

PROGETTO SOFTWARE ARCHITECTURE DESIGN

Prof. Anna Rita Fasolino

Studenti

Riccio Emanuele M63001339

Riccitiello Alessandro M63001354

Anno accademico 2022-2023

Contents

1	Avvio del progetto	5
1.1	Introduzione	5
1.2	Parti interessate	6
1.3	Funzionalità del sistema	6
1.4	Glossario dei termini	7
2	Processo di Sviluppo	8
2.1	Scrum	8
2.2	Pratiche agili	11
2.3	Strumenti e software di supporto	12
3	Specifica dei requisiti	13
3.1	Diagramma dei casi d'uso	13
3.2	Descrizione dei casi d'uso	14
3.3	Scenari	15
3.4	Requisiti non funzionali	18
4	Analisi dei requisiti	19
4.1	Diagramma di contesto	19

<i>CONTENTS</i>	3
4.2 Diagramma delle classi	20
4.3 Analisi architetturale	22
4.4 Stima dei costi	23
4.4.1 Valutazione casi d'uso	24
4.4.2 Valutazione attori	24
4.4.3 Valutazione Fattori di Complessità Tecnica	25
4.4.4 Valutazione complessità dellambiente	26
5 Architettura e Progettazione	28
5.1 Vista dei Componenti	28
5.2 Architettura MVVM	30
5.2.1 Model, View, View-Model	31
5.2.2 Binder	31
5.2.3 Vincoli di MVVM	32
5.2.4 Vantaggi di MVVM	32
5.2.5 Architettura Client MVVM	33
5.2.6 Architettura Server	35
5.3 Diagrammi di sequenza	37
5.3.1 SingUp	37
5.3.2 MostraUtenti - ShowUsers	39
5.3.3 AvviaChat - StartChat	39
5.3.4 AvviaDiretta - StartLive	40
5.3.5 PartecipaDiretta - JoinLive	41
5.3.6 ModificaProfilo - EditProfile	43

<i>CONTENTS</i>	4
6 Implementazione	45
6.1 Streaming audio e video	46
6.2 Autenticazione con JWT	48
6.3 API REST	49
6.4 Framework e servizi utilizzati	50
6.4.1 Node.js	50
6.4.2 React	51
6.4.3 MongoDB Atlas	51
6.4.4 Auth0	53
6.4.5 Interfaccia	54
6.5 Diagramma di deployment	58
6.5.1 Vercel & OnRender	58
7 Test	61
7.1 Unit Test	61
7.2 End-to-end Testing	61
7.3 Testing delle API	63

Chapter 1

Avvio del progetto

1.1 Introduzione

Si vuole realizzare una web app che permetta a più utenti di interagire tra di loro per migliorare e apprendere nuove lingue. Lo scopo è quello, quindi, di facilitare l'apprendimento di nuove skills linguistiche sfruttando una chat real-time scegliendo, secondo i propri interessi e quelli altrui, con chi intraprendere una conversazione. Gli utenti che vengono mostrati dal sistema sono solamente utenti compatibili. Un utente è definito compatibile se la lingua che desidera insegnare è presente tra le preferenze linguistiche di un altro utente.

L'applicazione permetterà a tutti gli utenti di creare o partecipare a dirette. Le dirette sono di tipo broadcast, i viewer possono sfruttare la chat della diretta per comunicare tra di loro. L'accesso alla diretta è soggetto a un earning system: chattando con altri utenti è possibile guadagnare punti spendibili in accesso a dirette.

1.2 Parti interessate

Il sistema software è utilizzato dall'attore **Utente**.

- **Utente:** è in grado di effettuare le operazioni di registrazione e login, è in grado di scambiare messaggi con altri utenti. L'utente può partecipare a dirette broadcast tenute da altri utenti, oppure avviare una propria diretta. Infine l'Utente ha la possibilità di visualizzare i profili degli altri Utenti con le relative informazioni, oppure modificare il proprio profilo.

1.3 Funzionalità del sistema

Il sistema deve:

- Permettere a un utente di registrarsi;
- Permettere all'utente di effettuare il login;
- Permettere all'utente di inserire e modificare i propri dati personali e interessi;
- Permettere all'utente di visualizzare i propri dati personali e il punteggio complessivo;
- Permettere all'utente di modificare le preferenze sulla lingua;
- Permettere all'utente di visualizzare utenti a esso compatibili;
- Permettere all'utente di aggiungere altri utenti ai propri contatti;

- Permettere all'utente di avviare una nuova conversazione;
- Tenere traccia dello storico delle conversazioni degli utenti;
- Associare e aggiornare il punteggio di ogni utente in base alle chat effettuate;
- Permettere all'utente di iniziare una diretta audio-video;
- Permettere all'utente di visualizzare le dirette trasmesse da altri utenti compatibili;
- Permettere all'utente di inviare e ricevere messaggi durante una diretta.

1.4 Glossario dei termini

Termine	Descrizione	Attinenza
Utente	Colui che usufruisce del servizio	Utente
Conversazione	Scambio di messaggi tra due (o pi) utenti	Utente
Chat	Sinonimo di Conversazione	Conversazione
Contatto	Utente compatibile con cui si effettua una chat	Chat, Utente
Diretta	Trasmissione audio/video real-time	Utente
Lingua madre	Lingua che l'utente è disposto a insegnare	Chat, Dirette, Utente
Lingue da imparare	Elenco di lingue che l'utente desidera apprendere	Chat, Diretta, Utente
Broadcaster	Utente che trasmette una diretta	Diretta, Utente
Viewer	Utente che partecipa a una diretta	Diretta, Utente
Earning System	Sistema di guadagno punti da parte di un utente	Chat, Utente
Punteggio	Numero di punti ottenuti da un utente	Chat, Earning System, Utente

Chapter 2

Processo di Sviluppo

Per la realizzazione di questo progetto è stato deciso di adottare un approccio agile, seguendo, in particolar modo, il framework Scrum.

2.1 Scrum

Scrum è un framework di gestione dei progetti Agile che aiuta i team a strutturare e gestire il proprio lavoro attraverso un insieme di valori, principi e pratiche. Il framework Scrum incoraggia i team a imparare attraverso l'esperienza, a organizzarsi in modo autonomo mentre lavorano su un problema e a riflettere sui risultati conseguiti e sugli insuccessi per migliorare continuamente. Scrum è, dunque, un framework euristico basato sull'apprendimento continuo e sull'adattamento a fattori variabili. Un approccio di tipo agile deve tener conto del fatto che il team non dispone di tutte le conoscenze all'inizio di un progetto e che si evolverà progressivamente con acquisizione di esperienza ed il consolidamento del team di lavoro.

Lo sviluppo secondo Scrum è basato su un insieme di artefatti:

- *Incremento*: Passo concreto verso l'obiettivo del prodotto. Ogni incremento è additivo rispetto a tutti quelli precedenti e viene accuratamente verificato, per garantire che tutti gli incrementi funzionino insieme. Per fornire valore, l'incremento deve essere utilizzabile.
- *Product Backlog*: Elenco di lavori prioritari per il team di sviluppo che deriva dalla roadmap e dai suoi requisiti. Gli elementi pi importanti sono indicati in cima al backlog del prodotto, in modo che il team sappia cosa consegnare per primo.
- *Sprint Backlog*: Insieme degli elementi del product backlog che vengono selezionati per uno sprint. Il backlog contiene anche le informazioni necessarie per consegnare l'incremento del prodotto e realizzare l'obiettivo dello sprint (sprint goal).

Lo scrum team è composto da diverse figure:

- *Product owner*: Figura responsabile di massimizzare il valore del prodotto da realizzare nonch è unico responsabile del product backlog. Ha il compito di interfacciarsi con il cliente ma anche avere una visione del valore che lo scrum team sta apportando al progetto. Inoltre, ha la responsabilità di bilanciare le esigenze degli stakeholder dell'organizzazione.
- *Scrum master*: Figura che aiuta il product owner a comprendere e comunicare meglio il valore delle pratiche Scrum, gestire al meglio il backlog, pianificare il lavoro con il team e suddividere il lavoro per ottenere un apprendimento pi efficace. Aiuta il team ad auto-organizzarsi,

superare gli impedimenti riscontrati e gestire le interazioni tra team di sviluppo ed il resto dell'organizzazione.

- *Development team*: Insieme di professionisti responsabili della realizzazione degli incrementi di cui prodotto ha bisogno. Secondo la Guida Scrum, può essere composto da tutti i tipi di persone, compresi designer, scrittori, programmatori, ecc. La dimensione del team di sviluppo può variare dai 3 ai 9 elementi. Il ruolo di sviluppatore in Scrum indica un membro del team che ha le giuste competenze, come parte del team per svolgere il lavoro.

Scrum durante il processo di sviluppo prevede diversi eventi riportati di seguito:

- *Sprint*: Breve periodo di tempo, o timebox, in cui un team Scrum collabora per completare una determinata quantità di lavoro [Durata: 15 giorni].
- *Pianificazione dello Sprint*: Evento in cui si pianifica il lavoro da svolgere durante uno sprint. In particolare viene definito cosa deve essere realizzato e come deve essere svolto il lavoro. Durante tale evento viene definito lo sprint goal, ovvero l'obiettivo da raggiungere al termine dello Sprint [Durata: 4 ore].
- *Daily Scrum*: Evento di breve durata per gli sviluppatori del team Scrum. Per ridurre la complessità, si tiene alla stessa ora e nello stesso luogo ogni giorno lavorativo. Si pianifica il lavoro da svolgere nelle successive 24 ore [Durata: 15 minuti].

- *Sprint Review*: Evento in cui avviene la revisione dell'incremento rilasciato al termine dell'ultimo sprint concluso. In questa fase avviene il confronto con gli stakeholder e vengono proposti spunti per l'incremento successivo [Durata: 3 ore].
- *Sprint Retrospective*: Evento che avviene nell'ambito del team allo scopo di migliorare il proprio rendimento in occasione dello sprint successivo [Durata: 1 ora].

2.2 Pratiche agili

Le metodologie agili hanno l'intento di presentare nel minor tempo possibile un risultato concreto ed eseguibile e soprattutto meglio adattabile a cambiamenti dei requisiti, per questo motivo, la produzione della documentazione del software è presente in misura minore nelle prime fasi del progetto, è viene costantemente aggiornata e raffinata fino al rilascio del prodotto finale.

Per documentare il software, allora, è stato deciso di introdurre i soli diagrammi indispensabili alla comprensione e le opportune viste, aggiornando progressivamente sulla base dei cambiamenti e risultati ottenuti. L'utilizzo di un approccio agile ha favorito alcune pratiche che favoriscono l'interazione e la collaborazione tra i membri del development team. Alcune di queste sono:

1. Comunicazione stretta tra i membri del team;
2. Pair programming;
3. Code refactoring;

4. Time boxing;
5. Semplicità di progettazione.

2.3 Strumenti e software di supporto

Per poter mettere in pratica il framework SCRUM, è stato sufficiente utilizzare la piattaforma Jira Software, essa permette agli utenti di creare un team di lavoro e di utilizzare un insieme di funzionalità tra le quali:

1. Definire i product goal;
2. Definire product backlog;
3. Definire un task;
4. Gestione delle storie utente;
5. Avvio di uno sprint;
6. Gestione di ruoli e responsabilità.

Per la gestione del versioning del software e allo scopo di favorire la collaborazione tra i membri del development team è stato utilizzato GitHub.

Chapter 3

Specifica dei requisiti

3.1 Diagramma dei casi d'uso

Il diagramma dei casi d'uso descrive le interazioni tra gli attori e il sistema.

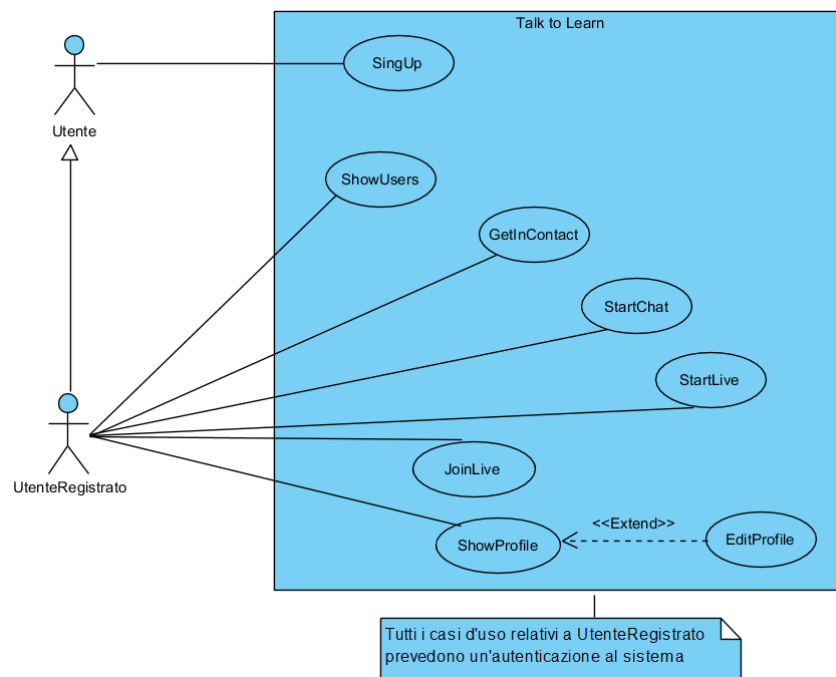


Figure 3.1: Casi d'uso

è stato possibile, a valle dell'analisi dei requisiti individuare due attori:

- Utente;
- Utente Registrato.

3.2 Descrizione dei casi d'uso

- *SignUp*: l'utente non ancora registrato chiede al sistema di registrarsi. Il sistema richiede l'inserimento di una mail e di una password.
- *MostraUtenti*: l'utente richiede al sistema di mostrare user compatibili al proprio profilo. Vengono mostrati tutti gli account compatibili con l'utente che ha fatto richiesta. Un utente è compatibile se desidera apprendere la "Lingua Madre" dell'utente con cui si interfaccia.
- *VisualizzaInformazioni*: dalla schermata di "Connect" è possibile cliccare su gli altri utenti per visualizzare il profilo dell'utente selezionato.
- *AggiungiAiContatti*: dal profilo di un utente non ancora "connected" è possibile scegliere di aggiungerlo o meno ai propri contatti per poi poter iniziare una chat.
- *AvviaChat*: Dalla lista degli utenti compatibili, viene avviata una chat con uno di essi.
- *StartLive*: Dalla relativa schermata, viene richiesta la creazione di una diretta. L'utente che ne fa richiesta inizia a trasmettere in broadcast. La trasmissione è di tipo audio-video.

- *JoinLive*: Dalla relativa schermata, viene richiesta la partecipazione a una diretta esistente. Il sistema controlla che l'utente che ne ha fatto richiesta soddisfi i requisiti (10 pnt).
- *ShowProfile*: L'utente richiede che gli sia mostrato il proprio profilo personale.
- *EditProfile*: L'utente che sta visualizzando la propria area personale può decidere di modificare alcune delle informazioni a lui legate.

3.3 Scenari

Caso d'uso	SignUp
Attore primario	Utente
Precondizioni	<p>L'utente non è ancora registrato e non è loggato</p> <ol style="list-style-type: none"> 1. L'utente, non ancora registrato, chiede al sistema di registrarsi; 2. Il sistema mostra all'utente il modulo di registrazione; 3. L'utente inserisce un nuovo indirizzo e-mail e password; 4. Il sistema controlla la validità della mail e della password; <ol style="list-style-type: none"> 4.1. if email già utilizzata <ol style="list-style-type: none"> 4.1.1 SYSTEM Error: email già in uso
Sequenza di eventi	<ol style="list-style-type: none"> 4.2. else if email non in forma standar <ol style="list-style-type: none"> 4.2.1. SYSTEM ERROR: email incorretta 4.3.elsif la password non rispetta i requisiti <ol style="list-style-type: none"> 4.3.1 SYSTEM Error: password non valida 4.4. else l'utente viene registrato correttamente <p>end if</p> <ol style="list-style-type: none"> 5. Il sistema completa il profilo utente facendo inserire dati personali e interessi
Postcondizioni	L'utente è stato registrato correttamente

Caso d'uso	Login
Attore primario	Utente
Precondizioni	L'utente è registrato 1. Il caso d'uso inizia quando l'utente chiede di effettuare il login 2. L'utente inserisce email e password 3. Il sistema controlla i dati inseriti;
Sequenza di eventi	3.1 if email o password errato 3.1.1 SYSTEM Error end if 4. L'utente effettua il login
Postcondizioni	L'utente ha effettuato correttamente il login
Casi d'uso correlati	Nessuno

Caso d'uso	MostraUtenti
Attore primario	UtenteRegistrato
Precondizioni	Devono esistere altri utenti compatibili
Sequenza di eventi	1. L'utente richiede di visualizzare utenti compatibili 2. Il sistema restituisce una lista di utenti compatibili.
Postcondizioni	L'utente può selezionare il profilo di uno degli utenti mostrati
Casi d'uso correlati	VisualizzaInformazioni, AggiungiAiContatti

Caso d'uso	VisualizzaInformazioni
Attore primario	UtenteRegistrato
Precondizioni	Devono esistere utenti compatibili 1. Dalla lista degli utenti compatibili si sceglie
Sequenza di eventi	l'utente del quale voler vedere le informazioni 2. Il sistema mostra le informazioni dell'utente selezionato.
Postcondizioni	Nessuna.
Casi d'uso correlati	Nessuno.

Caso d'uso	AggiungiAiContatti
Attore primario	UtenteRegistrato
Precondizioni	Si sta visualizzando il profilo di un utente compatibile. 1. L'utente aggiunge una nuova persona ai propri
Sequenza di eventi	contatti 2. Il sistema lo aggiunge ai contatti.
Postcondizioni	è possibile avviare una chat tra i due utenti
Casi d'uso correlati	Nessuno.

Caso d'uso	AvviaChat
Attore primario	UtenteRegistrato
Precondizioni	L'utente con cui si vuole avviare una chat deve essere aggiunto ai contatti.
Sequenza di eventi	1. Dalla lista dei contatti l'utente sceglie quello con cui iniziare una conversazione; 2. Il sistema apre la chat con l'altro utente e mostra le conversazioni avute in precedenza; 3. L'utente può scrivere e inviare il messaggio.
Postcondizioni	Il messaggio viene inviato.
Casi d'uso correlati	Nessuno

Caso d'uso	AvviaDiretta
Attore primario	UtenteRegistrato
Precondizioni	Nessuna.
Sequenza di eventi	1. L'utente (Broadcaster) che intende trasmettere avvia una diretta dall'apposita sezione; 2. Il sistema apre la diretta; 3. Il sistema permette ad altri utenti di accedervi.
Postcondizioni	La trasmissione è aperta e accessibile.
Casi d'uso correlati	Nessuno

Caso d'uso	PartecipaDiretta
Attore primario	UtenteRegistrato
Precondizioni	Deve essere avviata almeno una diretta.
Sequenza di eventi	1. Dalla schermata delle dirette l'utente decidere a quale prender parte come Viewer; 2. Il sistema controlla i requisiti d'accesso; 2.1 if l'utente non ha disposizione abbastanza crediti; 2.1.1. SYSTEM Error; 2.2. else l'utente entra a far parte della diretta. endif
Postcondizioni	L'utente può assistere e commentare la diretta.
Casi d'uso correlati	Nessuno.

Caso d'uso	VisualizzaProfilo
Attore primario	UtenteRegistrato.
Precondizioni	Nessuna.
Sequenza di eventi	1. L'utente accede alla propria area personale; 2. Il sistema ritorna tutte le informazioni dell'utente che ne ha fatto richiesta.
Postcondizioni	Viene visualizzato il proprio profilo personale.
Casi d'uso correlati	ModificaProfilo.

Caso d'uso	ModificaProfilo
Attore primario	UtenteRegistrato.
Precondizioni	L'utente deve star visualizzando il proprio profilo.
Sequenza di eventi	1. L'utente accede alla sezione di "Modifica Profilo"; 2. L'utente effettua le modifiche desiderate; 3. L'utente conferma le modifiche.
Postcondizioni	Le modifiche vengono applicate.
Casi d'uso correlati	Nessuno.

3.4 Requisiti non funzionali

- **Sicurezza:** Il sistema deve essere sicuro, non vi devono essere fughe di dati, deve essere protetto da accessi non autorizzati.
- **Performance:** Il sistema deve essere in grado di garantire sempre l'istantaneità della comunicazione, a prescindere dalla mole di utenti connessi.
- **Disponibilità:** Il sistema deve essere disponibile sempre essendo una chat real-time.
- **Usabilità:** Il sistema deve essere intuitivo, facile da comprendere e di rapido apprendimento, ogni utente dal primo accesso già deve capire quale è il modo di utilizzare la web-app.

Chapter 4

Analisi dei requisiti

4.1 Diagramma di contesto

Il primo dei diagrammi di analisi è il diagramma di contesto. Definisce il confine tra il sistema, o parte di un sistema e il suo ambiente, mostrando le entità che interagiscono con esso. Costituisce una vista ad alto livello in cui il sistema viene trattato come una black box. In questo caso, il sistema usufruisce di servizi esterni (Database).

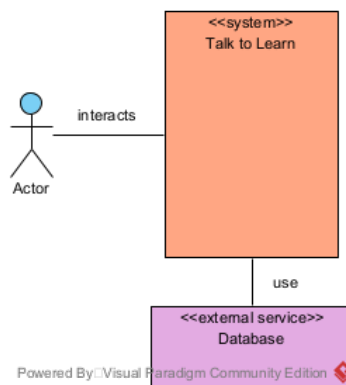


Figure 4.1: Diagramma di contesto

4.2 Diagramma delle classi

Uno dei primi diagrammi di analisi da prendere in considerazione quando si vuole modellare un sistema è il diagramma delle classi. È un diagramma statico che descrive la struttura di un sistema mostrando le classi del sistema, i loro attributi, i metodi e le relazioni tra gli oggetti.

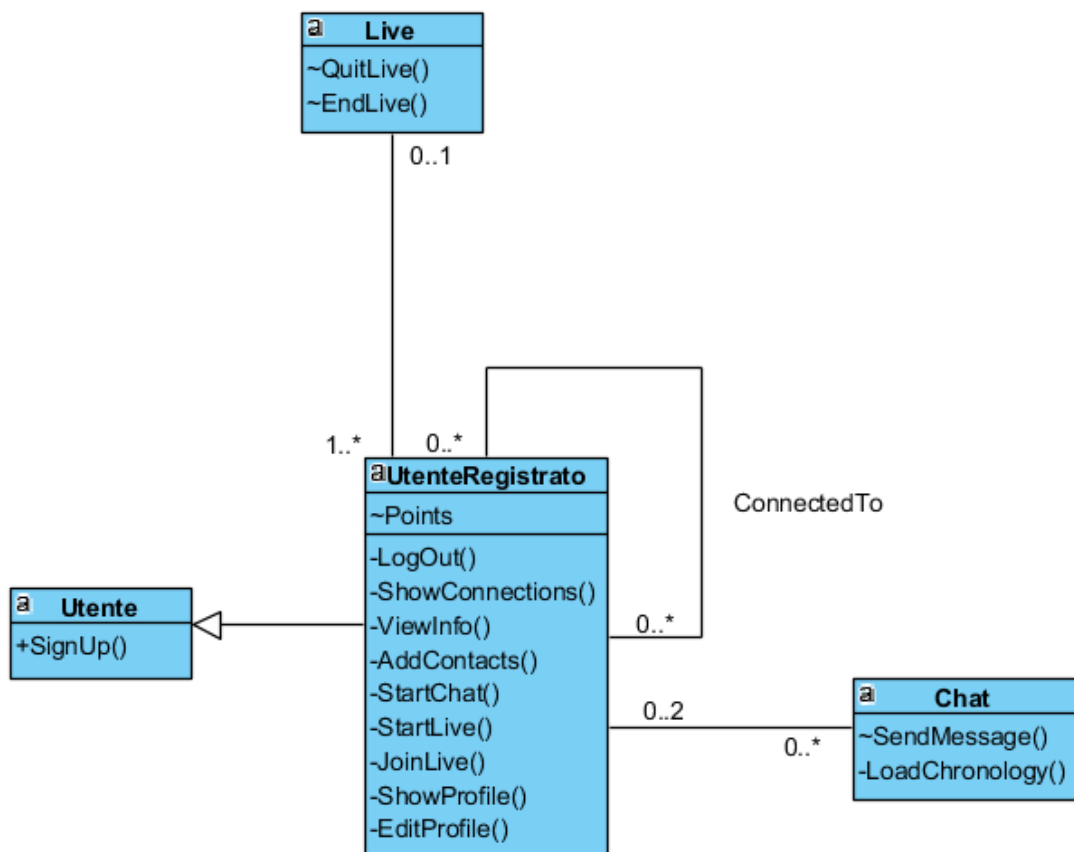


Figure 4.2: Diagramma delle classi

Il diagramma delle classi base presenta le relazioni elementari tra gli oggetti del sistema che si vuole modellare. In particolare si ha una classe *”UtenteRegistrato”*, legato alla classe *”Utente”* da un legame di specializ-

zazione, il quale è in associazione con tutte le altre classi. Si noti, inoltre, che ogni *"UtenteRegistrato"* è in associazione con s è stesso.

Procedendo con un'analisi pi raffinata è possibile introdurre due classi di supporto che permettono di assegnare e suddividere in maniera pi chiara le responsabilità. Sono state aggiunte una classe *"Vista"* che gestire le comunicazioni tra Utente e Sistema, non è altro che la prima classe posta, oltre la UI che riceve le richieste dall'utente. La classe *"Vista"* inoltre delega alla classe *"Control"* la coordinazione delle operazioni del sistema. In questo caso è utilizzato dalla classe *"Live"* e *"Chat"* e ad esso è demandata la responsabilità di aggiornare la View. Lo stereotipo *"Singleton"*, infine, stabilisce che soltanto un'unica istanza della classe stessa possa essere creata all'interno del programma.

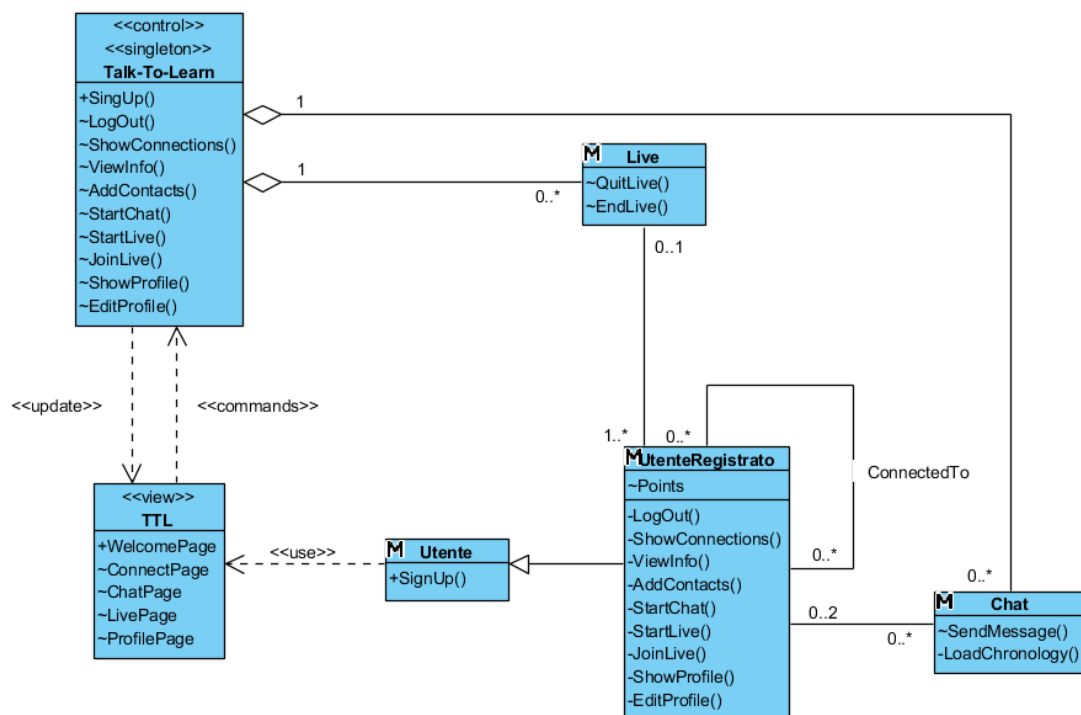


Figure 4.3: Diagramma delle classi raffinato

4.3 Analisi architetturale

La web app si configura come un software distribuito, in cui gli utenti utilizzano un client per interagire con l'interfaccia ed avere accesso alle funzionalità offerte. Un server, allora, riceve le richieste dell'utente implementando la logica della web app, gestendo la Chat real-time e le Live, inoltre il Server dialoga con un servizio esterno per la gestione dei dati.

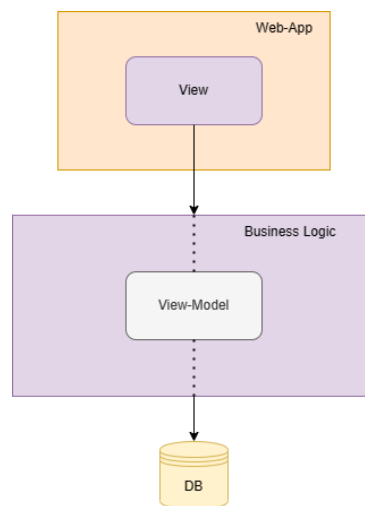


Figure 4.4: Schema logico dell'architettura "client-server"

Un modello di questo tipo, tipicamente, avrà almeno tre strati fisici differenti, con la possibilità di adottare uno schema 3 tier, in cui i livelli sono client, server e database, gestendo però opportunamente la presenza di un livello logico aggiuntivo nel server, dovendo gestire l'interfacciamento tra le diverse componenti del Server. Più avanti nella trattazione, sarà ulteriormente analizzato lo schema dell'architettura più adatto per il sistema software, soprattutto al fine di ottenere i requisiti di qualità chiave.

4.4 Stima dei costi

Per stimare i costi di sviluppo e, in generale, tenere traccia della dimensione del progetto, può essere necessario conoscere la produttività (basandosi su dati passati o approssimabili) ma anche la dimensione del team di sviluppo e l'ambiente di lavoro. La produttività, in particolare, può essere stimata adottando diversi criteri: in questo caso, è possibile considerare le Function-related measures basate su una stima della funzionalità del software rilasciato (Use-Case Point). Più complessi sono i casi d'uso, più tempo è necessario per la produzione. Essi dipendono da molteplici fattori tra i quali attori e ambiente in cui verrà sviluppato il software.

Prima di procedere con la stima vera e propria, si esaminano il numero di transazioni che interessa ogni singolo caso d'uso e si individua la tipologia dei relativi attori.

	Transazione complessive	Actor type
Registrazione	5	1
Login	4	1
MostraUtenti	2	1
VisualizzaInformazioni	2	1
AggiuntiContatti	2	1
AvviaChat	3	2
AvviaDiretta	3	3
PartecipaDiretta	2	3
VisualizzaProfilo	2	1
ModificaProfilo	3	2

4.4.1 Valutazione casi d'uso

Dalla stima della complessità di ogni caso d'uso si può ottenere un insieme di UUCW (Unadjusted Use Case Weight) considerando il numero di transazioni moltiplicate per il rispettivo peso assegnato:

$$\sum_{i=1}^3 w_i n_i \quad (4.1)$$

UseCase complexity	Weight	Number of use cases	Product
Simple	5	20	100
Average	10	6	60
Complex	15	2	30
Total		28	190

4.4.2 Valutazione attori

Un fattore correttivo che può essere considerato, riguarda la complessità dell'interazione dell'attore in ciascuno dei casi d'uso.

$$\sum_{i=1}^3 w_i n_i \quad (4.2)$$

Actor type	Weight	Number of use cases	Product
Simple	1	6	6
Average	2	2	4
Complex	3	2	6
Total			16

Ottenuto il fattore di UAW (Unadjusted Actor Weight) si possono ot-

tenere gli UUCP (Unadjusted Use Case Points):

$$UUCP = UAW + UUCW \tag{4.3}$$

Unadjusted Use Case Weight - UUCW	190
Unadjusted Actor Weight - UAW	16
Unadjusted Use Case Points - UUCP	206

4.4.3 Valutazione Fattori di Complessità Tecnica

è necessario considerare dei fattori di complessità tecnica per la realizzazione del progetto (Technical Correction Factor). Il valore di TFactor si ottiene mediante la tabella tramite 13 fattori:

$$TFactor = \sum_{i=1}^{13} w_i v_i \tag{4.4}$$

Factor	Description	Weight	Assessment	Impact
T1	Distributed system	2	5	10
T2	Performance objectives	1	5	5
T3	End-user efficiency	1	3	3
T4	Internal processing complexity	1	3	3
T5	Reusable code	1	3	3
T6	Easy to install	0.5	1	0.5
T7	Easy to use	0.5	5	2.5
T8	Portable	2	5	10
T9	Easy to change	1	4	4
T10	Concurrent use	1	4	4
T11	Security	1	5	5
T12	Access for third partes	1	1	1
T13	Training needs	1	1	1
	Total			52

Assegnati i fattori ed ottenuto il totale (TFactor), si pu calcolare il fattore di correzione definitivo secondo la formula:

$$TCF = 0.6 + (0.01 * TFactor) \quad (4.5)$$

Technical Complexity Factor - TCF 1.12

4.4.4 Valutazione complessità dellambiente

Infine si considerano i fattori relativi allambiente di lavoro. In questo caso il totale EFactor si ottiene mediante 8 fattori:

$$EFactor = \sum_{i=1}^8 w_i v_i \quad (4.6)$$

Factor	Description	Weight	Assessment	Impact
E1	Familiar with the development process	1.5	2	3
E2	Application experience	0.5	1	0.5
E3	Object-oriented experience	1	3	3
E4	Lead analyst capability	0.5	2	1
E5	Motivation	1	3	3
E6	Stable requirements	2	4	8
E7	Part-time staff	-1	1	-1
E8	Difficult programming language	-1	3	-3
Total				14.5

Assegnati i fattori ottenuto il totale ($EFactor$) si può calcolare il fattore di correzione definito secondo la formula:

$$EF = 1.4 + (-0.03 * EFactor) \quad (4.7)$$

Environmental Factors - EF 0.965

A questo punto è possibile ottenere il valore finale degli *Use Case Points*:

$$UUCP = UUCP * TCF * EF \quad (4.8)$$

Use Case Points - UCP 222.6448

Chapter 5

Architettura e Progettazione

A partire dal system context diagram e dalle analisi svolte sulla base dei requisiti funzionali e non effettuate nei capitoli precedenti, è necessario, nella fase di progettazione, intraprendere delle decisioni di design architetturale.

5.1 Vista dei Componenti

Per prima cosa andiamo a definire il diagramma dei componenti che rappresenta le relazioni tra le singole strutture che compongono un sistema in una vista di progettazione statica. Nel farlo possono essere considerati aspetti di modellazione sia logici che fisici.

Nel contesto UML, i componenti sono parti modulari di un sistema, che sono indipendenti e possono essere sostituiti da componenti equivalenti. Sono chiusi in sé e incapsulano tutte le strutture complesse desiderate. Per gli elementi incapsulati sono possibili contatti con altri componenti solo tramite interfaccia. I componenti possono mettere a disposizione le proprie interfacce,

ma anche ricorrere a interfacce di altri componenti, per avere accesso ad esempio alle loro funzioni o servizi. Allo stesso tempo, in un diagramma dei componenti, le interfacce documentano le relazioni e interdipendenze di un'architettura software.

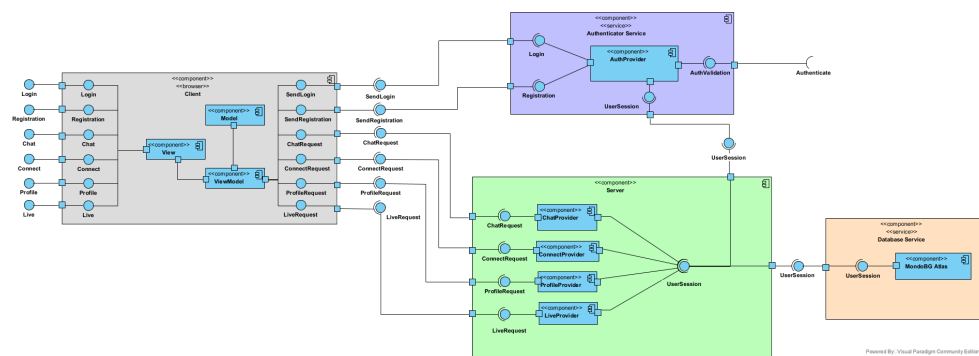


Figure 5.1: Diagramma dei componenti

Il diagramma, nello specifico presenta quattro componenti principali:

- **Client:** Web browser che offre delle interfacce per interagire tramite chat con altri utenti, per avviare e partecipare a dirette audio-video, per effettuare registrazione e/o login. La logica applicativa è quasi interamente demandata al server, al quale vi ha accesso tramite API;
- **Server:** Web server che espone risorse al client. Ha il compito di gestire tutti gli aspetti legati alla business-logic:
 - Gestisce la creazione delle dirette audio-video da parte di un utente broadcaster;
 - Gestisce la partecipazione delle dirette da parte di utenti viewers;
 - Gestisce il funzionamento della chat tra utenti;

- Gestisce la logica per fornire ad un utente altri utenti a lui compatibili;
 - Si interfaccia con il database per salvare e prelevare dati relativi a chat, dirette e utenti.
-
- **Authenticator:** Servizio esterno di auteficazione che realizza tutto il corredo di operazioni legato alle operazioni di registrazione e autenticazione di un utente. Le interazioni tra le entità esterne e questo servizio riguardano il deposito e il prelievo dei dati relativi allautenticazione di un utente.
 - **Database:** Servizio di archiviazione esterno che tiene traccia di tutte le informazioni utilizzate dal sistema, permette al sistema di gestire la persistenza dei dati.

5.2 Architettura MVVM

Per la realizzazione dell'applicazione web è scelto di adottare il pattern architetturale MVVM (Model - View - ViewModel).

La principale motivazione per cui è stato selezionato MVVM risiede nella struttura dell'architettura: indicata maggiormente per quel tipo di applicazioni che richiedono numerosi transizioni tra utente e il sistema. In particolare il pattern MVVM vuole semplificare la programmazione a eventi: un paradigma di programmazione che è caratterizzato dalla presenza di numerosi eventi esterni.

5.2.1 Model, View, View-Model

L'architettura MVVM si compone di 3 componenti principali:

- **View:** Responsabile della definizione della struttura, si occupa di realizzare la logica UI e di gestire le interazioni con l'utente.
- **View-Model:** E' un intermediario tra View e Model. Fornisce dati provenienti dal Model ad uno o più componenti View UI e trasforma gli input provenienti dalla View in azioni sul Model. Si occupa anche di trasformare i dati provenienti dal Model per prepararli alla presentazione, attraverso, ad esempio, conversioni di formato. Gestisce le dinamiche del sistema
- **Model:** Comprende la logica di dominio e di accesso al database.

5.2.2 Binder

Come ultimo componente di interesse vi è il **Binder**. Meccanismo fondamentale per questo pattern grazie al quale il view model e la view vengono costantemente sincronizzati. Le modifiche ai dati apportate dall'utente attraverso la view verranno automaticamente riportate nel view model, senza che questo onere spetti allo sviluppatore. Allo stesso modo, eventuali modifiche apportate ai dati contenuti nel view model verranno automaticamente rappresentate nella view.

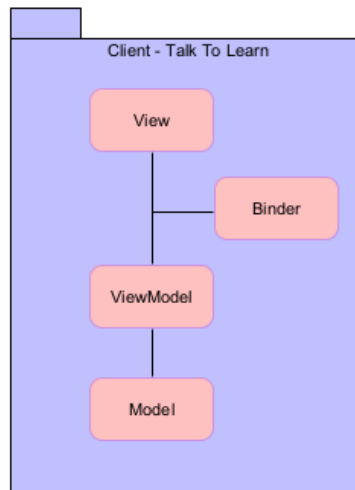


Figure 5.2: Architettura MVVM: Model - View - ViewModel

5.2.3 Vincoli di MVVM

Il pattern MVVM impone alcuni vincoli da rispettare:

- La view non deve contenere alcuna logica a meno di quella di presentazione: deve solamente implementare la UI;
- La view conosce il view-model, ma non vale il viceversa;
- La view e il model possono interagire tra loro solo attraverso il view-model: esso gestisce tutte le loro interazioni.

5.2.4 Vantaggi di MVVM

L'architettura MVVM è basata sul principio della separazione degli interessi (separation of concerns). Tale modello di programmazione si basa sull'assunzione che le classi con il quale l'utente si interfaccia non devono gestire alcun altro tipo di logica se non quella di presentazione. Su questa

base si possono elencare alcuni dei vantaggi del paradigma di separation of concerns:

- Le classi di UI risultano molto più leggere e quindi performanti;
- Si riduce la probabilità di errore;
- Favorisce il riuso, la leggibilità e la manutenzione del codice.

Inoltre il pattern MVVM è nativamente supportato da Android: in ottica di sviluppi futuri si potrebbe rilevare particolarmente utile nel caso si voglia realizzare anche una app mobile.

5.2.5 Architettura Client MVVM

Secondo le dinamiche del pattern, nell'applicazione Talk To Learn, si possono individuare i classici elementi: Model, View e ViewModel.

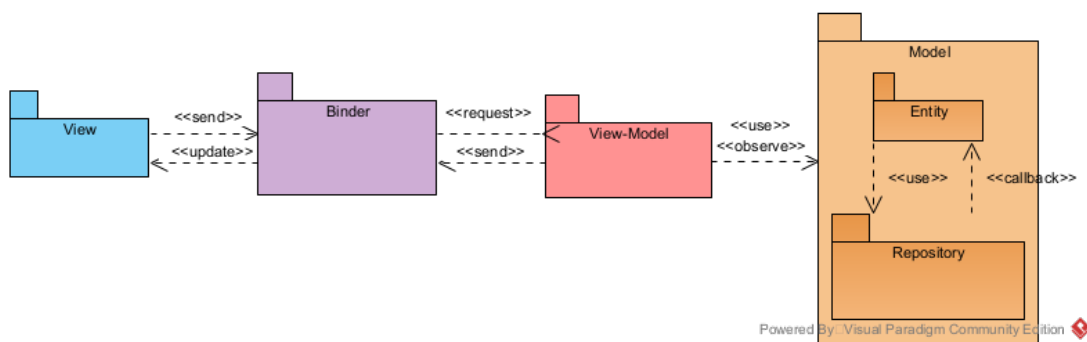


Figure 5.3: Diagramma dei package client

Il Model

Ha il compito di realizzare tutte le funzionalità necessarie per l'interazione con il database. Il **Model** presenta le stesse entità previste nel class diagram,

conservando le medesime relazioni. è possibile, quindi, individuare:

- *UserModel*;
- *ConnectModel*;
- *ChatModel*;
- *LiveModel*.

La View

Costituisce l'interfaccia permettendo la comunicazione tra utente ed entità del sistema. è possibile individuare diverse viste:

- *RegisterView*: Mostra all'utente l'intera procedura di registrazione;
- *LoginView*: Mostra all'utente i campi da compilare per effettuare il login;
- *ProfileView*: Presenta tutte le informazioni legate all'utente;
- *ConnectView*: Sono mostrati gli utenti compatibili;
- *ChatView*: L'utente utilizza questa interfaccia per interagire con altri utenti tramite chat;
- *LiveView*: Presenta una interfaccia che permette di creare o partecipare a dirette audio-video.

Il ViewModel

Utilizzato per gestire l'interazione tra View e Model, recupera i dati dal Model per ritornarli in View all'utente. Il ViewModel in **TalkToLearn** si compone di:

- *RegisterViewModel*: Si occupa della registrazione dell'utente;
- *LoginViewModel*: Gestisce login e logout;
- *ConnectViewModel*: Permette di visualizzare gli utenti compatibili;
- *ChatViewModel*: Gestisce l'interazione tra utenti tramite chat;
- *LiveViewModel*: Gestisce le dinamiche delle dirette.

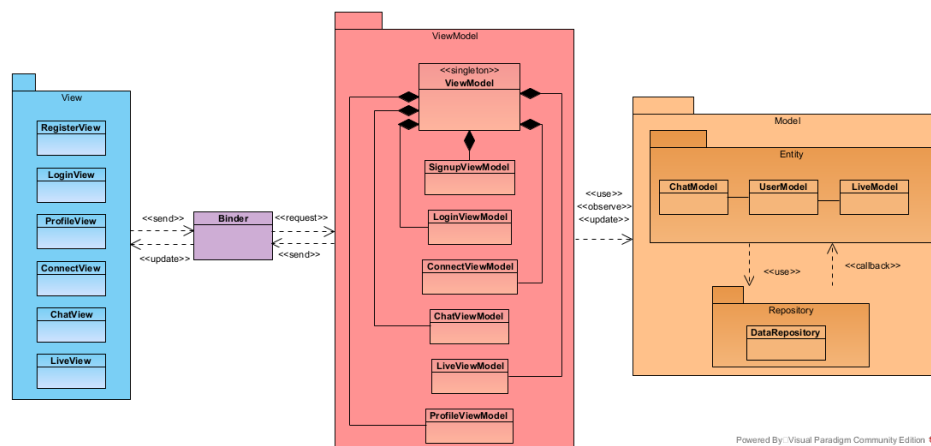


Figure 5.4: Diagramma delle classi Client

5.2.6 Architettura Server

Anche il server può essere suddiviso in classi, ognuna con i propri metodi. Il server si occupa di servire tutte quelle richieste effettuate dall'utente tramite client.

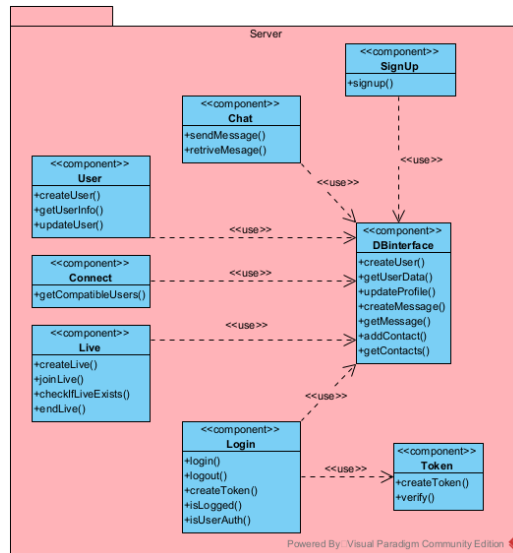


Figure 5.5: Diagramma delle classi Server

Nel sistema **TalkToLearn** si possono individuare le seguenti classi con le relative responsabilità:

- *SingUp*: Gestisce tutto ciò che è legato alla registrazione dell'utente al sistema;
- *Login*: Si occupa della autenticazione degli utenti. Esso utilizza il componente *Token* per creare un token unico da fornire all'utente al momento dell'autenticazione. Lo stesso token viene utilizzato dall'utente per utilizzare le funzionalità del sistema;
- *Chat*: Si fa carico della gestione delle chat tra utenti. Garantisce la consegna di messaggi e ricezione di notifiche e gestire la persistenza delle chat;
- *Live*: Dotato dei metodi necessari alla gestione delle dirette, si compone di:

- *Signaling*: Handshaking tra Broadcaster e Viewer, permette lo scambio delle configurazioni del tipo di codifica e bitrate;
 - *IP Candidate*: Fase nella quale Broadcaster e Viewer lavorano per sincronizzare le proprie configurazioni IP;
 - *Streaming*: Consegna dei frame audio-video; viene garantita sincronizzazione tra Broadcaster e Viewer.
-
- *Connect*: Si occupa di elaborare la lista di utenti compatibili e della loro visualizzazione quando richiesto dall'utente;
 - *User*: Gestite tutte le operazioni di CRUD relative agli utenti;
 - *dbInterface*: Utilizzato da tutte le altre classi, ha la responsabilità di coordinare l'accesso al database.

5.3 Diagrammi di sequenza

I diagrammi di sequenza (Sequence Diagram) sono utilizzati per descrivere in maniera più dettagliata un caso d'uso. Un sequence diagram, quindi, descrive le relazioni che intercorrono, in termini di messaggi, tra gli *Attori*, *Oggetti* ed *Entità* del sistema perso in esame.

5.3.1 SingUp

Un nuovo utente che vuole effettuare l'accesso al sito deve prima registrarsi, cliccando sull'apposita sezione. L'interazione viene effettuata con una schermata di View che invia una richiesta al ViewModel che si occuperà della cor-

retta gestione della registrazione. La richiesta viene inoltrata al Server di autenticazione (AuthServer in figura). In un primo istante, AuthServer ritorna un modulo di registrazione dove viene richiesto l'inserimento di email e password. L'utente, dopo aver inserito quanto richiesto, riceverà una risposta: un valore vero o falso a seconda dell'esito della registrazione. In caso di correttezza della registrazione, viene richiesto all'utente di inserire anche le sue generalità, che verranno memorizzate poi nel Database, al quale è stato già richiesto di creare un nuovo utente sulla base della e-mail fornita nella prima fase di registrazione. Tutte le varie interazioni passano sempre per la View e per il ViewModel. Al termine viene mostrato all'utente la propria pagina di profilo.

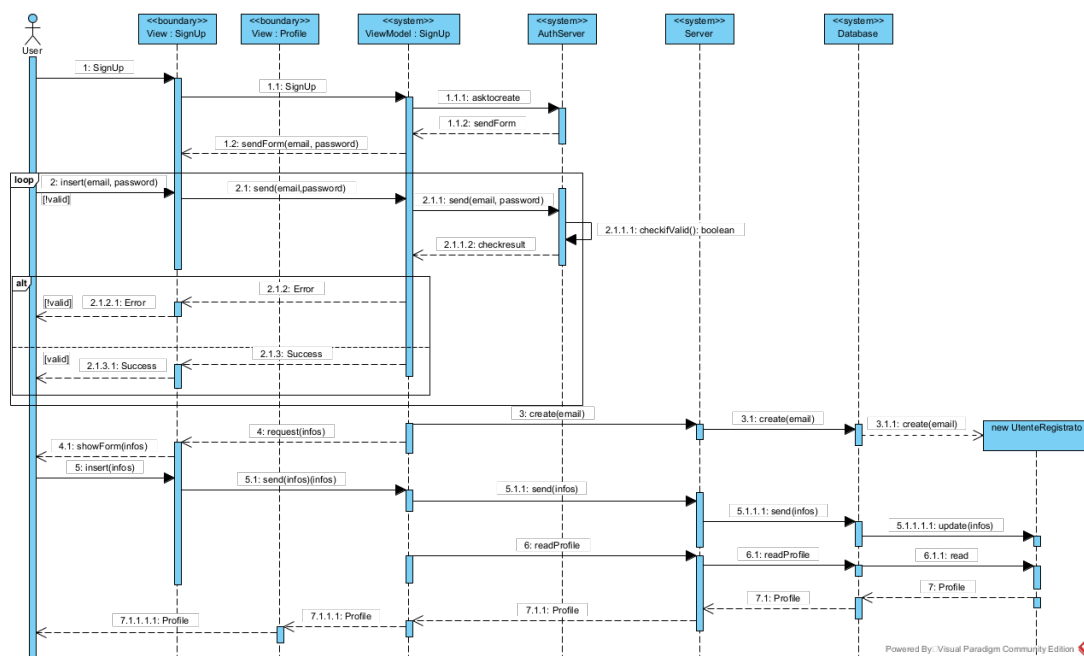


Figure 5.6: Diagramma di sequenza "SignUp"

5.3.2 MostraUtenti - ShowUsers

L'utente, per visualizzare utenti a lui compatibili, può contattare l'apposita pagina di Contacts. La vista invia una richiesta al ViewModel (Sistema) che la inoltra al server con i dati dell'utente che ne ha fatto richiesta. Il server, adesso, contatta il Database esterno chiedendo di restituire la lista degli utenti compatibili basandosi sulle preferenze linguistiche dell'utente. L'utente visualizza un messaggio di errore nel caso non ci siano utenti compatibili, oppure la lista restituita elaborata dal Database.

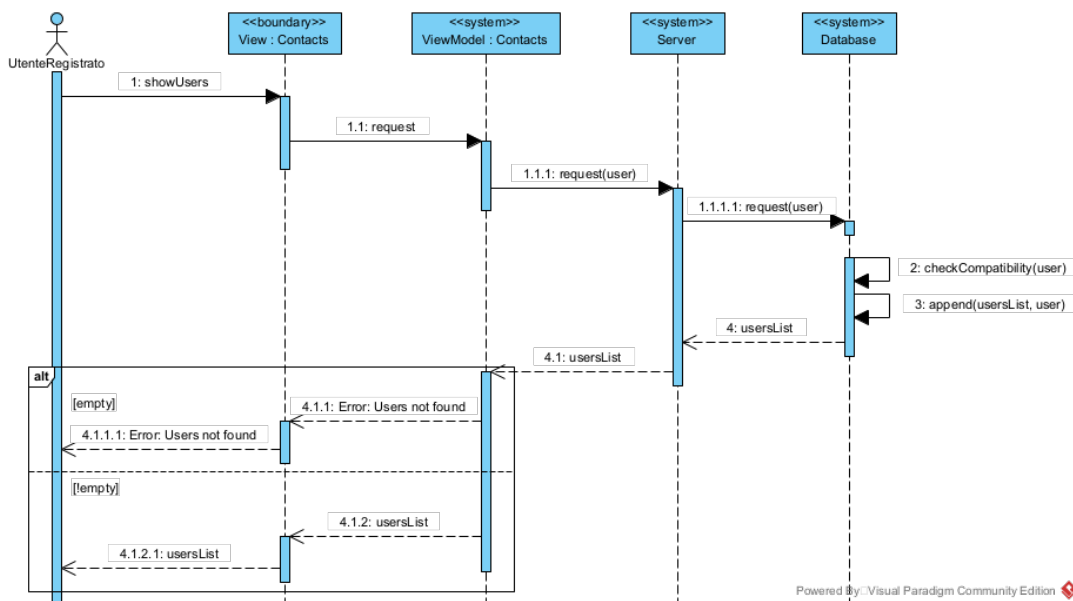


Figure 5.7: Diagramma di sequenza "MostraUtenti - ShowUsers"

5.3.3 AvviaChat - StartChat

L'utente che vuole iniziare una nuova Chat (o riprenderne una vecchia) va nella sezione apposita di "Chat" interfacciandosi con la relativa View. L'input viene rimandato al View-Model che, tramite il Server, chiede al

Database di ritornare tutti i contatti relativi all'utente attore. Da questo momento, l'utente, nel caso in cui vi siano utenti a lui collegati, può sceglierne uno qualsiasi per avviare la chat. Ancora una volta la View rimanda l'input al View-Model che, grazie all'interazione con il Server, chiede al Database di recuperare la cronologia di conversazione e di mostrarla all'utente. Ora l'utente può liberamente chattare con il suo contatto.

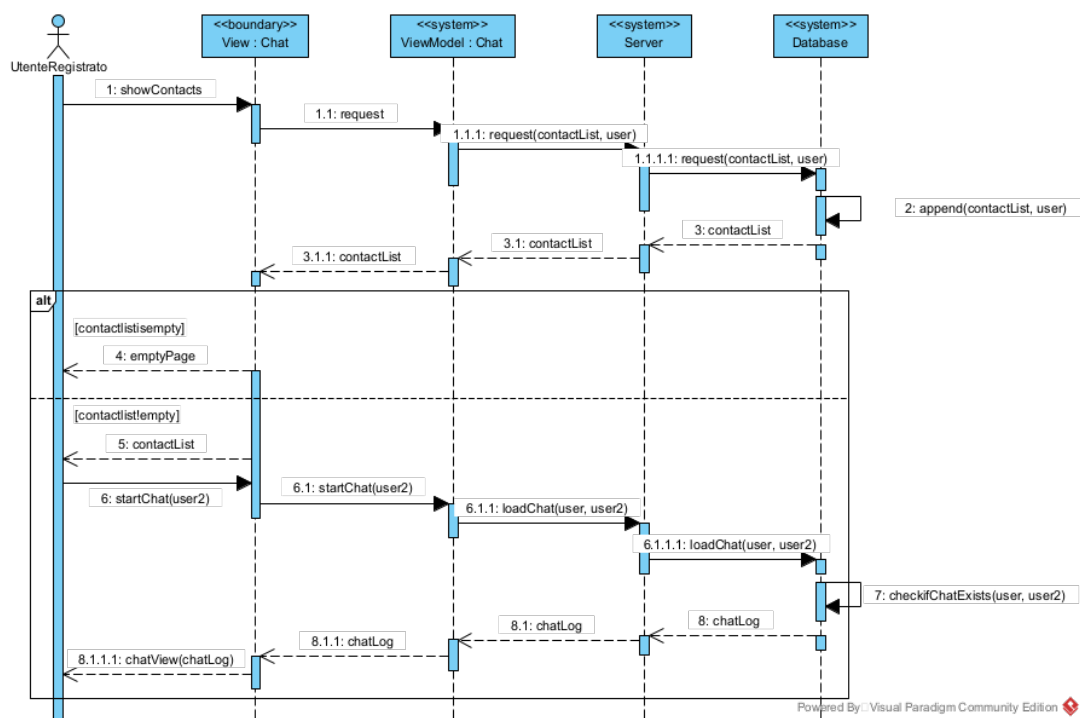


Figure 5.8: Diagramma di sequenza "AvviaChat - StartChat"

5.3.4 AvviaDiretta - StartLive

L'Utente può creare una nuova diretta streaming utilizzando la vista apposita di "Live". L'input viene inoltrato, poi, al View-Model che rimanda al Server la richiesta di creazione di una live, quest'ultimo la valuta, controllando che non ci sia già una diretta in corso da parte sua. Se il procedimento va a buon

fine viene creata una diretta pubblica a cui altri utenti possono assistere, viene inoltre etichettato come utente "Broadcaster". Al termine di tutto l'utente riceverà un messaggio da parte del server (inoltrato prima al View-Model e poi alla View). Il messaggio sarà: di successo se la diretta stream è stata avviata correttamente, di errore in caso contrario.

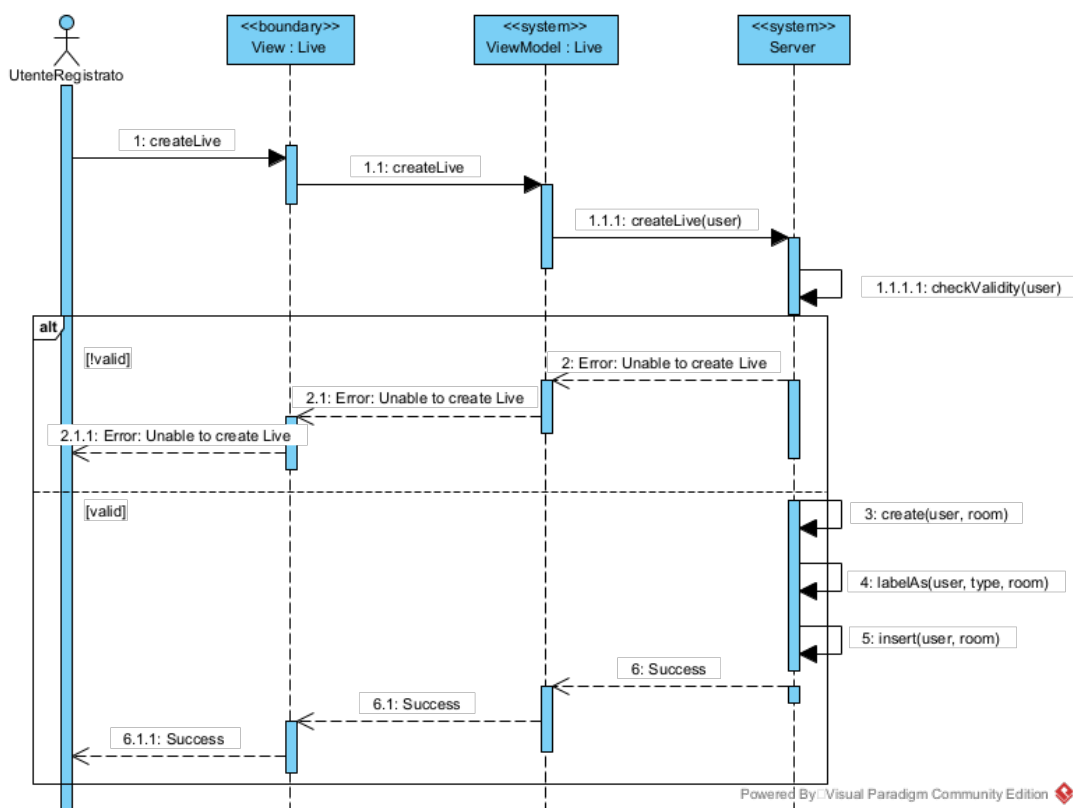


Figure 5.9: Diagramma di sequenza "AvviaDiretta - StartLive"

5.3.5 PartecipaDiretta - JoinLive

Un qualsiasi utente registrato può assistere a dirette streaming create e gestite da terzi. Per farlo è necessario utilizzare la View "Live". L'input viene rimandato al sistema che, interfacciandosi con il Server, rimanda all'utente la

lista di tutte le live attualmente in streaming. In caso non ve ne sia nessuna viene mostrato a video un messaggio di avviso. Adesso l'utente è libero di scegliere a quale live partecipare. La richiesta viene inoltrata, attraverso View e ViewModel, al Server. Il Server si occupa di controllare la validità della richiesta richiedendo al Database i dati necessari. In caso l'utente non abbia punti a sufficienza, gli viene inviato un messaggio di errore; altrimenti vengono effettuati gli update sul profilo dell'utente e viene etichettato come Viewer. Da ora in poi l'utente Broadcaster e il Viewer sono in contatto: creano la propria offerta e risposta rispettivamente e procedono al matching di IP. Una volta che questa procedura è giunta al termine, l'utente Broadcaster trasmette effettivamente al Viewer che può interagire con la chat e ascoltare la diretta streaming.

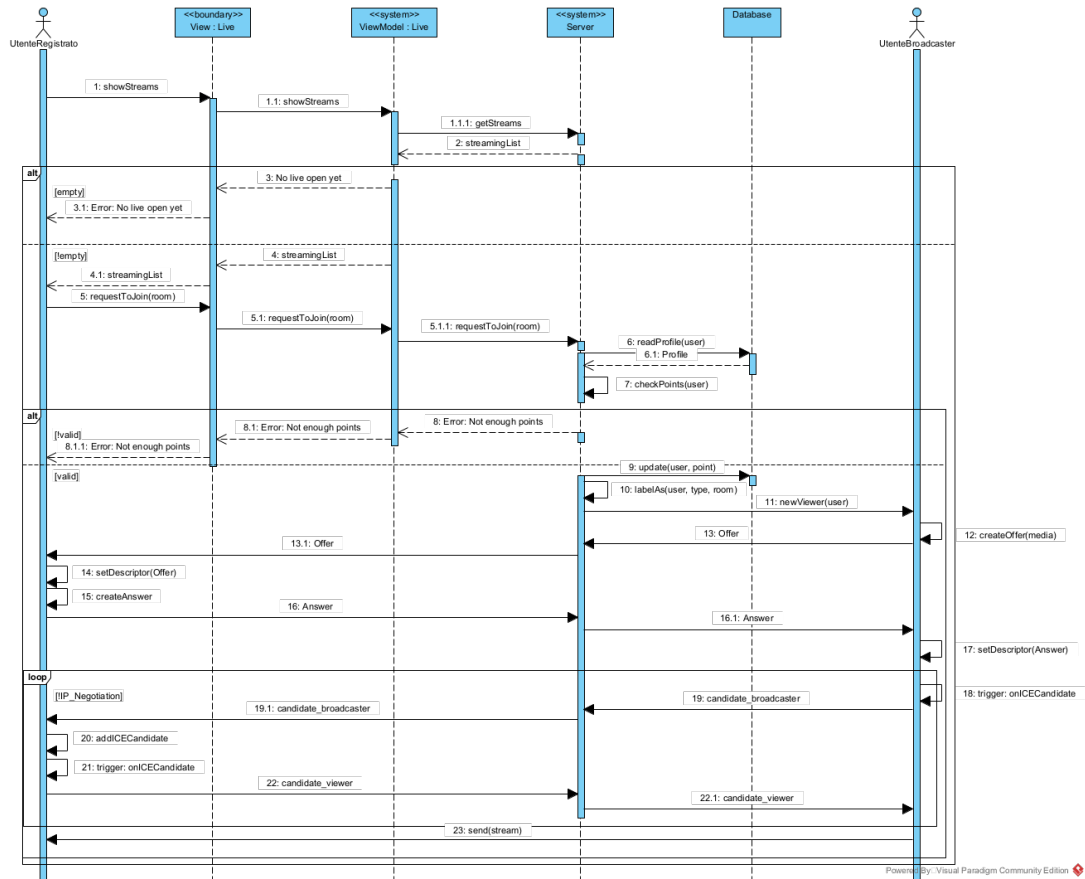


Figure 5.10: Diagramma di sequenza "PartecipaDiretta - JoinLive"

5.3.6 ModificaProfilo - EditProfile

Dalla sezione di "profilo personale", l'utente può effettuare modifiche riguardanti i propri interessi. Il sistema prende in carico le richieste e le rimanda al Server che, contattando il Database, si occupa di portare a termine le modifiche richieste. Infine verrà mostrato il profilo modificato.

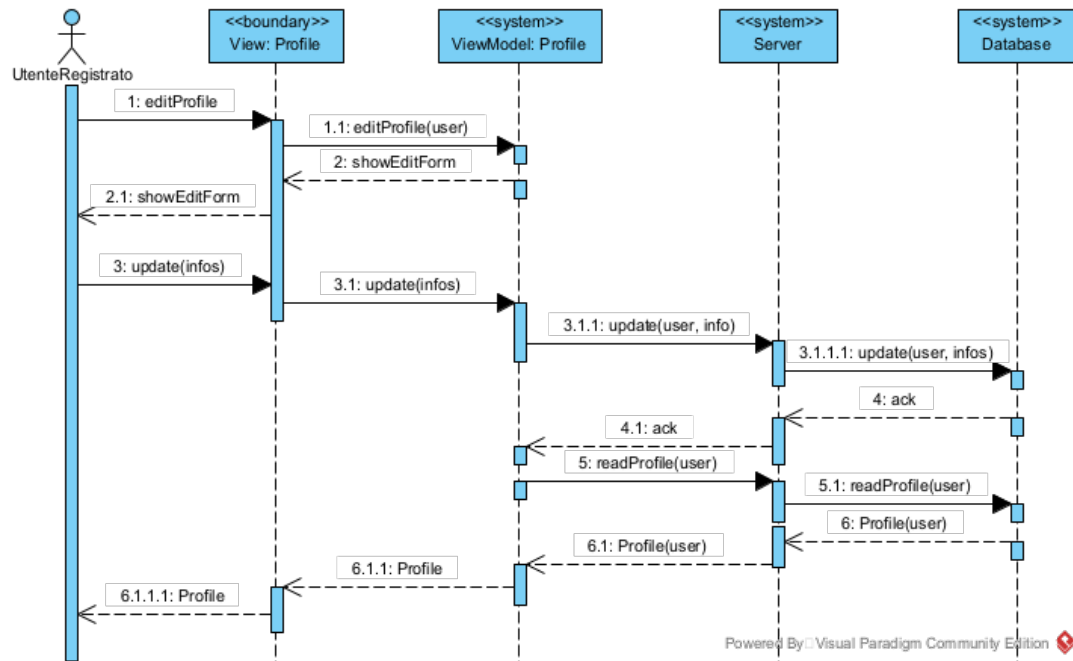


Figure 5.11: Diagramma di sequenza "ModificaProfilo - EditProfile"

Chapter 6

Implementazione

Lo sviluppo di una chat multiutente, con dirette audio-video che permetta ad utenti di utilizzare i servizi offerti dal software da remoto, ha permesso di indirizzare alcune scelte implementative rivolgendole alla usabilità del sistema, ed alle performance . Per lo sviluppo del client è stato utilizzato React, permettendo di ottenere interfacce grafiche semplici ed accattivanti per gli utenti che vi interagiscono; la scelta di React consente inoltre una elevata portabilità dell'applicazione ed una semplice configurazione.

I servizi erogati dal Server sono stati implementati utilizzando Node.js, la scelta è ricaduta su questo framework dato che esso consente di gestire in modo efficiente molteplici connessioni e comunicazioni simultanee con molti client, inoltre il framework consente di ottenere una bassa latenza per le comunicazioni. Il Server si interfaccia con il database, realizzato tramite MongoDB noSQL.

Il servizio per l'autenticazione è stato implementato utilizzando sempre Node.js, in particolare, esso si interfaccia con un servizio esterno, la pi-

piattaforma Auth0 con la quale è possibile gestire email e password associati agli utenti registrati alla piattaforma.

La descrizione specifica di ciascuno dei servizi utilizzati è fornita nella sezione successiva con particolare riferimento agli obiettivi di qualità raggiungibili.

6.1 Streaming audio e video

Il protocollo **WebRTC** (web real-time communication) è una tecnologia standard di comunicazione in tempo reale basata sul web, utilizzata per la trasmissione di audio, video e dati tra due o più dispositivi collegati a internet.

WebRTC è stato creato con l'intento di migliorare l'esperienza delle comunicazioni online, eliminando la necessità di utilizzare plugin di terze parti o di scaricare e installare software specifico. È ideale per applicazioni di comunicazione come videoconferenze, giochi in tempo reale, streaming multimediale, chat e trasferimento di file.

Il protocollo è basato su diverse tecnologie open source, tra cui VP8 e H.264 per la codifica video, Opus per la codifica audio e DTLS-SRTP per la sicurezza dei dati. WebRTC consente anche la condivisione di schermo e la gestione dei dati in tempo reale con l'API WebRTC JavaScript.

La comunicazione in tempo reale tra i dispositivi WebRTC avviene attraverso una connessione peer-to-peer, ovvero diretta, senza l'utilizzo di un server centrale. In pratica, i dati vengono trasmessi direttamente tra i browser o le applicazioni senza passare attraverso un server di terze parti.

Cio significa che i dati del protocollo webrtc sono piu sicuri e veloci rispetto ad altri protocolli di comunicazione.

In generale, il protocollo webrtc utilizza una serie di componenti, tra cui l'api `getUserMedia`, che consente di accedere alla fotocamera e al microfono del dispositivo, l'api `RTCPeerConnection`, che gestisce la connessione peer-to-peer, e l'api `RTCDataChannel`, utilizzata per trasmettere i dati in tempo reale.

Per stabilire una connessione webrtc tra due utenti, il browser dell'utente che inizia la chiamata invia una richiesta al browser dell'utente destinatario, indicando il tipo di media che vuole inviare (audio, video o dati). Se il browser destinatario accetta la richiesta, i due browser stabiliranno una connessione peer-to-peer utilizzando una tecnologia chiamata *interactive connectivity establishment* (ice), che consente di negoziare la migliore strada di connessione tra i due browser. Una volta stabilita la connessione peer-to-peer, webrtc utilizza i codec audio e video per codificare e decodificare i dati di comunicazione.

Al termine della conversazione, la connessione viene chiusa e il browser libera tutte le risorse utilizzate. Per garantire la sicurezza dei dati, il protocollo webrtc utilizza la crittografia end-to-end, ovvero i dati vengono crittografati dal mittente e decrittati dal destinatario senza che possano essere intercettati durante la trasmissione.

Infine, grazie al protocollo webrtc, gli sviluppatori possono creare applicazioni per la comunicazione in tempo reale, come le videoconferenze o le chat, in modo rapido, semplice e sicuro, senza la necessita di utilizzare server di terze parti.

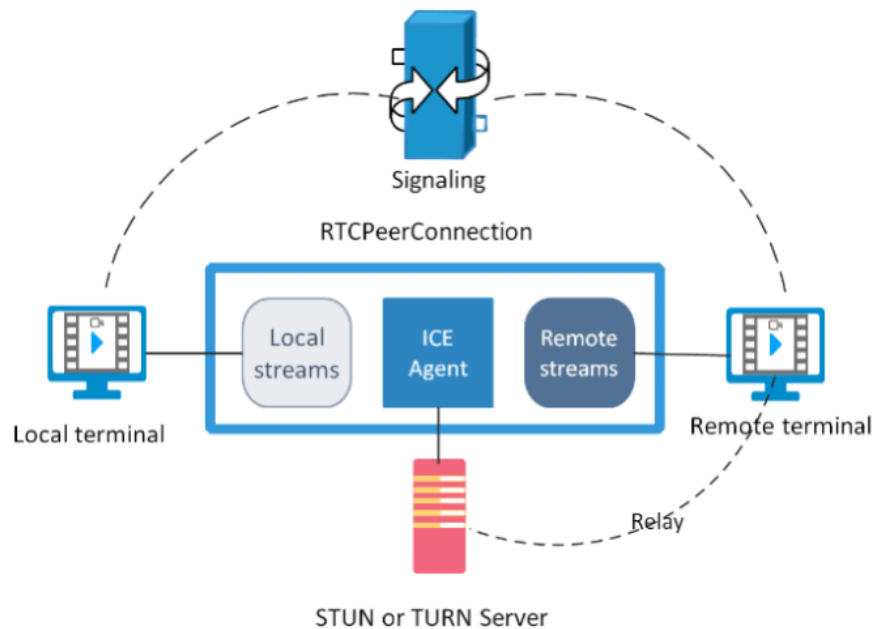


Figure 6.1: Architettura WebRTC

6.2 Autenticazione con JWT

Il meccanismo di autenticazione implementato, prevede il rilascio di un token. JWT è l'acronimo di JSON Web Token ed è un sistema di cifratura e di contatto in formato JSON per lo scambio di informazioni tra i vari servizi di un server. Ogni volta che un utente, comunica al server le sue credenziali di accesso questo risponde rilasciando un token di accesso che resta valido (per un certo tempo) per gli accessi dell'utente al sistema. Ogni token dispone, oltre di header e payload, anche di una firma, utilizzata per implementare un meccanismo di cifratura a chiave pubblica. Il client comunica username e password all'autenticazione service. Se le credenziali di accesso sono corrette, il server invia in risposta un JWT. Il client potrà usufruire degli altri servizi del

sistema, senza effettuare nuovamente l'accesso e sfruttando il token precedentemente assegnatogli. Nello specifico la generazione e la gestione dei token sono affidati ad un servizio esterno denominato Auth0, il quale dialoga con il Server.

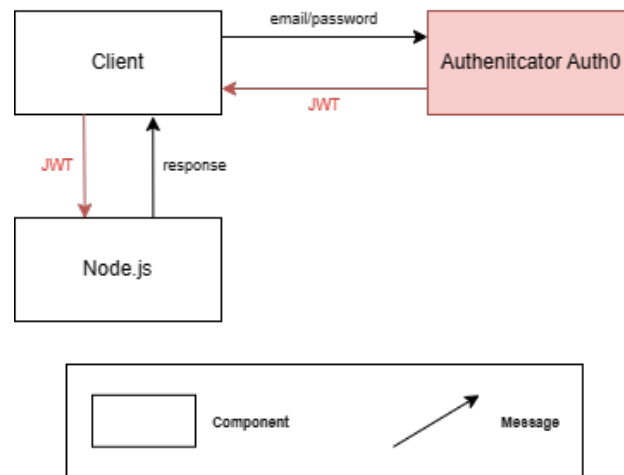


Figure 6.2: Logica di autenticazione

6.3 API REST

Per realizzare la comunicazione tra Client e Server è stato adottato l'interfaccia API REST. Le API REST sono un tipo di architettura software per la creazione di servizi web. Consentono a diverse applicazioni di comunicare tra loro attraverso la trasmissione di dati in formato JSON o XML. In pratica, le API REST funzionano come delle interfacce attraverso cui si possono inviare richieste ad un server per ottenere informazioni o per eseguire operazioni. Ad esempio, un'applicazione di e-commerce potrebbe utilizzare le API REST di un servizio di pagamento per elaborare le transazioni di pagamento. L'acronimo REST sta per "Representational State Transfer", ovvero

trasferimento di stato rappresentazionale. Cio è, le API REST si basano sulla rappresentazione degli oggetti e delle risorse (come ad esempio prodotti, utenti, transazioni) in formato testuale, che possono essere letti e modificati tramite richieste HTTP. Di seguito si riporta il link alla documentazione dell'API REST del progetto TalkToLearn realizzata con Postman.

Documentazione TalkToLearn API REST

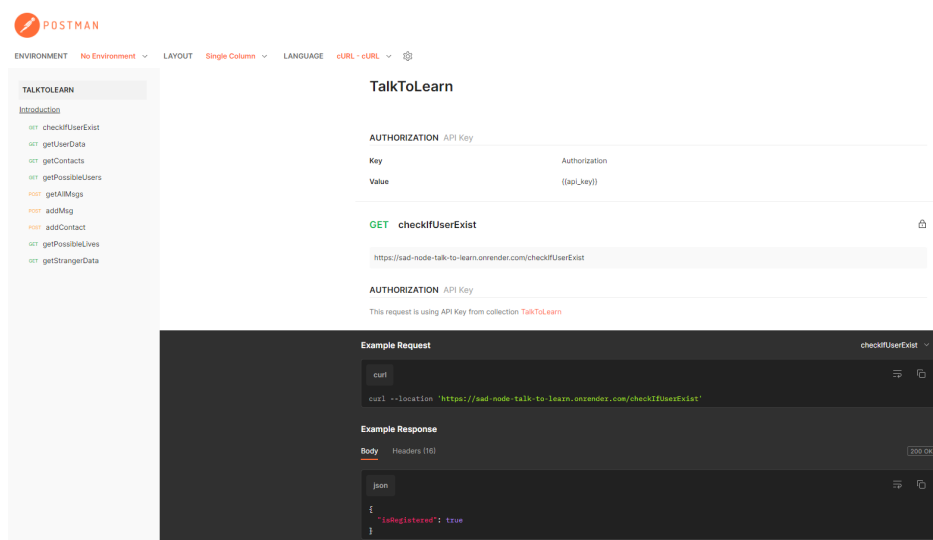


Figure 6.3: Documentazione Postman

6.4 Framework e servizi utilizzati

6.4.1 Node.js

Runtime system open source multiplatforma orientato agli eventi per l'esecuzione di codice JavaScript. Consente di utilizzare JavaScript anche per scrivere codice da eseguire lato server, ad esempio per la produzione del contenuto delle pagine web dinamiche prima che la pagina venga inviata al

browser dell'utente. Node.js in questo modo permette di implementare il cosiddetto paradigma JavaScript everywhere, unificando lo sviluppo di applicazioni Web intorno ad un unico linguaggio di programmazione. Ha un'architettura orientata agli eventi che rende possibile I/O asincrono. Questo design punta ad ottimizzare il throughput e la scalabilità nelle applicazioni web con molte operazioni di input/output. È inoltre ottimo per applicazioni web sistema real-time (programmi di comunicazione in tempo reale o browser game).

6.4.2 React

React rende semplice la creazione di UI interattive. Permette la progettazione di interfacce per ogni stato dell'applicazione. Ad ogni cambio di stato React aggiorna efficientemente solamente le parti della UI che dipendono da tali dati. La natura dichiarativa dell'UI rende il codice più prevedibile e facile da debuggare. Consente la creazione di componenti in isolamento, componibili per creare UI complesse. Dato che interazioni e logica per i componenti sono implementate in JavaScript si può facilmente passare ed accedere strutture dati complesse in vari punti dell'applicazione. Si è scelto di utilizzare questo web framework in quanto può effettuare rendering lato server con Node e in applicazioni mobile grazie a React Native.

6.4.3 MongoDB Atlas

MongoDB è un sistema di gestione di database documentale, open source e distribuito. MongoDB consente di archiviare e recuperare dati non strut-

turati attraverso un sistema di rappresentazione dei dati chiamato BSON, una versione binaria di JSON, questo gli conferisce un'alta flessibilità. MongoDB è progettato per scalare su più nodi senza compromettere le prestazioni, in questo modo è possibile anche replicare i dati in modo da garantirne un'alta disponibilità dei dati e una migliore tolleranza ai guasti. I database NoSQL sono in grado di gestire grandi volumi di dati in modo rapido ed efficiente, senza dover utilizzare complesse relazioni tra tabelle come nei RDBMS.

In questo progetto è stato utilizzato il servizio MongoDB Atlas, il quale fornisce un'istanza del database MongoDB, fornisce un'interfaccia per accedere ai dati, strumenti avanzati per l'amministrazione e il monitoraggio dei database.

In sintesi, i database NoSQL sono utilizzati per fornire una soluzione di data management agnostica rispetto ai modelli relazionali, offrendo benefici in termini di scalabilità, flessibilità, costi, prestazioni e disponibilità.

Di seguito si riporta la documentazione ufficiale utilizzata per integrare il servizio MongoDB nel sistema:

Guida Node.js MongoDB Atlas

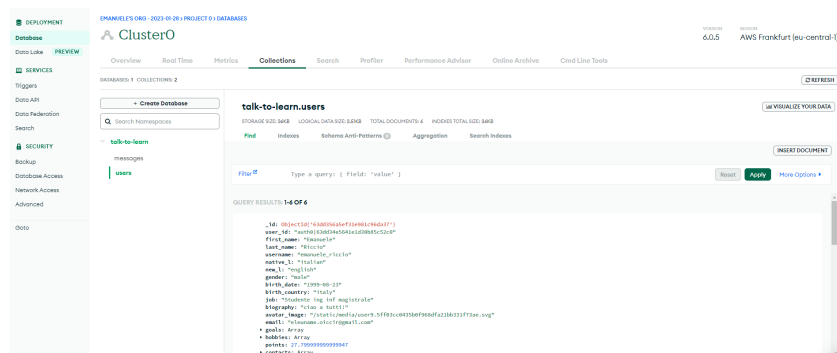


Figure 6.4: MongoDB Atlas

6.4.4 Auth0

Auth0 è un servizio di gestione delle identità e degli accessi, progettato per semplificare la creazione, l'autenticazione e l'autorizzazione degli utenti nei siti web e nelle applicazioni. Il suo obiettivo principale è offrire un'esperienza utente sicura e fluida, eliminando le complicazioni delle registrazioni tradizionali.

Auth0 fornisce un'ampia gamma di funzionalità, tra cui:

- *Autenticazione personalizzata*: consente agli utenti di accedere al sito utilizzando le proprie credenziali social media o di terze parti;
- *Controllo degli accessi*: offre un sistema di autorizzazione per gestire l'accesso agli utenti in base al loro ruolo e alle autorizzazioni necessarie;
- *Integrazioni*: permette l'integrazione con molte piattaforme di terze parti, riducendo al minimo la necessità di scrivere codice personalizzato;
- *Monitoraggio metriche*: consente di monitorare le prestazioni e la sicurezza di un'applicazione e di fornirne analisi approfondite.

Il servizio Auth0 è accessibile tramite una dashboard online, dove gli sviluppatori possono impostare le proprie preferenze di autenticazione e autorizzazione. Inoltre, è possibile personalizzare l'esperienza di autenticazione degli utenti per garantire che sia coerente con lo stile e il design generale del sito web o dell'applicazione.

In sintesi, Auth0 è una soluzione di autenticazione e autorizzazione sicura ed efficiente, ideale per le aziende e per gli sviluppatori che cercano di semplificare la gestione delle identità e degli accessi degli utenti.

La configurazione è molto semplice, di seguito si riporta la guida ufficiale per implementare agevolmente l'autenticazione nel proprio sistema, integrando Frontend e Backend:

Guida React Auth0

Guida Node.js Auth0

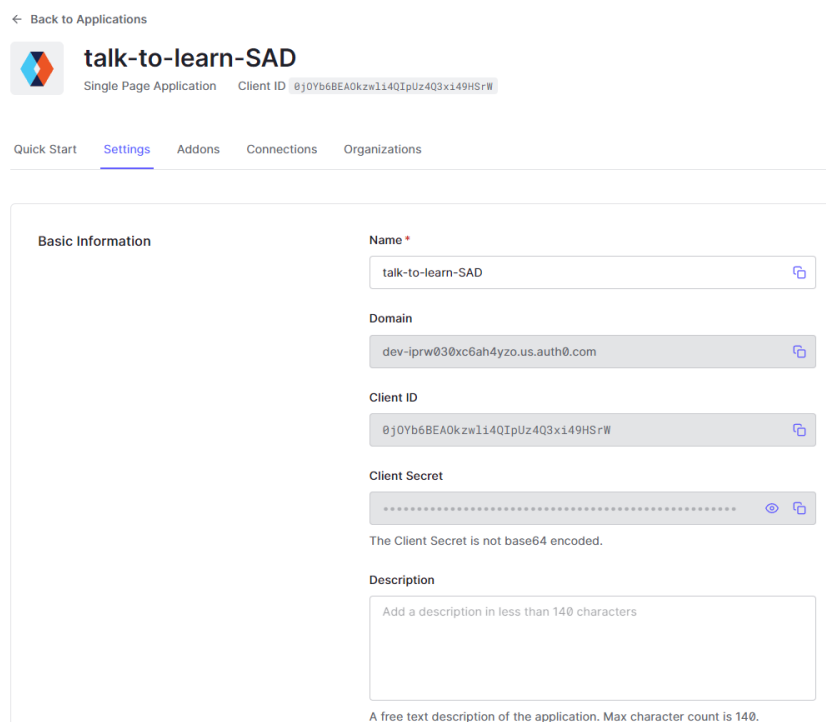


Figure 6.5: Auth0

6.4.5 Interfaccia

Più volte è stato sottolineato quanto è importante disporre di un folto bacino di utenti da cui attingere. Il successo di una chat multiutente si può misurare soprattutto in funzione dell'affluenza di utenti nel tempo. La chiave per venire incontro alle esigenze di utenti diversi per età e abilità nell'utilizzo di dispositivi digitali, rende molto importante la cura dell'usabilità e della portabilità

dell'applicativo. Se la scelta di React/Node garantisce un ottimo grado di portabilità, altrettanto importante è la gestione delle interfacce grafiche.

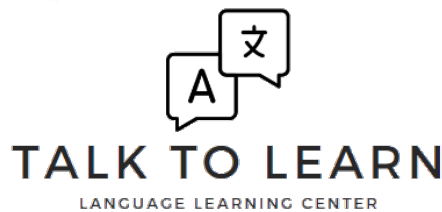



Figure 6.6: Logo


Per rendere l'applicazione unica e riconoscibile è stato creato un logo che richiamasse la funzionalità della piattaforma, cioè quella di imparare lingue tramite chat. L'applicazione presenta una pagina di login composta da form che consentono l'inserimento di username e password e, nell'eventualità di nuovi utenti, un link che ne consente la registrazione. Di seguito si riportano le interfacce grafiche delle diverse sezioni del sistema:



TALK TO LEARN
LANGUAGE LEARNING CENTER

Welcome

Log in



[Forgot password?](#)

Continue

Don't have an account? [Sign up](#)

Figure 6.7: Login page

1

Finished

This is a description.

2

In Progress

This is a description.

3

Waiting

This is a description.

TALK TO LEARN

Learn a new language

Setting your data

First Name

Last Name

Username

Native Language

New Language

Select your gender

Your birth date

Select your native country

Your job

Write a short description of you

Next

Figure 6.8: SingUp Page

TALK TO LEARN

Learn a new language

CONNECT

CHAT

LIVE

PROFILE

LOG OUT

alericc

Hello! I love studying SAD!

EDIT

male

1999-03-23

England

student

Full Name

alessandro riccitelli

Email

utente5@utente5.it

Wants to teach

English

Wants to learn

Italian

Points

2.2

Why I want to learn a new language

travel

Knowledge

study

Hobbies

party

play

canoa

Figure 6.9: Profile page

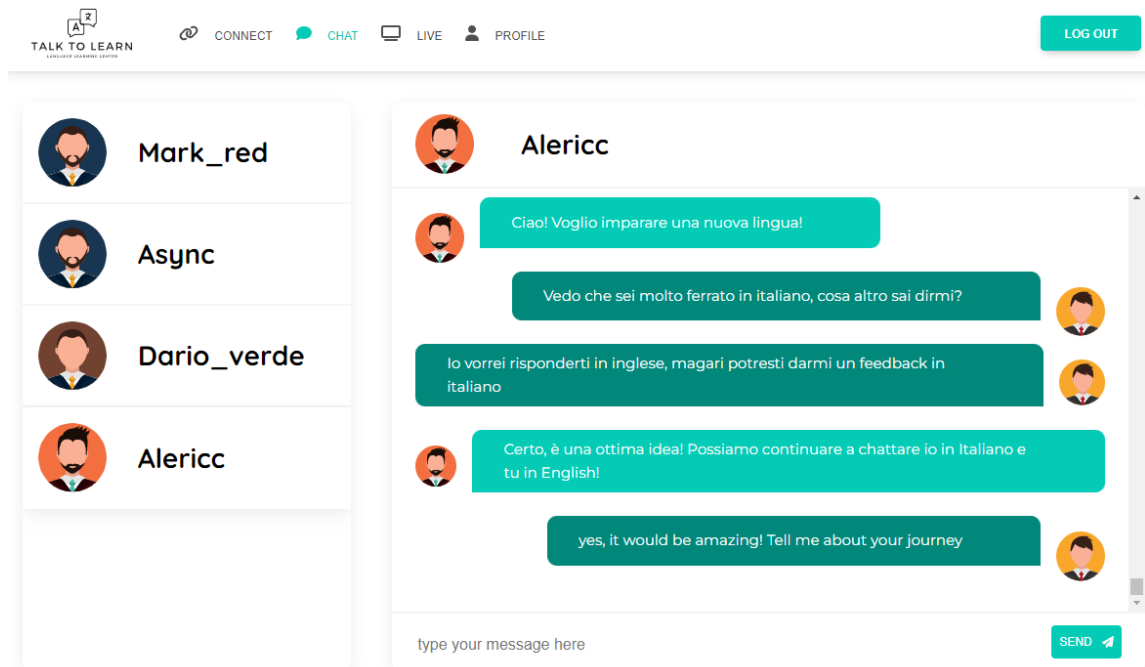


Figure 6.10: Chat page

6.5 Diagramma di deployment

6.5.1 Vercel & OnRender

Vercel & OnRender sono servizi di distribuzione di applicazioni e servizi Web sviluppati in Java, .NET, Node.js, Python etc. su server comuni come Apache, Nginx, etc. Caricando il proprio codice, gestiscono automaticamente il deployment, dal provisioning di capacità e autoscaling al monitoraggio della salute della applicazione. Al contempo, l'utente mantiene il controllo sulle risorse a cui l'applicazione può attingere. In particolare l'applicativo client è stato caricato su Vercel mentre il server su OnRender.

Panoramica di funzionamento

Per usare i servizi è necessario creare un'applicazione oppure collegare il proprio repository Github. La piattaforma avvia automaticamente un ambiente per poi creare e configurare le risorse necessarie per eseguire il codice. Una volta avviato l'environment, esso può essere gestito e aggiornato con il caricamento di nuove versioni dell'applicazione sul repository Github. Lo schema seguente illustra il flusso di lavoro:

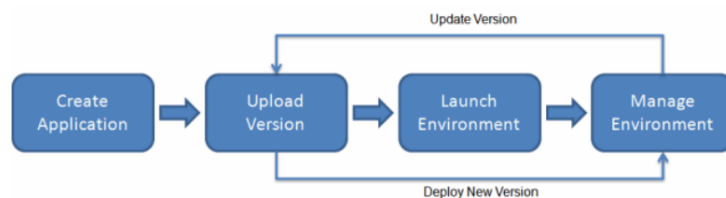


Figure 6.11: Funzionamento flusso di lavoro

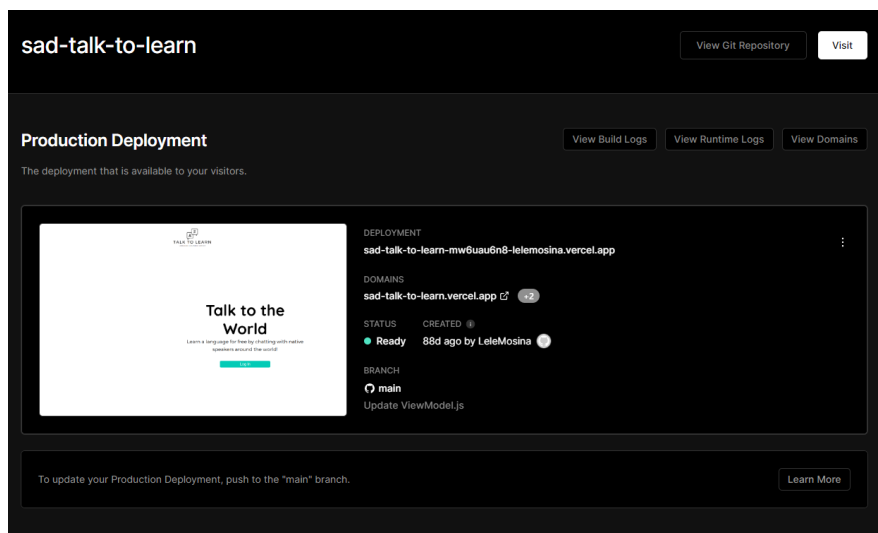


Figure 6.12: Vercel Dashboard

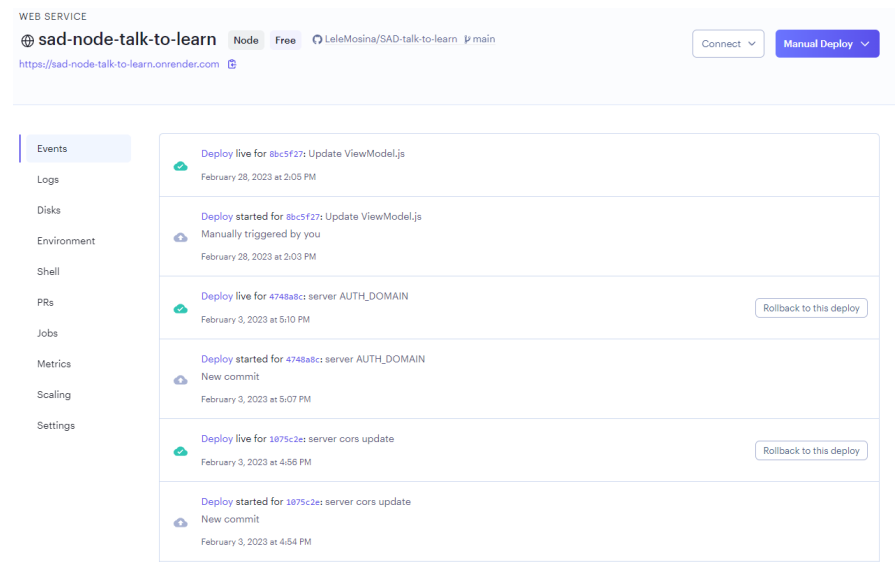


Figure 6.13: OnRender Dashboard

Per illustrare come l'applicazione è stata distribuita sui servizi precedentemente descritti è stato realizzato il seguente Deployment Diagram.

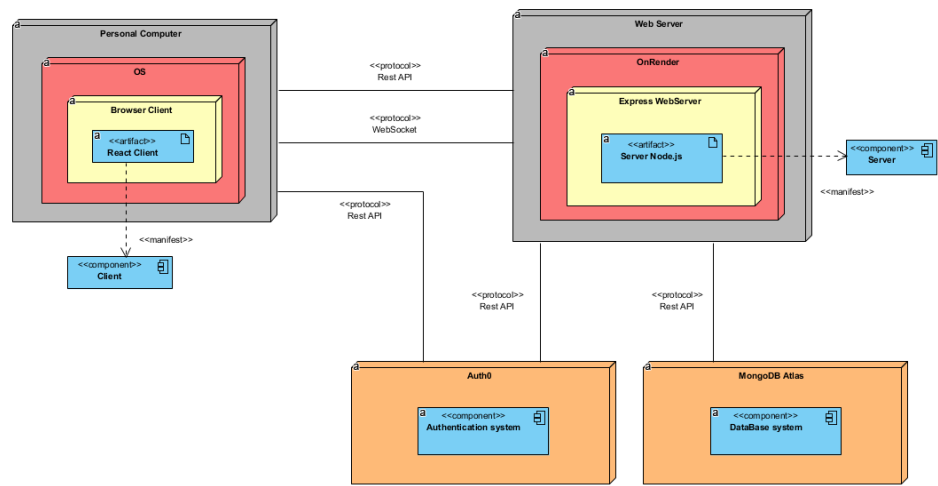


Figure 6.14: Deployment Diagram

Chapter 7

Test

7.1 Unit Test

Serve per testare singole unità software, ovvero, piccoli componenti che hanno un funzionamento autonomo. Nelle applicazioni web permette anche di verificare che una determinata porzione di codice continui a funzionare anche quando le librerie o i tool che utilizza vengono aggiornati. Nel caso specifico sono stati effettuati test automatici in quanto le dipendenze dei singoli moduli sono difficilmente simulabili.

7.2 End-to-end Testing

Il testing end-to-end è una metodologia utilizzata nel ciclo di vita dello sviluppo del software per testare la funzionalità e le prestazioni di un'applicazione in circostanze quasi reali. L'obiettivo è quello di simulare uno scenario reale per l'utente, dall'inizio alla fine. Il completamento di questo test non serve

solo a convalidare il sistema in esame, ma anche a garantire che i suoi sottosistemi funzionino e si comportino come previsto. Comprende il test di interfacce e delle dipendenze esterne, come l'ambiente in cui viene eseguito e il backend. Possono essere verificate non solo proprietà funzionali ma anche proprietà non funzionali e prestazioni. L'intera applicazione è stata testata utilizzando Cypress: un software utile a valutare l'efficienza di qualsiasi cosa venga eseguita in un web browser. Per tutte le pagine previste dall'applicazione, si controlla che tutto ciò che compone l'interfaccia sia visibile. Inoltre si procede a testare separatamente:

- **Login Page:** Controlla la correttezza dell'inserimento di email e password;
- **SingUp Page:** Controlla la corretta registrazione dell'utente con relativo reindirizzamento;
- **Home Page:** Si valuta se al click dei bottoni viene settato l'URL corretto;
- **Chat Page:** Si simula l'invio di un messaggio via chat.

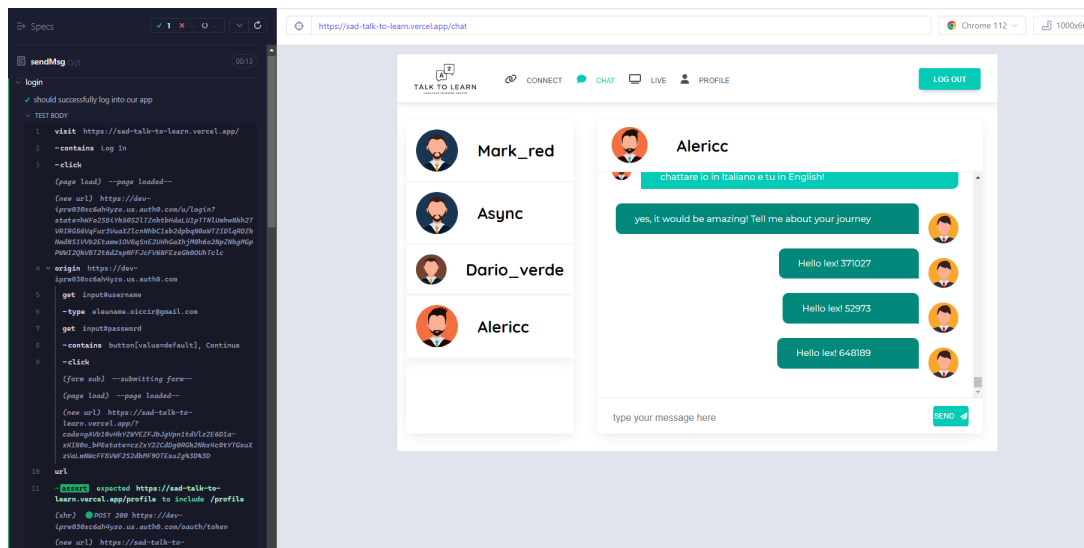


Figure 7.1: Test Chat page

7.3 Testing delle API

Il test delle API determina se le API (Application Programming Interface) soddisfano le specifiche di funzionalità, coerenza, efficienza, usabilità, prestazioni e sicurezza. Inoltre, aiuta a scoprire bug, anomalie o discrepanze dal comportamento previsto di un'API. Durante lo sviluppo di un'API, cicli di feedback rapidi aiutano a garantire che funzioni nel modo desiderato e restituisca i dati previsti. Il testing è stato automatizzato utilizzando Insomnia. Insomnia è un potente client API REST multiplatforma che è molto facile da usare e ha un'interfaccia utente ben congegnata e intuitiva. Contiene tutte le funzionalità di base per testare le API REST come la creazione di richieste HTTP con la possibilità di aggiungere intestazioni e autenticazione o acquisire la risposta e visualizzare lo stato, il corpo, le intestazioni e i cookie. I test vengono scritti selezionando una delle richieste dalla scheda Debug e

quindi facendo asserzioni sui dati restituiti dal server. Si possono eseguire test singoli o unintera suite di test.

Sono state testate le API per la registrazione, in particolare quella per il controllo di un utente già registrato, si è provato il caso in cui si inserisce un token errato, oppure corretto nel caso di un utente già registrato.

The screenshot displays the 'New Collection - Run results' interface. At the top, it shows 'Run on Today, 16:40:52' and a link to 'View all runs'. Below this is a summary table with the following data:

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	862ms	6	121 ms

Below the table, there are tabs for 'All Tests', 'Passed (6)', 'Failed (0)', and 'Skipped (0)'. The 'All Tests' tab is selected. Underneath, it shows 'Iteration 1' and three test cases, each with a green 'GET' icon and a green vertical bar indicating success:

- checkIfUserExist true**
https://sad-node-talk-to-learn.onrender.com/checkIfUserExist
 - PASS status 200
 - PASS test user exists
- checkIfUserExist false**
https://sad-node-talk-to-learn.onrender.com/checkIfUserExist
 - PASS status 200
 - PASS test user doesn't exists
- checkIfUserExist wrong token**
https://sad-node-talk-to-learn.onrender.com/checkIfUserExist
 - PASS status 401
 - PASS test invalid token

Figure 7.2: Test controllo utente registrato