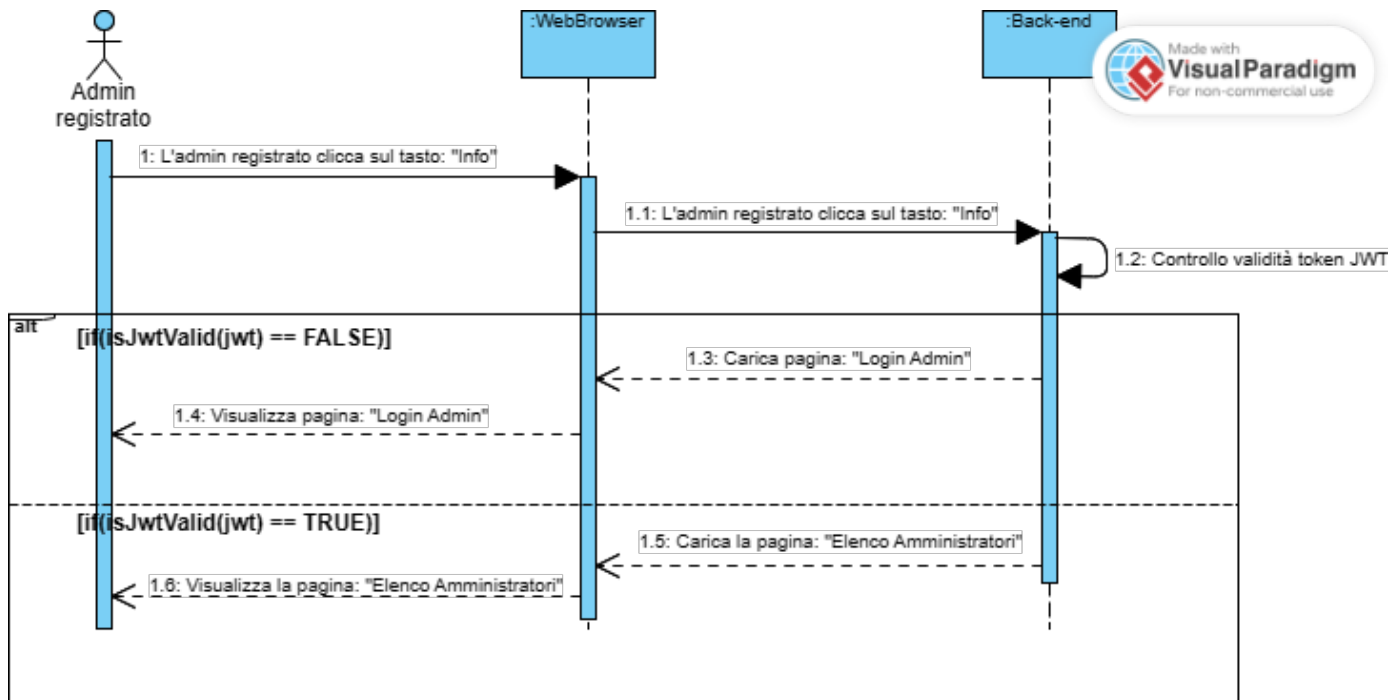
Attenzione!

| Per una visione migliore del diagramma clicca [qui](#)!

3. Sequence Diagram: Logout



4. Sequence Diagram: Visualizza lista admins



Nota bene

Un approccio analogo vale anche in corrispondenza del diagramma di sequenza associato al caso d'uso: "Visualizza classi" ed è per questo motivo che lo si omette dalla raffigurazione.

5. Sequence Diagram: Invita admin

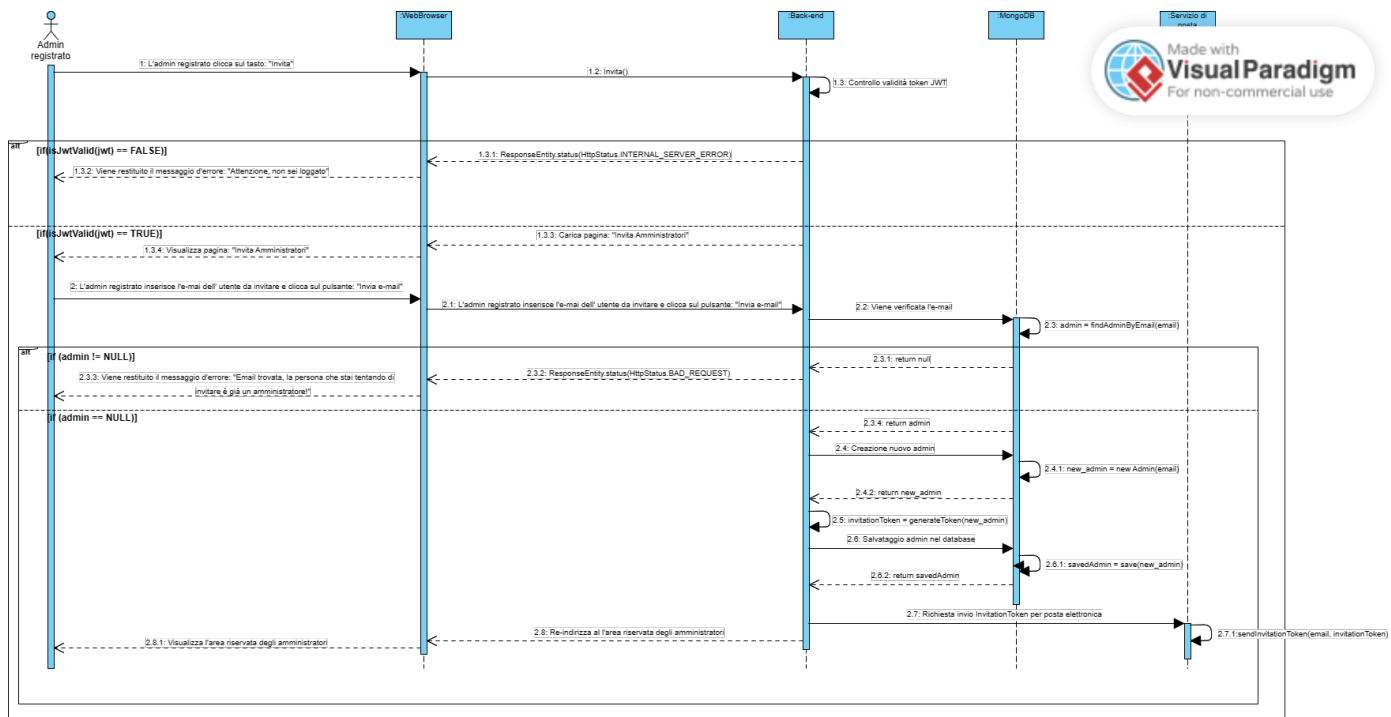


Figura 11: Sequence Diagram relativo il caso d'uso: **"Invita admin"**; per non appesantire la schematizzazione, si è evitato di eseguire il controllo nel caso in cui l'invio dell'e-mail per posta elettronica non fosse andato a buon fine.



Attenzione!

| Per una visione migliore del diagramma clicca [qui!](#)

6. Sequence Diagram: Registrati con InvitationToken

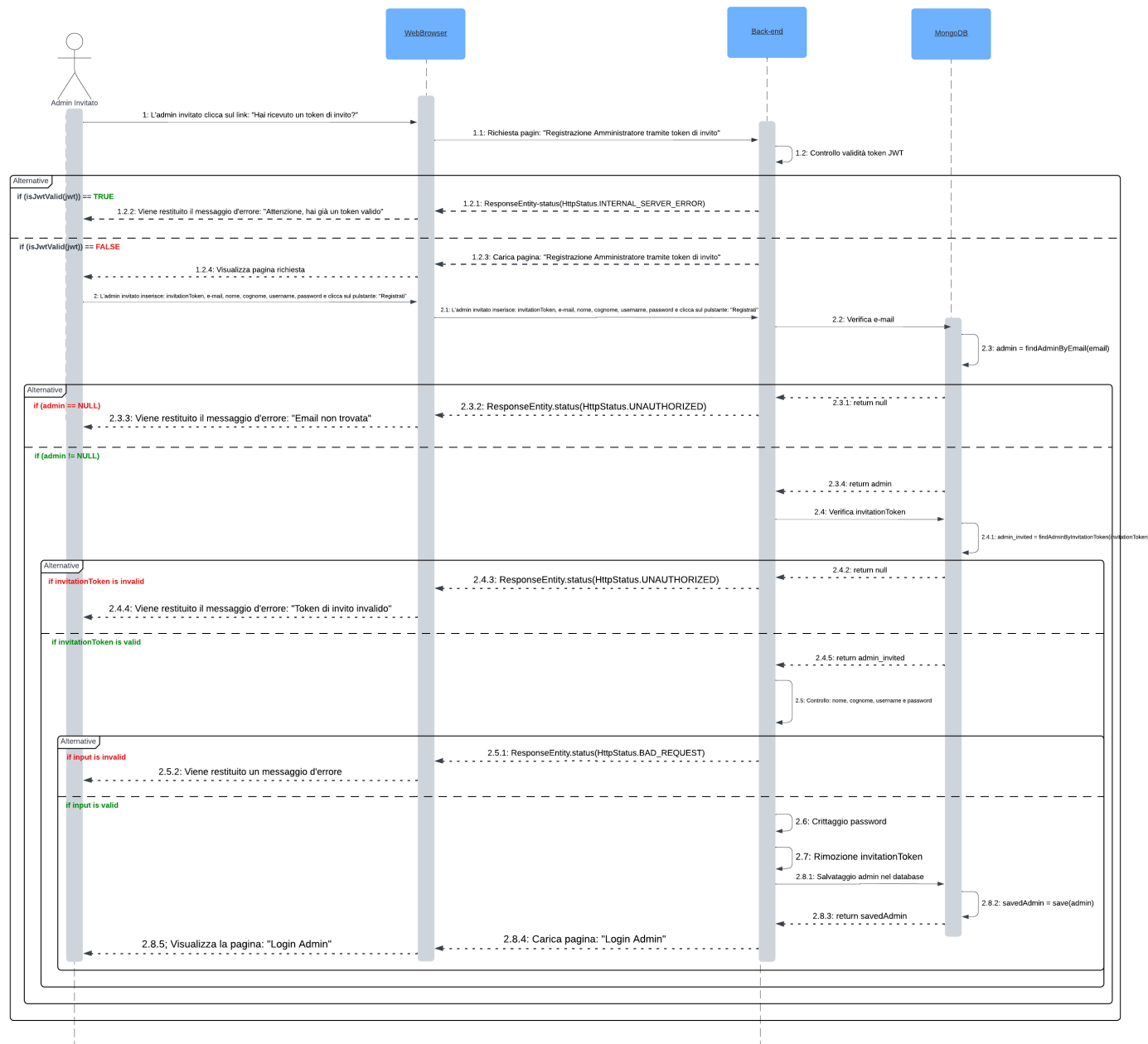


Figura 12: Sequence Diagram relativo il caso d'uso: "Registrati con invitationToken".

3.2 Sequence Diagrams relativi i task T23

In questo paragrafo dettaglieremo, mediante **Diagramma di Sequenza**, esclusivamente il caso d'uso: "Registrati con Facebook":

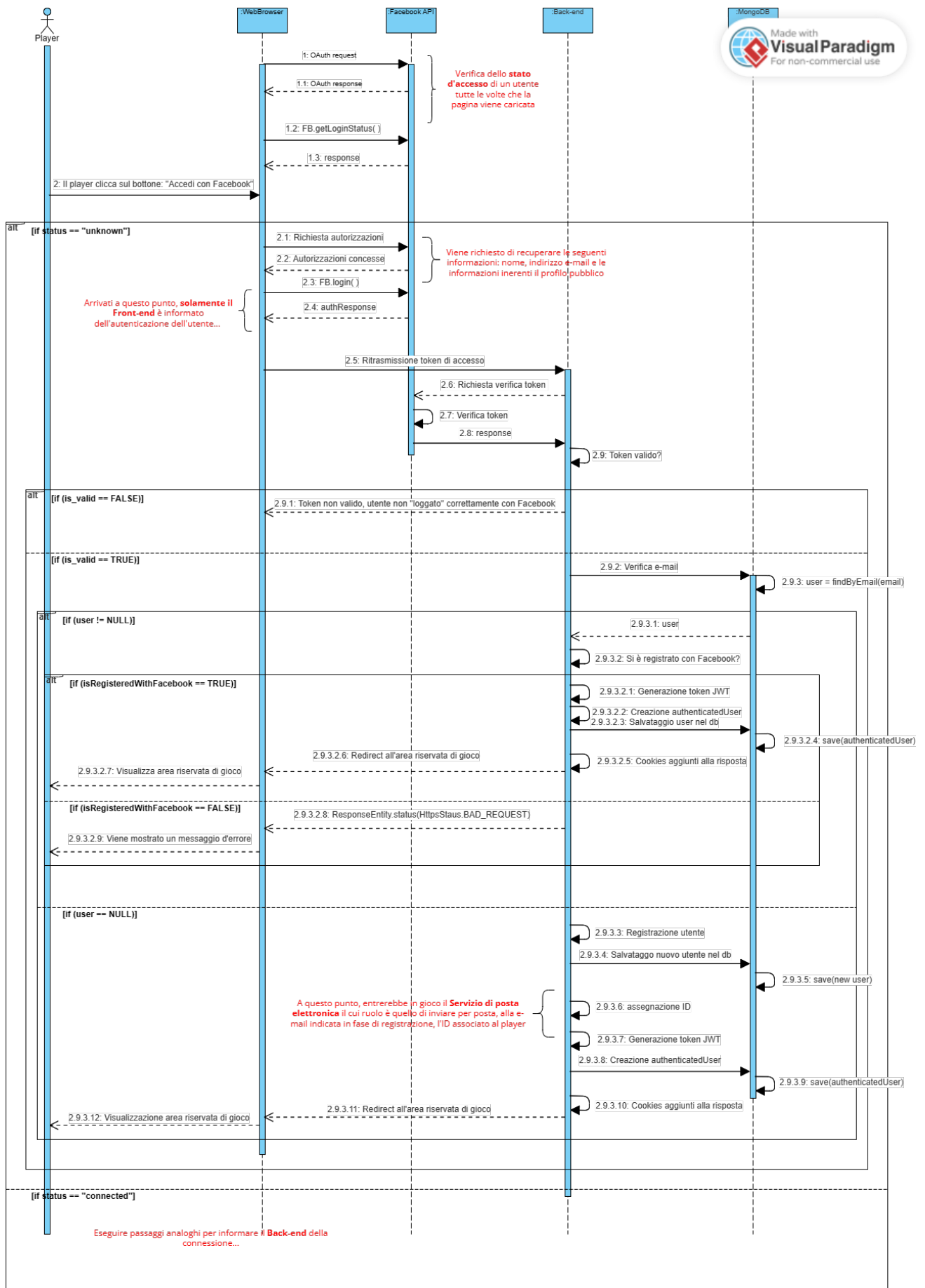


Figura 13: Sequence Diagram relativo il caso d'uso: "Registrati con Facebook".

4 Implementazione

Per evitare di appesantire eccessivamente la documentazione, si prega di fare riferimento ai seguenti files di log: [log.txt](#) e [log2.txt](#) ottenuti eseguendo i comandi:

```
git diff > log.txt  
git diff > log2.txt
```

Si tratta di un comando estremamente utile durante il processo di sviluppo del software poiché consente di visualizzare sinteticamente quali sono state tutte le modifiche apportate al progetto sia localmente, sia rispetto altri rami o commit.

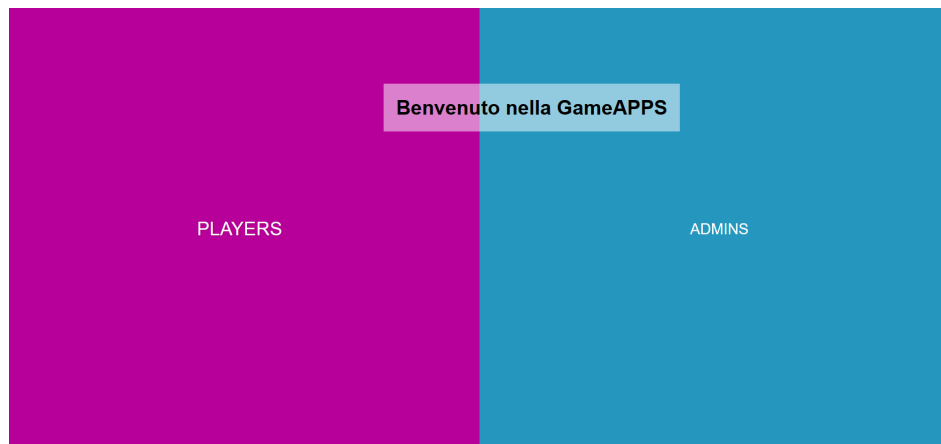


Figura 14: Nuovo menù aggiunto

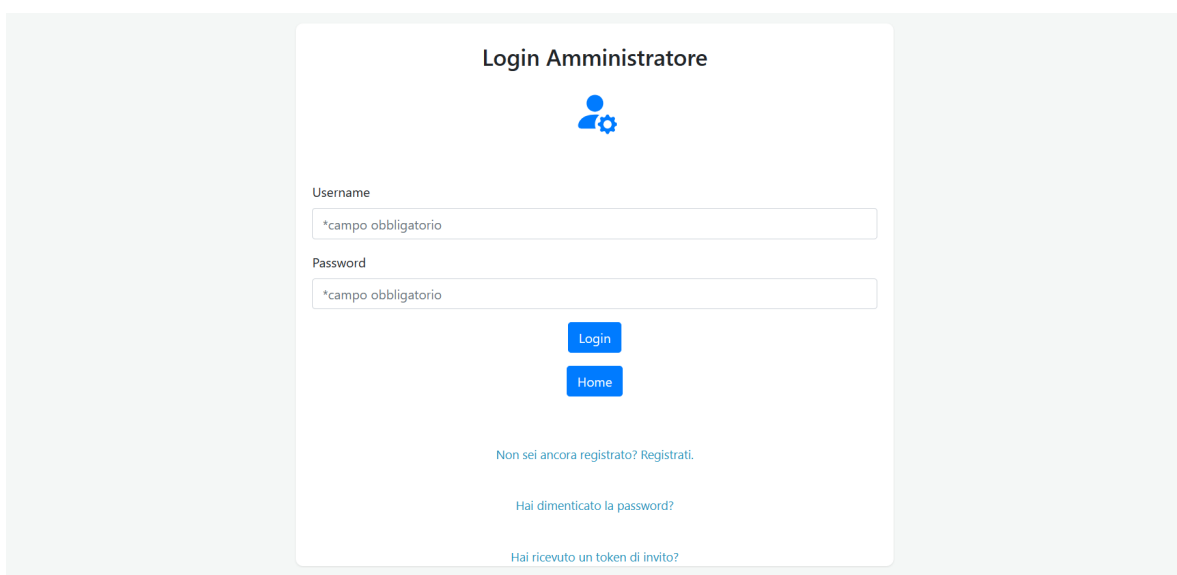



Figura 15: Pagina: "Login Amministratore" aggiornata

Reimposta password



Inserisci il tuo indirizzo email e ti invieremo un token per reimpostare la password.

Indirizzo e-mail


*campo obbligatorio

Invia e-mail

[Hai già ricevuto il token?](#)

Figura 16: Pagina: **"Reimposta password"** aggiunta

Modifica password



Indirizzo e-mail

*campo obbligatorio

Token ricevuto

*campo obbligatorio


Nuova password

*campo obbligatorio

Reimposta

Figura 17: Pagina: **"Modifica password"** aggiunta

Registrazione Amministratore tramite Token di invito



Token di invito

*campo obbligatorio

Nome

*campo obbligatorio

Cognome

*campo obbligatorio

Username

*campo obbligatorio

Password

*campo obbligatorio

Registrati

Figura 18: Pagina: **"Registrazione Amministratore tramite Token di invito"** aggiunta

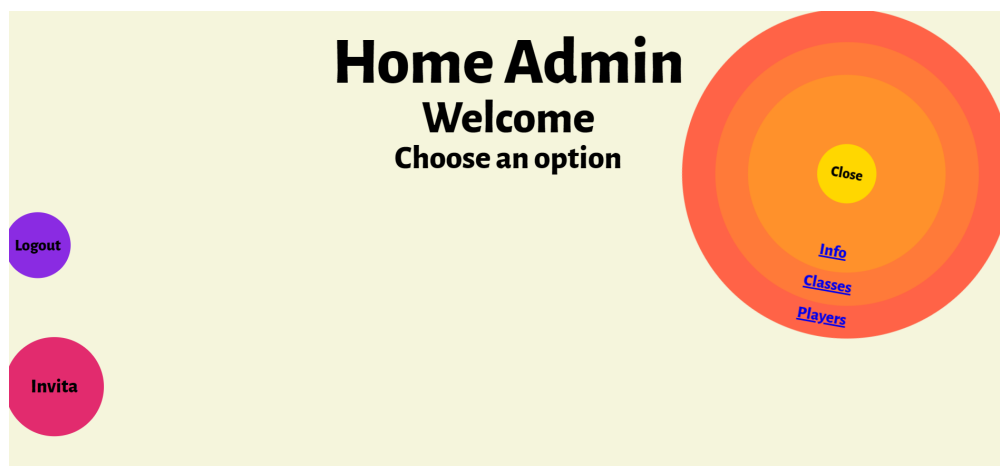


Figura 19: Cruscotto admin aggiornato

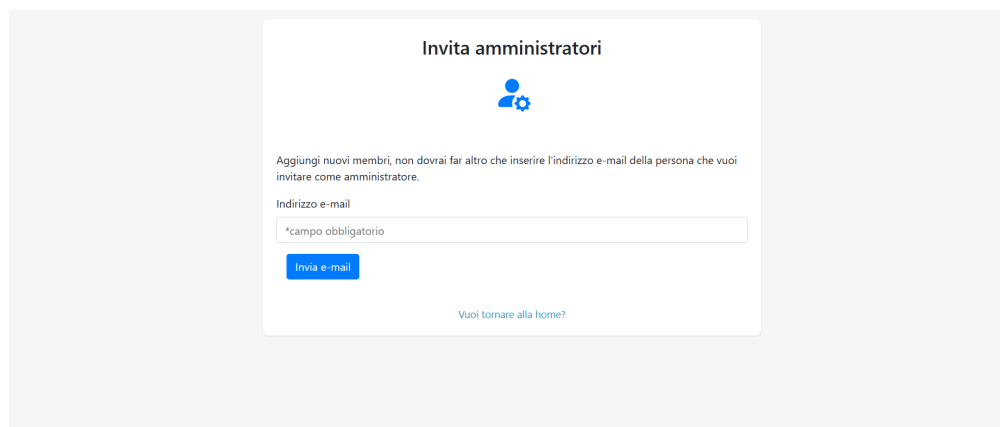


Figura 20: Pagina: "Invita amministratori" aggiunta

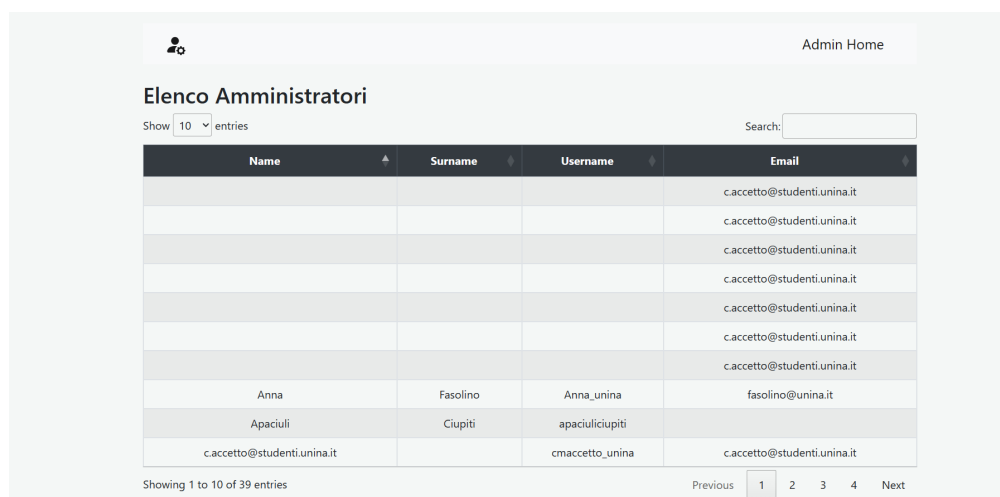


Figura 21: Pagina: "Invita amministratori" aggiunta

4.1 Extras

4.1.1 Aggiunta degli API endpoints

Per garantire che gli endpoints appena sviluppati siano "visibili" dall'UI Gateway, componente preposto alla gestione di tutte le richieste in entrata, è necessario inserire le loro rotte all'interno del suo file di configurazione: `default.conf`.

```
server {
    listen 80;
    #timeout attesa back end
    proxy_read_timeout 6000s;

    #parametri relativi al cliente
    client_max_body_size 5M;
    client_body_timeout 600s;

    location ~ / {
        return 301 /login;
    }

    error_page 500 /error.html;

    #MODIFICA (12/02/2024) : Aggiunta servizio logout dell'admin
    #MODIFICA (15/02/2024) : Aggiunta servizio di visualizzazione di tutti gli amministratori del sistema
    #MODIFICA (15/02/2024) : Aggiunta servizio di cambio psw
    #MODIFICA (16/02/2024) : Aggiunta servizio di invio amministratori
    #MODIFICA (18/02/2024) : Aggiunta menu unico per indirizzare verso: o /login oppure verso /loginAdmin

    location ~ ^/(login|login_with_invitation|invite_admins|password_reset_admin|password_change_admin|info|admins_list|logout_admin|class|player|loginAdmin|
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://manvsclass-controller-1:8080;
    }

    location ~ ^/(menu|login|logout|register|mail_register|password_change|password_reset|t23|students_list) {
        include /etc/nginx/includes/proxy.conf;
        proxy_pass http://t23-gl-app-1:8080;
    }

    location ~ ^/(main|editor|report|t5) {
        include /etc/nginx/includes/proxy.conf;
        proxy_pass http://t5-app-1:8080;
    }

    location ^~ /tests {
        include /etc/nginx/includes/proxy.conf;
        proxy_pass http://prototipo20-t8_generazione-1:3080/tests;
    }

    location ~ ^/(robots|turns|games) {
        include /etc/nginx/includes/proxy.conf;
    }
}
```

Figura 22: Aggiornamento del file di configurazione (`default.conf`) dell' **UI Gateway** per inserire le rotte degli API endpoints che sono stati sviluppati nel corso del progetto.

4.1.2 Automazione del processo di compilazione e building dei container mediante task in VSCode

Per semplificare il flusso di lavoro, migliorare l'efficienza ed automatizzare tutte quelle attività ripetitive, nel nostro caso: la compilazione ed building dei container, si è deciso di adoperare i task in Visual Studio Code [9].

Ricordiamo che la versione attuale, per rendere effettive le modifiche apportate all'applicazione, richiede: **(a)** di posizionarsi nella shell all'interno della cartella del servizio modificato contenente i files `mvnw` e digitare il comando:

```
mvn package
```

dopodiché, **(b)** dopo essersi posizionati all'interno della cartella contenente il file `Dockerfile`, digitare il comando:

```
docker compose up -d --build
```

Per snellire una procedura tediosa come questa e che deve essere eseguita per ogni modifica apportata, accorrono in nostro aiuto i task, definibili nel file: `tasks.json` all'interno della cartella `.vscode` del progetto.

Per poterli eseguire è necessario rispettare i seguenti passaggi:

1. da VSCode importare l'intera "workspace folder" poiché la funzionalità di automazione del processo di deployment tramite task non è disponibile quando si lavora su files singoli; per verificare di aver eseguito correttamente la procedura, controllare nel menù "Explorer" (prima icona nella barra laterale sinistra), di essere in grado di visualizzare interamente il contenuto della cartella.

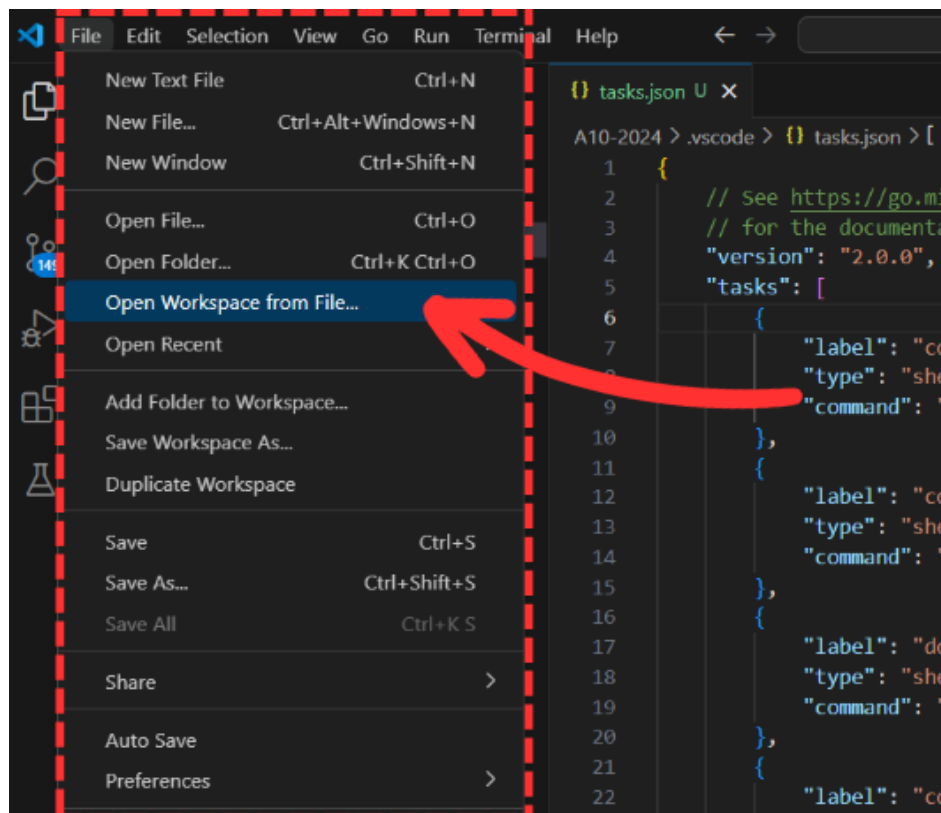


Figura 23: Configurazione task in VSCode

2. Posizionarsi con il cursore in corrispondenza del terminale e cliccare il pulsante: "Launch Profile...", dal menù appena comparso,

selezionare la voce: "Run Task..."³.

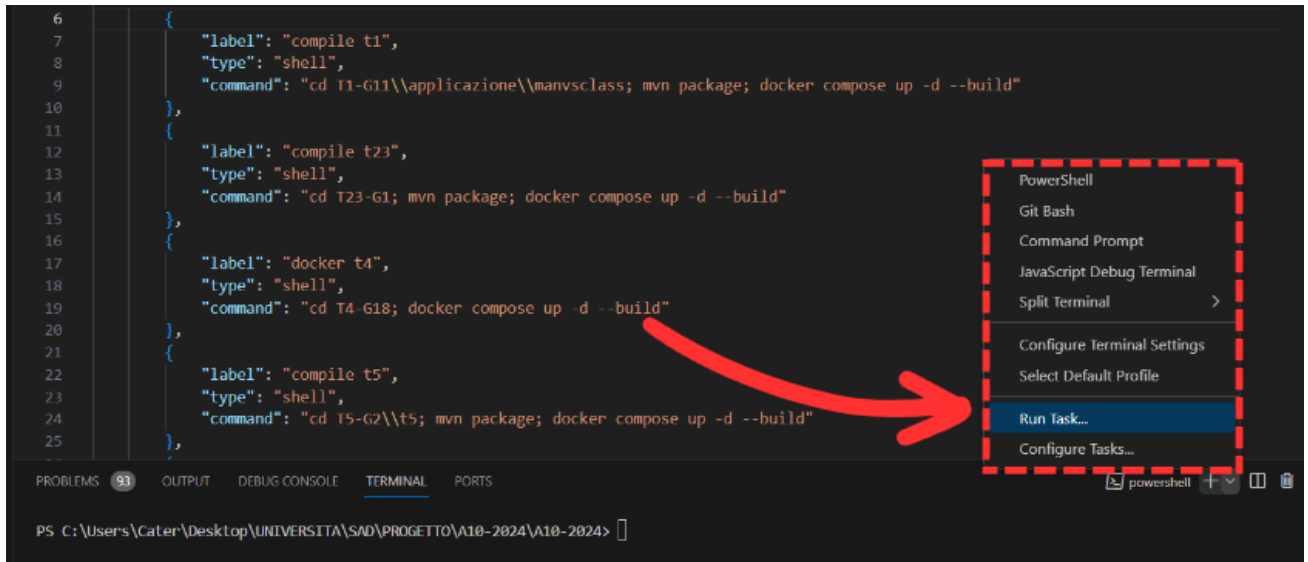


Figura 24: Esecuzione task in VSCode (1)

3. Dal menù a tendina, selezionare il task che si intende eseguire.

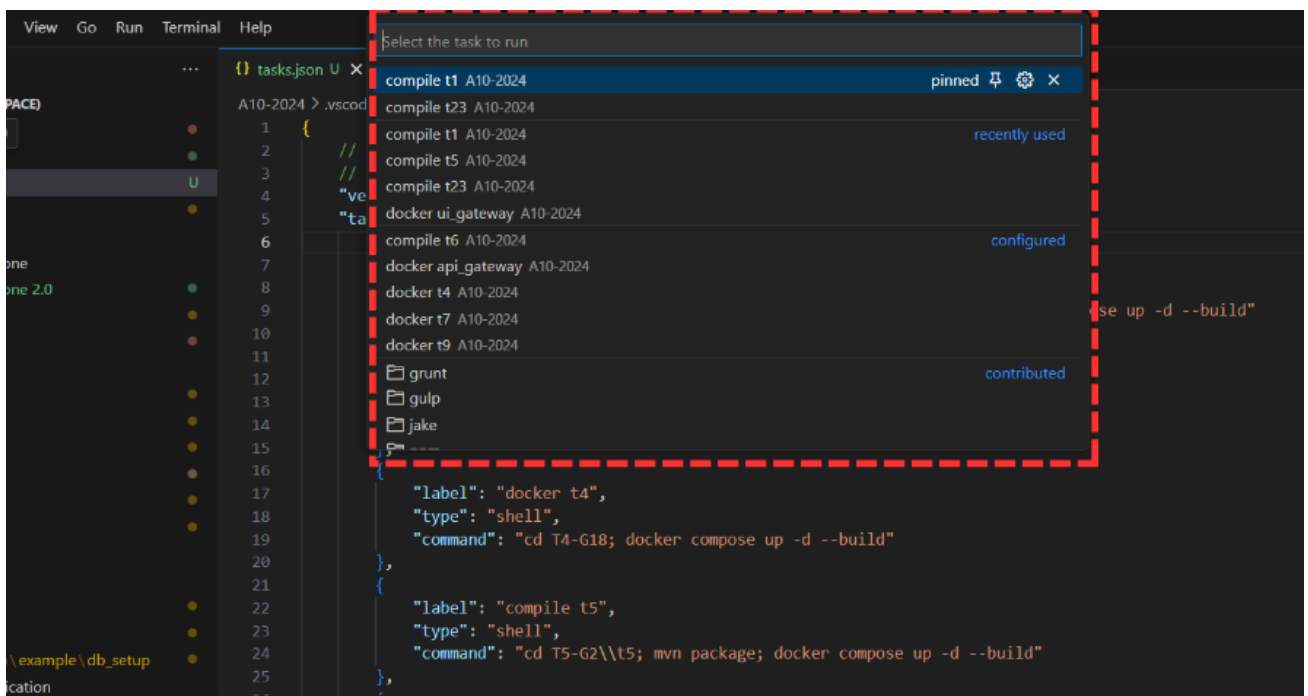


Figura 25: Esecuzione task in VSCode (2)

³Ricordarsi che ogni comando che coinvolge il Docker richiede che il container stesso sia in esecuzione!

```

{
// See https://go.microsoft.com/fwlink/?LinkId=733558
// for the documentation about the tasks.json format
"version": "2.0.0",
"tasks": [
  {
    "label": "compile t1",
    "type": "shell",
    "command": "cd T1-G11\\applicazione\\manvsclass;
      ↪ mvn package; docker compose up -d --build"
  },
  {
    "label": "compile t23",
    "type": "shell",
    "command": "cd T23-G1; mvn package; docker
      ↪ compose up -d --build"
  },
  {
    "label": "docker t4",
    "type": "shell",
    "command": "cd T4-G18; docker compose up -d --
      ↪ build"
  },
  {
    "label": "compile t5",
    "type": "shell",
    "command": "cd T5-G2\\t5; mvn package; docker
      ↪ compose up -d --build"
  },
  {
    "label": "compile t6",
    "type": "shell",
    "command": "cd T6-G12\\T6; mvn package; docker
      ↪ compose up -d --build"
  },
  {
    "label": "docker t7",

```

```

        "type": "shell",
        "command": "cd T7-G31\\RemoteCCC; docker compose
            ↪ up -d --build"
    },
    {
        "label": "docker t9",
        "type": "shell",
        "command": "cd T9-G19\\Progetto-SAD-G19-master;
            ↪ docker compose up -d --build"
    },
    {
        "label": "docker api_gateway",
        "type": "shell",
        "command": "cd api_gateway; docker compose up -d
            ↪ --build"
    },
    {
        "label": "docker ui_gateway",
        "type": "shell",
        "command": "cd ui_gateway; docker compose up -d
            ↪ --build"
    }
]
}

```

5 Testing

Per il testing di alcuni degli API endpoints aggiunti, si è deciso di fare affidamento ai potenti strumenti messi a disposizione dal programma **Postman** allo scopo di inviare richieste HTTP alle API e visualizzare agevolmente ed in maniera compatta, le risposte ottenute.

Eseguiamo, sostanzialmente, le seguenti verifiche:

1. Verifica della validità dei token JWT

Vogliamo verificare che solamente gli *amministratori* dotati di un token JWT valido abbiano la possibilità di accedere alle loro aree riservate (/home_adm, /class, /player, /info, /logout_admin, /admins_list, /invite_admins, /menu), in caso contrario, dovranno venir re-indirizzati alla pagina di login:

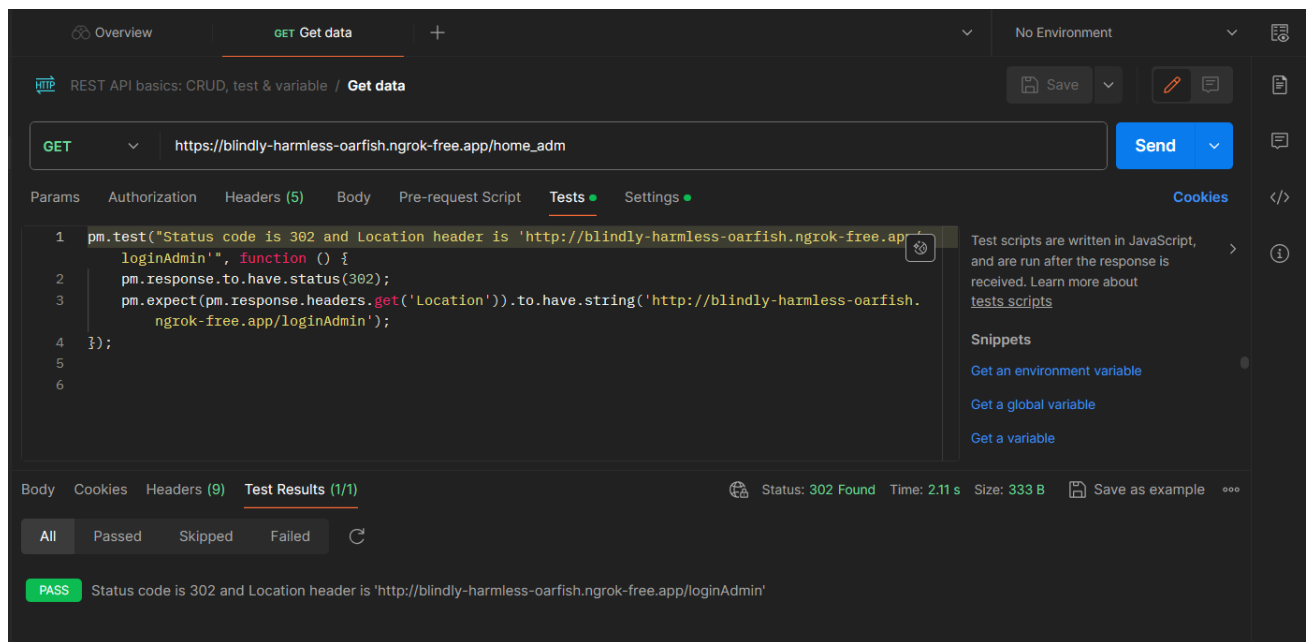


Figura 26: Validazione dei token JWT; solamente gli amministratori con dei token validi, avranno la possibilità di accedere presso le loro aree riservate, in caso contrario, verranno re-indirizzati (**codice di stato 302**) alla pagina di login (/loginAdmin).

2. Verifica della funzionalità: "Richiesta reset password"

Vogliamo verificare che solamente gli *amministratori* correttamente registrati e con dei token JWT invalidi, abbiano la possibilità di richiedere di impostare una nuova password, di conseguenza ci aspettiamo che, nel caso in cui l'e-mail inserita non dovesse venir trovata nel database, venga restituito un messaggio d'errore e che l'operazione non venga completata.

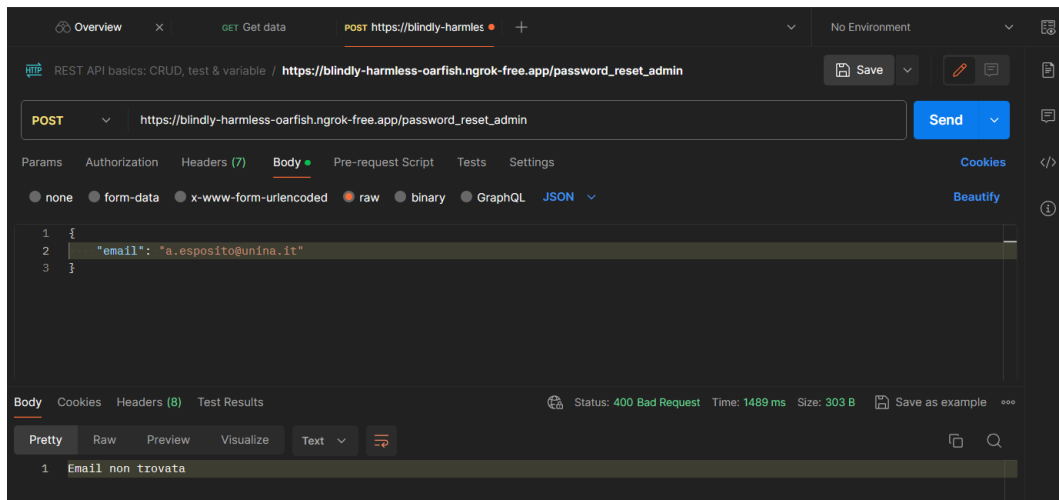


Figura 27: Verifica della funzionalità: **”Richiesta reset password”**; solamente gli amministratori correttamente registrati e con dei token JWT invalidi, avranno la possibilità di richiedere di impostare una nuova password, in caso contrario, se l’indirizzo di posta elettronica non dovesse venir trovato nel database, verrà restituito un messaggio d’errore (**codice di stato 400**) e l’operazione non viene completata.

3. Verifica della funzionalità: **”Invita admin”**

Vogliamo verificare che solamente gli *amministratori* correttamente registrati e con dei token JWT validi, abbiano la possibilità di invitare nuovi amministratori, a patto che l’e-mail inserita non venga trovata all’interno del database, in qual caso si andrà a restituire un messaggio d’errore e l’operazione non viene completata.

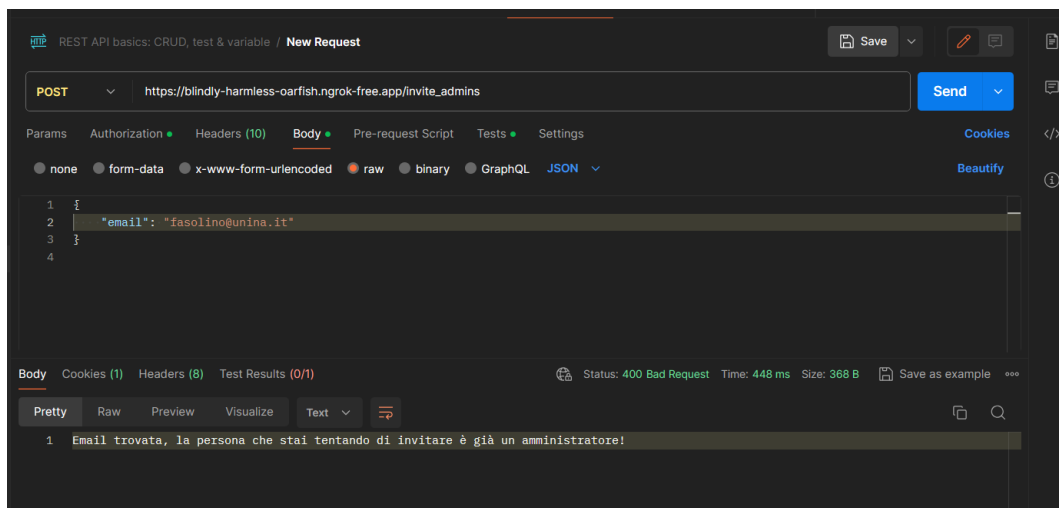


Figura 28: Verifica della funzionalità: **”Invita admin”**; solamente gli amministratori correttamente registrati e con dei token JWT validi, avranno la possibilità di invitare nuovi amministratori inserendo un indirizzo e-mail non già precedentemente memorizzato nel database, in caso contrario, se l’indirizzo di posta elettronica dovesse venir trovato nel database, verrà restituito un messaggio d’errore (**codice di stato 400**) e l’operazione non viene completata.

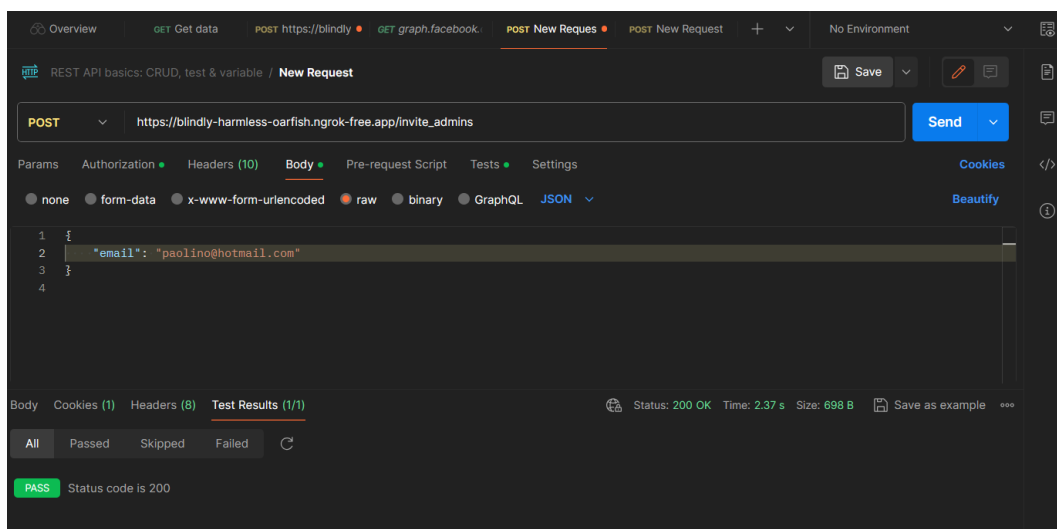


Figura 29: Verifica della funzionalità: **"Invita admin"**; solamente gli amministratori correttamente registrati e con dei token JWT validi, avranno la possibilità di invitare nuovi amministratori inserendo un indirizzo e-mail non già precedentemente memorizzato nel database (**codice di stato 200**) .

4. Controlli sugli input nel form di registrazione

In fase di registrazione, il sistema prevede: **(a)** la possibilità di inserire utenti con più nomi separati, eventualmente, da uno spazio, **(b)** la possibilità di inserire utenti con cognomi costituiti da più parole separate da spazi oppure apostrofi, **(c)** un meccanismo per distinguere gli amministratori dai normali giocatori accettando esclusivamente username del seguente formato: [username di lunghezza tra 2-30 caratteri]_unina.

Sarà possibile registrarsi inserendo: nome, cognome, username, e-mail istituzionale (@unina.it oppure @studenti.unina.it) e password. L'applicazione accetterà solamente password lunghe tra gli 8 e i 16 caratteri e contenenti: almeno una lettera maiuscola, una minuscola, un numero ed un carattere speciale e se la password inserita rispetterà tali criteri, non verrà memorizzata in chiaro all'interno del database, bensì crittata mediante un algoritmo di hashing.

Il sistema controllerà che tutti i campi del form di registrazione vengano compilati e che gli input siano validi, in caso di errore, visualizzerà a video un messaggio di errore. L'applicazione, prima di consentire la registrazione, controllerà la validità del token JWT assegnato all'utente, in caso di invalidità del token, si potrà procedere con l'operazione.

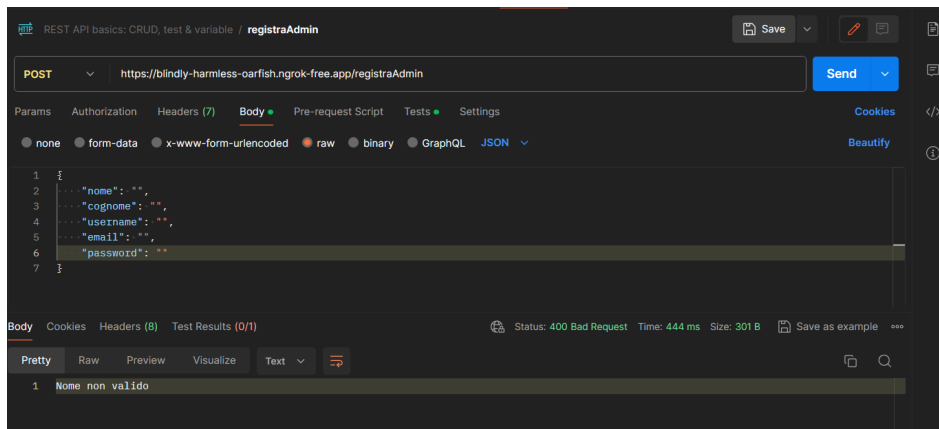


Figura 30: Verifica della funzionalità: **"Registra admin"**; Il sistema deve controllare che tutti i campi del form di registrazione vengano compilati e che gli input siano validi.

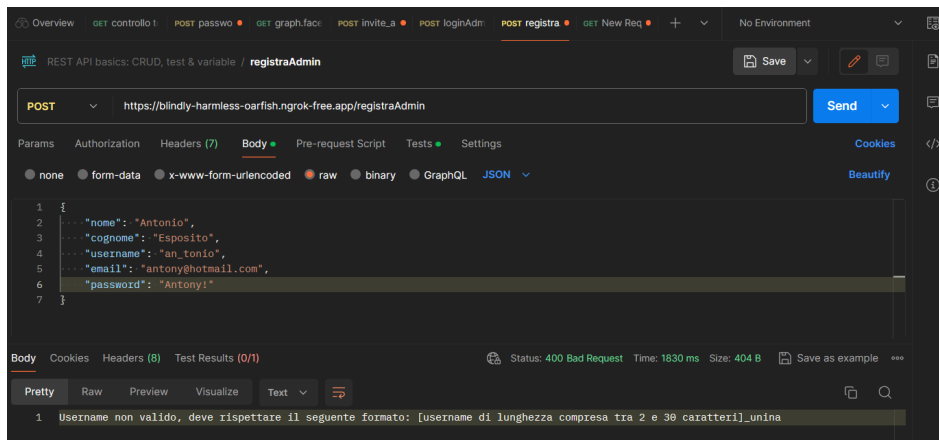


Figura 31: Verifica della funzionalità: **"Registra admin"**; Il sistema deve accettare esclusivamente l'indirizzo di posta elettronica istituzionale.

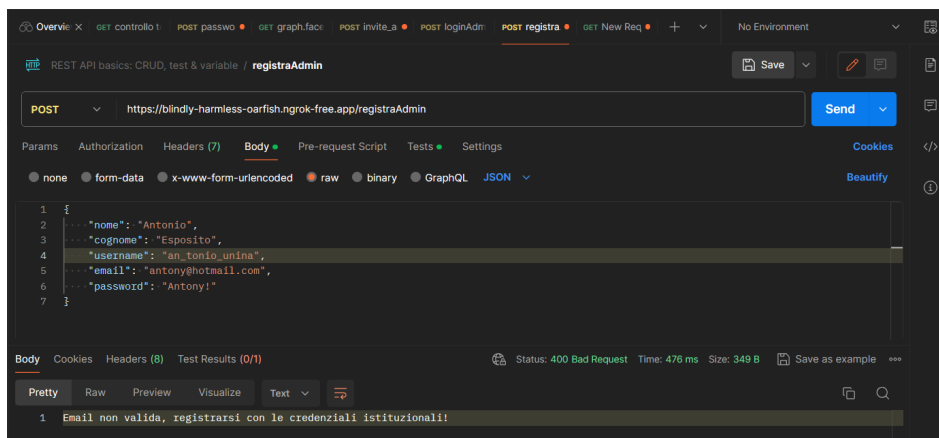


Figura 32: Verifica della funzionalità: **"Registra admin"**; Il sistema deve accettare esclusivamente username del seguente formato: [username di lunghezza tra 2-30 caratteri]_unina.

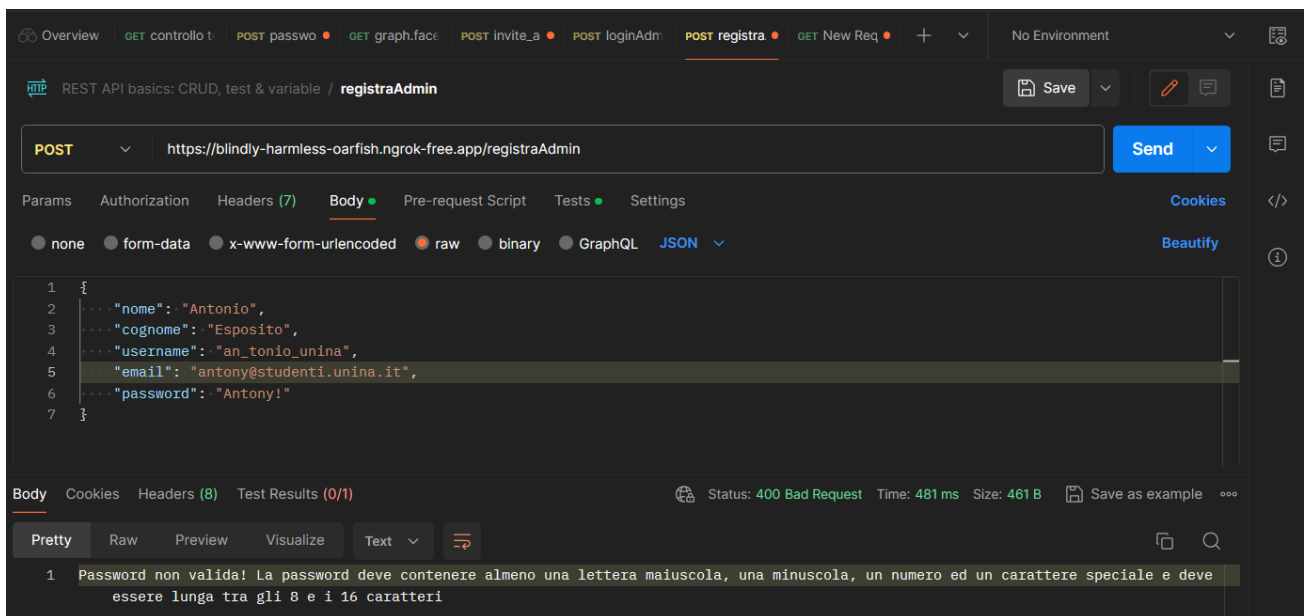


Figura 33: Verifica della funzionalità: **"Registra admin"**; Il sistema deve accettare esclusivamente password lunghe tra gli 8 e i 16 caratteri e contenenti: almeno una lettera maiuscola, una minuscola, un numero ed un carattere speciale

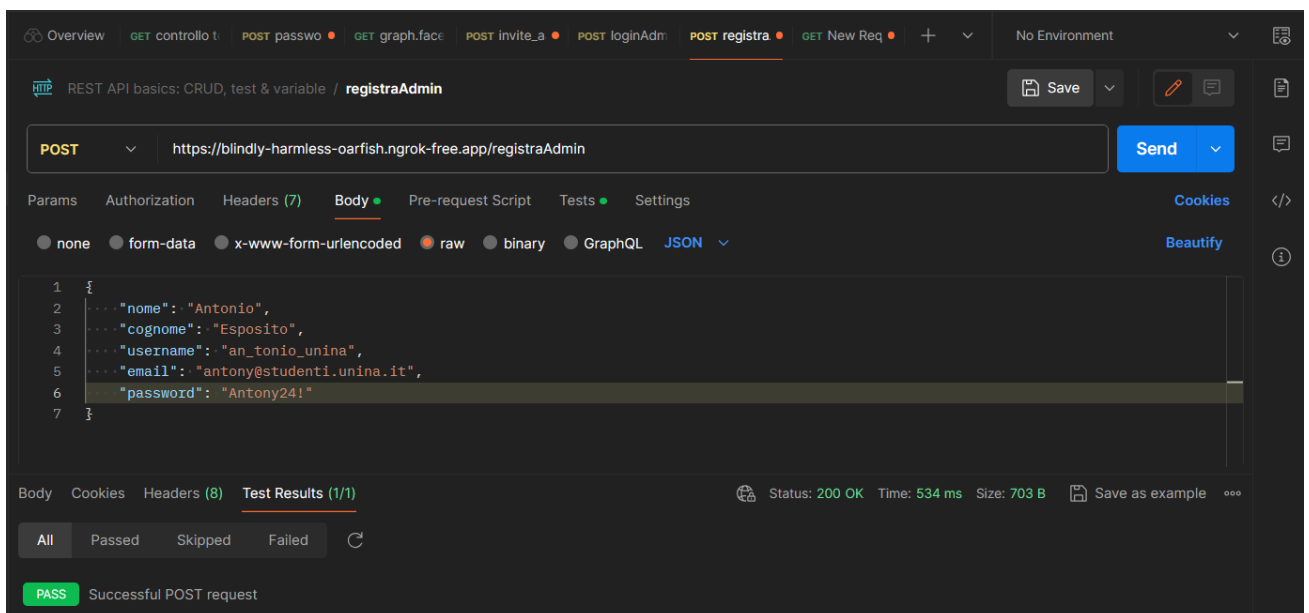


Figura 34: Verifica della funzionalità: **"Registra admin"**; registrazione avvenuta con successo

6 Guida per sviluppatori in erba

Questo spazio è interamente dedicato ai giovani valorosi che spinti dalla temerarietà e dalla paura dell'ignoto, sono riusciti a spingersi nella lettura fino a questo punto (innanzitutto complimenti!) e che probabilmente si staranno domandando: "e adesso?"; non temete, questa sezione vuole essere un baluardo, ma soprattutto un esempio pratico, di come approcciare alla progettazione, sviluppo ed implementazione di un requisito effettivo da integrare all'interno dell'applicazione.

Si è deciso di approfondire il caso d'uso: "**Registrati con Facebook**", ritenuto particolarmente interessante non solo perchè prevede l'interazione con API sviluppate da terze parti, ma anche perchè consente di comprendere appieno e di apprezzare, come Front-end e Back-end interagiscano tra di loro scambiandosi dati ed informazioni.

Articoleremo la trattazione in vari step ma prima di procedere e "sporcarci" le mani, bisogna che vi sia chiarezza: **(a)** riguardo l'architettura di riferimento e che **(b)** soprattutto si sappia a quale servizio "appartenga" il requisito che si sta tentando di dettagliare, per questo motivo, si invita caldamente alla lettura, o riletture, del [sottoparagrafo 1.1.1 \(Comprensione dell'architettura di riferimento\)](#).

Dopo questa doverosa premessa, analizziamo dettagliatamente il requisito proposto:

Task di appartenenza	Descrizione
T23	Il sistema deve consentire agli utenti di decidere se registrarsi tramite il classico form (e-mail + password) oppure effettuare il social login tramite Facebook e completare in questo modo la procedura di registrazione

Rispondiamo ad una serie di domande preliminari:

1. A cosa serve e perché può risultare utile integrare nella propria applicazione la funzionalità di social login?

Grazie al social login, gli utenti hanno la possibilità di creare un account all'interno della propria applicazione in maniera facile e veloce senza la necessità di dover impostare alcuna password che rischierebbe di venir dimenticata, senza contare il fatto che, così facendo, una volta creato un account su di una piattaforma, è possibile, anche in un solo click, accedere all'applicativo da qualsiasi altra piattaforma.

2. Perché è stato scelto proprio Facebook e non, per esempio, GitHub?

La risposta può venir sintetizzata nei seguenti tre punti principali:

- La soluzione proposta da Facebook è molto ben documentata: vengono descritti chiaramente e con molti esempi, tutti i passaggi necessari per integrare efficacemente il ”**Facebook login**” all’interno delle proprie applicazioni ed implementare, in questo modo, degli efficaci meccanismi di autenticazione ed autorizzazione; un aspetto tutt’altro da ignorare, è la possibilità di poter consultare una guida (ufficiale) [10] completamente in italiano, un enorme vantaggio nel caso in cui si volessero fugare eventuali dubbi di sorta.
- Facebook, così come Google, hanno sviluppato e diffuso delle speciali librerie JavaScript, i cosiddetti: **SDK**, per (a) accelerare, (b) ridurre la complessità dell’implementazione e (c) semplificare lo sviluppo delle applicazioni fornendo agli sviluppatori tutti gli strumenti e le risorse necessarie per implementare determinate funzionalità oppure per garantire l’interazione con determinati servizi e piattaforme.

Una soluzione di questo promuove il riuso e l’integrazione di funzionalità sviluppate da terze parti.



Suggerimento per future integrazioni

Per migliorare l’esperienza di utilizzo ed ingrandire il bacino d’utenza, prevedendo la possibilità che non tutti i giocatori che intendono registrarsi all’applicazione dispongano di un account Facebook, potrebbe essere utile integrare anche il social login tramite Google [11].

- Le API di GitHub, a differenza di quelle di Facebook e Google, non supportano l’impiego del protocollo di autenticazione ed autorizzazione: **OpenID Connect (OIDC)**, estremamente semplice e versatile ma soprattutto in grado di proteggere gli account degli utenti da potenziali attacchi malevoli.



Attenzione!

Se si volesse comunque procedere nell'implementazione di un social login tramite GitHub, bisognerebbe sfruttare il protocollo OAuth 2.0, che garantisce la sola autorizzazione, dopodiché implementare degli appositi meccanismi per abilitare l'autenticazione degli utenti ed evitare potenziali attacchi.

Considerato il ruolo di rilievo che viene ricoperto dal protocollo OpenID Connect (OIDC), ma soprattutto per comprendere il meccanismo di autorizzazione ed autenticazione di Facebook, è bene analizzare approfonditamente il suo funzionamento:

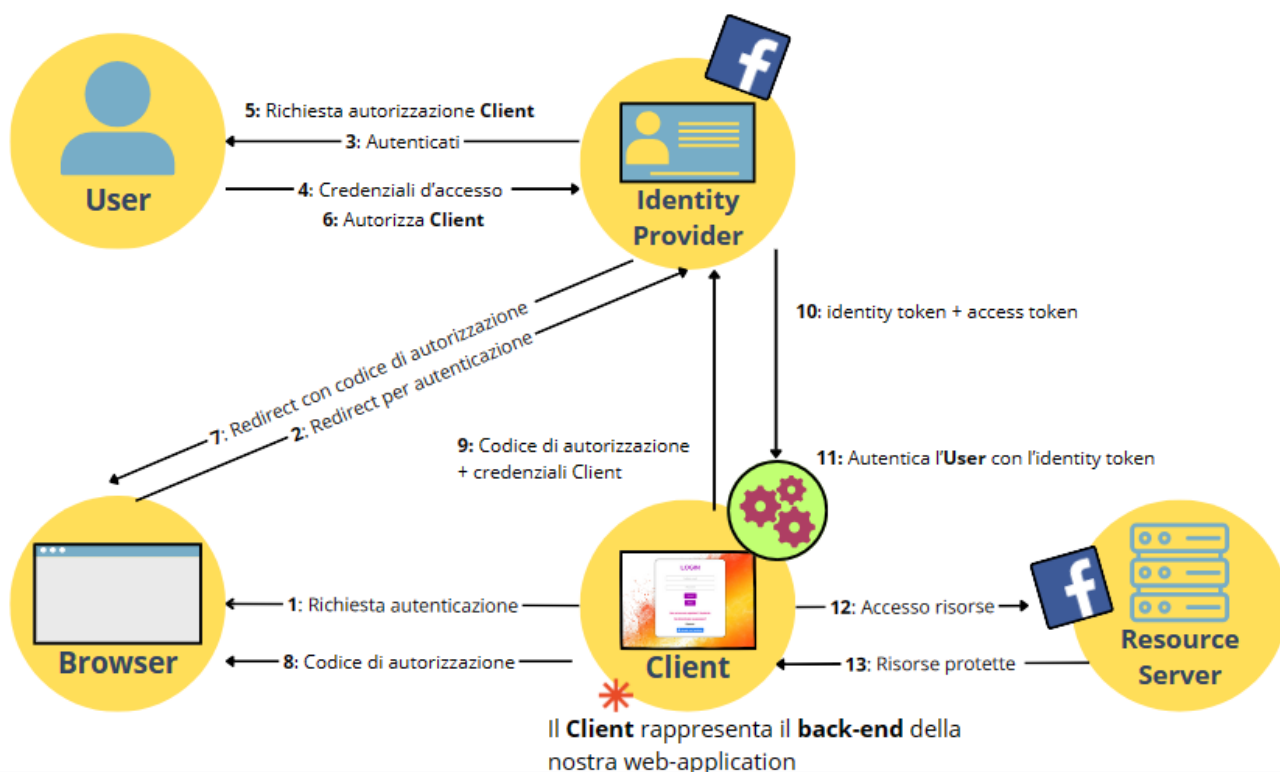


Figura 35: Flusso di autorizzazione del protocollo OIDC.

In un contesto di questo tipo, è bene definire adeguatamente che cosa si intende per: (a) **codice di autorizzazione (authorization code)**, un codice temporaneo e di breve durata che il Client fornisce all'Identity Provider in cambio di un token di accesso, e per (b) **token di accesso (access token)**, un codice che consente all'applicativo di richiedere a Facebook, in questo caso, informazioni sull'User. E' bene anche sottolineare, per evitare che si crei confusione, che gli elementi: **Identity Provider** e **Resource Server**, qui presentate come entità separate, in realtà sono entrambe delle API di Facebook; la prima avrà come mansione quella di restituirci un token di accesso mentre il compito della seconda, sulla scorta dell'access token appena recuperato, di eseguire azioni per conto dell'utente come, ad esempio, richiedere informazioni riguardo il suo profilo.

Abbiamo anche chiacchierato abbastanza, come si fa ad implementare effettivamente questo Facebook Login ?

1. Creazione account sviluppatore di Facebook

Come suggerisce la documentazione, il primo step per poter utilizzare: servizi, funzionalità, API e chi più ne ha più ne metta, forniti da Meta[™], è quello di registrarsi come sviluppatore.

2. Creazione di una applicazione

L'obiettivo è creare una applicazione alla quale verrà associato un identificativo univoco (ID dell'app).

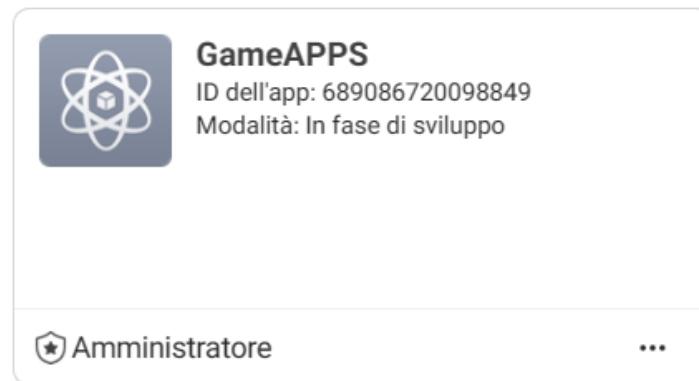


Figura 36: La nostra web-app e relativo ID

3. Aggiunta dell'SDK di Facebook per JavaScript all'interno nella nostra pagina web

Inseriamo, come script, in fondo alla pagina HTML preesistente di login (login.html), il seguente stralcio di codice:

Listing 2: SDK di Facebook per JavaScript inserito in una sezione di scripting in login.html

```
(function(d, s, id){  
  var js, fjs = d.getElementsByTagName(s)[0];  
  if (d.getElementById(id)) {return;}  
  js = d.createElement(s); js.id = id;  
  js.src = "https://connect.facebook.net/it-IT/sdk.js";  
  fjs.parentNode.insertBefore(js, fjs);  
}(document, 'script', 'facebook-jssdk'));
```

4. Abilitazione dell'SDK JavaScript per il Facebook Login

All'interno della *Dashboard gestione app*, abilitare le seguenti impostazioni:

Casi d'uso > Personalizza > Impostazioni

Impostazioni di Facebook Login

Personalizza il tuo caso d'uso in modo che la tua app funzioni correttamente in base alle tue esigenze.

[Indietro](#)

Impostazioni del client OAuth

<input checked="" type="checkbox"/> Si	Accesso client OAuth Abilita il flow standard del token client OAuth. Proteggi la tua app ed evitane l'uso improprio bloccando gli URI di reindirizzamento dei token consentiti con le opzioni di seguito. Disabilita completamente se non lo usi. [?]
<input checked="" type="checkbox"/> Si	Accesso OAuth web Abilita l'accesso OAuth client basato sul web. [?]
<input type="checkbox"/> No	Forza la riautenticazione OAuth Web Se l'opzione è attiva, le persone sono chiamate a inserire la password Facebook per accedere al Web. [?]
<input checked="" type="checkbox"/> Si	Usa la modalità con limitazioni per gli URI di reindirizzamento Consenti solo reindirizzamenti che corrispondono esattamente agli URI di reindirizzamento OAuth validi. Fortemente consigliato. [?]
<input checked="" type="checkbox"/> Si	Applica HTTPS Applica l'uso di HTTPS per gli URI di reindirizzamento e l'SDK JavaScript. Fortemente consigliato. [?]
<input type="checkbox"/> No	Accesso OAuth al browser incorporato Abilita gli URI di reindirizzamento della visualizzazione web per l'accesso OAuth client. [?]

Figura 37: Assicurarsi di abilitare il flusso del client OAuth per garantire la funzionalità di autenticazione e proteggere, in questo modo, gli account degli utilizzatori della web-app.



Attenzione!

Di default, l'integrazione del Facebook login all'interno delle proprie applicazioni, ed in particolare le azioni di autenticazione mediante SDK, richiedono che venga obbligatoriamente applicato il protocollo HTTPS e qui sorgono i primi problemi, infatti, come abbiamo già osservato nei capitoli precedenti ([per maggiori informazioni clicca qui](#)), la versione attuale dell'applicazione, comunica in rete mediante il protocollo HTTP.

Il problema può venire arginato, recuperando, a partire dal container (`flamboyant_tu`) all'interno del quale è stato installato il servizio di tunneling **Ngrok**, un indirizzo "sicuro" ([vedi Figura 17 alla pagina seguente](#)):


```
ngrok (Ctrl+C to quit)

Take our ngrok in production survey! https://forms.gle/aXlBFHvzEA36DudFn6

Session Status      online
Account             Caterina Maria Accetto (Plan: Free)
Update              update available (version 3.6.0, Ctrl-U to update)
Version             3.5.0
Region              Europe (eu)
Latency              29ms 4-03-01T12:36:48.482619667Z
Web Interface        http://0.0.0.0:4848
Forwarding            https://blindly-harmless-oarfish.ngrok-free.app -> http://localhost:80

Connections          ttl    opn    rt1    rt5    p50    p90
                    2      0      0.00   0.00   65.76  66.28

HTTP Requests
-----
GET /t23/css/login.css      200
GET /login                  200
GET /t23/css/images/login_page_background_yellow.jpg 200
GET /favicon.ico            404 Not Found
GET /login                  200
GET /t23/css/login.css      200
GET /t23/css/images/login_page_background_yellow.jpg 200
```

Figura 38: Migrazione dall'indirizzo HTTP (`http://localhost:80`) verso un indirizzo HTTPS (`https://blindly-harmless-oarfish.ngrok-free.app`), indispensabile per poter abilitare il social login tramite Facebook

Per completare l'abilitazione, non bisognerà far altro che: **(a)** inserire l'URL di re-indirizzamento, sulla scorta dell'indirizzo appena recuperato, nell'apposito campo per garantire una corretta autorizzazione, **(b)** specificare che si sta utilizzando l'SDK di JavaScript per effettuare l'accesso ed infine **(c)** definire il dominio della pagina ospitante l'SDK all'interno della lista dei domini consentiti; quest'ultima operazione garantisce che tutti i token d'accesso vengano restituiti solamente a callback appartenenti alla lista dei domini consentiti.

URI di reindirizzamento OAuth validi

Il parametro `redirect-uri` specificato manualmente e usato con l'accesso sul web deve corrispondere esattamente a uno degli URI indicati qui. Questo elenco è usato anche dall'SDK JavaScript per i browser in-app che bloccano i pop-up. [?]

`https://blindly-harmless-oarfish.ngrok-free.app/main` ✕

Copia negli appunti

☐ No Accesso dai dispositivi
Abilita il flusso di accesso del client OAuth per dispositivi come smart TV [?]

☒ Si Accedi con l'SDK JavaScript
Abilita l'accesso e la funzionalità di accesso con l'SDK JavaScript. [?]

Domini consentiti per l'SDK JavaScript

L'accesso e le funzionalità di accesso dell'SDK JavaScript saranno disponibili solo su questi domini. [?]

`https://blindly-harmless-oarfish.ngrok-free.app/` ✕

Figura 39: Completamento procedura di abilitazione dell'SDK JavaScript per il Facebook Login

5. Inizializzazione SDK

In coda allo script mostrato nel [Listing 2](#), inserire il seguente stralcio di codice per poter inizializzare l'SDK:

Listing 3: Inizializzazione dell'SDK di Facebook inserito in una sezione di scripting in login.html

```
window.fbAsyncInit = function() {  
    //Inizializzazione dell'SDK  
    FB.init({  
        appId : '689086720098849', //ID della propria  
            ↪ applicazione  
        xfbml : false, //analisi DOM per trovare ed  
            ↪ inizializzare qualsiasi plug-in social  
            ↪ usando XFBML  
        cookie : true,  
        version : 'v19.0' //versione del Graph API  
    });  
  
    FB.getLoginStatus(function(response) {  
        statusChangeCallback(response);  
        console.log(response.authResponse.accessToken);  
    });  
  
    FB.AppEvents.logPageView();  
  
    FB.login(function(response) {  
        if (response.authResponse) {  
            console.log('Welcome! Fetching your  
                ↪ information.... ');  
  
            //Recupero nome ed email tramite GET all'  
            ↪ API di Facebook  
            FB.api('/me', {fields: 'name, email'},  
                ↪ function(response) {  
                    document.getElementById("profile").  
                        ↪ innerHTML = "Good to see you, " +  
                        ↪ response.name + ". i see your  
                        ↪ email address is " + response.  
                        ↪ email  
                    console.log(JSON.stringify(response));  
                });  
        } else {  
            console.log('User cancelled login or did  
                ↪ not fully authorize.');        }  
    });  
  
    FB.logout(function(response) {  
        //Utente sloggato e ricaricamento della pagina  
        location.reload();  
    });  
};
```

6. Verifica dello stato di accesso di un utente

Una volta aver inizializzato l'SDK per JavaScript, verrà chiamata la funzione: `FB.getLoginStatus` per recuperare lo stato d'accesso dell'utente che sta attualmente visitando la web-app nel browser.

E' possibile che venga restituito uno dei seguenti tre possibili stati:

- `status: 'connected'` connesso nella web-app
- `status: 'not_authorized'` accesso effettuato su Facebook ma non sulla web-app
- `status: 'unknown'` mancato accesso su Facebook, non si sa se l'utente si sia "loggato" nella web-app.

Listing 4: Verifica dello stato d'accesso di un utente

```
//Lo stato d'accesso viene gestito nella funzione di callback
FB.getLoginStatus(function(response) {
    statusChangeCallback(response);
    console.log(response.authResponse.accessToken);
});
```

7. Gestione dello stato d'accesso di un utente

La funzione `statusChangeCallback` viene chiamata con la risposta restituita da `FB.getLoginStatus`:

Listing 5: Gestione dello stato d'accesso (1)

```
//statusChangeCallback viene chiamata con i risultati restituiti da
↪ FB.getLoginStatus
function statusChangeCallback(response) {
    console.log('statusChangeCallback');
    console.log(response);
}
```

Si controlla se l'utente sia connesso oppure no, in caso affermativo (`response.status==='connected'`), si recupera il token di accesso e si effettua una chiamata all'API di Facebook per ottenere una serie di informazioni sull'utente come : il nome e l'e-mail:

Listing 6: Gestione dello stato d'accesso (2)

```

//Siamo loggati?
if(response.status==='connected') {
  //se lo stato 'connected', i parametri authResponse
  ↪ vengono inclusi automaticamente nella risposta
  const access_token = response.authResponse.accessToken;

  //Richiesta all'API per recuperare email e nome
  FB.api('/me', {fields: 'name, email'}, function(
    ↪ response) {
    //document.getElementById("profile").
    ↪ innerHTML = "Good to see you, " +
    ↪ response.name + ". i see your
    ↪ email address is " + response.
    ↪ email
    console.log(JSON.stringify(response));
  }
}

```

Le informazioni appena recuperate (token di accesso, nome ed indirizzo di posta elettronica), vengono inviate al Back-end, il nostro Controller, rimasto all'oscuro delle operazioni effettuate dagli utenti (bisogna tenere presente che arrivati a questo punto, solamente il Front-end è a conoscenza dello stato d'accesso dell'utente) tramite una richiesta POST presso un URL specifico:

Listing 7: Gestione dello stato d'accesso (3)

```

//Invio dati back-end (POST): accessToken + email + nome
//Dati da inviare nella richiesta POST (da convertire in formato
↪ JSON)

const data = {
  email: response.email,
  nome: response.name,
  access_token: access_token
};

// Costruzione URL che includa i parametri di interesse (nome,
↪ email ed access_token)
const url = 'https://blindly-harmless-
↪ oarfish.ngrok-free.app/
↪ login_with_facebook?nome=${data.
↪ nome}&email=${data.email}&
↪ access_token=${data.access_token}'

//Configurazione della richiesta

```

```

const options = {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
    // Specifica il tipo di contenuto
    ↪ come JSON
  },
  redirect: 'follow'
  // Segui il reindirizzamento
};

// Esecuzione della richiesta POST
↪ utilizzando la fetch()
fetch(url, options)
  .then(response => {
    if (!response.ok) {
      throw new Error('Errore nella
        ↪ richiesta POST');
    }
    location.reload()
  })
  .catch(error => {
    console.error('Si e' verificato un
      ↪ errore:', error);
  });
});
}
}

```

Le informazioni inviate per mezzo del metodo POST (nome, e-mail ed access_token), dovranno venire adeguatamente gestite dal Backend (**Controller.java**) per garantire che la procedura di login con Facebook venga eseguita correttamente:

```

@PostMapping("/login_with_facebook")
public ResponseEntity<String> login_with_facebook(
    ↪ @RequestParam("email") String email,
    ↪ @RequestParam("nome") String name,
    ↪ @RequestParam("access_token") String tokenFb,
    ↪ @CookieValue(name = "jwt", required = false)
    ↪ String jwt, HttpServletRequest request,
    ↪ HttpServletResponse response) {

    //Contattare FB e validare il token d'accesso
    //Invio GET presso end-point debug-token

    // URL dell'endpoint a cui inviare la richiesta
    ↪ GET
    String url = "https://graph.facebook.com/
    ↪ debug_token?input_token="+tokenFb+"&
    ↪ access_token="+app_token;

    // Esegue la richiesta GET e ottiene la risposta
    ↪ come oggetto ResponseEntity<String>
    ResponseEntity<String> login_with_facebook =
    ↪ restTemplate.getForEntity(url, String.class
    ↪ );

    String responseBody = null;
    boolean is_valid = false;

    // Verifica lo stato della risposta
    if (login_with_facebook.getStatusCode() ==
    ↪ HttpStatus.OK) {
        // Richiesta andata a buon fine, puoi
        ↪ accedere ai dati della risposta
        responseBody = login_with_facebook.getBody();
        System.out.println("Risposta ricevuta:");
        System.out.println(responseBody);
    } else {
        // Gestisci il caso in cui la richiesta non

```

```

        ↪ sia andata a buon fine
    System.out.println("Errore nella richiesta: "
        ↪ + login_with_facebook.getStatusCode())
        ↪ ;
}

//Deserializzare risposta
ObjectMapper objectMapper = new ObjectMapper();
System.out.println("Deserializzazione JSON...");

try {
    // Converti il corpo della risposta in un
    ↪ oggetto Java (es. MyResponseClass)
    MyResponseClass responseObj = objectMapper.
        ↪ readValue(responseBody, MyResponseClass
        ↪ .class);

    // Ora puoi accedere ai campi dell'oggetto
    ↪ responseObj
    is_valid = responseObj.getData().isIs_valid()
        ↪ ;

    // Esegui le operazioni desiderate con i dati
    ↪ della risposta
    System.out.println("is_valid: " + is_valid);

} catch (IOException e) {
    e.printStackTrace();
}

//Token valido?
if (is_valid==true) {

    System.out.println("Token valido");

    //Ti sei gia' registrato?
    User user = userRepository.findByEmail(email)
        ↪ ;

```

```

if(user != null) {

    //Utente esiste (mail trovata nel database
    ↪ )
    System.out.println("Utente gia' registrato
    ↪ (mail trovata nel database)");

    //Si e' gia' registrato con Facebook?

    if(user.isRegisteredWithFacebook){

        System.out.println("Utente registrato
        ↪ con Facebook");
        //Flusso JWT
        String token = generateToken(user);
        AuthenticatedUser authenticatedUser =
            ↪ new AuthenticatedUser(user,
            ↪ token);
        authenticatedUserRepository.save(
            ↪ authenticatedUser);
        System.out.println("authenticatedUser
            ↪ correttamente creato (
            ↪ login_with_facebook)");

        Cookie jwtTokenCookie = new Cookie("
            ↪ jwt", token);
        jwtTokenCookie.setMaxAge(3600);
        response.addCookie(jwtTokenCookie);
        System.out.println("Cookie aggiunto
            ↪ alla risposta (
            ↪ login_with_facebook)");

        try {
            response.sendRedirect("/main");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```



```

    } else {
        //Non si e' registrato con Facebook
        System.out.println("Utente non
            ↳ registrato con Facebook");
        return ResponseEntity.status(
            ↳ HttpStatus.BAD_REQUEST).body("Ti
            ↳ sei gia' registrato con email e
            ↳ password. Nella pagina di login
            ↳ , inserisci le tue credenziali."
            ↳ );
    }
} else {
    //Utente non si e' mai registrato
    System.out.println("Utente non si e' mai
        ↳ registrato");

    //Registrazione Utente
    System.out.println("Registrazione Utente
        ↳ ...");
    User n = new User();
    n.setName(name);
    n.setSurname("");
    n.setEmail(email);
    n.setPassword("");
    n.setRegisteredWithFacebook(true);
    n.setStudies(Studies.ALTRO);

    //Salvataggio
    System.out.println("Salvataggio...");
    userRepository.save(n);
    System.out.println("Salvataggio completato
        ↳ .");

    //Assegnazione ID
    Integer ID = n.getID();

    try {
        emailService.sendMailRegister(email,

```

```

        ↪ ID);
//Flusso JWT
String token = generateToken(n);
AuthenticatedUser authenticatedUser =
    ↪ new AuthenticatedUser(n, token);
authenticatedUserRepository.save(
    ↪ authenticatedUser);

Cookie jwtTokenCookie = new Cookie("
    ↪ jwt", token);
jwtTokenCookie.setMaxAge(3600);
response.addCookie(jwtTokenCookie);
try {
    response.sendRedirect("/main");
} catch (IOException e) {
    return ResponseEntity.status(
        ↪ HttpStatus.
        ↪ INTERNAL_SERVER_ERROR).body("
        ↪ Failed to confirm your
        ↪ registration");
    }
} catch (MessagingException e) {
    return ResponseEntity.status(
        ↪ HttpStatus.INTERNAL_SERVER_ERROR
        ↪ ).body("Failed to confirm your
        ↪ registration");
    }
}
} else {
    //token non valido, utente non loggato
    ↪ correttamente con facebook
}

return ResponseEntity.status(302).body("");
}

```

8. Connessione di un utente

Nel caso in cui un'utente dovesse aprire la pagina web dal browser e contestualmente non avesse ancora effettuato l'accesso su Facebook, gli è concessa la possibilità di autenticarsi in-app cliccando sul bottone: "Accedi con Facebook"; la pressione del bottone darà luogo ad una finestra di dialogo dove poter completare la procedura di autenticazione inserendo le credenziali associate al proprio account Facebook.

Listing 9: Connessione di un utente

```
//La funzione checkLoginState() viene invocata non appena viene  
  ↳ cliccato  
//il bottone: "Accedi con Facebook"  
function checkLoginState() {  
    FB.getLoginStatus(function(response) {  
        statusChangeCallback(response);  
    });  
}
```



Figura 40: Procedura di autenticazione tramite Facebook

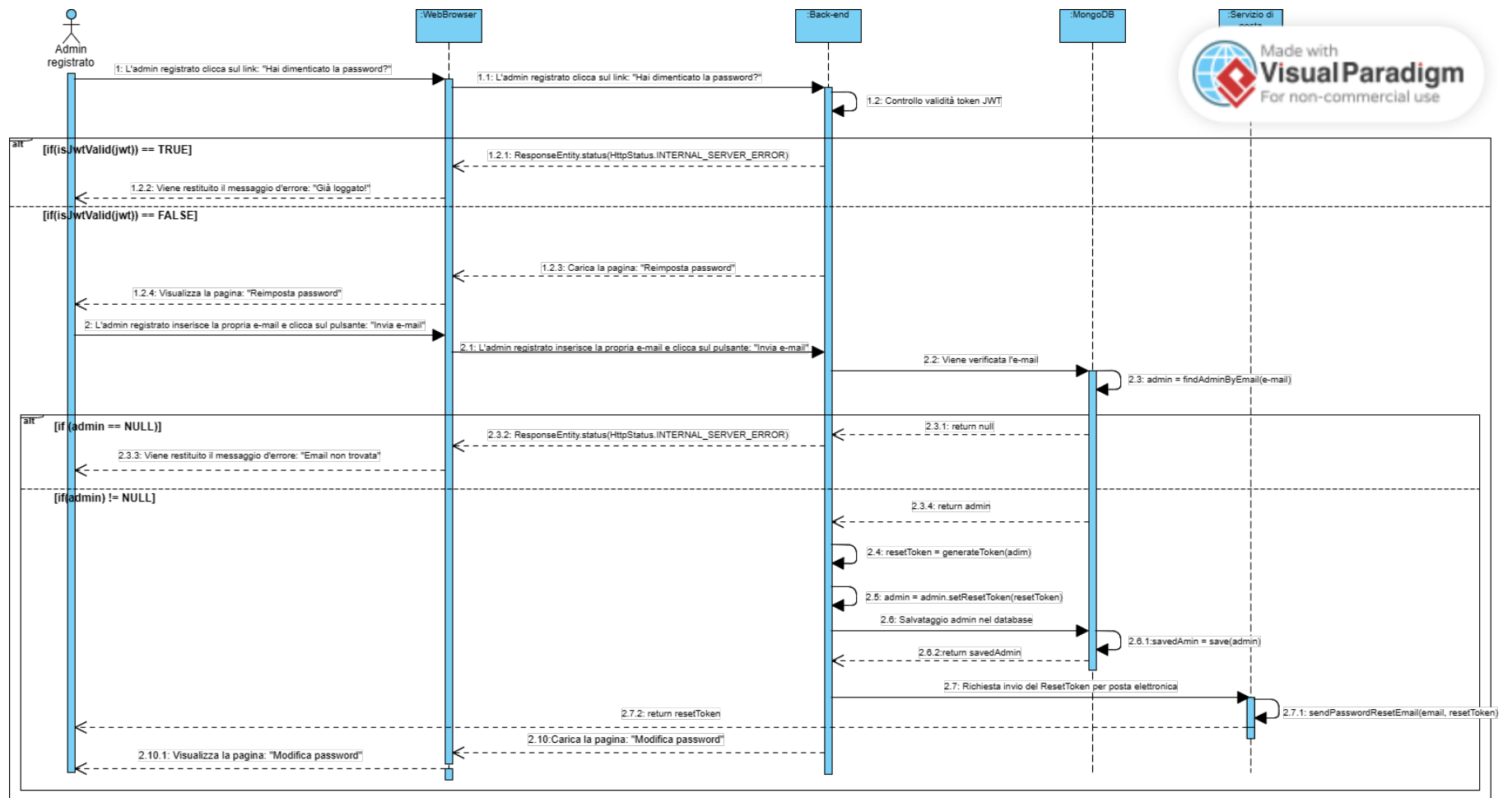


Figura 41: Sequence diagram relativo il caso d'uso: "Richiesta reset password"

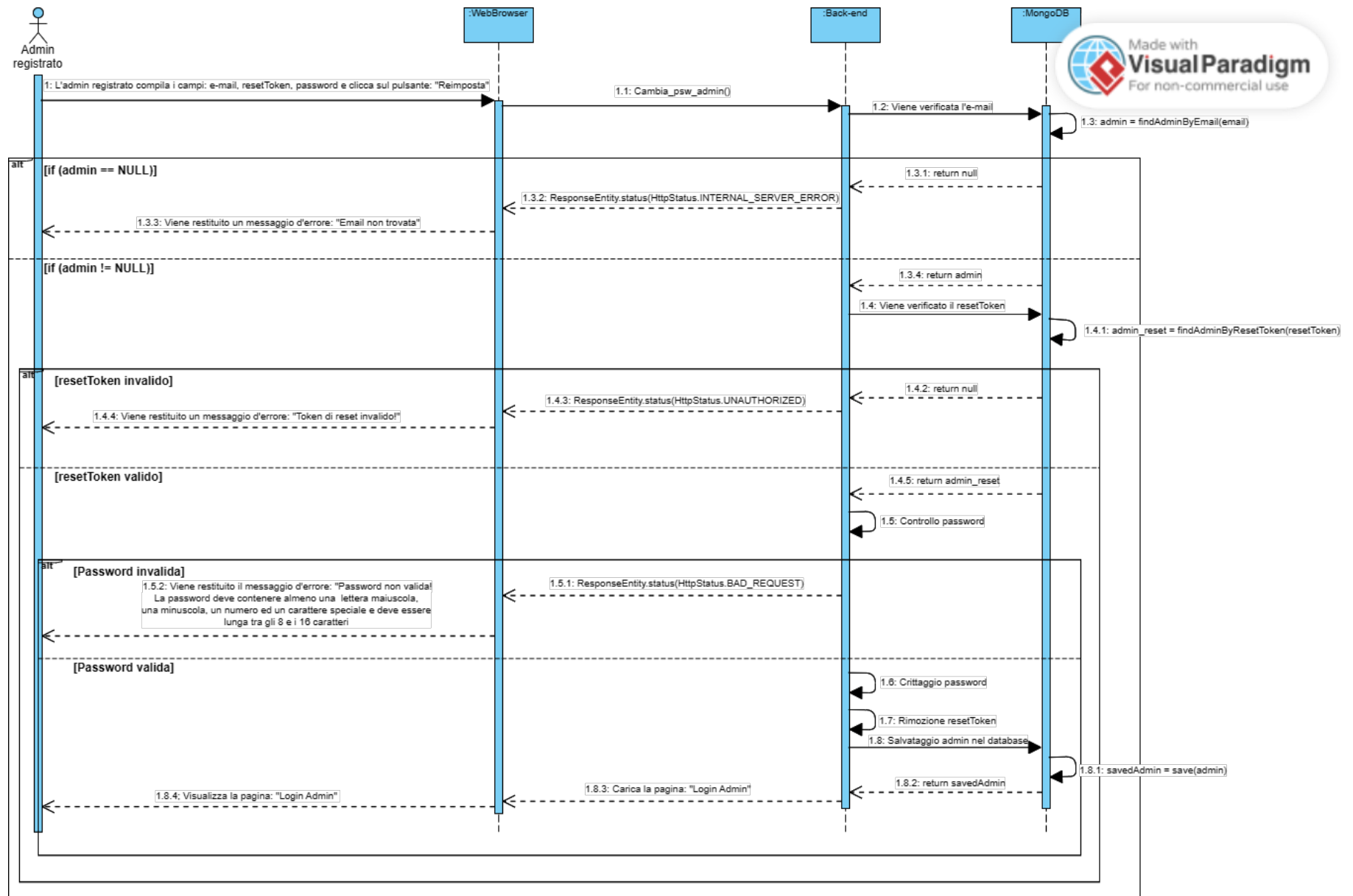


Figura 42: Sequence diagram relativo il caso d'uso: "Modifica password"

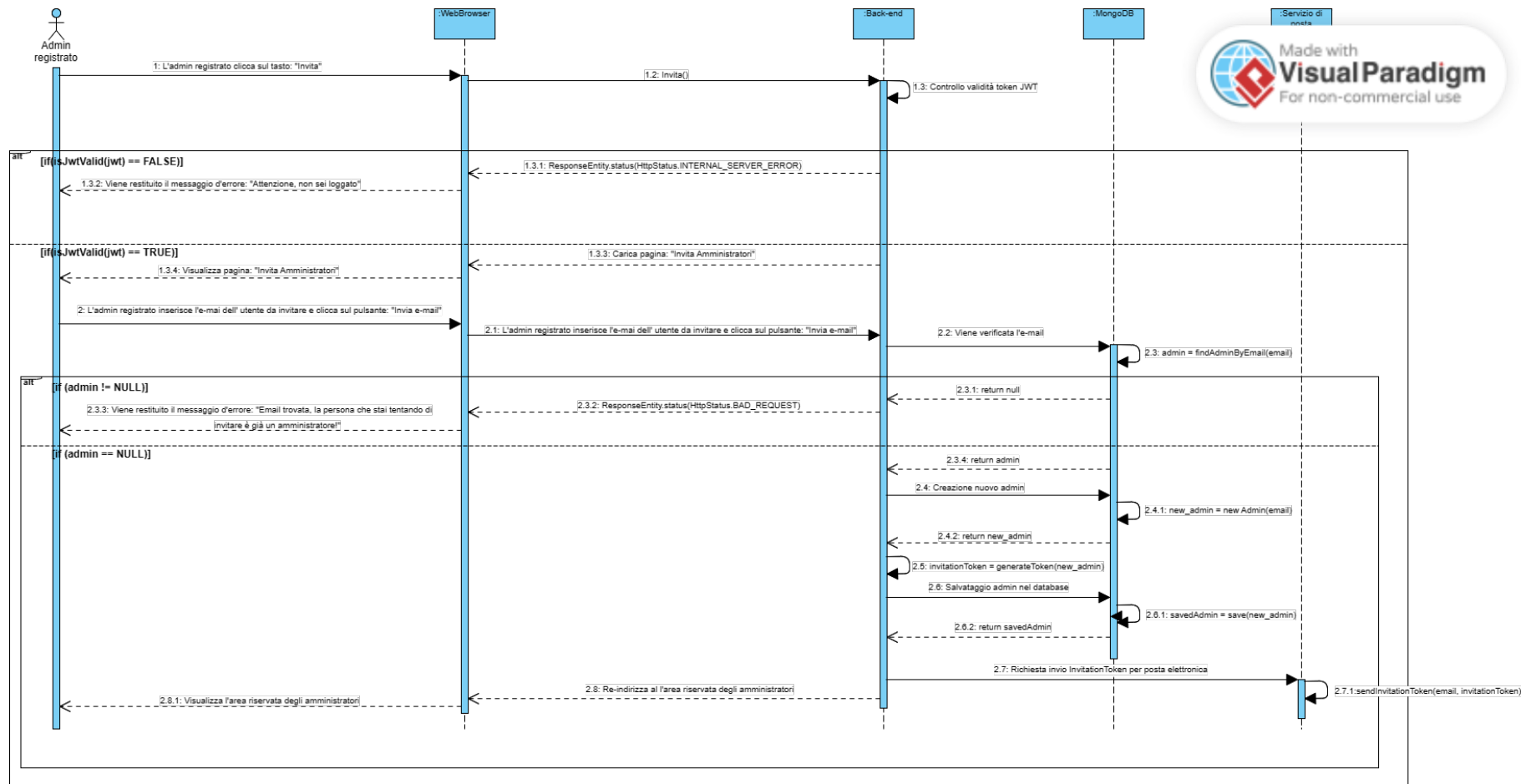


Figura 43: Sequence diagram relativo il caso d'uso: **"Invita admin"**

Riferimenti bibliografici

- [1] <https://enactest-project.eu/it/>.
- [2] <https://it.wikipedia.org/wiki/Gamification>.
- [3] <https://github.com/Testing-Game-SAD-2023>.
- [4] <https://github.com/Testing-Game-SAD-2023/A10-2024>.
- [5] <https://kinsta.com/it/knowledgebase/api-endpoint/>.
- [6] <https://datatracker.ietf.org/doc/html/rfc7519/>.
- [7] <https://github.com/Testing-Game-SAD-2023/A10-2024/blob/main/T1-G11/documentazione/Documentazione%20T1%20G11.pdf>.
- [8] https://github.com/Testing-Game-SAD-2023/A10-2024/blob/main/T23-G1/G1_T2_T3.pdf.
- [9] <https://code.visualstudio.com/Docs/editor/tasks/>.
- [10] <https://developers.facebook.com/docs/facebook-login/>.
- [11] <https://developers.google.com/identity/sign-in/web/sign-in?hl=it/>.