

Progettazione e Sviluppo di Sistemi Software: "Smart Learning"

Caso Lorenzo - Mat. M63/885
Farina Giorgio - Mat. M63/861
Fasano Francesco - Mat. M63/929
Giamattei Luca - Mat. M63/825

25 ottobre 2020

Indice

1	Processo di sviluppo	1
1.1	Releases	3
2	Avvio del progetto	5
2.1	Introduzione	5
2.1.1	Scopo del Progetto	5
2.1.2	Ambito del Progetto	6
2.2	Descrizione Generale	6
2.2.1	Classi Utente e Caratteristiche	6
2.2.2	Ambiente Operativo	6
2.2.3	Documentazione	6
2.2.4	Dipendenze	6
3	Specifiche dei requisiti	7
3.1	Interfacce Utente	7
3.2	Requisiti Funzionali	8
3.3	Requisiti Non Funzionali	9
4	Analisi dei requisiti	10
4.1	Casi d'uso	10
4.1.1	Descrizione degli Scenari	13
4.2	Activity Diagrams	21
4.3	System Domain Model	26
5	Architettura Software	29
5.1	Introduzione	29
5.2	Component Diagram	29
5.3	Application Server	31
5.3.1	Package Diagrams	32
5.3.2	Architettura Three-Layer	33
5.4	Client	35
5.5	Database Server	35
5.6	Servizi esterni: Janus	35
5.7	Deployment Diagram	36

6	Implementazione, configurazione ed esecuzione	38
6.1	Implementazione	38
6.1.1	Application Server	38
6.1.2	Database	41
6.2	Implementazione Client ed Esecuzione	43
6.3	Configurazione	47
7	Janus	49
7.1	Introduzione	49
7.2	Interazione con Janus	49
7.2.1	Gestione degli accessi al Server Janus	49
7.2.2	Gestione degli accessi alla Room	50
7.2.3	Mobile-Code (Code-on-demand)	51
7.3	Plugin VideoRoom	51
7.3.1	Publish - Subscribe	52
7.4	Script di riposizionamento	54
8	Testing	56
8.1	Data Layer	56
8.2	REST API	57

Capitolo 1

Processo di sviluppo

Per lo sviluppo dell'applicazione web Smart Learning si è utilizzato un processo di sviluppo di tipo **UP**, o **Unified Process**. Infatti, tale metodologia riconosce lo sviluppo del software come un'attività guidata dalla definizione dei requisiti funzionali, espressi attraverso i casi d'uso. L'analisi dei casi d'uso permette di definire le caratteristiche dell'architettura software che li realizza in modo integrato. Da questo punto di vista, l'UP è centrato sull'architettura, in quanto, una volta stabilita quella di base, il progetto procede per incrementi che integrano le funzionalità individuate nell'analisi.

Il progetto, seguendo lo Unified Process, attraversa diverse fasi, le quali possono essere reiterate più volte:

Fase	Numero di iterazioni	Inizio	Fine
Ideazione	1	Settimana 1	Settimana 2
Elaborazione	1	Settimana 2	Settimana 3
Costruzione	2	Settimana 3	Settimana 5
Transizione	2	Settimana 4	Settimana 6

Tabella 1.1: Distribuzione temporale del processo di sviluppo

Ogni fase è a sua volta attraversata da diversi flusso di lavoro, che vengono eseguiti ad ogni iterazione, ed inoltre essa si deve concludere con la realizzazione di determinati obiettivi (**milestone**):

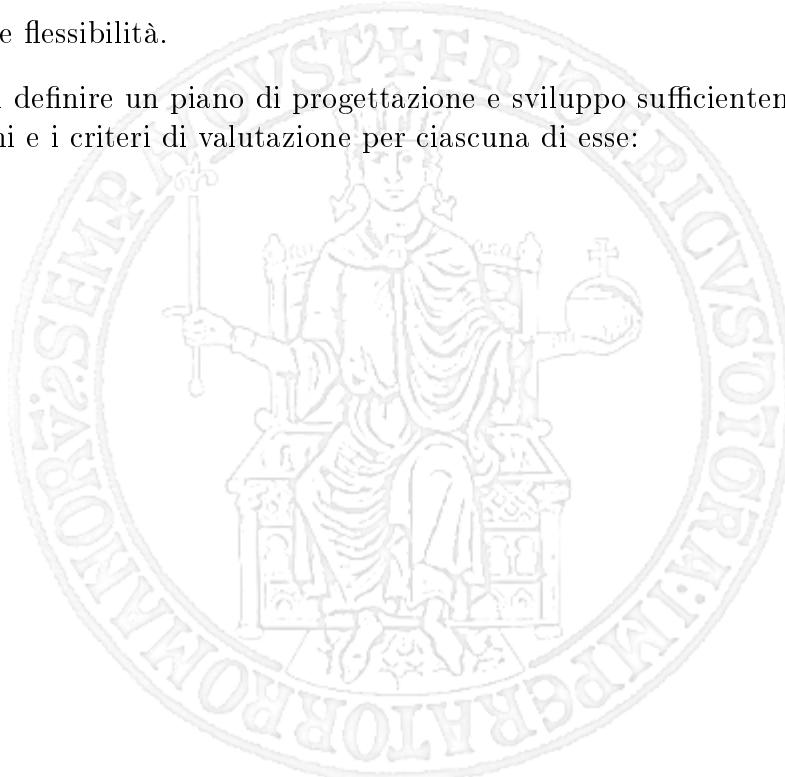
Fase	Descrizione	Milestone
Ideazione	Analisi di fattibilità, modellazione dei primi casi d'uso (20%), definizione dei requisiti, risk assessment	Studio di fattibilità, Use Case Diagram, Specifica dei requisiti
Elaborazione	Identificazione attori e casi d'uso (80%), definizione dei requisiti non funzionali, definizione piano di progettazione e sviluppo, analisi del dominio, progettazione dell'architettura	Prototipo architettonico, Piano di progettazione e sviluppo
Costruzione	Sviluppo software, integrazione e testing, documentazione dello sviluppo	Prodotto software stabile e utilizzabile da end-users
Transizione	Installazione, esecuzione ed analisi, bug fixes	Utenti soddisfatti

Tabella 1.2: Descrizione del processo di sviluppo

La suddivisione in iterazioni presenta numerosi vantaggi, quali:

- Valutare progressivamente l'avanzamento del progetto;
- Gestire al miglior modo i fattori rischio;
- Fornire rapidamente una versione operativa del sistema;
- Avere maggiore flessibilità.

Si è quindi deciso di definire un piano di progettazione e sviluppo sufficientemente dettagliato da mostrare le iterazioni e i criteri di valutazione per ciascuna di esse:



Fase	Iterazione	Descrizione	Criteri di valutazione
Ideazione	Iterazione preliminare	Analisi di fattibilità (architettura a microservizi e monolitica, tecnologie per scambio flussi multimediali - Janus WebRTC); Definizione requisiti e casi d'uso	Il contenuto dei casi d'uso principali è una rappresentazione accurata di ciò che il software fornirà; Fattibilità
Elaborazione	E1 Iteration – Develop Architectural Prototype	Analisi, design e sviluppo prototipo architettonico per R1	Prototipo architettonico stabile e approvato; Principali elementi di rischio identificati e risolti in modo credibile; Progetti e piani di sviluppo sufficientemente dettagliati, accurati e credibili
Costruzione	C1 Iteration – Develop R1 Release	Sviluppo R1, integrazione e testing; Analisi e design R2	R1 stabile e utilizzabile da end-users; Piano di sviluppo per R2 sufficientemente dettagliato ed approvato
	C2 Iteration – Develop R2 Release	Sviluppo R2, integrazione e testing;	R2 stabile e utilizzabile da end-users
Transizione	T1 Iteration – R1 Release	Deploy e bug fixes R1	Soddisfacimento per il set di funzionalità e il loro funzionamento
	T2 Iteration – R2 Release	Deploy e bug fixes R2	Soddisfacimento per il set di funzionalità e il loro funzionamento

Tabella 1.3: Descrizione del processo di sviluppo con iterazioni

1.1 Releases

Il documento descrive le prime due releases del sistema. Per la gestione delle release è stato utilizzato **GitHub**. Le funzionalità di base sono state implementate nella **prima release (R1)**, la quale include:

- La registrazione di un utente;
- L'upgrade di un profilo utente ad un profilo di tipo docente;
- Il login;

- La creazione di una lezione da parte di un docente;
- L'aggiunta di una programmazione per una determinata lezione da parte di un docente;
- La possibilità per un docente di visualizzare le lezioni create e le relative programmazioni;
- La ricerca di una lezione basata sul cognome del docente o sul topic della lezione.
- La prenotazione di una lezione da parte di un utente;
- La possibilità di visualizzare le lezioni prenotate da parte di un utente;

La **seconda release (R2)**, d'altra parte, include le seguenti funzionalità aggiuntive:

- L'avvio della videochiamata relativa ad una lezione da parte del docente;
- La possibilità di entrare in una videochiamata avviata da parte di un utente prenotato.
- La possibilità di terminare una lezione da parte di un docente (tale comando termina la videochiamata per tutti i partecipanti);
- La possibilità di terminare la videochiamata da parte di un utente generico;
- La condivisione simultanea di più schermi all'interno della stessa videochiamata;
- La possibilità di spegnere il microfono e la videocamera all'interno di una videochiamata.

Per quanto riguarda le **releases future**, invece, si è ipotizzato di progettare le seguenti funzionalità:

- Includere il sistema di pagamento PayPal per effettuare le prenotazioni;
- La possibilità da parte di un utente di recensire una lezione alla quale ha partecipato;
- Associare ad ogni docente uno “score” basato sulle recensioni ricevute ed una pagina contenente tali recensioni;
- Fornire all'utente la possibilità di esprimere una preferenza per un “topic” (tale funzionalità è stata implementata solo parzialmente, ovvero sono state implementate le API a livello Data Layer);
- Fornire ai docenti, quando creano la lezione, la possibilità di visualizzare i topic preferiti dagli utenti (tale funzionalità è stata implementata solo parzialmente, ovvero sono state implementate le API a livello Data Layer);
- Un sistema di notifiche tramite e-mail che:
 - Invii una notifica all'utente nel caso in cui un docente crei una lezione relativa ad uno dei topic per i quali ha espresso la propria preferenza;
 - Invii una notifica all'utente quando un docente avvia la videochiamata relativa ad una lezione prenotata;
- Aggiunta di un meccanismo di autenticazione e autorizzazione basato su JWT Token.

Capitolo 2

Avvio del progetto

2.1 Introduzione

2.1.1 Scopo del Progetto

Lo scopo del progetto è realizzare una piattaforma web per la gestione di videolezioni online private. Tale piattaforma interagirà con due tipologie di utenti:

- Gli **studenti** che potranno partecipare alle lezioni, dopo averle prenotate tramite un pagamento;
- I **docenti** che potranno creare e tenere le lezioni.

Gli utenti, per effettuare una qualsiasi operazione, devono loggarsi al sistema. Per ottenere un account bisogna registrarsi fornendo nome, cognome, indirizzo e-mail e password.

Inizialmente qualsiasi utente verrà registrato come uno studente, tuttavia è possibile effettuare l'upgrade ad un profilo di tipo docente in qualsiasi momento tramite l'apposita pagina, sarà necessaria solamente inserire un indirizzo e-mail associato ad un account PayPal e inserire il proprio Curriculum Vitae.

Il login si può effettuare inserendo l'e-mail di registrazione e la password.

I docenti possono creare una lezione specificandone il titolo, una descrizione, il topic e il numero massimo di studenti che vi possono partecipare. In seguito, egli potrà aggiungere più programmazioni per quella determinata lezione, specificandone data, orario di inizio, orario di fine e prezzo.

Gli studenti potranno ricercare una lezione tramite il cognome del docente o il topic al quale sono interessati. Trovata una lezione d'interesse, essi potranno valutare il docente che l'ha creata tramite le recensioni che egli ha ricevuto. Una volta che essi avranno scelto una lezione, vi si potranno prenotare, effettuando il pagamento tramite il servizio esterno **PayPal**.

D'altra parte un docente, potrà consultare una lista dei topic preferiti dagli utenti della piattaforma. Egli inoltre, nell'orario e nel giorno stabiliti per la lezione avvierà una videochiamata attraverso **Janus**, anch'esso un servizio esterno. Il sistema provvederà quindi ad inviare una notifica a tutti gli utenti che hanno prenotato tale lezione.

All'interno della videochiamata tutti gli utenti potranno disattivare microfono e videocamera ed inoltre sarà permesso loro di condividere lo schermo (anche più utenti contemporaneamente).

Una volta terminata la lezione gli studenti potranno recensirla, assegnando un punteggio (da 0,0 a 5,0) e scrivendo un commento.

2.1.2 Ambito del Progetto

Il sistema nasce con l'intento di facilitare gli studenti, soprattutto in ambito universitario, nella ricerca di un aiuto didattico valido e certificato (tramite Curriculum e recensioni). Inoltre, l'utilità di tale piattaforma aumenta se si considera il periodo storico che stiamo attraversando.

D'altra parte, gli studenti non saranno gli unici a trarre vantaggio dalla piattaforma, i docenti, infatti, potranno utilizzarla per lavorare da casa.

2.2 Descrizione Generale

2.2.1 Classi Utente e Caratteristiche

Il sistema è costituito da due classi utente: Studente e Docente. Entrambe dovranno registrarsi al sistema ed effettuare il login per usufruire dei suoi servizi. In particolare:

- **Studente:** utilizza il sistema per ricercare e selezionare una lezione, effettuando una prenotazione a pagamento (tramite PayPal) per parteciparvi. Dopo aver partecipato alla lezione (tramite Janus), ha la possibilità di recensirla.
- **Docente:** utilizza il sistema per creare una lezione riguardante un “topic”, avviando la videochiamata relativa alla lezione nella data e nell'orari prefissati. D'altra parte un utente di tipo Docente avrà, a differenza degli studenti, un account PayPal associato al proprio profilo, tramite il quale gli saranno accreditati i pagamenti, un Curriculum Vitae e una pagina contenente un punteggio e le recensioni delle lezioni effettuate.

2.2.2 Ambiente Operativo

Il sistema sarà una Web Application eseguito su un server **Tomcat 9.0**, deployato in cloud sulla piattaforma **Microsoft Azure**.

2.2.3 Documentazione

La documentazione allegata al presente documento sarà la seguente:

- Archivio contenente i diagrammi UML riguardanti il sistema (è stata effettuata una suddivisione in cartelle per favorire lo sviluppo in parallelo);
- File di documentazione per le interfacce REST (standard OpenAPI 3.0);
- File di documentazione Javadoc per le interfacce del Data Layer dell'Application Server.

2.2.4 Dipendenze

Il sistema presenta le seguenti dipendenze da sistemi esterni:

1. **Janus:** è un Server WebRTC, tecnologia open source che consente ai browser di effettuare in tempo reale la videochat;
2. **PayPal:** offre servizi di pagamento digitale e di trasferimento di denaro tramite Internet (release futura).

Capitolo 3

Specifiche dei requisiti

3.1 Interfacce Utente

Il sistema sarà costituito dalle seguenti interfacce:

- Utente non loggato:
 - Interfaccia per effettuare la registrazione;
 - Interfaccia per effettuare il login.
- Studente:
 - Interfaccia per effettuare l'upgrade ad un profilo docente;
 - Interfaccia per effettuare la ricerca di una lezione;
 - Interfaccia per visualizzare le lezioni cercate;
 - Interfaccia per visualizzare il profilo di un docente;
 - Interfaccia per effettuare una prenotazione;
 - Interfaccia per la gestione di tutte le lezioni prenotate;
 - Interfaccia videochiamata;
 - Interfaccia per sottoscrivere un “topic” di interesse;
 - Interfaccia per recensire una lezione.
- Docente:
 - Interfaccia per creare una lezione;
 - Interfaccia per la gestione di tutte le lezioni create;
 - Interfaccia per visualizzare le programazioni di una lezione;
 - Interfaccia per aggiungere una programmazione;
 - Interfaccia videochiamata.

3.2 Requisiti Funzionali

Sono stati elaborati i seguenti Requisiti Funzionali:

- **Registrazione:** l'utente per ottenere le credenziali di accesso deve registrarsi fornendo i propri dati (nome, cognome, e-mail e password) e verrà registrato come studente;
- **Login:** l'utente che intende utilizzare il sito deve inserire le sue credenziali di accesso (e-mail e password);
- **Registrazione come docente:** lo studente registrato inserisce l'indirizzo e-mail associato ad un account PayPal e il proprio Curriculum Vitae, per poter accedere alle funzioni dedicate ai docenti;
- **Crea lezione:** il docente crea una lezione specificandone il titolo, una descrizione, il topic e il numero massimo di studenti che vi possono partecipare;
- **Aggiungi programmazione:** il docente seleziona una lezione e ne aggiunge una programmazione, specificandone data, orario di inizio, orario di fine e prezzo;
- **Visualizza lezioni create:** il docente può visualizzare tutte le lezioni che ha creato;
- **Cerca lezione:** lo studente effettua una ricerca tra le lezioni create in base al cognome del docente o al topic della lezione;
- **Prenota lezione:** lo studente prenota una lezione alla quale è interessato;
- **Visualizza profilo docente:** lo studente può visualizzare il profilo di un docente, contenente le lezioni che ha creato e lo score derivato dalle sue recensioni;
- **Visualizza lezioni prenotate:** lo studente può visualizzare tutte le lezioni che ha prenotato;
- **Avvia videochiamata:** il docente avvia la videochiamata relativa ad una lezione nel giorno e nell'orario programmato;
- **Partecipa a videochiamata:** lo studente si unisce ad una videochiamata relativa ad una lezione che ha prenotato;
- **Gestisci videochiamata:** l'utente che partecipa ad una videochiamata può disattivare il proprio microfono e la propria videocamera, oltre a poter condividere lo schermo;
- **Termina lezione:** il docente termina la videochiamata avviata per tutti i partecipanti;
- **Termina videochiamata:** lo studente termina la videochiamata avviata per se stesso;
- **Recensione lezione:** lo studente, al termine di una videochiamata, ha la possibilità di recensire una lezione attraverso uno "score" e una descrizione;
- **Sottoscrivi topic:** lo studente può esprimere una preferenza riguardante alcuni topic, in modo da essere notificato qualora un docente creasse una lezione relativa a quell'argomento.

3.3 Requisiti Non Funzionali

Sono stati elaborati i seguenti Requisiti non Funzionali:

- **Sicurezza:**
 - Il sistema deve assicurare che un utente non loggato non possa accedere alle funzionalità del sistema;
 - Il sistema deve assicurare che un utente con profilo studente non possa accedere alle funzionalità dedicate ai docenti.
- **Usabilità:** il tempo di formazione necessario ad un utente inesperto per poter usare produttivamente il sistema è di circa 30 minuti. Mentre il numero di iterazioni per svolgere le varie funzioni sono limitate rispetto ad applicativi di simile utilizzo.
- **Scalabilità, Manutenibilità e Riusabilità:** tali attributi di qualità sono garantiti dagli approcci architetturali utilizzati.



Capitolo 4

Analisi dei requisiti

4.1 Casi d'uso

Il documento dei casi d'uso individua e descrive gli scenari elementari di utilizzo del sistema da parte degli attori che si interfacciano con esso (esseri umani oppure da sistemi informativi esterni). I casi d'uso del sistema sono rappresentati dal seguente Use Case Diagram:



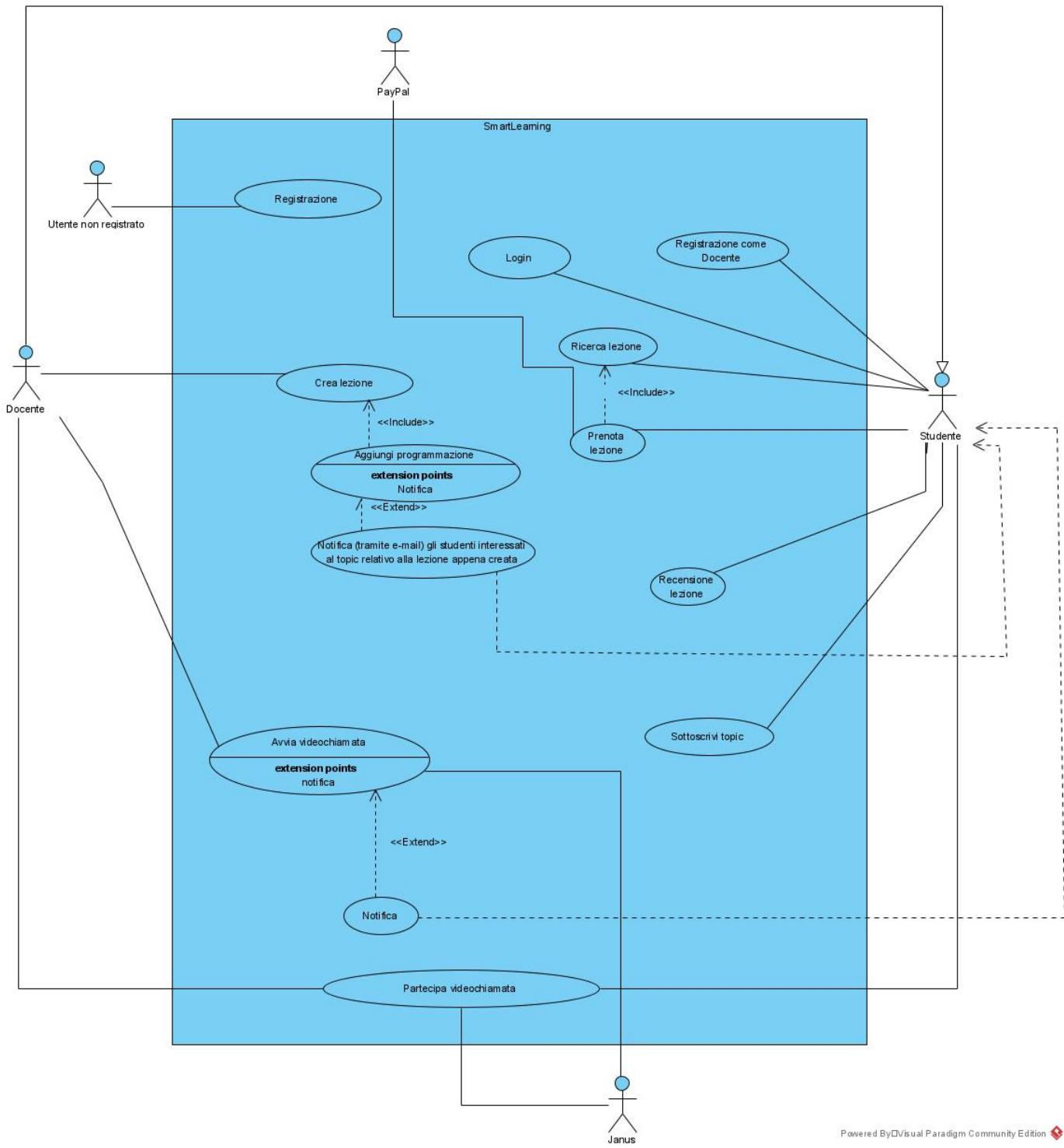


Figura 4.1: Use Case Diagram

Powered By Visual Paradigm Community Edition

Dall'analisi dei casi d'uso sono stati individuati i seguenti **attori**:

1. Utente non registrato: un utente il quale interagisce per la prima volta con il sistema e, di conseguenza, ancora deve compiere la registrazione. Tale utente, dopo aver effettuato la registrazione, diventa uno “Studente”.

2. Studente: un utente che interagisce con il sistema per usufruire delle lezioni messe a disposizione dai docenti presenti in esso. In particolare, egli potrebbe diventare un “Docente”, effettuando un “upgrade” tramite l’inserimento dell’e-mail associata ad un conto PayPal e il proprio Curriculum Vitae.
3. Docente: un utente che interagisce con il sistema per creare e tenere delle lezioni per gli studenti presenti in esso.
4. PayPal (secondario): è un attore secondario poiché è un servizio esterno, di cui l’applicazione usufruisce per il pagamento delle lezioni.
5. Janus (Secondario): è un attore secondario poiché è un servizio esterno, di cui l’applicazione usufruisce per la gestione delle videochiamate.

Sono stati, invece, individuati i seguenti **casi d’uso**:

1. Registrazione;
2. Registrazione come Docente;
3. Login;
4. Crea lezione;
5. Aggiungi programmazione;
6. Ricerca lezione;
7. Prenota lezione;
8. Avvia videochiamata;
9. Partecipa a videochiamata;
10. Sottoscrivi topic;
11. Recensione lezione.

Sono state effettuate le seguenti scelte implementative:

- Il caso d’uso “Ricerca lezione” è incluso in “Prenota lezione”, in quanto per effettuare la prenotazione della lezione è necessario precedentemente cercarla;
- Il caso d’uso “Aggiungi programmazione” è incluso in “Crea lezione”, in quanto per aggiungere la programmazione di una lezione è necessario averla precedentemente creata;
- Il caso d’uso “Avvia videochiamata” estende “Notifica”, in quanto il sistema provvederà a notificare tale evento a tutti gli studenti che hanno effettuato la prenotazione per quella lezione;
- Il caso d’uso “Crea lezione” estende “Notifica”, in quanto il sistema provvederà a notificare tale evento a tutti gli studenti che hanno sottoscritto il topic relativo alla lezione appena creata.

4.1.1 Descrizione degli Scenari

Segue una descrizione dettagliata degli scenari relativi ad i casi d'uso elencati in precedenza:

- **Registrazione**

Caso d'uso: Registrazione
ID: 1
Breve descrizione: L'utente inserisci i propri dati per registrarsi al sistema
Attori primari: Utente
Attori secondari: Nessuno
Precondizione: L'utente possiede un account di posta elettronica
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. L'utente seleziona “Registrati ora” (dalla homepage) 2. Il sistema mostra la pagina “Registrazione” 3. L'utente inserisce i propri dati (nome, cognome, e-mail, password) 4. L'utente seleziona “Conferma Registrazione” 5. Se l'e-mail è valida <ol style="list-style-type: none"> (a) Il sistema controlla che non sia già registrata, se non è già registrata <ol style="list-style-type: none"> i. Il sistema conferma la registrazione ii. Il sistema rimanda l'utente alla homepage (“Login”) (b) Altrimenti <ol style="list-style-type: none"> i. Il sistema restituisce un messaggio di errore 6. Altrimenti: <ol style="list-style-type: none"> (a) Il sistema restituisce un messaggio di errore
Postcondizione: L'utente è registrato
Sequenze di eventi alternative: E-mail non valida o già registrata

- **Registrazione come docente**

Caso d'uso: Registrazione come docente
ID: 2
Breve descrizione: L'utente registra il proprio profilo come docente
Attori primari: Utente (Studente)
Attori secondari: Nessuno
Precondizione: L'utente deve essere registrato al sito
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. L'utente seleziona “Upgrade Docente” dalla Navigation Bar 2. Inserisce l'e-mail associata ad un conto PayPal ed un file contenente il suo Curriculum Vitae 3. Seleziona “Upgrade”
Postcondizione: L'utente è registrato
Sequenze di eventi alternative: Nessuna

- Login

Caso d'uso: Login
ID: 3
Breve descrizione: L'utente inserisce le proprie credenziali per accedere al sito
Attori primari: Utente
Attori secondari: Nessuno
Precondizione: L'utente deve essere registrato al sito
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. L'utente seleziona la pagina di “Login” 2. L'utente inserisce e-mail e password 3. L'utente seleziona “Login” 4. Il sistema verifica le credenziali 5. Se le credenziali sono corrette <ol style="list-style-type: none"> (a) L'utente effettua il Login (b) Il sistema mostra la homepage 6. Altrimenti <ol style="list-style-type: none"> (a) Il sistema restituisce un messaggio d'errore (b) Il sistema ritorna alla pagina di Login
Postcondizione: L'utente è loggato
Sequenze di eventi alternative: Credenziali non corrette

- Crea lezione

Caso d'uso: Crea lezione
ID: 4
Breve descrizione: Il docente crea una lezione
Attori primari: Docente
Attori secondari: Nessuno
Precondizione: L'utente deve essere loggato come docente
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. Il docente seleziona “Crea lezione” dalla scheda “Lezioni mie” 2. Il docente inserisce il nome, la descrizione, il topic (tra quelli esistenti) e il numero massimo di studenti della lezione 3. Il docente seleziona “Crea lezione”
Postcondizione: La lezione è creata e si possono aggiungere programmazioni relative ad essa
Sequenze di eventi alternative: Nessuna

- Aggiungi programmazione

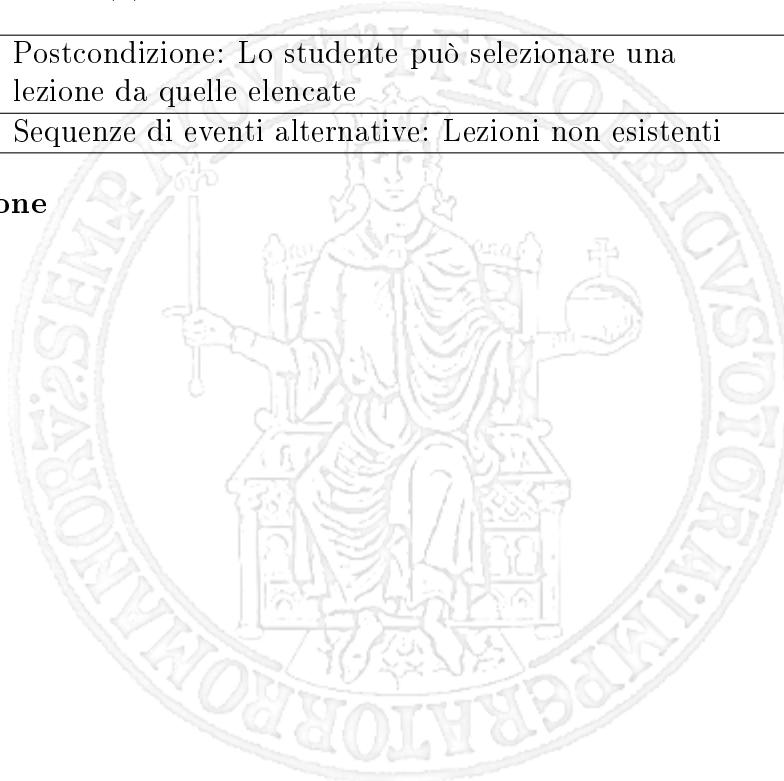


Caso d'uso: Aggiungi programmazione
ID: 5
Breve descrizione: Il docente aggiunge una programmazione ad una lezione creata
Attori primari: Docente
Attori secondari: Nessuno
Precondizione: L'utente deve essere loggato e deve aver creato la lezione
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. Il docente seleziona “Vedi le programmazioni” da una lezione creata e presente nella scheda “Lezioni mie” 2. Seleziona “Aggiungi una programmazione” 3. Inserisce data, orario di inizio, orario di fine e prezzo 4. Seleziona “Programma” 5. Il sistema controlla se non esiste una programmazione per la stessa lezione in quello orario, se essa non esiste: <ol style="list-style-type: none"> (a) La lezione è stata programmata correttamente 6. Altrimenti <ol style="list-style-type: none"> (a) Restituisce errore
Postcondizione: La lezione è programmata e gli studenti possono prenotarla
Sequenze di eventi alternative: Errore di programmazione

- Ricerca lezione

Caso d'uso: Ricerca lezione
ID: 6
Breve descrizione: Lo studente ricerca una lezione da prenotare
Attori primari: Studente
Attori secondari: Nessuno
Precondizione: Lo Studente deve essere loggato
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. Lo studente seleziona la barra di ricerca 2. Lo studente inserisce il topic o il cognome del docente 3. Lo studente seleziona “Cerca” 4. Se le lezioni cercate esistono <ol style="list-style-type: none"> (a) Il sistema mostra le lezioni ricercate (b) Lo studente può selezionare una lezione da prenotare 5. Altrimenti <ol style="list-style-type: none"> (a) Lo studente può cercare un'altra lezione
Postcondizione: Lo studente può selezionare una lezione da quelle elencate
Sequenze di eventi alternative: Lezioni non esistenti

- **Prenota lezione**



Caso d'uso: Prenota lezione
ID: 7
Breve descrizione: Lo studente prenota una lezione ricercata, effettuando un pagamento
Attori primari: Studente
Attori secondari: Sistema di pagamento esterno (PayPal)
Precondizione: Lo Studente ha trovato una lezione di interesse
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. Lo studente seleziona lezione 2. Lo studente seleziona una programmazione della lezione 3. Lo studente seleziona “Prenota ora” <ol style="list-style-type: none"> (a) Se la lezione ha posti disponibili <ol style="list-style-type: none"> i. Il sistema apre la pagina di pagamento PayPal ii. Lo studente inserisce le credenziali PayPal iii. Lo studente seleziona “Conferma” iv. Se il pagamento è avvenuto correttamente <ul style="list-style-type: none"> – Il sistema conferma la prenotazione – Il sistema aggiorna i posti disponibili i. Altrimenti <ul style="list-style-type: none"> – Il sistema mostra un messaggio di errore e torna alla pagina di pagamento Paypal (b) Altrimenti <ol style="list-style-type: none"> i. Il sistema restituisce un messaggio d'errore ii. Lo studente può selezionare un'altra programmazione
Postcondizione: Lo studente è prenotato per quella lezione
Sequenze di eventi alternative: Posti non disponibili per la lezione, errore nel pagamento

- Avvia videochiamata

Caso d'uso: Avvia videochiamata
ID: 8
Breve descrizione: Il docente avvia la videochiamata relativa ad una lezione
Attori primari: Docente
Attori secondari: Janus
Precondizione: Il docente ha creato la lezione, vi è un numero minimo di partecipanti, ci troviamo nell'orario stabilito in fase di creazione
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. Il docente seleziona la scheda “Lezioni mie” 2. Seleziona una lezione 3. Seleziona una programmazione 4. Il sistema controlla se l'orario della programmazione è successivo a quello corrente: <ol style="list-style-type: none"> (a) Se l'orario è successivo <ol style="list-style-type: none"> i. Il sistema mostra il tasto “Avvia videochiamata” ii. Il docente seleziona “Avvia videochiamata” iii. Il sistema invia una notifica ai partecipanti iv. Il sistema apre la schermata di videochiamata (Janus) (b) Altrimenti <ol style="list-style-type: none"> i. Il docente attende l'orario di inizio prestabilito (ritorna al punto 4.a.i)
Postcondizione: La lezione è avviata
Sequenze di eventi alternative: Orario precedente alla programmazione

- Partecipa a videochiamata

Caso d'uso: Partecipa a videochiamata
ID: 9
Breve descrizione: Lo studente partecipa alla videochiamata relativa ad una lezione
Attori primari: Studente
Attori secondari: Janus
Precondizione: Il docente ha avviato la videochiamata della lezione e lo studente ha prenotato la lezione
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. Lo studente seleziona la “Lezioni prenotate” 2. Seleziona una lezione prenotata 3. Seleziona una programmazione della lezione 4. Se la lezione è stata avviata dal docente <ol style="list-style-type: none"> (a) Il sistema mostra il tasto “Partecipa” (b) Lo studente seleziona “Partecipa” (c) Il sistema apre la schermata di videochiamata (Janus) 5. Altrimenti <ol style="list-style-type: none"> (a) Lo studente attende l'avvio della videochiamata (ritorna al punto 4.a)
Postcondizione: Lo studente partecipa alla videochiamata
Sequenze di eventi alternative: Lezione non avviata

- **Sottoscrivi topic**

Caso d'uso: Sottoscrivi topic
ID: 10
Breve descrizione: Lo studente esprime la preferenza per un topic
Attori primari: Studente
Attori secondari: Nessuno
Precondizione: Lo studente è loggato nel sistema
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. Lo studente seleziona la scheda “Topic” 2. Seleziona un topic di interesse 3. Seleziona “SottoscrivI”
Postcondizione: Lo studente ha sottoscritto un topic
Sequenze di eventi alternative: Nessuna

- **Recensione lezione**

Caso d'uso: Recensione lezione
ID: 11
Breve descrizione: Lo studente esprime una preferenza di un topic a cui è interessato
Attori primari: Studente
Attori secondari: Nessuno
Precondizione: Lo studente ha partecipato alla lezione, la lezione è terminata
Sequenza degli eventi:
<ol style="list-style-type: none"> 1. Lo studente termina la videochiamata relativa ad una lezione 2. Il sistema apre una pagina di recensione 3. Lo studente seleziona uno “score” per la lezione (da 0,0 a 5,0) 4. Se lo studente vuole <ol style="list-style-type: none"> (a) Aggiunge una descrizione per la sua recensione (b) Seleziona “Pubblica recensione” 5. Altrimenti <ol style="list-style-type: none"> (a) Seleziona “Pubblica recensione”
Postcondizione: La lezione viene recensita e lo score del docente viene aggiornato
Sequenze di eventi alternative: Nessuna

4.2 Activity Diagrams

Gli Activity Diagrams descrivono i casi d'uso graficamente, mostrando le attività e l'ordine in cui esse vengono eseguite. Per ogni caso d'uso è stato implementato un diagramma differente, tuttavia si mostrano in seguito solo i diagrammi più rilevanti, i restanti sono consultabili nella documentazione allegata.

- **Registrazione**

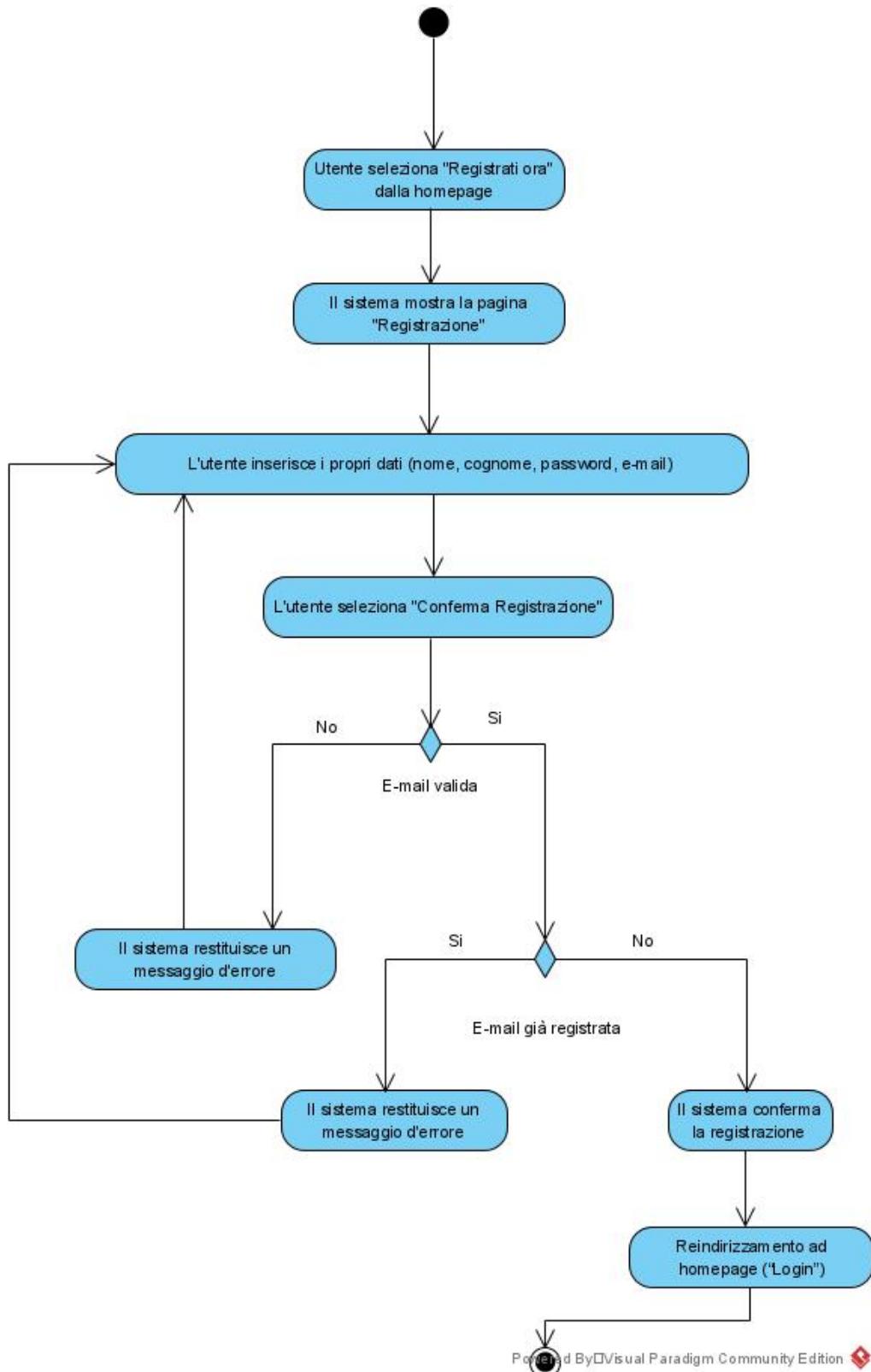


Figura 4.2: Activity Diagram - Registrazione

- Login

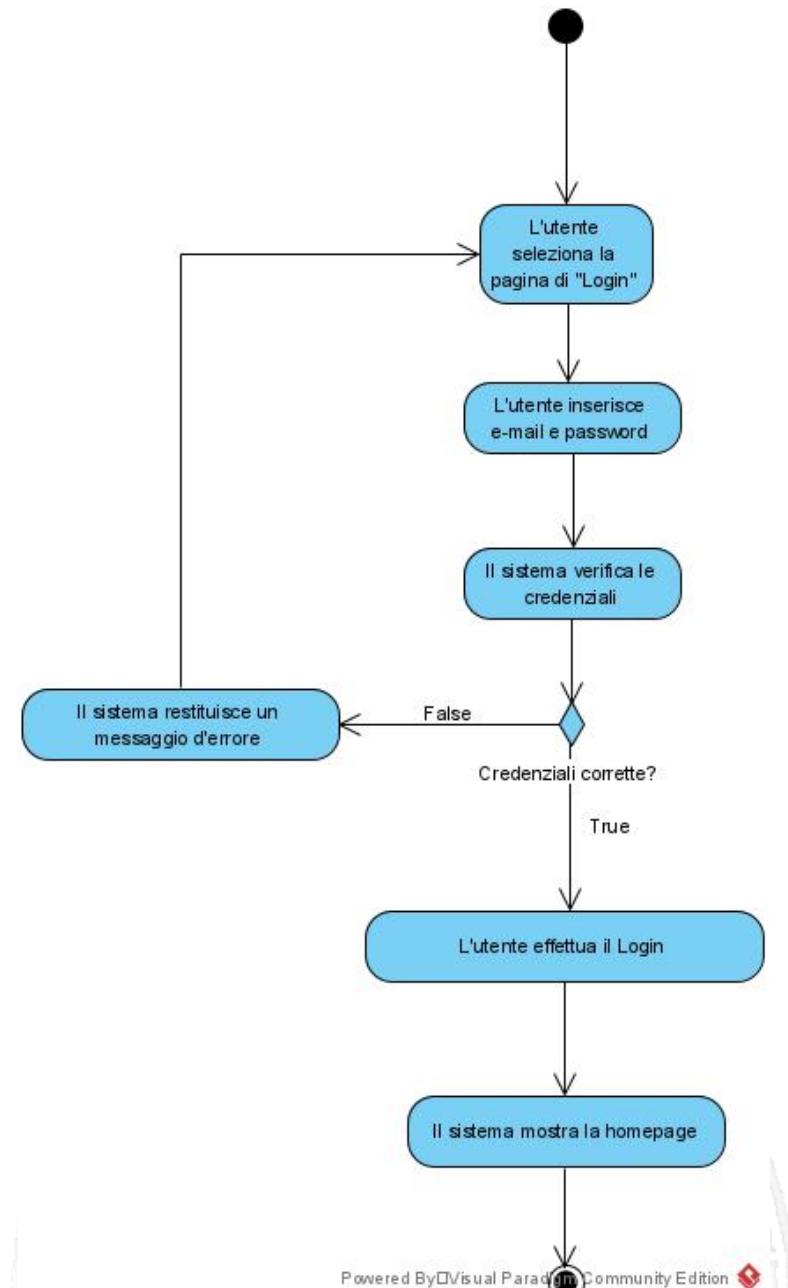


Figura 4.3: Activity Diagram - Login

- Prenota lezione

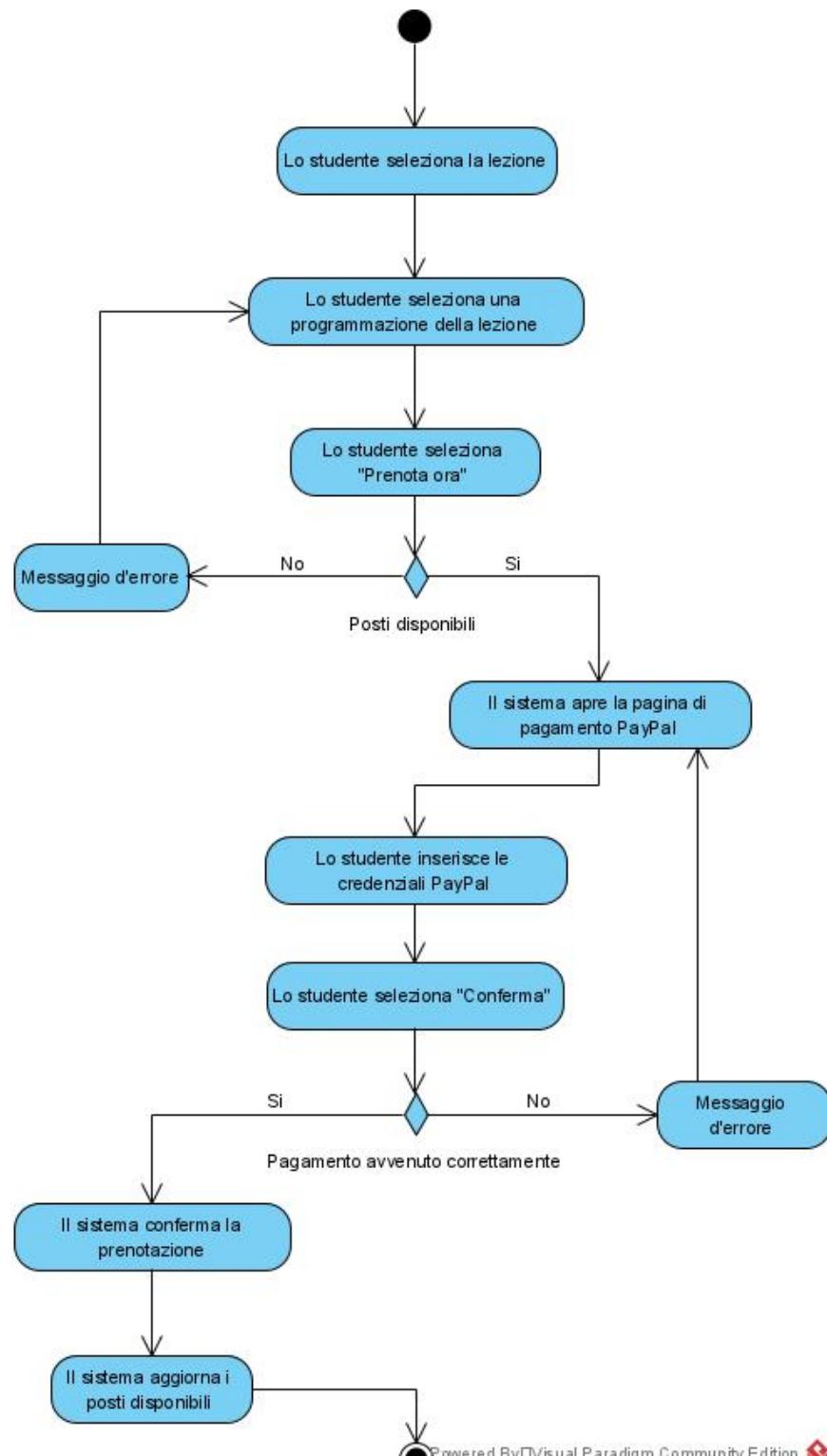


Figura 4.4: Activity Diagram - Prenota lezione

- Avvia videochiamata

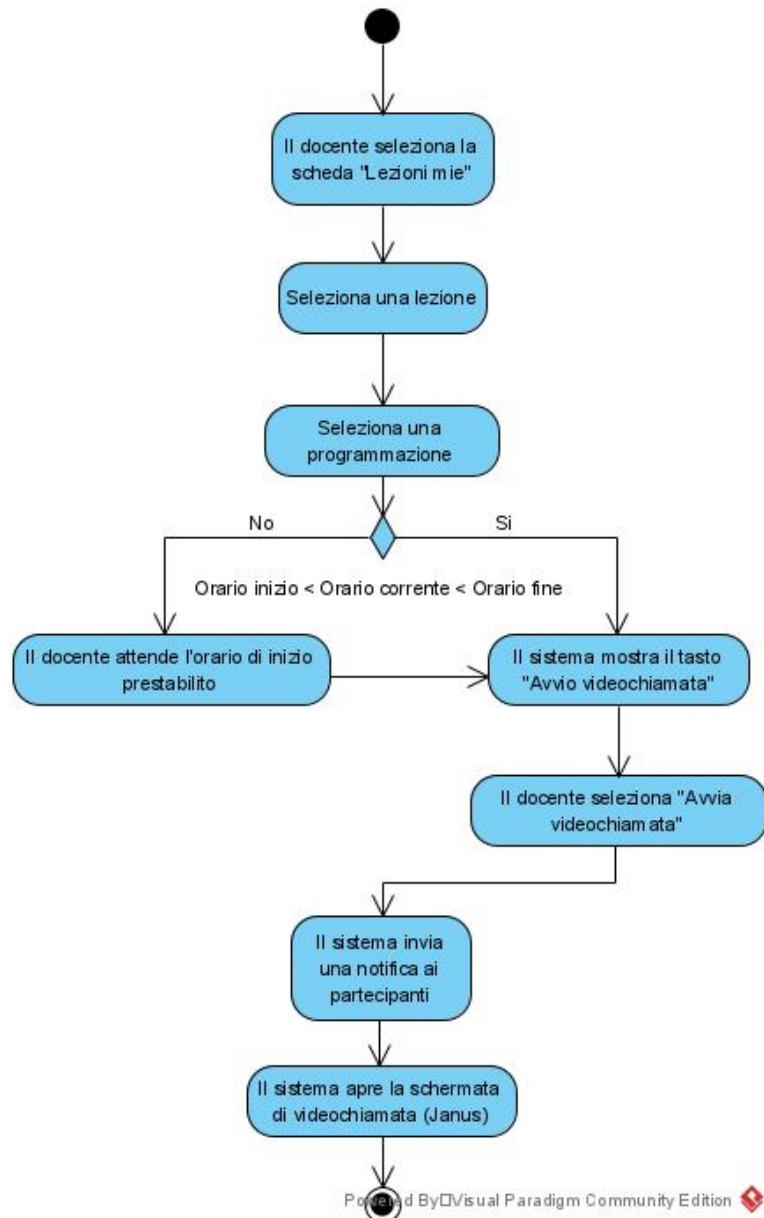


Figura 4.5: Activity Diagram - Avvia videochiamata

- Partecipa a videochiamata

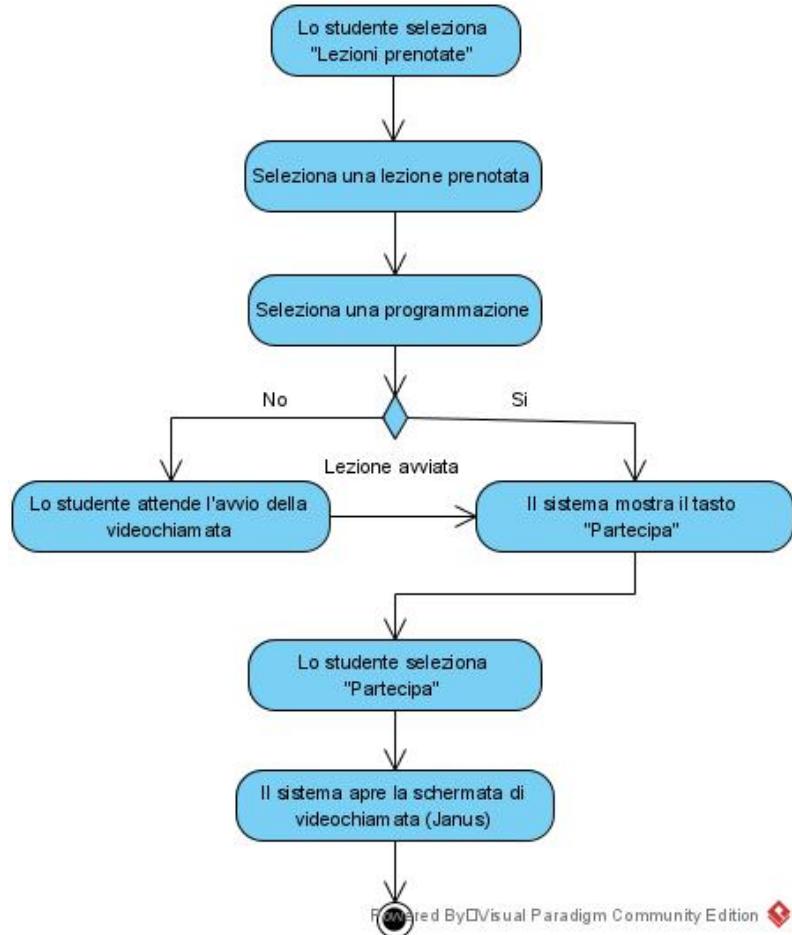


Figura 4.6: Activity Diagram - Partecipa a videochiamata

4.3 System Domain Model

Il System Domain Model denota un modello concettuale di un sistema. Esso descrive le varie entità che fanno parte o hanno rilevanza nel sistema e le loro relazioni:

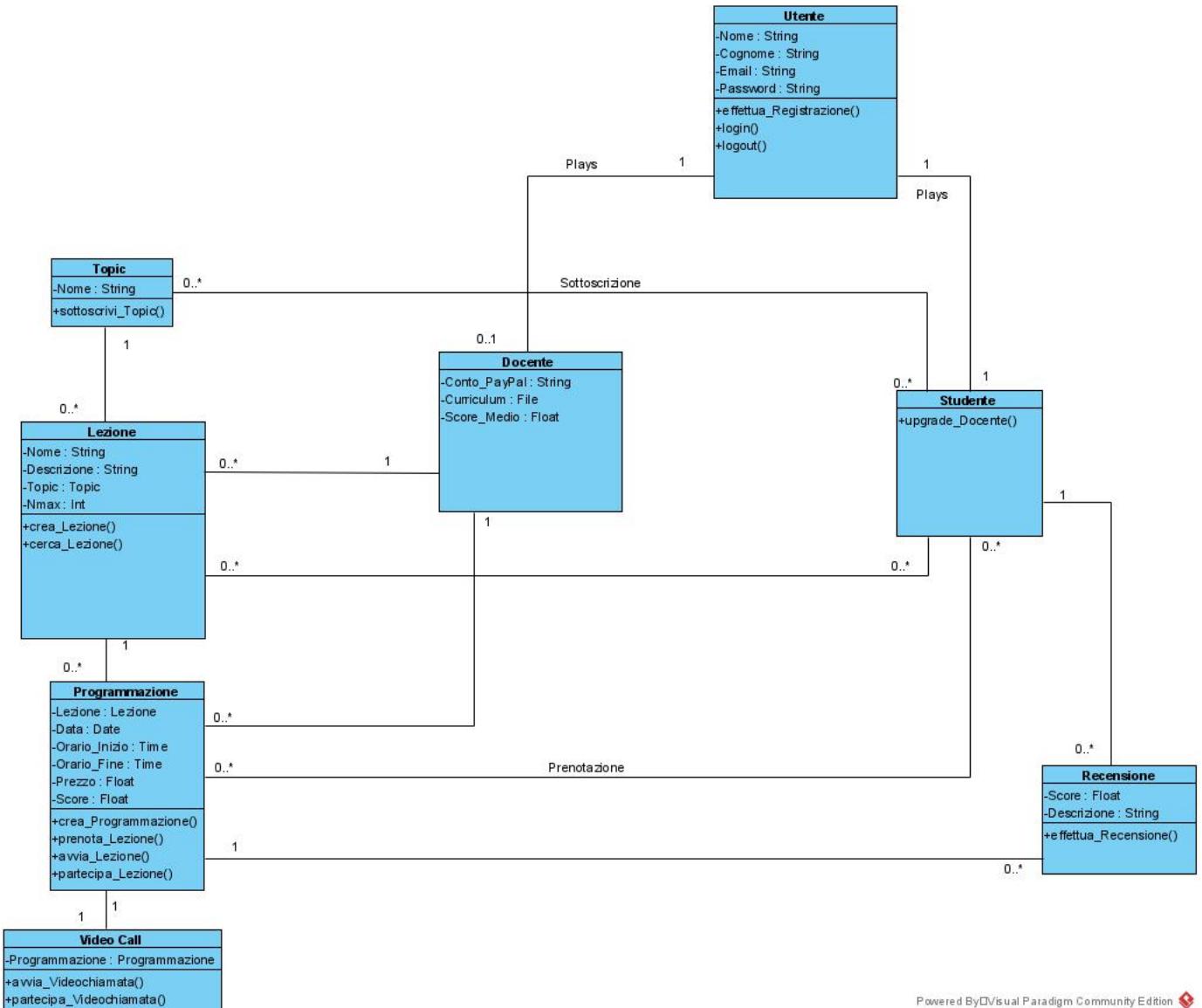
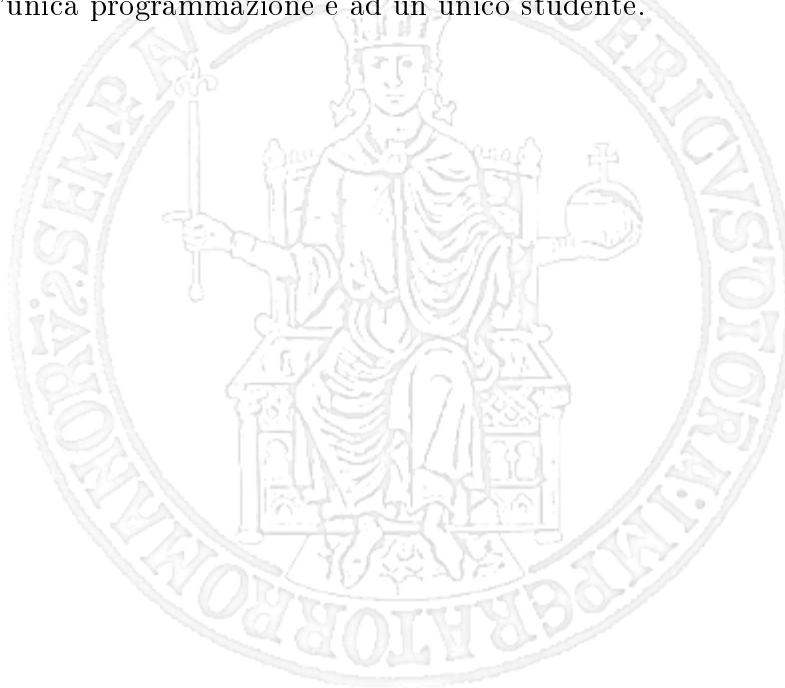


Figura 4.7: System Domain Model

Il sistema è composto da otto entità fondamentali:

- **Utente**: un generico utente che raggiunge il sito ha la possibilità di registrarsi, inserendo nome, cognome, e-mail e password e, dopo di ciò, di effettuare il login e il logout. Un utente può essere di due tipologie “Studente” o “Docente”.
- **Studente**: un utente, una volta registratosi avrà un account di tipo “Studente”, di conseguenza tale entità ha gli stessi attributi di “Utente”, tuttavia si è preferito differenziarlo per distinguere i due ruoli che un utente può recitare all’interno del sistema. Uno studente può effettuare la ricerca di una lezione e la prenotazione di una o più lezioni, partecipare a una videochiamata relativa ad una lezione che ha prenotato e, dopo averla terminata, scrivere una recensione.

- **Docente:** un utente generico, definito come studente, può in qualsiasi momento effettuare l'upgrade del proprio account “Studente” ad un account di tipo “Docente”, semplicemente inserendo l'e-mail relativa ad un account PayPal valido e caricando il proprio Curriculum Vitae; egli inoltre avrà associato al proprio profilo uno “Score medio” relativo alle recensioni ricevute alle proprie lezioni. Un docente può creare le lezioni e le programmazioni relative ad esse e, successivamente, può avviare la videochiamata relativa ad una lezione.
- **Lezione:** una lezione è caratterizzata da nome, descrizione, topic e numero massimo di studenti; essa può essere creata da un docente e prenotata da uno studente. Una lezione può avere più programmazioni relative, mentre si riferisce ad un solo topic.
- **Programmazione:** una programmazione si riferisce ad una lezione, essa infatti si può vedere come l'esecuzione materiale di una lezione, caratterizzata, infatti, da data, orario di inizio e fine, prezzo e score (elaborato in base alle recensioni). Una programmazione inoltre ha una videochiamata associata, di conseguenza avviare o partecipare ad una lezione significa avviare o partecipare ad una videochiamata.
- **Video Call:** una videochiamata si riferisce ad una programmazione, essa può essere avviata solo dal docente che ha creato la relativa lezione e vi possono partecipare solo gli studenti che hanno effettuato la prenotazione per quella lezione.
- **Topic:** un topic, caratterizzato da un nome, è un argomento al quale si può riferire una lezione. Gli studenti possono sottoscrivere i topic per ricevere una notifica nel caso in cui un docente crea una lezione relativa a quel topic.
- **Recensione:** uno studente, dopo aver partecipato ad una lezione, può effettuare una recensione caratterizzata da uno “Score” e una descrizione. Naturalmente una recensione si riferisce ad un'unica programmazione e ad un unico studente.



Capitolo 5

Architettura Software

5.1 Introduzione

La web application è strutturata secondo un'**architettura three-tier**.

Il Client non comunica direttamente con il server del database, ma con il server dell'applicazione. Il Client svolge solo il compito di interfaccia utente e la logica dell'applicazione è inserita nel server applicativo.

Server applicativo e server di database possono risiedere nella stessa macchina o su macchine diverse collegate in rete: nel nostro caso risiedono entrambi su **Azure Cloud**, ma su due locazioni differenti.

Il Client è un semplice browser web, mentre il server applicativo è un server web: come server applicativo è stato utilizzato **Apache Tomcat 9.0**.

Per quanto riguarda il server del database, invece, è stato adoperato come DBMS **MySQL 8.0**.

Il web browser del client invia le proprie richieste, tramite il protocollo HTTP/HTTPS, al livello intermedio ovvero al web server. Quest'ultimo interpreta e serve tali richieste, interagendo anche con il DBMS, per poi generare una risposta in formato XML da inviare allo stesso browser, che la interpreterà e la presenterà all'utente sotto forma di Web.

D'altra parte l'applicazione fa ricorso ad un servizio esterno, **Janus**, per implementare la funzionalità di videochiamata, il quale comunicherà direttamente con il Client.

5.2 Component Diagram

Per illustrare l'architettura del sistema, si è fatto ricorso al Component Diagram, il quale è un diagramma che ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini dei suoi componenti principali e delle relazioni fra di essi. Per componente si intende una unità software dotata di una precisa identità, nonché responsabilità e interfacce ben definite.

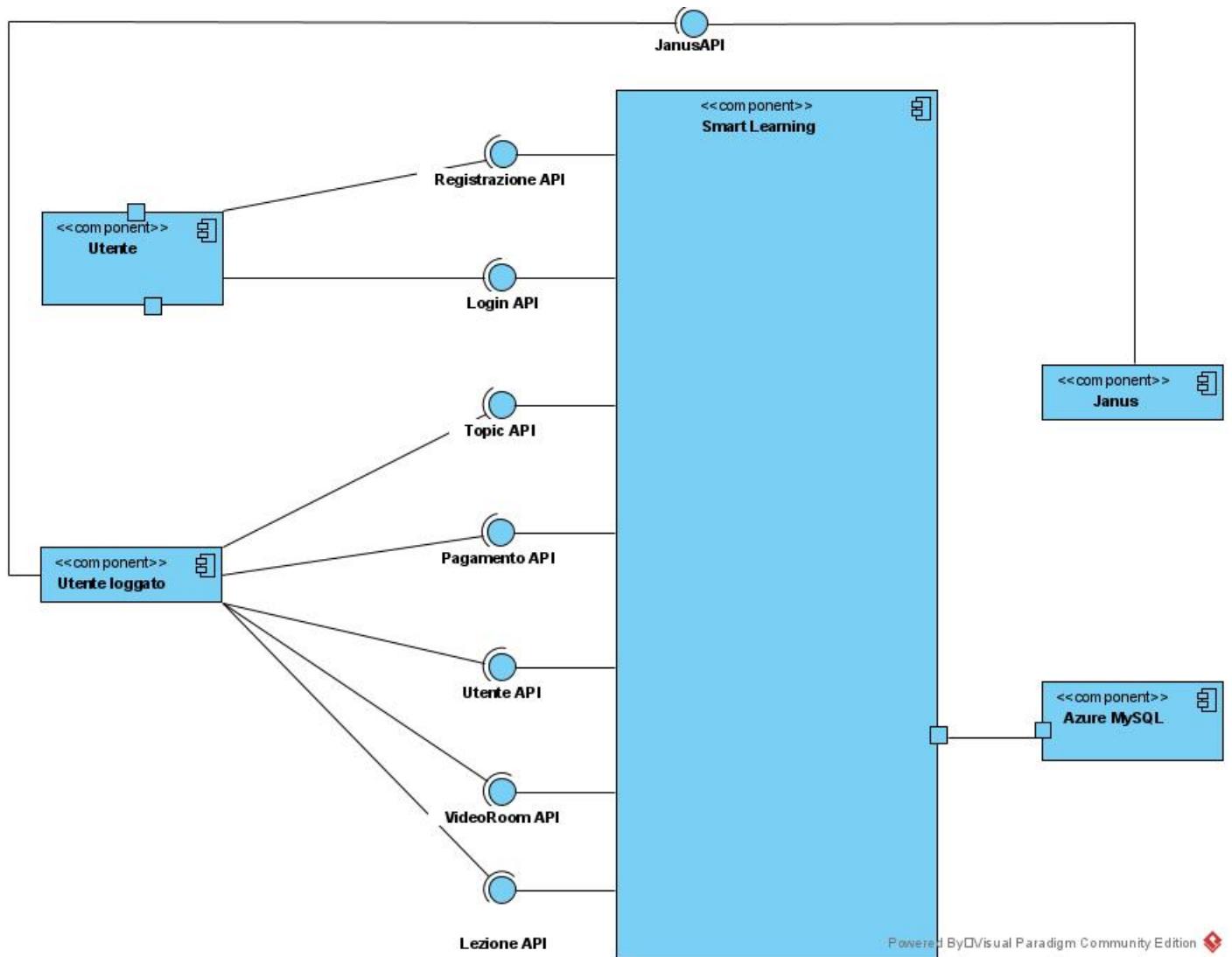


Figura 5.1: Component Diagram

Com'è possibile evincere dalla figura, è possibile scomporre il sistema nei seguenti componenti:

- Una Web Application: Smart Learning;
- Due User Interface: Utente ed Utente loggato;
- Un Service (servizi esterni adoperati dal sistema): Janus;
- Un Database (necessario per gestire la persistenza dei dati): MySQL;
- Sette API interposte tra Client e Web Application (alcune ulteriormente scomponibili in REST API):
 - Registrazione API (REST API);
 - Login API (REST API);
 - Utente API;

- * Utente API (REST API);
- * UpgradeToDocente API (REST API);
- Lezione API:
 - * LezioniPrenotate API (REST API);
 - * Programmazioni API (REST API);
 - * LezioniUtente API (REST API);
 - * ProgrammazioneDocente API (REST API);
 - * LezioneDocente API (REST API);
 - * LezioniDocente API (REST API);
- Pagamento API:
 - * Prenotazione API (REST API);
 - * Prenotazioni API (REST API);
- Topics API (REST API);
- Videoroom API:
 - * VideoCallUtente API (REST API);
 - * VideoCallDocenteAPI (REST API).

5.3 Application Server

Le funzionalità dell'applicazione Smart Learning, precedentemente definite nella specifica dei requisiti, possono essere raggruppate nei seguenti moduli: Registrazione, Login, Utente, Lezione, Pagamento, Topic e Videoroom.

Dato il numero dei servizi offerti, sono state valutate due possibili soluzioni architetturali: un'architettura a microservizi o un'architettura three-layer.

In particolare, l'architettura a microservizi sembra adattarsi perfettamente al sistema. D'altra parte essa, grazie alla separazione dei componenti in microservizi indipendenti (ognuno con la propria base di dati), attribuirebbe al software flessibilità, scalabilità ed estensibilità (nuovi servizi), requisiti ottimali per un'applicazione sviluppata ex-novo.

Inoltre, i servizi sarebbero sviluppati e distribuiti in maniera indipendente, risultando più facili da manutenere, correggere e aggiornare, rispetto ad una classica applicazione monolitica.

Di contro, lo sviluppo di un'architettura a microservizi richiederebbe un grosso sforzo implementativo, dovuto alla complessità di una tale tipologia di architettura. Di conseguenza, in seguito ad un'analisi di fattibilità, si è constatato che non si sarebbero potuti rispettare i tempi di consegna del progetto.

In particolare la problematica principale è rappresentata dal mantenimento della consistenza dei dati, distribuiti su basi di dati indipendenti; essa prevederebbe una comunicazione asincrona tra i microservizi (tramite broker), evitando l'overhead della comunicazione sincrona (*two-phase-commit*), e degli orchestratori delegati a sfruttare questa comunicazione asincrona per implementare la logica di business e dei meccanismi affidabili di rollback.

Per tale motivo, si è optato verso un'architettura più semplice da implementare, ovvero un'**architettura three-layer** (monolitica), in cui saranno presenti servizi esterni.

Essa prevede la suddivisione dell'applicazione in tre diversi moduli: Web Layer, Service Layer e Data Layer.

Nell'architettura a tre livelli, ciascuna delle funzionalità principali è isolata dalle altre, in modo che il Web Layer sia indipendente dal Service Layer, che a sua volta è separato dai dati (Data Layer).

D'altra parte quando l'applicazione è relativamente "piccola", l'architettura monolitica può avere numerosi vantaggi; essa risulta infatti semplice da sviluppare, testare, deployare e scalare, inoltre l'applicazione di cambiamenti radicali è facilitata.

Tuttavia, al crescere delle dimensioni, lo sviluppo, i test, il deploy e lo scaling diventano sempre più difficili.

5.3.1 Package Diagrams

I Package Diagrams sono utilizzati per modellare la struttura modulare del sistema, d'altra parte essi evidenziano i differenti strati dell'architettura che si rifletteranno nel codice sorgente.

Nel caso in esame sono stati elaborati due differenti diagrammi:

- Un Package Diagram generale, il quale mostra la struttura generale del sistema;
- Un Package Diagram specifico, il quale entra nel dettaglio dei vari moduli del sistema.

Tale distinzione è stata fatta per fornire una versione più "leggibile" dell'architettura ed una più dettagliata.

Package Diagram generale

Come è possibile evincere dalla figura sottostante, l'applicazione lato server è strutturata secondo un'architettura **three-layer** (a tre livelli), soluzione ampiamente utilizzata per le applicazioni web.

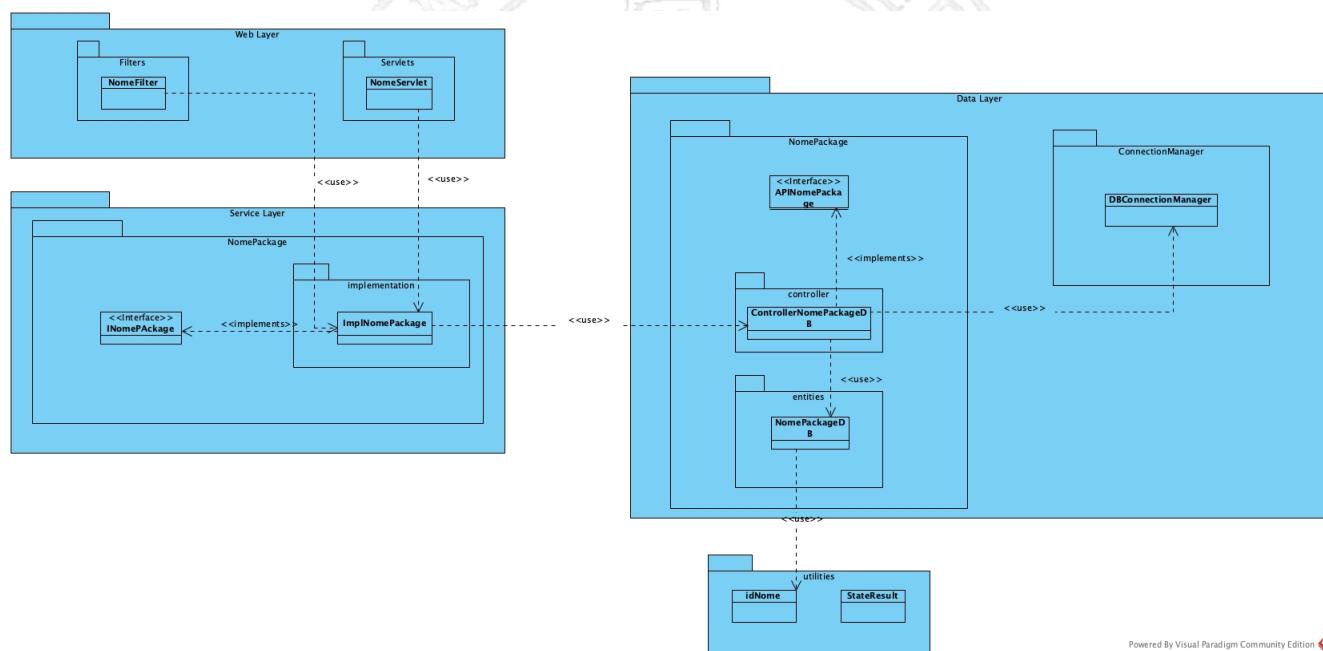


Figura 5.2: Package Diagram generale

Package Diagram specifico

Tale diagramma specifica puntualmente la struttura ed i contenuti dei vari package del sistema, tuttavia risulta poco leggibile a causa delle dimensioni dello stesso.



Figura 5.3: Package Diagram specifico

5.3.2 Architettura Three-Layer

Come detto in precedenza il Server applicativo è a sua volta suddiviso secondo un'architettura a tre livelli:

- **Web Layer:** si occupa dell'interfacciamento con il Client.
- **Service Layer:** si occupa delle elaborazioni dei dati in base alla cosiddetta *business logic*. Esso crea un ponte tra il Web Layer e il Data Layer.
- **Data Layer:** il livello dati garantisce il meccanismo di archiviazione persistente dei dati. Esso supporta la connessione con il database e l'esecuzione delle operazioni di inserimento, aggiornamento, eliminazione e recupero dei dati.

Web Layer

Il Web Layer è composto da **filtri** e **servlet**.

Le richieste effettuate dal Client sono in formato HTTP, esse sono ricevute dai filtri, che dopo averle processate le inoltreranno alle servlet, che si occupano di accedere alla risorsa richiesta; ogni risorsa ha una servlet associata.

Viene così a crearsi una catena costituita, per l'appunto, da filtri e servlet.

Quest'ultime si occupano di richiamare le interfacce esposte dai servizi, che implementano le operazioni per servire le richieste, e di formattare la risposta da inviare al Client in formato XML.

Le interfacce esposte dalle servlet sono state realizzate adottando lo stile architetturale **REST**.

L'architettura REST si basa sul protocollo HTTP. Il funzionamento prevede una struttura degli URL ben definita che identifica univocamente una risorsa o un insieme di risorse e l'utilizzo dei metodi HTTP specifici per il recupero di informazioni (GET), per la modifica (POST, PUT, PATCH, DELETE) e per altri scopi (OPTIONS, ecc.).

Le interfacce REST sono state interamente documentate secondo lo standard **OpenAPI 3.0** in un file allegato, come mostrato nella seguente figura:

VideoCallUtente	
GET <code>/riservate/videocallutente</code> Permette a un studente di ottenere il token della videocall e il nome della stanza della videocall associata ad una programmazione pagata dall'utente	
VideoCallDocente	
GET <code>/riservate/docente/videocalldocente</code> Permette a un docente di ottenere le informazioni per effettuare il setup della chiamata, tra cui anche la generazione dei token degli studenti per accedere alla videochiamata.	
DELETE <code>/riservate/docente/videocalldocente</code> Elimina l'istanza della videocall	
Upgradetodocente	
POST <code>/riservate/upgradetodocente</code> Permette a uno studente di fornire le info necessarie per fare l'upgrade a docente tra cui il curriculum e l'indirizzo email del conto paypal.	

Figura 5.4: REST API

Service Layer

Il Service Layer si occupa delle elaborazioni dei dati in base alla cosiddetta business logic; le elaborazioni del livello intermedio generano i risultati richiesti dall'utente.

Il Service Layer è composto dai seguenti moduli:

- **Registrazione:** si occupa della logica di registrazione di nuovi utenti;
- **Login:** gestisce la logica sul quale si basa il Login;
- **Utente:** ha la responsabilità di gestire tutto ciò che riguarda il profilo degli utenti (incluso l'Upgrade ad un profilo “Docente”);
- **Lezione:** include la logica di gestione delle lezioni e del calendario a loro associato;
- **Pagamento:** include la gestione dei pagamenti e tiene traccia delle persone che hanno pagato una lezione;
- **Topic:** include la logica di gestione dei topic;
- **Videoroom:** ha la responsabilità di gestire ciò che riguarda le videochiamate relative alle lezioni.

Si è scelto quindi di adottare uno stile architetturale Service-Oriented, di conseguenza i principali componenti di questo layer sono stati realizzati come servizi indipendenti, inoltre essi sono richiamati attraverso un'interfaccia indipendente dalla specifica implementazione; ciò garantisce modificabilità e riusabilità.

Data Layer

Il Data Layer ha il compito di gestire la persistenza dei dati, quindi di comunicare con il DBMS relativo al database utilizzato.

Tale livello è strutturato secondo il pattern architettonale **DAO (Data Access Object)**: è stata implementata una classe, con i relativi metodi, per ogni entità tabellare del DBMS.

Tali classi sono utili per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) ovvero al data layer da parte della business logic, creando un maggiore livello di astrazione ed una più facile manutenibilità.

Infatti, il Data Layer espone ai componenti presenti nel Service Layer un insieme di interfacce (API) per accedere ai dati, ma è compito del DAO gestire la connessione al database e lo svolgimento delle operazioni CRUD; tutto ciò favorisce il disaccoppiamento tra il Service Layer con il Database persistente.

La connessione al database è gestita tramite una classe DBConnectionManager, dalla quale sono dipendenti tutte le altre classi del livello.

5.4 Client

Il Client è strutturato secondo un'architettura detta **Thin Client**, ovvero esso svolge solo il compito di interfaccia utente, mentre la logica dell'applicazione e la gestione dei dati è demandata ai server.

Nel caso in questione il Client è un browser che comunica con il server applicativo tramite il protocollo HTTP/HTTPS.

Il browser, attraverso il proprio motore di rendering, genera l'interfaccia utente, interpretando le pagine realizzate tramite HTML, CSS e tecnologia JavaScript. D'altra parte esso comunicherà direttamente con il server Janus, per la fruizione del servizio di videochiamata.

5.5 Database Server

Il Database Server è stato implementato tramite il DBMS MySQL, utilizzato per la gestione della persistenza dei dati e la loro interrogazione tramite JDBC, ricevendo e soddisfacendo le richieste di lettura/scrittura sul DB da parte della logica applicativa.

JDBC è un connettore per database, che consente l'accesso ad una base di dati da qualsiasi programma scritto in JAVA, indipendentemente dal tipo di DBMS utilizzato. Esso è costituito da un'API object oriented orientata ai database relazionali.

Il database è di tipo **relazionale** e le entità tabellari da cui è composto corrispondono alle classi del Data Layer, come definito dal pattern Data Access Object.

5.6 Servizi esterni: Janus

L'applicazione è basata sulla tecnologia open source WebRTC, disponibile su tutti i moderni browser, che consente di effettuare videochat P2P in tempo reale. A supporto di tale scelta tecnologica si è deciso di utilizzare il WebRTC server Janus. Questo infatti fornisce delle API in javascript che astraggono i dettagli implementativi della tecnologia e ne consentono un rapido e semplice utilizzo

in svariati casi d'uso. In particolare sono messi a disposizione una serie di “plugin” da aggiungere al codice base così da personalizzarne l'utilizzo. In questo caso si è deciso di utilizzare il plugin “VideoRoom” che fornisce l'implementazione di una **SFU (Selective Forwarding Unit)**.

Maggiori dettagli sul Janus sono forniti nel capitolo 7.

5.7 Deployment Diagram

Il Deployment Diagram è utilizzato per descrivere un sistema in termini di risorse hardware, dette nodi, e di relazioni fra di esse. Se unito al Component Diagram, mostra come le componenti software siano distribuite rispetto alle risorse hardware disponibili sul sistema.

Come è possibile evincere dalla figura sottostante, il sistema è “deployato” nella seguente modalità:

- **Nodo 1:** Azure Cloud contenente il server Apache Tomcat, sul quale viene eseguito “Smart Learning”;
- **Nodo 2:** Azure Cloud contenente il database MySQL;
- **Nodo 3:** Azure Cloud contenente il server Janus.



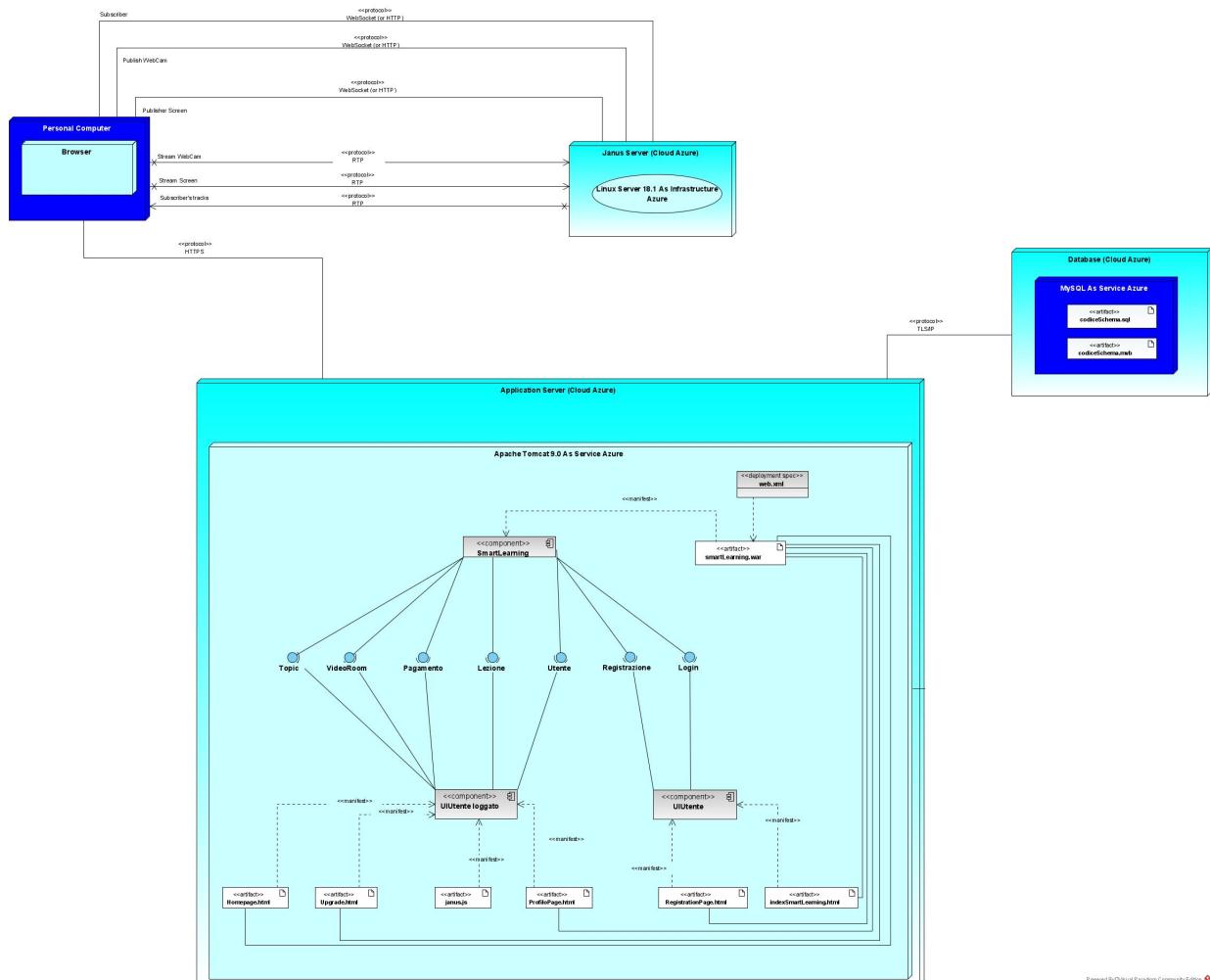


Figura 5.5: Deployment Diagram

Capitolo 6

Implementazione, configurazione ed esecuzione

6.1 Implementazione

6.1.1 Application Server

Web Layer

È il livello che espone le funzionalità offerte dal sistema verso il Client. In particolare, vi sono implementati:

- 2 Filtri derivati dalla classe `Filter`:
 - `AuthFilter` filtro per il controllo dell'autenticazione dell'utente all'interno del sistema;
 - `DocFilter` gestisce l'autorizzazione per le funzioni dedicate al docente.
- 16 Servlet per identificare le risorse. Ogni Servlet è una classe derivata da `HttpServlet`. Il flusso di esecuzione di una servlet è incentrato su due componenti fondamentali: la richiesta (request, inviata dal client verso il server) e la risposta (response, inviata dal server verso il client). In Java, questi due componenti sono identificati, rispettivamente, dalle seguenti interfacce: `javax.servlet.http.HttpServletRequest` e `javax.servlet.http.HttpServletResponse`.

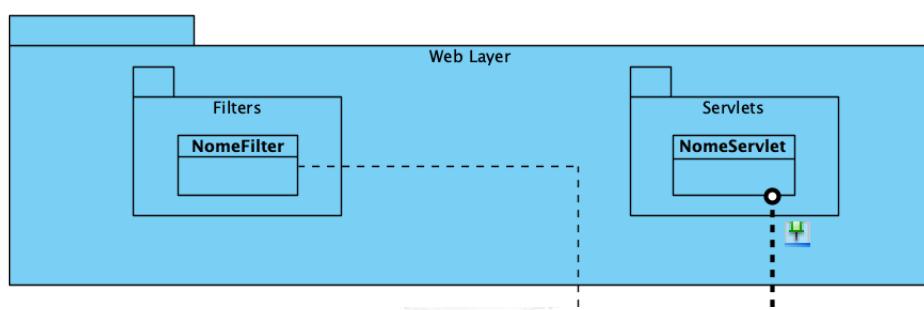


Figura 6.1: Package Web Layer

Service Layer

Nel Service Layer troviamo l'implementazione della logica di business. Tale package è composto da:

- Un package per ogni servizio offerto al cui interno troviamo:
 - Un'interfaccia con la definizione dei metodi relativi ai servizi;
 - Un package di implementazione dentro il quale vi è la classe di implementazione dei metodi definiti nell'interfaccia relativi a quel servizio.

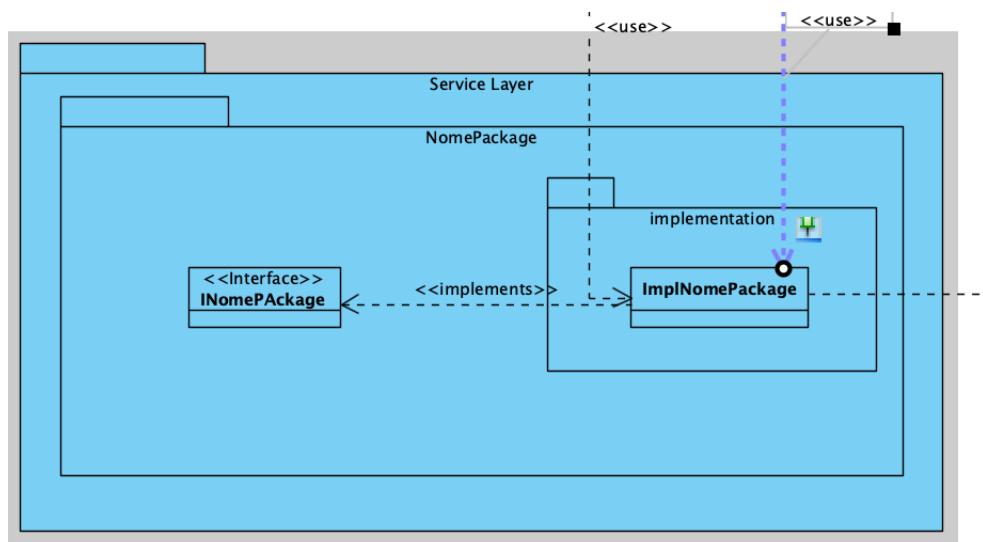


Figura 6.2: Package Service Layer

Data Layer

Il package Data Layer è composto da:

- Un package che prende il nome dell'entità da implementare al cui interno troviamo:
 - Il package “entities” nel quale vi è la classe che funge da contenitore d'informazioni dell'entità;
 - Il package “controller” nel quale vi è il controller, classe che implementa l'interfaccia che espone le funzioni pubbliche sull'entità;
 - L'interfaccia della classe con la definizione dei metodi relativi all'entità, i quali possono essere richiamati dal Service Layer. È possibile consultare la documentazione di tutte le interfacce, generata tramite Javadoc, all'interno della cartella “Codice\WebAppSmartLearning\doc”. Ogni interfaccia segue il seguente standard:
 - * ogni funzione ritorna sempre uno “StateResult” che rispecchia lo stato di completamento della funzione;

CAPITOLO 6. IMPLEMENTAZIONE, CONFIGURAZIONE ED ESECUZIONE

- * le funzioni utilizzano come contenitori di informazioni solo le entità presenti nel relativo package “entities”;
- * le informazioni ottenute dall’elaborazioni delle funzioni sono restituiti tramite parametri di I/O.

Nella seguente figura è mostrata come esempio la documentazione Javadoc relativa a “API_LezioneDB”:

Method Summary	
All Methods	Instance Methods
Modifier and Type	Method and Description
StateResult	addFasciaOraria(idUser idOwnerUser, idLesson idLezione, FasciaOraria orari) Questa funzione permette di aggiungere una fascia oraria alla lezione con il vincolo di non poter essere sovrapposta ad altre lezioni dello stesso utente.
StateResult	attachSlotsToLessonsDocente(java.util.Vector<LezioneDB> lezioni) Questa funzione permette di arricchire le lezioni di un docente con le relative fasce orarie (anche quelle non visibili agli utenti)
StateResult	attachSlotsToLessonsUtente(java.util.Vector<LezioneDB> lezioni) Questa funzione permette di collegare gli slot alle lezioni (ogni lezione deve contenere almeno l'id utente e l'id lezione)
StateResult	createLesson(LezioneDB infoLezione, java.lang.String nomeTopic) Questa funzione permette di creare una lezione
StateResult	getFascePayedStillUpByLesson(idUser idUser, idLesson idlez, java.util.Vector<FasciaOraria> fasce) Questa funzione permette di ottenere una le le fasce orarie relative ad una lezione pagate da un utente
StateResult	getFasciaOraria(FasciaOraria fascia) Questa funzione permette di ottenere una fascia oraria visibile
StateResult	getFasciaOrariaidUser id, FasciaOraria fascia) Verifica che la fascia oraria esista e sia stata generata da uno specifico docente.
StateResult	getFasciaOrariaNoCatalogo(idUser id, FasciaOraria fascia) Questa funzione permette di ottenere una fascia oraria in base ad un utente, anche se scaduta
StateResult	getLessonsByDocente(idUser idOwnerUser, java.util.Vector<LezioneDB> lezioni) Questa funzione permette di selezionare tutte le lezioni di un docente (senza controllo su visibile in fasce orarie)
StateResult	getLessonsbyTitle(java.lang.String title, java.util.Vector<LezioneDB> lezioni) Questa funzione permette di ottenere un vettore di lezioni che riguardano un determinato nome di una lezione
StateResult	getLessonsbyTopics(idTopic infoTopic, java.util.Vector<LezioneDB> lezioni) Questa funzione permette di ottenere un vettore di lezioni che riguardano un determinato topic
StateResult	getLessonsByUser(idUser idOwnerUser, java.util.Vector<LezioneDB> lezioni) Questa funzione permette di selezionare tutte le lezioni di un docente (con controllo visibile su fasce orarie = '1')
StateResult	getLessonsPayedStillUp(idUser idUser, java.util.Vector<LezioneDB> lezioni) Questa funzione restituisce tutte le lezioni con relative fasce pagate da un utente e ancora non "scadute"
StateResult	updateFasciaOraria(idUser id, FasciaOraria slotAggiornato) Questa funzione permette di aggiornare una fascia oraria di una lezione.
StateResult	validLezione(idlesson idLesson) Questa funzione permette di validare l'id di una lezione

Figura 6.3: Documentazione Javadoc API_LezioneDB

- Il package “ConnectionManager” in cui troviamo la classe che implementa le funzioni per la connessione con il database e tutte le funzioni che effettuano le query al database;

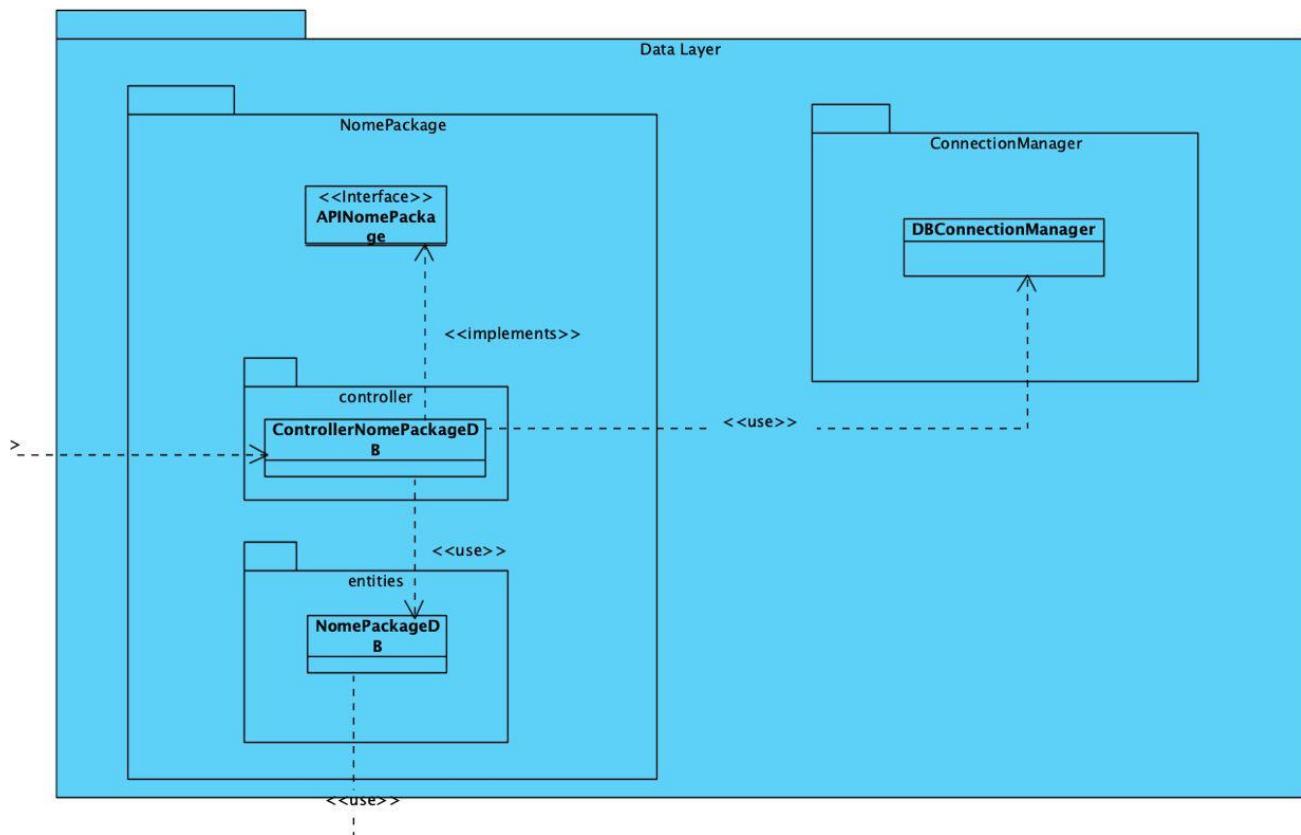


Figura 6.4: Package Data Layer

Utilities

Nel package “utilities” si trovano le classi relative ai vari id, che si riferiscono alle varie tabelle del database, e una classe enum “StateResult” che rappresenta lo stato di fine elaborazione di una funzione.

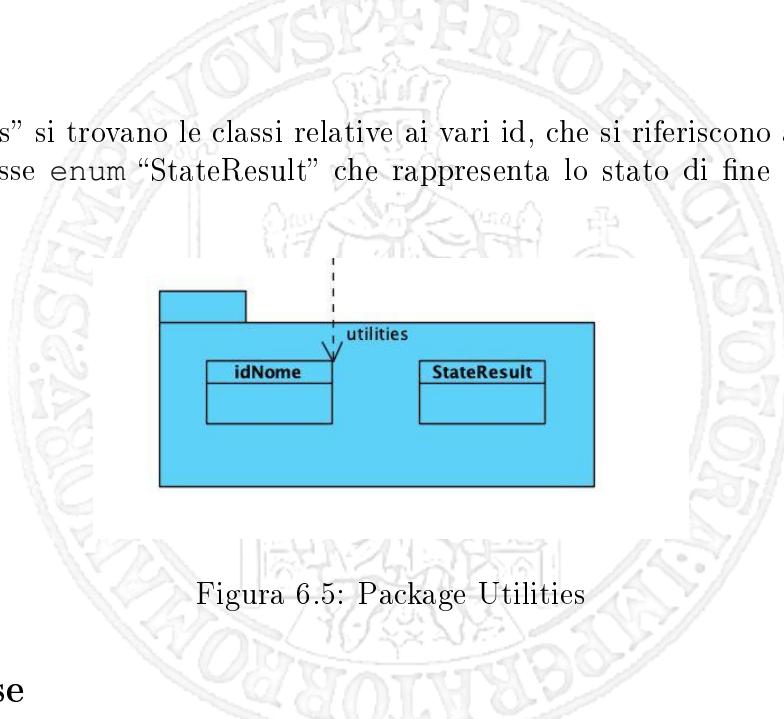


Figura 6.5: Package Utilities

6.1.2 Database

Il Database è di tipo **relazionale** ed è rappresentabile attraverso il seguente **modello E-R** (entità-relazione):

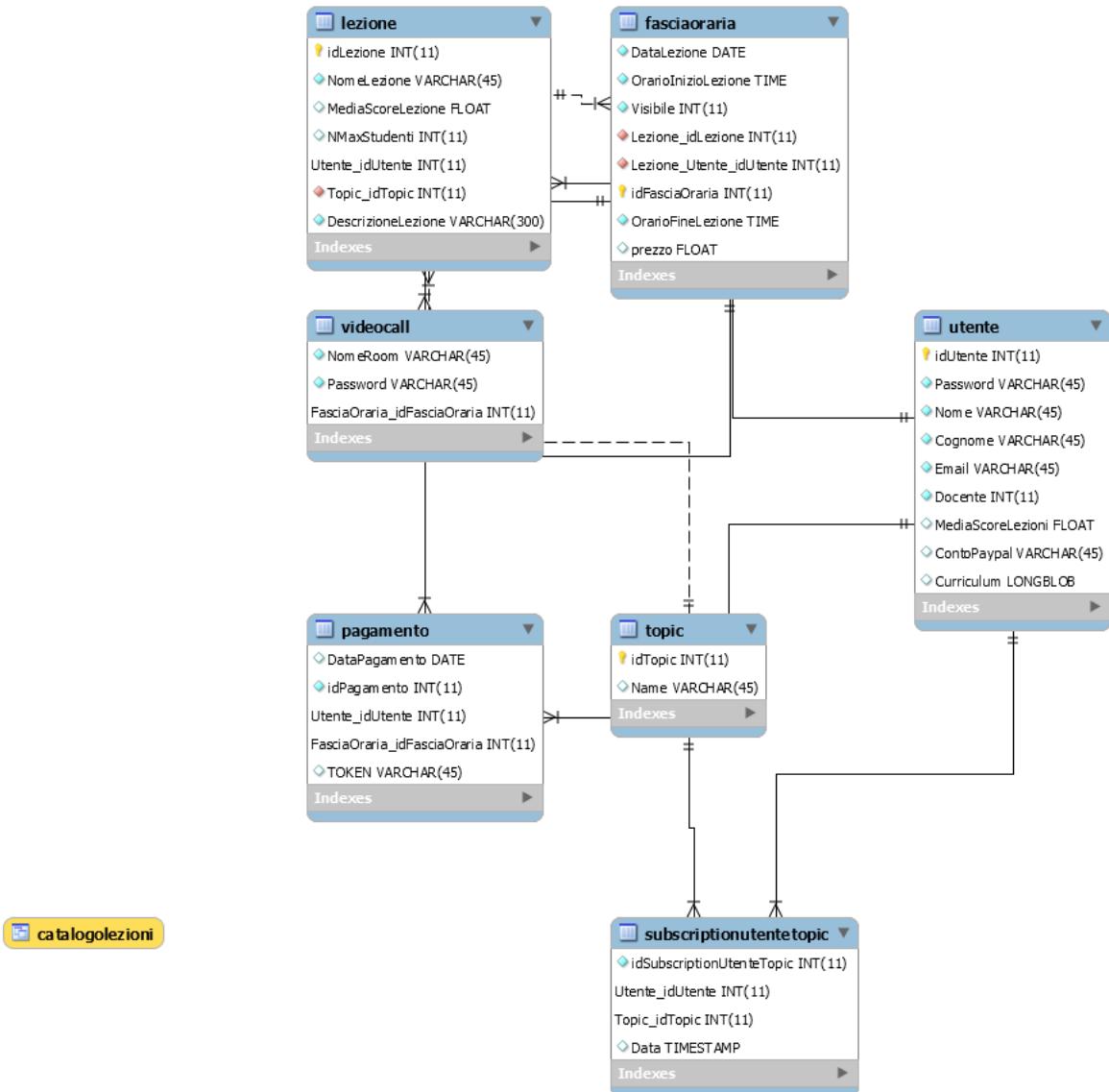


Figura 6.6: Modello E-R

Esso, inoltre, è caratterizzato dall'utilizzo di una vista denominata "catalogolezione", e delle stored procedures. In particolare:

- "catalogolezione" raggruppa i dati della tabella "lezione" e di "fasciaoraria" in modo da poter effettuare i controlli sulla data e sugli orari per il retrieve di una fascia oraria di una lezione non scaduta. La data di creazione deve essere la data corrente oppure maggiore della data corrente, l'orario di inizio di una lezione deve essere maggiore o uguale dell'orario corrente;

```

1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = 'root'@'localhost'
4     SQL SECURITY DEFINER
5     VIEW `prova2`.`catalogolezioni` AS
6     SELECT
7         `prova2`.`lezione`.`idLezione` AS `idLezione`,
8         `prova2`.`lezione`.`NomeLezione` AS `NomeLezione`,
9         `prova2`.`lezione`.`MediaScoreLezione` AS `MediaScoreLezione`,
10        `prova2`.`lezione`.`NMaxStudenti` AS `NMaxStudenti`,
11        `prova2`.`fasciaoraria`.`DataLezione` AS `DataLezione`,
12        `prova2`.`fasciaoraria`.`OrarioInizioLezione` AS `OrarioInizioLezione`,
13        `prova2`.`fasciaoraria`.`OrarioFineLezione` AS `OrarioFineLezione`,
14        `prova2`.`fasciaoraria`.`prezzo` AS `prezzo`,
15        `prova2`.`fasciaoraria`.`Visibile` AS `visible`,
16        `prova2`.`lezione`.`Utente_idUtente` AS `Utente_idUtente`,
17        `prova2`.`lezione`.`Topic_idTopic` AS `Topic_idTopic`,
18        `prova2`.`fasciaoraria`.`idFasciaOraria` AS `idFasciaOraria`,
19        `prova2`.`lezione`.`DescrizioneLezione` AS `DescrizioneLezione`
20
21     FROM
22     (
23         `prova2`.`lezione`
24         JOIN `prova2`.`fasciaoraria` ON (((`prova2`.`lezione`.`idLezione` = `prova2`.`fasciaoraria`.`Lezione_idLezione`)
25             AND (`prova2`.`lezione`.`Utente_idUtente` = `prova2`.`fasciaoraria`.`Lezione_Utente_idUtente`)))
26
27     WHERE
28         ((`prova2`.`fasciaoraria`.`DataLezione` > CURDATE())
29             OR ((`prova2`.`fasciaoraria`.`DataLezione` = CURDATE()
30                 AND (`prova2`.`fasciaoraria`.`OrarioInizioLezione` >= (NOW() - INTERVAL 2 HOUR))))

```

Figura 6.7: Vista “catalogolezione” (MySQL)

- la procedura "updateCond" permette di aggiornare una fascia oraria di una lezione e la data di una lezione effettuando un controllo sulla nuova fascia oraria rispetto alla precedente evitando che si incrocino;
- la procedura "inserisciVideoCall" inserisce una nuova room con un id di fascia oraria e un token docente generato in modo randomico;
- la procedura "inserisciCond" permette di inserire una nuova fascia oraria ad una lezione creata evitando che il docente inserisca una fascia oraria che si incrocia con una già creata;
- la procedura "genTokens" crea un token in modo randomico per ogni prenotazione per ogni studente pagante per una fascia oraria.

6.2 Implementazione Client ed Esecuzione

Per l'esecuzione dell'applicazione web è necessario collegarsi all'indirizzo: <https://webappsmartlearning.azurewebsites.net>

Per l'interfacciamento con il client si sono sviluppate diverse pagine html, in particolare:

- indexSmartLearning.html: pagina di login;
- RegistrationPage.html: pagina di registrazione;
- Homepage.html: homepage (ricerca lezioni);
- ProfiloPage.html: pagina del profilo, contenente informazioni sull'utente, le lezioni alle quali è prenotato, le proprie lezioni in caso di Docente. Su questa pagina è inoltre possibile effettuare le videocall.

Inoltre si è deciso di utilizzare diversi file css per la visualizzazione degli elementi. In particolare, per tutte le pagine elencate si utilizzeranno i css della libreria “bootstrap” così come il file “demo.css”, appositamente sviluppato. L’ultimo css utilizzato è “index.css”, utile ad una corretta visualizzazione della interfaccia di videochiamata.

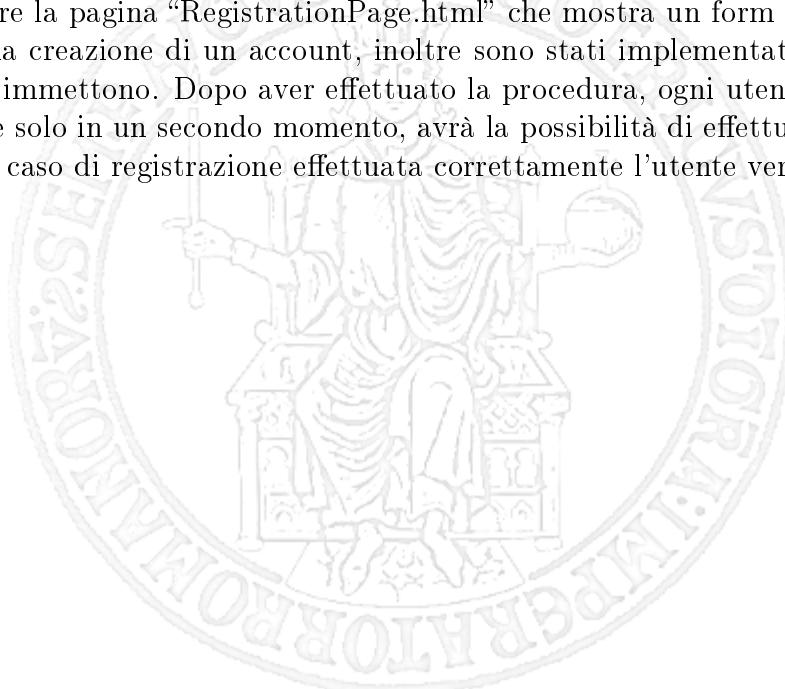
Oltre ai file html e css, lato client risiederanno anche degli script Javascript, embedded nelle pagine oppure in file separati. Si è deciso di lasciare gli script embedded nelle pagine laddove le funzioni sono specifiche e non riusabili per altre pagine. Tuttavia è possibile trovare 3 file Javascript:

- janus.js: libreria fornita da janus per l’interfacciamento clientside verso il server Janus;
- mainpage.js: file contenente script che utilizzano funzioni contenute in janus.js ed eseguono l’effettiva creazione degli handler e l’esecuzione delle operazioni per l’interazione con il server Janus e la trasmissione/ricezione dei flussi multimediali;
- jsSmartLearning.js: contiene tutte e sole le funzioni che vengono utilizzate da più pagine.

In seguito si spiegheranno, con l’ausilio di alcuni diagrammi, i principali flussi di esecuzione sviluppati.

Login e Registrazione

La prima pagina che verrà visualizzata nel caso in cui non si sia effettuato il login è “indexSmartLearning.html”. Come si può notare dalla figura sottostante, tale pagina consente di effettuare l’accesso all’applicazione se si possiedano già le credenziali. In caso di esito positivo, verrà caricata “Homepage.html”, ovvero la pagina principale dell’applicazione, dove sarà possibile effettuare la ricerca delle lezioni. Se invece non si possiedono delle credenziali, tramite il testo “registrati ora” sarà possibile caricare la pagina “RegistrationPage.html” che mostra un form attraverso il quale è possibile effettuare la creazione di un account, inoltre sono stati implementati gli opportuni controlli sui dati che si immettono. Dopo aver effettuato la procedura, ogni utente sarà registrato in qualità di studente e solo in un secondo momento, avrà la possibilità di effettuare l’upgrade ad un account docente. In caso di registrazione effettuata correttamente l’utente verrà reindirizzato alla pagina di login.



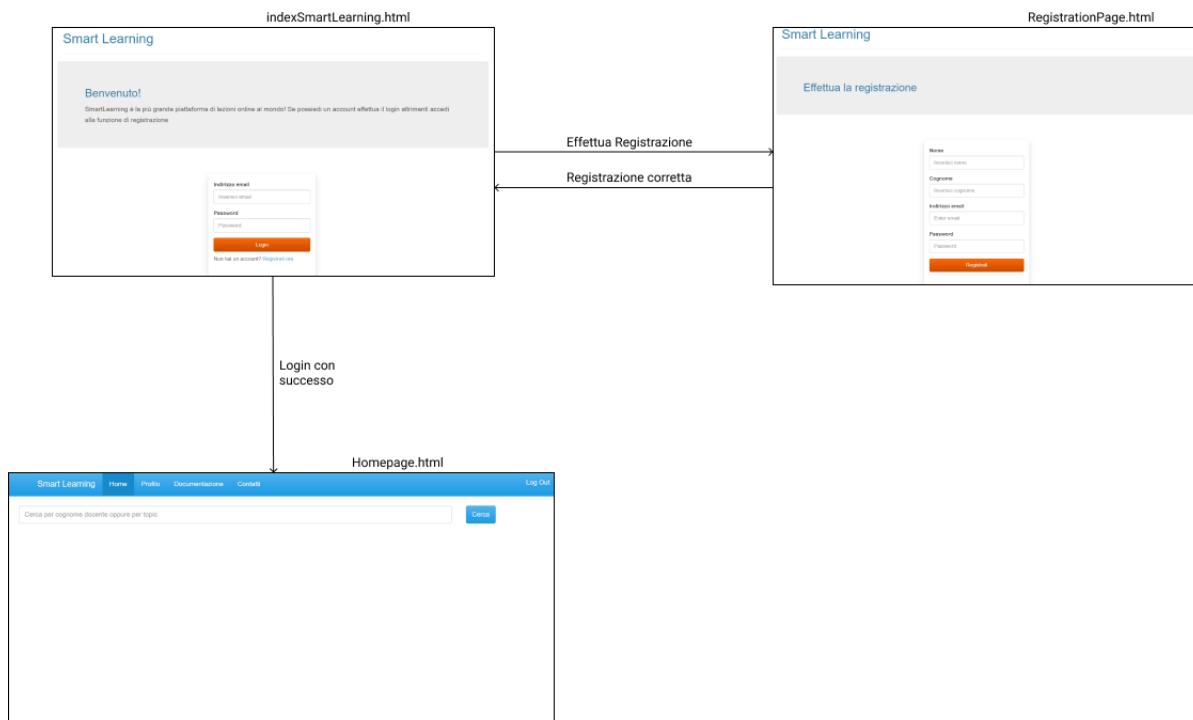


Figura 6.8: Mockup Login - Registrazione

Interfacce Studente

Effettuato correttamente il login, un utente si ritroverà nella homepage. Con riferimento alla figura sottostante, si fa notare che dalla homepage è possibile effettuare la ricerca di una lezione secondo due criteri:

- Il cognome del docente
- Il topic della lezione

Digitando un qualsiasi valore all'interno della barra di ricerca, questo verrà confrontato dapprima con i cognomi dei docenti presenti nel DB e, in caso di riscontro positivo, verranno visualizzate tutte le lezioni create dal docente, con relative programmazioni. Viceversa, se non vi è un riscontro positivo con un docente, il valore verrà confrontato con i topic, restituendo tutte le lezioni relative al topic digitato. In caso di esito negativo della ricerca, verrà mostrato un alert.

Se la ricerca ha prodotto almeno un risultato, cliccando su “prenota ora” è possibile vedere tutte le programmazioni di una lezione e prenotarsi.

Cliccando, invece, su “Profilo” nella barra di navigazione principale è possibile raggiungere la pagina “ProfiloPage.html”, la quale racchiude la maggior parte delle funzionalità fin ora sviluppate. In questa pagina si avrà a disposizione un’ulteriore barra di navigazione verticale, tramite la quale è possibile accedere a diverse informazioni:

- Al caricamento della pagina verrà mostrato il tab “utente”, dove è possibile reperire tutte le informazioni relative all’utente loggato.
- Il tab “Lezioni Sottoscritte”, il quale permette di vedere tutte le informazioni relative alle lezioni prenotate. Successivamente, cliccando su “vedi programmazioni” è possibile vedere

CAPITOLO 6. IMPLEMENTAZIONE, CONFIGURAZIONE ED ESECUZIONE

nello specifico tutte le prenotazioni per una lezione e, se la lezione è stata avviata, prendere parte alla videochiamata cliccando su “partecipa”.

- Mentre il tab “Lezioni mie” riguarda funzionalità del docente, approfondite nel paragrafo successivo.

Tornando alla barra principale, se è stato effettuato il login con un account Studente, sarà presente un tasto “Upgrade Docente”, attraverso il quale è possibile raggiungere la pagina “upgrade.html”. Qui sarà presentato un form all’interno del quale è possibile inserire l’email relativa ad un account PayPal ed il proprio Curriculum Vitae; infatti, queste informazioni sono necessarie per creare un account di tipo Docente.

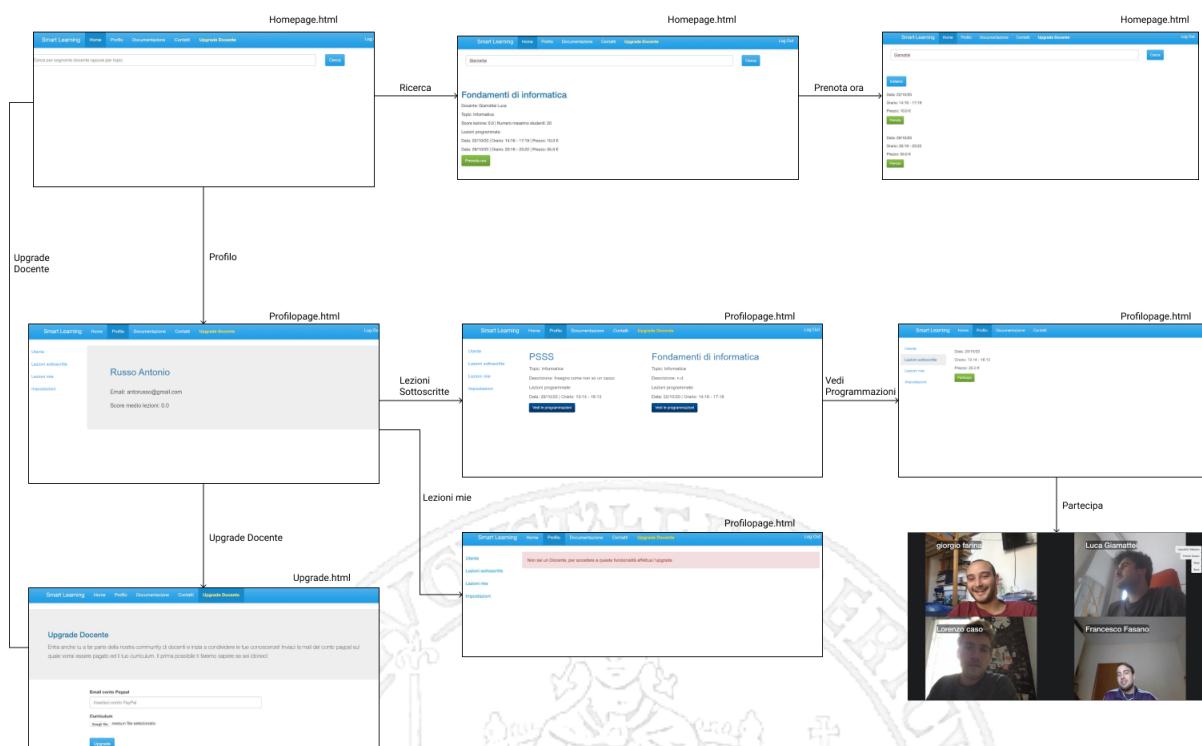


Figura 6.9: Mockup Studente

Interfacce Docente

L’account docente è un’estensione dell’account Studente e, per questo motivo, include tutte le funzionalità precedentemente illustrate.

Tuttavia il docente avrà accesso alle funzionalità del tab “Lezioni mie” nella pagina “Profilo-Page.html”. Dove è possibile vedere le proprie lezioni create con relative programmazioni, oltre a poter creare nuove lezioni e programmazioni. Nella figura sottostante è possibile vedere il form da compilare per la creazione di una nuova lezione, così come quello per la creazione di una nuova programmazione. Aperta la visualizzazione delle programmazioni, nel caso in cui ci si trovi nell’orario relativo ad una di queste, verrà visualizzato il bottone “avvia videochiamata”, che consente la creazione e l’accesso alla VideoRoom relativa alla lezione.

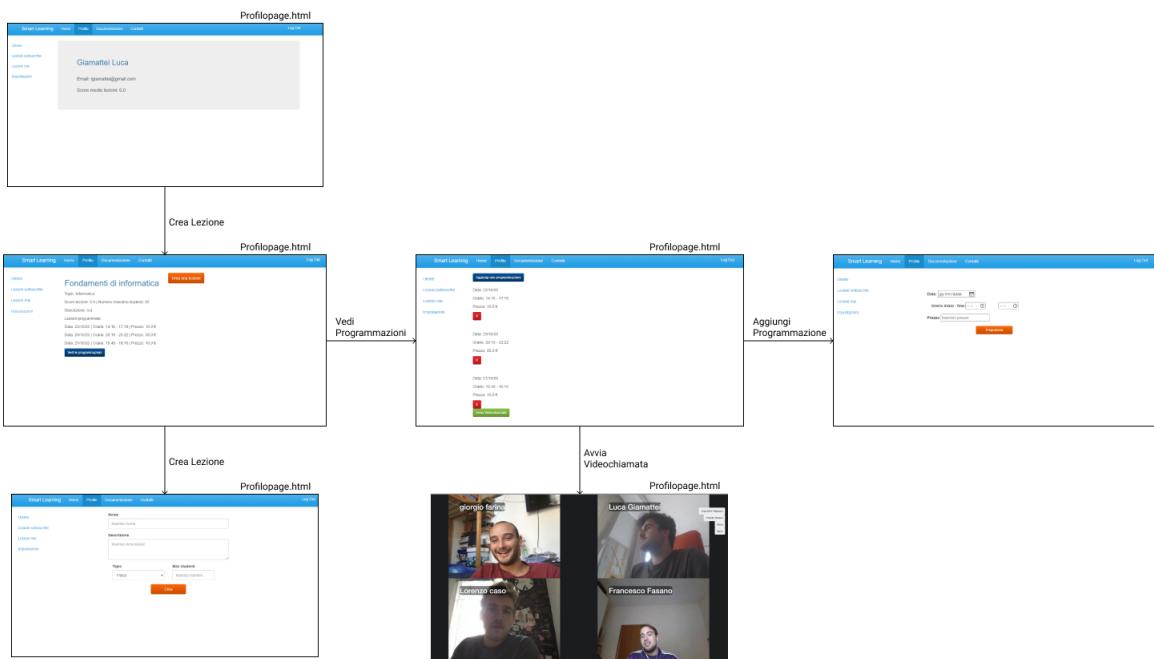


Figura 6.10: Mockup Docente

6.3 Configurazione

Per eseguire l'applicazione è necessario un web browser, nei successivi esempi è stato adoperato Google Chrome (consigliato).

Per una corretta esecuzione, in particolare delle funzionalità relative a Janus, è necessario effettuare la seguente procedura:

1. Digitare la seguente stringa all'interno della barra degli indirizzi:

```
chrome://flags/#unsafely-treat-insecure-origin-as-secure
```

2. Inserire il seguente URL all'interno dell'opzione denominata "**Insecure origins treated as secure**":

```
http://20.49.195.171:8088
```

3. Selezionare "Enabled" dal menù corrispondente;

4. Riavviare il browser mediante il comando "Relaunch Now".

Infatti il server Janus è considerato come un'origine insicura, questa procedura permetterà di trattarlo come sicuro ed eseguire la funzionalità di videochiamata.

Il procedimento è mostrato nella seguente figura:

CAPITOLO 6. IMPLEMENTAZIONE, CONFIGURAZIONE ED ESECUZIONE

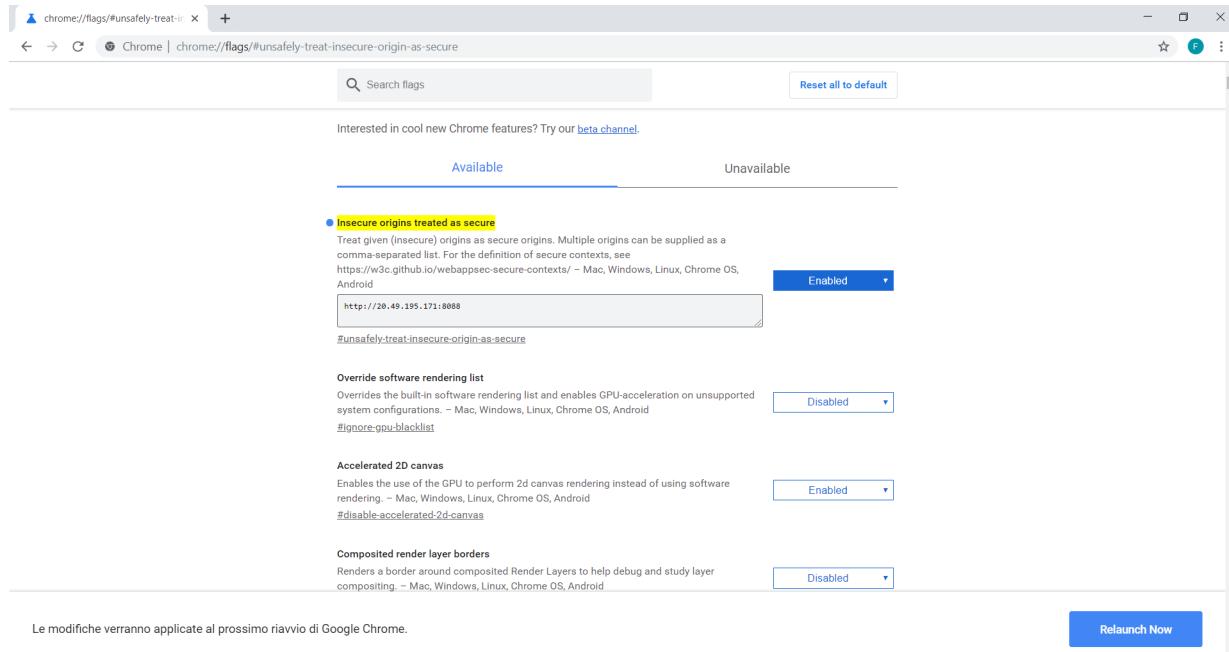


Figura 6.11: Configurazione Google Chrome



Capitolo 7

Janus

7.1 Introduzione

WebRTC è una tecnologia OpenSource, disponibile su tutti i moderni browser, che consente di effettuare videochat P2P in tempo reale.

A supporto di tale scelta tecnologica si è deciso di utilizzare il WebRTC Server Janus, in particolare di utilizzare il plugin “**VideoRoom**” di Janus che fornisce l’implementazione di una **SFU (Selective Forwarding Unit)**.

Per quanto riguarda il client, Janus fornisce delle API in javascript (`janus.js`) che astraendo i dettagli implementativi della tecnologia consentono al client di interfacciarsi con semplicità ai servizi offerti da Janus.

7.2 Interazione con Janus

Il Server Janus è stato installato e mandato in esecuzione su una macchina virtuale con sistema operativo Ubuntu 16.04 su Azure ed il suo servizio è disponibile sulla porta 8088.

La comunicazione tra il Client e il Server Janus avviene tramite protocollo **WebSocket** oppure, nel caso in cui quest’ultimo non sia disponibile, tramite **HTTP** (Long Poll). Questi protocolli permettono di avere un canale bidirezionale in modo tale che il client possa ricevere eventi asincroni.

Di seguito sono mostrati alcune problematiche alle quali si è andato incontro durante la progettazione dell’interazione con il server Janus.

7.2.1 Gestione degli accessi al Server Janus

Inizialmente, nell’architettura prescrittiva, si era deciso di fare in modo che l’interazione con Janus, per quanto riguarda la creazione e l’eliminazione di una Room, fosse limitata solo al Web Application Server Smart Learning.

Tuttavia, dopo uno studio di fattibilità, si è constatato che tale scelta implementativa avrebbe richiesto di potere eseguire codice Javascript nell’application server per fare uso della libreria “`janus.js`” messa a disposizione da Janus.

Non essendo possibile eseguire codice Javascript su Tomcat, si potrebbe delegare tale onere a un altro server, come nodeJS.

In ogni caso tale soluzione richiederebbe un grande spreco di risorse, significherebbe mantere attivo in modo continuativo un ulteriore server per ospitare un handler per comunicare Janus.

Tuttavia, una soluzione più semplice, che in questo momento non è stata adottata ed è rimandata alla prossima release, è quella di permettere l'accesso alla funzionalità di creazione e di eliminazione di una room, non all'Application Server, ma ai docenti per un certo periodo di tempo tramite dei Ticket di servizio. In tal caso, l'Application Server fungerebbe da fornitore dei token per poter accedere a tali funzionalità.

7.2.2 Gestione degli accessi alla Room

U'ulteriore problematica riguarda la gestione degli accessi alla room una volta creata, che consiste nel limitare l'accesso alla room ai soli studenti prenotati e al docente.

A tal fine la sequenza di passi eseguite da un docente per avviare una videocall è la seguente:

- Egli contatta l'Application Server per ottenere un nome della Room (univoco) e un insieme di token (il suo e quello degli studenti prenotati).
- Il Docente (Client Side) a quel punto, in modo automatico, si occuperà lui stesso di andare a istanziare una nuova Room, fornendo a Janus il nome della Room e i token, e configurando la stanza in modo tale da essere l'unico utente privilegiato, quindi in grado di poterla eliminare e fare il “kickout” degli studenti.
- Una volta creata la Room, egli vi accede con una semplice operazione di “Join”, fornendo il proprio token.

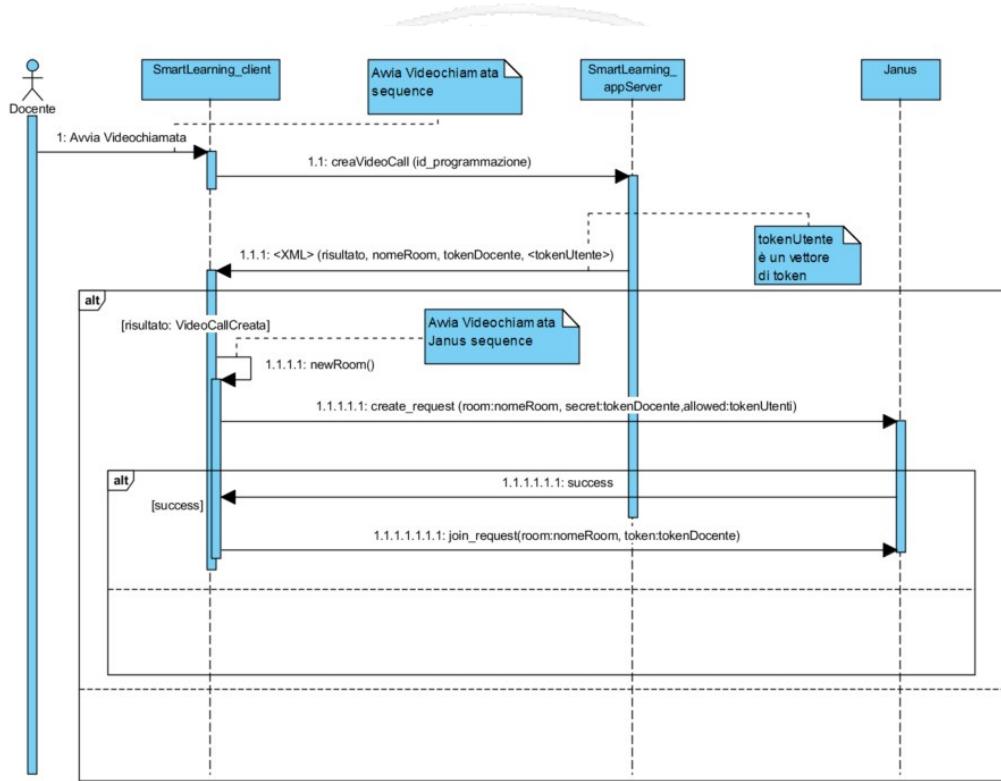


Figura 7.1: Sequence Diagram - Docente avvia videocall

In modo duale, uno studente per partecipare alla videocall:

- Egli contatta l’Application Server per ottenere il token a lui riservato per accedere alla Room;
- Lo Studente (Client Side) a quel punto si può interfacciare con Janus ed eseguire una semplice “Join” fornendo il proprio token.

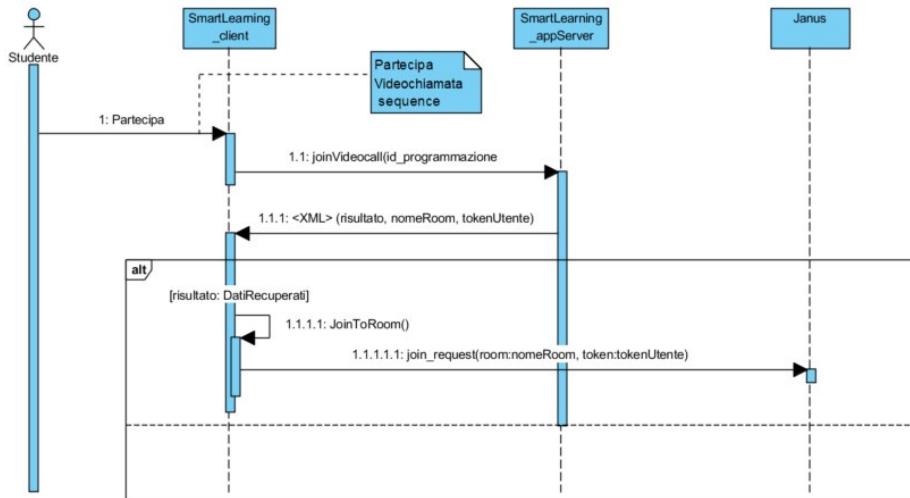


Figura 7.2: Sequence Diagram - Studente partecipa a videocall

7.2.3 Mobile-Code (Code-on-demand)

L’architettura del sistema Smart Learning, nella corrente release, prevede quindi che l’interfacciamento con Janus sia Client-Side. Infatti, per ora, si sta assumendo che l’unico possibile client sia un browser, oramai sempre provvisto di engine sia per HTML, che per Javascript.

Di conseguenza, avendo il potenziale client dell’applicazione le risorse per poter eseguire il programma, è possibile delegargli l’interazione con Janus trasmettendogli sia la libreria “janus.js” che l’applicativo “mainpage.js”.

Il vantaggio di questa scelta di design è quello di alleggerire il carico dell’Application Server, distribuendolo così sui vari client.

D’altra parte tale scelta può portare a problematiche di sicurezza: lo script javascript potrebbe essere malizioso, tuttavia l’engine javascript di cui sono dotati i browser è progettato per essere Client-Side e, di conseguenza, sono implementate delle policy di sicurezza (ad esempio che proteggono l’accesso al filesystem, alla videocamera, o al microfono) limitando le potenziali funzionalità del codice javascript.

7.3 Plugin VideoRoom

Come detto in precedenza, si è adottato il plugin “VideoRoom” di Janus, il quale fornisce l’implementazione di una SFU.

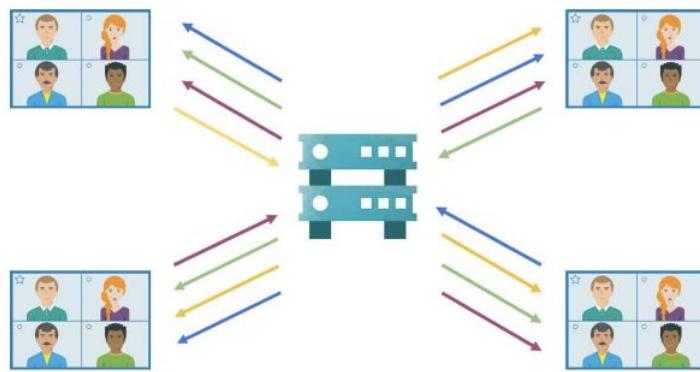


Figura 7.3: Selective Forwarding Unit

Una **SFU (Selective Forwarding Unit)** è un'entità in grado di ricevere più stream e decidere ognuno di questi a quali partecipanti deve essere reindirizzato. Uno dei principali vantaggi di questa architettura a confronto con una P2P è la scalabilità, infatti con un P2P ogni peer connesso caricherà flussi multimediali (audio, video, ecc.) $N-1$ volte (dove N è il numero di persone nella chiamata).

L'utilizzo di una SFU consente ai peer di caricare file multimediali e flussi di dati solo una volta. Il server quindi gestisce la distribuzione agli $N-1$ partecipanti. Questo essenzialmente aiuta a limitare la larghezza di banda consumata da tutti i partecipanti e consente di avere un numero maggiore di partecipanti alla chiamata.

La versione più aggiornata di Janus, inoltre, supporta il **multistream**, ovvero la possibilità di aggiungere più track dello stesso tipo ad una singola peerConnection. Nel nostro caso, ad esempio, è stato possibile gestire tutti gli stream in ricezione di un peer con una sola istanza PeerConnection.

L'utilizzo di una sola connessione presenta alcuni vantaggi, come la riduzione dei tempi di stabilimento della chiamata, poiché sono necessari meno Round di ICE(Interactive Connectivity Establishment – il protocollo per lo stabilimento della connessione tra browser) e un minor spreco di risorse lato client (comporta l'utilizzo di un minor numero di porte). Tuttavia questo approccio risulta essere più complesso nell'implementazione oltre che meno flessibile nell'aggiungere e rimuovere dinamicamente partecipanti.

7.3.1 Publish - Subscribe

Lo script “**mainpage.js**” che interagisce con Janus può istanziare fino a tre handler.

In generale, un handler instaura un canale bidirezionale di controllo e di notifica con Janus, permettendo al server Janus di fare la “push” di eventi verso i client, oltre a generare un canale dati unidirezionale (protocollo RTP) su cui viaggeranno i dati multimediali.

Nello specifico, due handler (publishers) sono entrambi dedicati a pubblicare un insieme di track che possono essere l'audio e il video della webcam o il video dello schermo, mentre il terzo handler (subscriber) è dedicato a ricevere lo stream di tracks degli altri utenti.

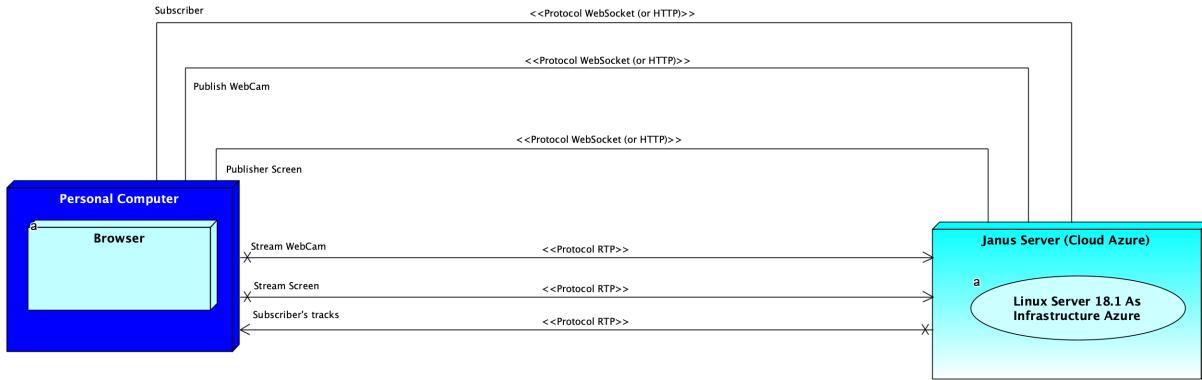


Figura 7.4: Deployment Diagram Client/Janus

Infatti, Janus adotta lo stile architetturale **Publish - Subscribe**, in particolare i topic a cui i subscribers possono sottoscriversi sono le singole track dei publishers mentre il Server Janus funge da intermediario.

Alcuni degli eventi generati da Janus verso i client sono:

- unpublished: comunica al subscriber che un publisher (a cui il subscriber era sottoscritto) ha smesso di pubblicare il proprio stream;
- publishers: comunica al subscriber che l'elenco delle tracks a cui è possibile sottoscriversi è stato aggiornato;
- leaving: comunica che un utente è uscito dalla room.

E' possibile gestire questi eventi tramite l'handler di sottoscrizione, il quale adopera un meccanismo di callback.

Per una maggiore documentazione sulla scelte implementative e, quindi, la definizione delle varie callback adoperate dai vari handler, si rimanda al codice e ai sequence diagrams.

Il vantaggio di avere Janus come intermediario è il disaccoppiamento dei client, esso è di tre diversi tipi:

- Spaziale: un client non ha bisogno di avere informazioni riguardo la raggiungibilità degli altri client;
- Temporale: i client possono essere volatili;
- Di sincronizzazione: non c'è bisogno che i client si sincronizzino, ciò evita numerosi blocchi.

D'altra parte, in una comunicazione multimediale tra N client il disaccoppiamento spaziale è indispensabile.

Lo svantaggio principale è il seguente: in questo modo Janus diventa un Single Point of Failure, ovvero nel caso di fallimento del server Janus la comunicazione tra i vari client non è più possibile.

7.4 Script di riposizionamento

Ogni utente può entrare/uscire dalla room, così come può condividere o meno lo schermo e/o la webcam. Ciascuna di queste azioni comporta una modifica nell'interfaccia degli stream. Di conseguenza, è stato implementato uno **script di ricalcolo e riposizionamento** dei video, il quale permette di avere in ogni momento una griglia ottimale rispetto alle dimensioni dello schermo dal quale si effettua la chiamata.

```

1 function recalculateLayout() {
2   Janus.log("RECALCUL");
3
4   const aspectRatio = 16 / 9;
5
6   const screenWidth = document.body.getBoundingClientRect().width;
7   const screenHeight = (document.body.getBoundingClientRect().height);
8   const videoCount1 = document.getElementsByClassName("Video-Stream").length
9     ;
10  const videoCount2 = document.getElementsByClassName("Video-Stream hide").length;
11
12  Janus.log("calcolo",videoCount1-videoCount2);
13
14  const videoCount = (videoCount1-videoCount2);
15
16 // or use this nice lib: https://github.com/fzembow/rect-scaler
17 function calculateLayout(
18   containerWidth,
19   containerHeight,
20   videoCount,
21   aspectRatio,
22 ) {
23   let bestLayout = {
24     area: 0,
25     cols: 0,
26     rows: 0,
27     width: 0,
28     height: 0
29   };
30
31 // brute-force search layout where video occupy the largest area of the
32 // container
33 for (let cols = 1; cols <= videoCount; cols++) {
34   const rows = Math.ceil(videoCount / cols);
35   const hScale = containerWidth / (cols * aspectRatio);
36   const vScale = containerHeight / rows;
37   let width;
38   let height;
39   if (hScale <= vScale) {
40     width = Math.floor(containerWidth / cols);
41     height = Math.floor(containerWidth / cols * aspectRatio);
42   } else {
43     width = Math.floor(containerHeight / rows);
44     height = Math.floor(containerHeight / rows * aspectRatio);
45   }
46   const area = width * height;
47   if (area > bestLayout.area) {
48     bestLayout = {
49       area: area,
50       cols: cols,
51       rows: rows,
52       width: width,
53       height: height
54     };
55   }
56 }
57
58 return bestLayout;
59 }
```

```

39     height = Math.floor(width / aspectRatio);
40 } else {
41     height = Math.floor(containerHeight / rows);
42     width = Math.floor(height * aspectRatio);
43 }
44 const area = width * height;
45 if (area > bestLayout.area) {
46     bestLayout = {
47         area,
48         width,
49         height,
50         rows,
51         cols
52     };
53 }
54 return bestLayout;
55 }
56
57 const { width, height, cols } = calculateLayout(
58     screenWidth,
59     screenHeight,
60     videoCount,
61     aspectRatio
62 );
63
64
65 let root = document.documentElement;
66 root.style.setProperty("--width", width + "px");
67 root.style.setProperty("--height", height + "px");
68 root.style.setProperty("--cols", cols + "");
69 }
70

```

Capitolo 8

Testing

8.1 Data Layer

Le API del data layer sono state testate tramite **JUnit5**, infatti i dinamici sono risultati utili in quanto i casi di test sono accomunati da una forte standardizzazione. È stato generato un test case per ogni API presente nel livello.

Nella seguente figura è mostrato un esempio del caso di test implementato con JUnit5 per l'API di Utente.

```
/*
 * @author PsssTeam
 * Il test richiede che nel db vi sia una tupla in Utente con password = "testPassword" e idUser = 1
 */
class TestUtente {
    private static ControllerUtenteDB controller;

    /**
     * @throws java.lang.Exception
     */
    @BeforeAll
    static void setUpBeforeClass() throws Exception {
        controller = new ControllerUtenteDB();
    }

    /**
     * Test method for {@link dataLayer.user.controller.ControllerUtenteDB#validateUser(dataLayer.utilities.idUser)}.
     */
    @Test
    void testValidateUser() {
        if (controller.validateUser(new idUser(1))==StateResult.VALID) {
            System.out.println("testValidateUser: Output VALID verificato");
        }
    }

    /**
     * Test method for {@link dataLayer.user.controller.ControllerUtenteDB#createUser(dataLayer.user.entities.UtenteDB, java.lang.String)}.
     */
    @Test
    void testCreateUser() {
        UtenteDB utente = new UtenteDB("Giorgio", "Farina", "giorgio1996.fari96@gmail.com");
        if (controller.createUser(utente, "testPassword")==StateResult.CREATED) {
            System.out.println("createUser: Id Utente: Output CREATED "+utente.getId().toString()+"\n");
        }
    }

    /**
     * Test method for {@link dataLayer.user.controller.ControllerUtenteDB#retrieveUser(dataLayer.utilities.idUser, dataLayer.user.entities.UtenteDB)}.
     */
    @Test
    void testRetrieveUser() {
        UtenteDB utente = new UtenteDB();
        if (controller.retrieveUser(new idUser(1), utente)==StateResult.VALID) {
            System.out.println("retrieveUser: Output VALID "+utente.toString()+"\n");
        }
    }
}
```

Figura 8.1: Esempio di TestCase JUnit 5

8.2 REST API

Il paradigma architetturale REST è basato sul protocollo HTTP. Anche in questo caso non è stato progettato un vero e proprio suite di casi di test per testare ogni risorsa REST, piuttosto, su Swagger, le API sono state documentate secondo lo standard OpenAPI3.0 ottenendo come risultato un ambiente di test, nel quale, come è possibile vedere dall'immagine seguente, ogni URI può essere testata facilmente modificando i parametri di ingresso alla funzione. Inoltre ad ogni metodo di ogni URI è stato documentato uno schema per la risposta con il quale è possibile sia progettare i test, ma anche validare la risposta.

ProgrammazioneDocente

POST /riservate/docente/programmazionedocente Permette a un docente di aggiungere una programmazione ad una lezione.

Permette a un docente di aggiungere una programmazione ad una lezione. La richiesta post richiede nel body l'identificativo della lezione, l'identificativo del docente e le informazioni sulla programmazione (data, orainizio, orafine, prezzo). Un docente non può aggiungere una programmazione in una data con un orario già occupato da un'altra sua programmazione.

Parameters Cancel

No parameters

Request body application/x-www-form-urlencoded

Elementi di autenticazione da validare

idlez
idlez * required
integer identificativo della lezione del docente

requesterId
requesterId * required
integer identificativo del docente

data
data * required
string data della programmazione

orainizio
orainizio * required
string orario inizio della programmazione

orafine
orafine * required
string orario fine della programmazione

prezzo
prezzo * required
string prezzo della programmazione

Execute

Figura 8.2: Esempio di Interfaccia Swagger per testare il metodo POST dell'URI “ProgrammazioneDocente”

Responses		
Code	Description	Links
200	<p>La risposta varia a seconda se è possibile aggiungere una programmazione in quella specifica data con quell'orario.</p> <p>Media type <input checked="" type="button" value="application/xml"/> <input type="button" value="application/json"/></p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>rispostaprogrammazionedocentePost ↴ { ↵ oneOf -> string example: lezioneProgrammata string example: lezioneNonProgrammata } </pre>	No links
default	errore inaspettato	No links

Figura 8.3: Schema di risposta alla richiesta POST dell'URI “ProgrammazioneDocente”

