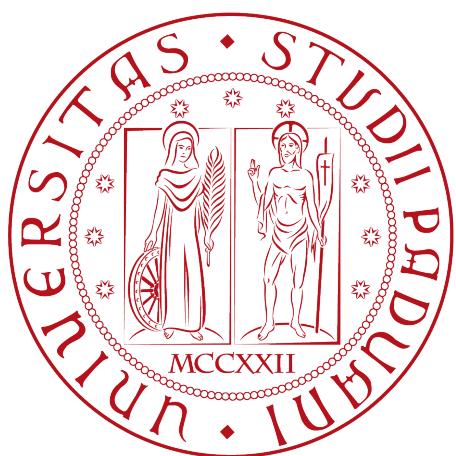


Università degli Studi di Padova
DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"
CORSO DI LAUREA IN INFORMATICA



**Un applicativo web per la gestione di
attività sportive: aggiunta di nuove
funzionalità mediante Spring e Angular**

Tesi di laurea triennale

Relatore

Prof. Paolo Baldan

Laureando

Lorenzo Matterazzo

ANNO ACCADEMICO 2020-2021

*Un applicativo web per la gestione di attività sportive: aggiunta di nuove funzionalità
mediante Spring e Angular*
Tesi di laurea triennale
Lorenzo Matterazzo, © Dicembre 2021.

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di 320 ore, dal laureando Lorenzo Matterazzo presso l'azienda Sync Lab S.r.l, situata a Padova. Lo *stage* ha avuto come argomento principale l'implementazione di nuove funzionalità nel contesto dell'applicazione **SportWill**, una *web app* che dà modo all'utente di divulgare la sua intenzione (*will*^[g]) di effettuare un'attività sportiva. La piattaforma faceva vedere tutto a tutti, rendendo troppo caotica la fruizione, quindi l'esigenza era di dare la possibilità all'utente di creare uno o più gruppi a cui utenti "amici" possano unirsi, vedendo quindi solo le *will* di gruppo. Le attività svolte nel corso dello *stage* sono due:

- la prima è un insieme di attività che sono legate al *back end* della *web app*, come lo sviluppo di tre **microservizi**^[g] mediante il **framework**^[g] Spring Java. In particolare:
 1. il primo **microservizio** consente l'effettuazione delle funzionalità **Create Read Update Delete (CRUD)** per la gestione dei gruppi e la visualizzazione delle *will* degli utenti appartenenti allo stesso gruppo;
 2. il secondo è l'implementazione di un **API Gateway**, che permette di esporre le **Application Program Interface (API)** dei vari servizi presenti in un unico punto di accesso;
 3. il terzo è l'implementazione di un **Eureka Server**^[g] che contiene le informazioni di tutti i servizi che si registrano nel suo server.

Oltre all'implementazione di nuovi **microservizi**, quelli esistenti sono stati modificati affinché le *will* possano essere visualizzate o da tutti gli utenti oppure solo dagli utenti appartenenti agli stessi gruppi. Ultime le attività lato *back end*, è stata effettuata la **containerizzazione**^[g] di tutti i **microservizi** su Docker.

- La seconda attività è stata la modifica del *front end*, mediante il **framework** Angular, per adeguarla alle nuove funzionalità del *back end*.

L'esito dello *stage* è stato molto positivo: le attività obbligatorie e facoltative sono state portate a termine con successo abbastanza facilmente e con un po' di anticipo, permettendomi così di implementare anche la parte *front end* dell'applicazione.

“Limits, like fears, are often just an illusion”

— Michael Jordan

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Paolo Baldan, relatore della mia tesi, per l'aiuto e il sostegno fornитоми durante la stesura del lavoro. Nonostante non abbia avuto modo di assistere alle sue lezioni di persona, il suo stile di insegnamento e l'entusiasmo per l'argomento mi hanno fatto una forte impressione e ho sempre portato con me ricordi positivi delle sue lezioni. Vorrei inoltre ringraziarlo per la sua assistenza e il suo coinvolgimento dedicato in ogni fase dello sviluppo della tesi.

Ringrazio, poi, Fabio e tutti i dipendenti di Sync Lab S.r.l, per avermi dato la possibilità di svolgere lo stage in un ambiente sereno, che mi ha permesso di mettermi in gioco e fare un'esperienza che sarà preziosa per il mio futuro.

Desidero ringraziare Mirko ed Elton, per avermi ascoltato, sostenuto e per aver reso i pomeriggi di studio più leggeri negli ultimi tre anni.

Desidero ringraziare con affetto mia madre e mio padre, che mi hanno permesso di fare questo percorso, incoraggiandomi a dare il meglio di me.

Desidero, infine, ringraziare i miei amici, per tutti i momenti di spensieratezza che mi avete regalato e per l'affetto sincero che provate per me. Grazie per aver reso questo traguardo davvero speciale.

Padova, Dicembre 2021

Lorenzo Matterazzo

Indice

1	Introduzione	1
1.1	Convenzioni tipografiche	1
1.2	L'azienda	1
1.3	Lo stage proposto	2
1.4	Strumenti utilizzati	2
1.4.1	Visual Studio Code	2
1.4.2	Figma	3
1.4.3	Git	3
1.4.4	Postman	3
1.4.5	DbVisualizer	3
1.5	Prodotto ottenuto	3
1.6	Organizzazione del testo	3
2	Descrizione dello stage	5
2.1	Analisi preventiva dei rischi	5
2.2	Requisiti e obiettivi	6
2.3	Pianificazione del lavoro	7
3	Analisi dei requisiti	9
3.1	Casi d'uso	9
3.2	Tracciamento dei requisiti	23
4	Progettazione e codifica	27
4.1	Tecnologie	27
4.2	Progettazione	28
4.2.1	Back end	28
4.2.2	Front end	30
4.3	Design Pattern utilizzati	33
4.3.1	Microservizi	33
4.3.2	API Gateway	33
4.3.3	Client-Side Service Discovery e Service Registry	34
4.3.4	Dependency Injection	34
4.3.5	Singleton	36
4.3.6	Feature Service	36
4.3.7	Lazy Loading	36
4.3.8	Observer	37
4.4	Codifica	38
4.4.1	Back end	38

4.4.2	Eureka Server	44
4.4.3	Front end	45
4.4.4	Servizi	57
5	Verifica e validazione	59
5.1	Accessibilità	59
5.1.1	Elementi di accessibilità	59
5.2	Verifica del GruppiService	59
5.2.1	Test effettuati	60
6	Conclusioni	63
6.1	Raggiungimento degli obiettivi	63
6.2	Conoscenze acquisite	63
6.3	Valutazione personale	64
	Glossario	65
	Acronimi	69
	Bibliografia	71

Elenco delle figure

1.1	Logo dell'azienda	1
3.1	Diagramma generale dei casi d'uso	10
3.2	UC1: Vis. lista dei gruppi	11
3.3	UC1.1: Vis. singolo gruppo in lista	11
3.4	UC5: Vis. dettagli di un gruppo	13
3.5	UC5.3: Vis. lista dei partecipanti di un gruppo	14
3.6	UC5.3.1: Vis. singolo partecipante in lista	15
3.7	UC6: Modifica di un gruppo	17
3.8	UC7: Crea un nuovo gruppo	19
3.9	UC11: Vis. lista <i>will</i> degli utenti appartenenti agli stessi gruppi	22
3.10	UC11.1: Vis. singola <i>will</i>	22
4.1	Architettura a strati di Spring Boot	29
4.2	Spring Boot <i>workflow</i>	29
4.3	<i>Mockup</i> pagina per la visualizzazione dei gruppi creati dall'utente	31
4.4	<i>Mockup</i> pagina per la visualizzazione dei dettagli di un gruppo	32
4.5	<i>Mockup</i> pagina per la modifica dei dettagli di un gruppo	32
4.6	Diagramma concettuale che descrive l' <i>API Gateway</i>	33
4.7	Diagramma concettuale che descrive la registrazione dei microservizi all' <i>Eureka Server</i>	34
4.8	Tabella <i>Crew</i>	38
4.9	Pagina di homepage	46
4.10	Pagina dei gruppi creati o a cui partecipa un utente	46
4.11	Pagina dei gruppi	47
4.12	Pagina di creazione di un gruppo	48
4.13	Pagina di modifica di un gruppo	49
4.14	Pagina di visualizzazione dei dettagli di un gruppo	50
4.15	Componente <i>Homepage</i> di un utente autenticato	51
4.16	Componente <i>Homepage</i> di un utente non autenticato	51
4.17	Componente <i>WillCard</i>	52
4.18	Componente <i>CrewCard</i>	52
4.19	Componente <i>Crewpage</i> che visualizza i gruppi in cui partecipare	53
4.20	Componente <i>Crewpage</i> che visualizza la pagina di gestione dei gruppi dell'utente	53
4.21	Componente che permette di visualizzare i dettagli di un gruppo	54
4.22	Componente <i>CrewEditable</i>	55
4.23	Componente <i>DialogComponent</i>	56

4.24 Componente MapComponent	56
5.1 Code coverage della classe GruppiController	60

Elenco dei codici

4.1	Esempio di <i>Dependency Injection</i> con Spring	35
4.2	Esempio di creazione di un servizio	35
4.3	Esempio di <i>constructor injection</i>	35
4.4	Implementazione del <i>pattern Lazy loading</i> nel AppRoutingModule	36
4.5	Esempio implementazione <i>pattern Observer</i>	37
4.6	Rappresentazione JSON della classe Crew	38
4.7	Implementazione della classe AuthFilter	42
4.8	Configurazione dello <i>Spring Cloud Gateway</i> nel file application.yml del microservizio API Gateway	43
4.9	File application.yml del microservizio Eureka Server	45
4.10	Configurazione per la registrazione di un microservizio all'Eureka Server nel file application.properties	45

Elenco delle tabelle

3.2 Tabella del tracciamento dei requisiti funzionali	24
3.2 Tabella del tracciamento dei requisiti funzionali	25
3.2 Tabella del tracciamento dei requisiti funzionali	26
3.4 Tabella del tracciamento dei requisiti qualitativi	26
3.6 Tabella del tracciamento dei requisiti di vincolo	26
5.2 Test di unità	60
5.2 Test di unità	61
5.2 Test di unità	62
6.2 Tabella raggiungimento degli obiettivi	63

Capitolo 1

Introduzione

1.1 Convenzioni tipografiche

Durante la stesura del documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: [parola^{\[g\]}](#);
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

1.2 L'azienda

Sync Lab S.r.l nasce a Napoli nel 2002 come *software house* ed è rapidamente cresciuta nel mercato dell'[Information and Communications Technology \(ICT\)](#), tramutatasi in *System Integrator* e conquistando significative fette di mercato nei settori *mobile*, videosorveglianza e sicurezza delle infrastrutture informatiche aziendali.

Attualmente, Sync Lab S.r.l ha più di 150 clienti diretti e finali, con un organico aziendale di 200 dipendenti distribuiti tra le 5 sedi dislocate in tutta Italia.

Sync Lab S.r.l si pone come obiettivo principale quello di supportare il cliente nella realizzazione, messa in opera e governance di soluzione IT, sia dal punto di vista tecnologico, sia nel governo del cambiamento organizzativo.



Figura 1.1: Logo dell'azienda

1.3 Lo stage proposto

“Lo sport dà il meglio di sé quando ci unisce.”

Frank Deford

SportWill permette agli utenti di condividere le proprie *will* e partecipare a questa attività sportive.

Dal momento che allo stato dell'arte attuale non esiste ancora la suddivisione delle *will* per gruppi, lo *stage* proposto da Sync Lab S.r.l consiste nell'integrare alla piattaforma già esistente la visualizzazione delle *will* agli utenti che appartengono agli stessi gruppi. Gli obiettivi da raggiungere nel corso dello *stage* sono principalmente due:

- sviluppare un **microservizio** utilizzando il **framework** Spring Java per la creazione dei gruppi;
- modificare i **microservizi** esistenti affinché permettano la visualizzazione solo delle *will* di utenti appartenenti allo stesso gruppo.

Non è richiesta la modifica del *front end* in modo da adeguarlo alle nuove funzionalità del *back end*, a meno che non rimanga tempo da investire su questa attività.

1.4 Strumenti utilizzati

1.4.1 Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. Include il supporto per *debugging*, un controllo per Git integrato, Syntax highlighting, IntelliSense, Snippet e *refactoring* del codice.

Punto di forza di Visual Studio Code sono le estensioni, grazie alle quali è possibile ampliare notevolmente le funzionalità del programma.

Le estensioni utilizzate nel corso dello *stage* sono:

- **GitLens**: amplia le funzionalità di Git integrate in Visual Studio Code;
- **Spring Boot Extension Pack**: raccolta di estensioni per lo sviluppo con Spring Boot;
- **W3C Web Validator**: controlla la validità del *markup* dei documenti HTML e CSS;
- **Web Accessibility**: verifica l'accessibilità dei documenti HTML, evidenziando gli elementi che si potrebbe prendere in considerazione di cambiare e dando suggerimenti su come potrebbero essere modificati;
- **Docker Extension Pack**: raccolta di estensioni per lo gestione dei *container* Docker, Docker images, Dockerfile e file Docker-compose;
- **Angular Extension Pack**: raccolta di estensioni per lo sviluppo con Angular;
- **Extension Pack for Java**: raccolta di estensioni popolari che possono aiutare a scrivere, testare e fare il *debugging* di applicazioni Java.

1.4.2 Figma

Figma è un *tool* per la progettazione di interfacce, che si rivolge principalmente ai *web designer* che hanno bisogno di un *software* studiato appositamente per realizzare il *design* di siti *web* e applicazioni.

Nel contesto dello *stage* è stato utilizzato per la realizzazione delle pagine *web* per la visualizzazione, creazione e modifica di un gruppo.

1.4.3 Git

Software di versionamento utile a tracciare modifiche e cambiamenti di insiemi di file.

1.4.4 Postman

Strumento che permette di eseguire richieste HTTP ad un *server* di *back end*. Quando si lavora con un altro sviluppatore *back end* è possibile condividere le [API](#), ma la sua vera forza è quella di farci sapere tutto di una richiesta HTTP.

1.4.5 DbVisualizer

DbVisualizer è uno strumento *multi-database* intuitivo e ricco di funzionalità per sviluppatori, analisti e amministratori di *database*, che fornisce un'unica, potente interfaccia su un'ampia gamma di sistemi operativi.

DbVisualizer ha un'interfaccia chiara, facile da usare ed è uno strumento che funziona su tutti i principali sistemi operativi, supportando molte varietà di *database*.

1.5 Prodotto ottenuto

Al termine dello *stage* le integrazioni delle funzionalità con la *web app* sono state realizzata con successo. Tutte le chiamate alle [API](#) sono state testate e sono state integrate anche nel *front end*. L'integrazione delle nuove funzionalità permettono alla *web app* di:

- visualizzare le *will* appartenenti agli utenti che partecipano agli stessi gruppi;
- visualizzare le *will* con visibilità globale (quindi che non sono visibili solo agli utenti che partecipano agli stessi gruppi);
- visualizzare i gruppi a cui partecipa un utente;
- visualizzare e modificare i gruppi creati da un utente;
- visualizzare e cercare i gruppi;
- creare nuovi gruppi.

1.6 Organizzazione del testo

[Il secondo capitolo](#) descrive l'analisi preventiva dei rischi, gli obiettivi dello *stage* e la pianificazione delle ore di lavoro.

Il terzo capitolo approfondisce l'analisi dei requisiti del prodotto.

Il quarto capitolo approfondisce la fase di progettazione e codifica.

Il quinto capitolo approfondisce l'accessibilità e la fase di verifica e validazione.

Il sesto capitolo contiene un'analisi del lavoro svolto e le conclusioni tratte.

Capitolo 2

Descrizione dello stage

In questo capitolo è presente un'analisi preventiva dei rischi a cui si sarebbe potuto incomberre durante lo stage, la lista degli obiettivi da raggiungere e la pianificazione delle ore di lavoro.

2.1 Analisi preventiva dei rischi

Qui vengono analizzati i rischi emersi durante la fase di analisi iniziale a cui si potrebbe incombere. Per ogni rischio individuato, si è proceduto ad elaborare un piano di contingenza per far fronte a tali rischi.

1. Inesperienza tecnologica

Descrizione: le tecnologie da utilizzare sono nuove o esplorate parzialmente, il che può portare alla nascita di problemi operativi.

Piano di contingenza: le attività che richiedono maggior tempo oppure con un livello di capacità tecniche elevate saranno trattate per prime, in modo da migliorarle incrementalmente durante il periodo di stage.

2. Problematiche *software* di supporto

Descrizione: il *computer* in cui si sviluppa il prodotto potrebbe avere malfunzionamenti, e nel caso accadesse potrebbe causare gravi ritardi.

Piano di contingenza: ad ogni *commit* effettuato è necessario aggiornare la *repository* remota. Inoltre deve essere possibile replicare velocemente l'ambiente di lavoro in un altro *computer* in modo da tornare operativi nel minor tempo possibile. Un modo per ripristinare velocemente le impostazioni di lavoro è la creazione di una cartella da versionare chiamata `.vscode` in cui inserire:

- il file `settings.json`, in cui salvare le impostazioni dell'editor;
- il file `extensions.json`, in cui aggiungere gli identificativi delle estensioni utilizzate in Visual Studio Code;
- Il file `launch.json`, in cui configurare il debugger in Visual Studio Code.

3. Tempistiche

Descrizione: il tempo di apprendimento di nuove tecnologie potrebbe indurre in ritardi sulle scadenze previste. I ritardi verranno individuati nel caso in cui il lavoro

da effettuare si scostasse dalla pianificazione presente su [Trello^{\[g\]}](#).

Piano di contingenza: appena si rilevano difficoltà o scostamenti rispetto al piano di lavoro, si dovrà avvisare tempestivamente il *tutor* aziendale, con cui ci si potrà confrontare, e solo in caso estremo rimandare le scadenze prefissate.

4. Impegni personali

Descrizione: è possibile che vi siano degli impegni personali da adempiere e che di conseguenza abbia meno tempo da poter dedicare allo sviluppo del progetto.

Piano di contingenza: gli incarichi con le relative scadenze sono stati predisposti nel rispetto degli eventuali impegni personali. In caso di imprevisti, bisognerà immediatamente contattare il *tutor* aziendale.

5. Interpretazione errata o non sufficiente dei requisiti

Descrizione: Dopo una prima analisi dei requisiti, è possibile che si noti la necessità di modificare o aggiungere nuovi requisiti in un secondo momento.

Piano di contingenza: Due volte a settimana verrà fatto il punto della situazione con il *tutor* aziendale. Nel caso si notassero assenze nei requisiti, si procederà alla sua conseguente analisi e si deciderà come procedere in maniera da limitare un eventuale rallentamento sulla di sviluppo.

2.2 Requisiti e obiettivi

Notazione

Si farà riferimento ai requisiti secondo le seguenti notazioni:

- *O* per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- *D* per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- *F* per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo del requisito.

Obiettivi fissati

Si prevede lo svolgimento dei seguenti obiettivi:

- Obbligatori
 - O01: Acquisizione competenze sulle tematiche sopra descritte;
 - O02: Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma;
 - O03: Portare a termine l'implementazione dei [microservizi](#) richiesti con una percentuale di superamento pari al 80.
- Desiderabili

- D01: Portare a termine l'implementazione dei **microservizi** richiesti con una percentuale di superamento pari al 100.
- Facoltativi
 - F01: Utilizzo della *containerizzazione* per portare tutti i **microservizi** su Docker.

2.3 Pianificazione del lavoro

Pianificazione settimanale

- **Prima Settimana (40 ore)**
 - Incontro con persone coinvolte nel progetto per discutere i requisiti e le richieste relativamente al sistema da sviluppare;
 - Verifica credenziali e strumenti di lavoro assegnati;
 - Ripasso Java Standard Edition e tool di sviluppo (IDE ecc.);
 - Studio teorico dell'architettura a **microservizi**: passaggio da monolite a **microservizi** con pro e contro;
 - Ripasso principi della buona programmazione (SOLID, CleanCode);
 - Ripasso Java Standard Edition.
- **Seconda Settimana - (40 ore)**
 - Studio teorico dell'architettura a **microservizi**: passaggio da monolite ad architetture a **microservizi**;
 - Studio teorico dell'architettura a **microservizi**: Api Gateway, Service Discovery e Service Registry, Circuit Breaker e Saga Pattern;
 - Studio Spring Core/Spring Boot.
- **Terza Settimana - (40 ore)**
 - Studio servizi REST e *framework* Spring Data REST;
 - Studio ORM, in particolare il *framework* Spring Data JPA.
- **Quarta Settimana - (40 ore)**
 - Studio ORM, in particolare il *framework* Spring Data JPA.;
- **Quinta Settimana - (40 ore)**
 - Studio della piattaforma **SportWill** esistente;
 - Analisi nuova funzionalità da implementare.
- **Sesta Settimana - (40 ore)**
 - Implementazione del nuovo servizio.
- **Settima Settimana - (40 ore)**
 - Implementazione del nuovo servizio.
- **Ottava Settimana - Conclusione (40 ore)**
 - Considerazioni e collaudi finali.

Ripartizione ore

La pianificazione, in termini di quantità di ore di lavoro, sarà così distribuita:

Durata in ore	Descrizione dell'attività
160	Formazione sulle tecnologie
18	Studio Java Standard Edition e tool di sviluppo
18	Studio architettura a <i>microservizi</i>
4	Ripasso dei principi della buona programmazione (<i>SOLID</i> , <i>Clean-Code</i>)
10	Studio teorico dell'architettura a <i>microservizi</i> : passaggio da monolite ad architetture a <i>microservizi</i>
15	Studio teorico dell'architettura a <i>microservizi</i> : <i>Api Gateway</i> , <i>Service Discovery</i> e <i>Service Registry</i> , <i>Circuit Breaker</i> e <i>Saga Pattern</i>
15	Studio Spring Core/Spring Boot
20	Studio servizi REST e framework Spring Data REST
60	Studio ORM, in particolare il framework Spring Data JPA
40	Definizione architettura di riferimento e relativa documentazione
14	Analisi del problema e del dominio applicativo
22	Progettazione della piattaforma e relativi test
4	Stesura documentazione relativa ad analisi e progettazione
80	Implementazione del nuovo servizio
40	Collaudo Finale
30	Collaudo
6	Stesura documentazione finale
2	Incontro di presentazione della piattaforma con gli stakeholders
2	Live demo di tutto il lavoro di stage
Totale ore	320

Capitolo 3

Analisi dei requisiti

Il presente capitolo descrive in maniera dettagliata requisiti e casi d'uso individuati durante la fase di analisi del progetto di stage.

3.1 Casi d'uso

Attori principali

Nella fase di analisi del progetto di *stage* è emersa la presenza di un solo attore, ovvero l'**utente autenticato**, attore che rappresenta un utente che ha effettuato l'autenticazione all'interno dell'applicazione *web*. Ha quindi la possibilità di vedere tutte le informazioni sui gruppi che sarebbero altrimenti inaccessibili.

Elenco dei casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi dei casi d'uso. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo [Unified Modeling Language \(UML\)](#) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.

I casi d'uso riportati in seguito descrivono solo le funzionalità che dovranno essere implementate, senza quindi descrivere quelle già presenti nella *web app*.



Figura 3.1: Diagramma generale dei casi d'uso

UC1: Visualizzazione lista dei gruppi

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app* ed è autenticato.

Descrizione: L'utente vuole visualizzare la lista dei gruppi.

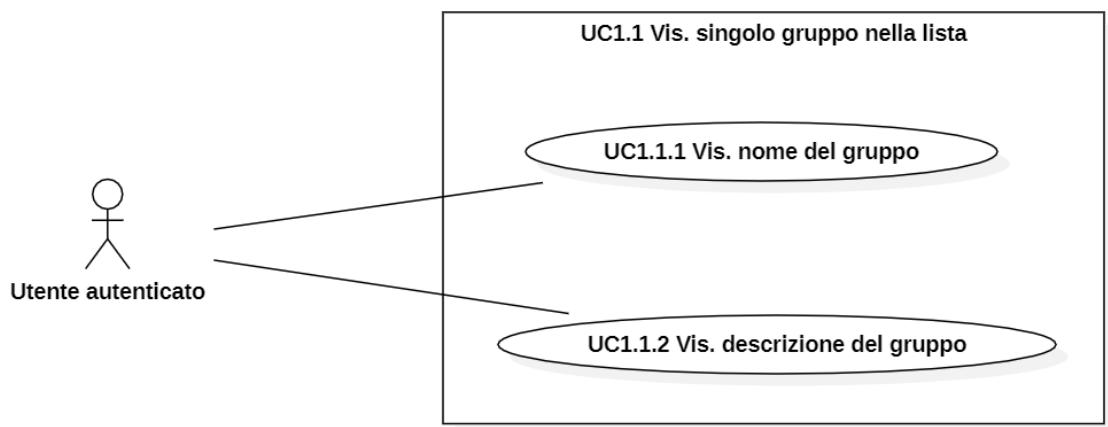
Postcondizioni: L'utente ha visualizzato la lista dei gruppi.

Scenario principale:

1. L'utente visualizza la lista dei gruppi.

**Figura 3.2:** UC1: Vis. lista dei gruppi**UC1.1: Visualizzazione singolo gruppo in lista****Attori Principali:** Utente autenticato.**Precondizioni:** L'utente ha aperto la *web app*, è autenticato e sta visualizzando la lista dei gruppi.**Descrizione:** L'utente vuole visualizzare un singolo gruppo nella lista dei gruppi.**Postcondizioni:** L'utente ha visualizzato un singolo gruppo nella lista dei gruppi.**Scenario principale:**

1. L'utente visualizza la pagina contenente la lista dei gruppi;
2. l'utente visualizza un singolo gruppo nella lista dei gruppi.

**Figura 3.3:** UC1.1: Vis. singolo gruppo in lista

UC1.1.1: Visualizzazione nome del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando un gruppo dalla lista dei gruppi.

Descrizione: L'utente vuole visualizzare il nome di un gruppo dalla lista dei gruppi.

Postcondizioni: L'utente ha visualizzato il nome di un gruppo dalla lista dei gruppi.

Scenario principale:

1. L'utente visualizza la pagina contenente la lista dei gruppi;
2. l'utente visualizza il nome di un gruppo dalla lista dei gruppi.

UC1.1.2: Visualizzazione descrizione del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando un gruppo dalla lista dei gruppi.

Descrizione: L'utente vuole visualizzare la descrizione di gruppo della lista dei gruppi.

Postcondizioni: L'utente ha visualizzato la descrizione di un gruppo dalla lista dei gruppi.

Scenario principale:

1. L'utente visualizza la pagina contenente la lista dei gruppi;
2. l'utente visualizza la descrizione di un gruppo dalla lista dei gruppi.

UC2: Visualizzazione lista dei gruppi creati dall'utente

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app* ed è autenticato.

Descrizione: L'utente vuole visualizzare la lista dei gruppi creati da lui.

Postcondizioni: L'utente ha visualizzato la lista dei gruppi creati da lui.

Scenario principale:

1. L'utente visualizza la lista dei gruppi creati da lui.

UC3: Visualizzazione lista dei gruppi a cui partecipa un utente

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app* ed è autenticato.

Descrizione: L'utente vuole visualizzare la lista dei gruppi a cui partecipa.

Postcondizioni: L'utente ha visualizzato la lista dei gruppi a cui partecipa.

Scenario principale:

1. L'utente visualizza la lista dei gruppi a cui partecipa.

UC4: Visualizzazione lista dei gruppi creati da altri utenti

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app* ed è autenticato.

Descrizione: L'utente vuole visualizzare la lista dei gruppi creati da altri utenti.

Postcondizioni: L'utente ha visualizzato la lista dei gruppi creati da altri utenti.

Scenario principale:

1. L'utente visualizza la lista dei gruppi creati da altri utenti.

UC5: Visualizzazione dettagli di un gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando la lista dei gruppi.

Descrizione: L'utente vuole visualizzare i dettagli di un gruppo.

Postcondizioni: L'utente ha visualizzato i dettagli di un gruppo.

Scenario principale:

1. L'utente clicca su un gruppo sul quale vuole ottenere i dettagli;
2. l'utente visualizza i dettagli di un gruppo.

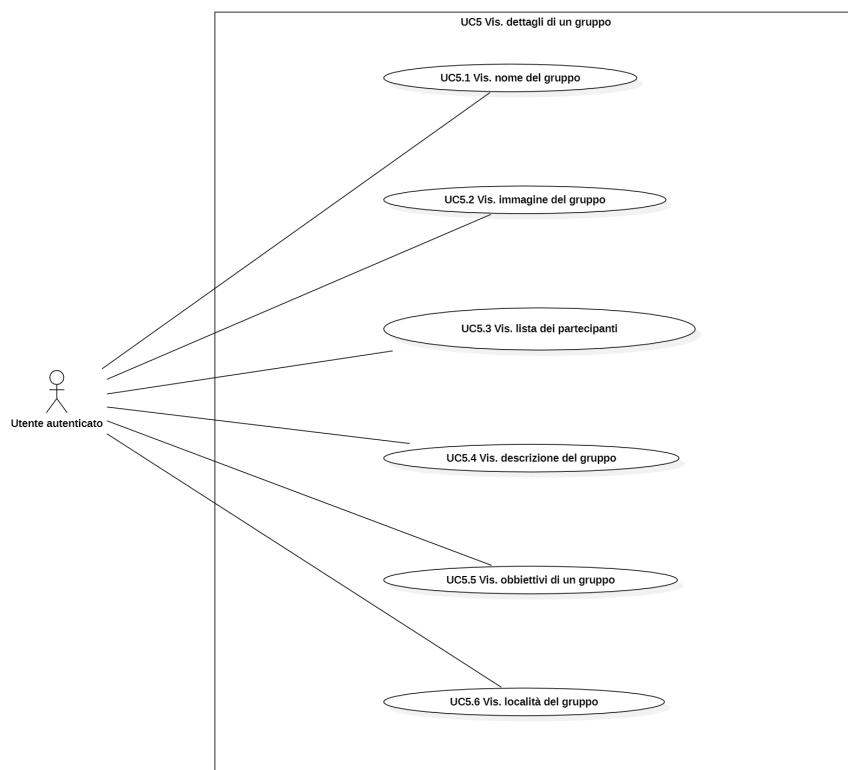


Figura 3.4: UC5: Vis. dettagli di un gruppo

UC5.1: Visualizzazione nome del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando i dettagli di un gruppo.

Descrizione: L'utente vuole visualizzare il nome di un gruppo.

Postcondizioni: L'utente ha visualizzato il nome di un gruppo.

Scenario principale:

1. L'utente visualizza il nome di un gruppo.

UC5.2: Visualizzazione immagine del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando i dettagli di un gruppo.

Descrizione: L'utente vuole visualizzare l'immagine di un gruppo.

Postcondizioni: L'utente ha visualizzato l'immagine di un gruppo.

Scenario principale:

1. L'utente visualizza l'immagine di un gruppo.

UC5.3: Visualizzazione lista dei partecipanti di un gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando i dettagli di un gruppo.

Descrizione: L'utente vuole visualizzare la lista dei partecipanti ad un gruppo.

Postcondizioni: L'utente ha visualizzato la lista dei partecipanti ad un gruppo.

Scenario principale:

1. L'utente visualizza la lista dei partecipanti ad un gruppo.

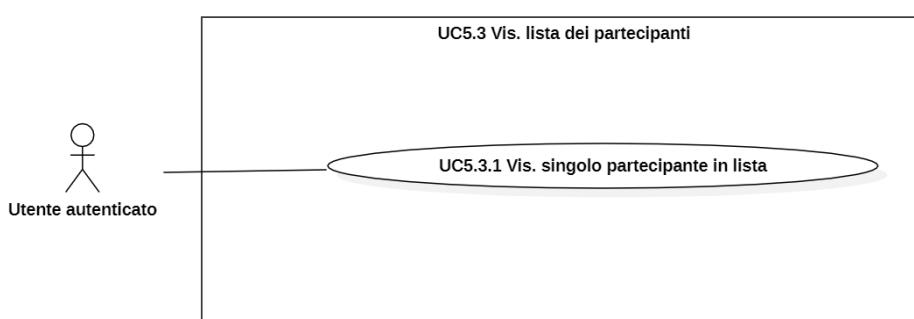


Figura 3.5: UC5.3: Vis. lista dei partecipanti di un gruppo

UC5.3.1: Visualizzazione singolo partecipante in lista

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando la lista dei partecipanti ad un gruppo.

Descrizione: L'utente vuole visualizzare un partecipante dalla lista dei partecipanti ad un gruppo.

Postcondizioni: L'utente ha visualizzato un partecipante dalla lista dei partecipanti ad un gruppo.

Scenario principale:

1. L'utente visualizza un partecipante dalla lista dei partecipanti ad un gruppo.

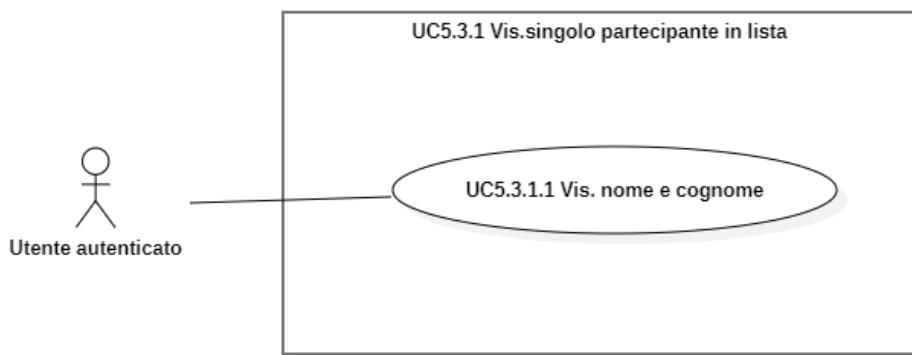


Figura 3.6: UC5.3.1: Vis. singolo partecipante in lista

UC5.3.1.1: Visualizzazione singolo partecipante in lista

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando un partecipante dalla lista dei partecipanti.

Descrizione: L'utente vuole visualizzare nome e cognome di un partecipante dalla lista dei partecipanti.

Postcondizioni: L'utente ha visualizzato nome e cognome di un partecipante dalla lista dei partecipanti.

Scenario principale:

1. L'utente visualizza nome e cognome di un partecipante dalla lista dei partecipanti.

UC5.4: Visualizzazione descrizione del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando i dettagli di un gruppo.

Descrizione: L'utente vuole visualizzare la descrizione di un gruppo.

Postcondizioni: L'utente ha visualizzato la descrizione di un gruppo.

Scenario principale:

1. L'utente visualizza la descrizione di un gruppo.

UC5.5: Visualizzazione obbiettivi del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando i dettagli di un gruppo.

Descrizione: L'utente vuole visualizzare gli obbiettivi di un gruppo.

Postcondizioni: L'utente ha visualizzato gli obbiettivi di un gruppo.

Scenario principale:

1. L'utente visualizza gli obbiettivi di un gruppo.

UC5.6: Visualizzazione località del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando i dettagli di un gruppo.

Descrizione: L'utente vuole visualizzare la località di un gruppo.

Postcondizioni: L'utente ha visualizzato la località di un gruppo.

Scenario principale:

1. L'utente visualizza la località di un gruppo.

UC6: Modifica di un gruppo

Attori Principali: Utente autenticato.

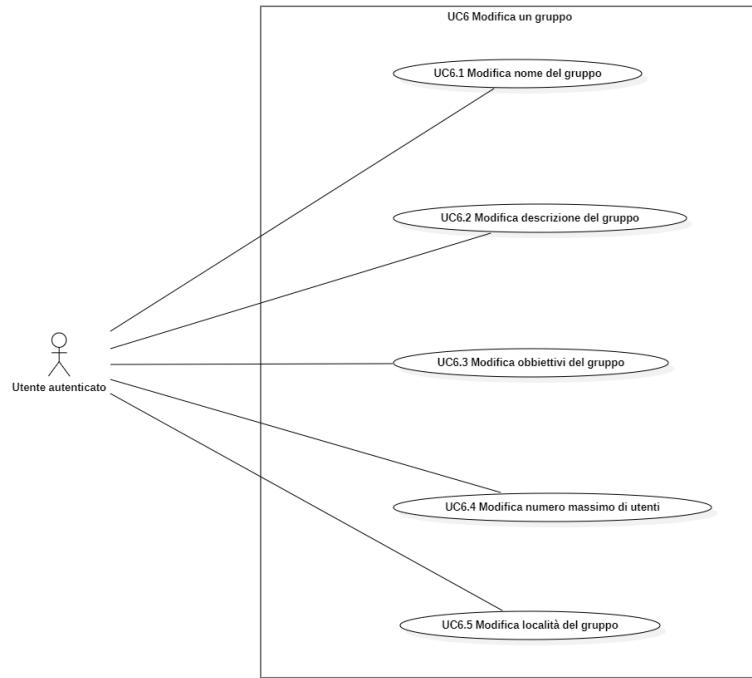
Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando la lista dei gruppi creati da lui.

Descrizione: L'utente vuole modificare i dettagli di un gruppo.

Postcondizioni: L'utente ha modificato i dettagli di un gruppo.

Scenario principale:

1. L'utente modifica i dettagli di un gruppo.

**Figura 3.7:** UC6: Modifica di un gruppo**UC6.1: Modifica nome del gruppo****Attori Principali:** Utente autenticato.**Precondizioni:** L'utente ha aperto la *web app*, è autenticato e sta modificando i dettagli di un gruppo.**Descrizione:** L'utente vuole modificare il nome di un gruppo.**Postcondizioni:** L'utente ha modificato il nome di un gruppo.**Scenario principale:**

1. L'utente modifica il nome di un gruppo.

UC6.2: Modifica descrizione del gruppo**Attori Principali:** Utente autenticato.**Precondizioni:** L'utente ha aperto la *web app*, è autenticato e sta modificando i dettagli di un gruppo.**Descrizione:** L'utente vuole modificare la descrizione di un gruppo.**Postcondizioni:** L'utente ha modificato la descrizione di un gruppo.**Scenario principale:**

1. L'utente modifica la descrizione di un gruppo.

UC6.3: Modifica obiettivi del gruppo**Attori Principali:** Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta modificando i dettagli di un gruppo.

Descrizione: L'utente vuole modificare gli obiettivi di un gruppo.

Postcondizioni: L'utente ha modificato gli obiettivi di un gruppo.

Scenario principale:

1. L'utente modifica gli obiettivi di un gruppo.

UC6.4: Modifica numero massimo di utenti

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta modificando i dettagli di un gruppo.

Descrizione: L'utente vuole modificare il numero massimo di utenti di un gruppo.

Postcondizioni: L'utente ha modificato il numero massimo di utenti di un gruppo.

Scenario principale:

1. L'utente modifica il numero massimo di utenti di un gruppo.

UC6.5: Modifica località del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta modificando i dettagli di un gruppo.

Descrizione: L'utente vuole modificare la località di un gruppo.

Postcondizioni: L'utente ha modificato la località di un gruppo.

Scenario principale:

1. L'utente modifica la località di un gruppo.

UC7: Crea un nuovo gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app* ed è autenticato.

Descrizione: L'utente vuole creare un nuovo gruppo.

Postcondizioni: L'utente ha creato un nuovo gruppo.

Scenario principale:

1. L'utente crea un nuovo gruppo.

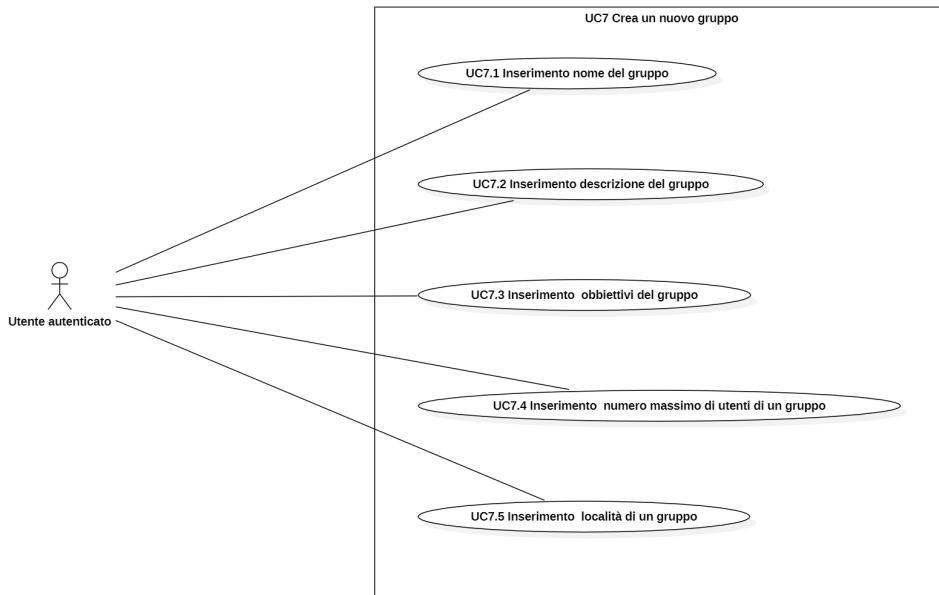


Figura 3.8: UC7: Crea un nuovo gruppo

UC7.1: Inserimento nome del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta creando un nuovo gruppo.

Descrizione: L'utente vuole inserire il nome del gruppo.

Postcondizioni: L'utente ha inserito il nome del gruppo.

Scenario principale:

1. L'utente inserisce il nome del gruppo.

UC7.2: Inserimento descrizione del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta creando un nuovo gruppo.

Descrizione: L'utente vuole inserire la descrizione del gruppo.

Postcondizioni: L'utente ha inserito la descrizione del gruppo.

Scenario principale:

1. L'utente inserisce la descrizione del gruppo.

UC7.3: Inserimento obiettivi del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta creando un nuovo gruppo.

Descrizione: L'utente vuole inserire gli obiettivi del gruppo.

Postcondizioni: L'utente ha inserito gli obiettivi del gruppo.

Scenario principale:

1. L'utente inserisce gli obiettivi del gruppo.

UC7.4: Inserimento numero massimo di utenti nel gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta creando un nuovo gruppo.

Descrizione: L'utente vuole inserire il numero massimo di utenti del gruppo.

Postcondizioni: L'utente ha inserito il numero massimo di utenti del gruppo.

Scenario principale:

1. L'utente inserisce il numero massimo di utenti del gruppo.

UC7.5: Inserimento località del gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta creando un nuovo gruppo.

Descrizione: L'utente vuole inserire la località del gruppo.

Postcondizioni: L'utente ha inserito la località del gruppo.

Scenario principale:

1. L'utente inserisce la località del gruppo.

UC8: Elimina un gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e si trova nella pagina di modifica di un gruppo.

Descrizione: L'utente vuole eliminare un gruppo.

Postcondizioni: L'utente ha eliminato un gruppo.

Scenario principale:

1. L'utente elimina un gruppo.

UC9: Partecipa ad un gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e si trova nella pagina dei dettagli di un gruppo a cui non partecipa.

Descrizione: L'utente vuole partecipare ad un gruppo.

Postcondizioni: L'utente ha partecipato al gruppo.

Scenario principale:

1. L'utente partecipa al gruppo.

UC10: Annulla partecipazione ad un gruppo

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e si trova nella pagina dei dettagli di un gruppo a cui partecipa.

Descrizione: L'utente vuole annullare la partecipazione al gruppo.

Postcondizioni: L'utente ha annullato la partecipazione al gruppo.

Scenario principale:

1. L'utente annulla la partecipazione al gruppo.

UC11: Visualizzazione lista delle will degli utenti appartenenti agli stessi gruppi

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app* ed è autenticato.

Descrizione: L'utente vuole visualizzare la lista delle *will* degli utenti appartenenti agli stessi gruppi.

Postcondizioni: L'utente ha visualizzato la lista delle *will* degli utenti appartenenti agli stessi gruppi.

Scenario principale:

1. L'utente visualizza la lista delle *will* degli utenti appartenenti agli stessi gruppi.

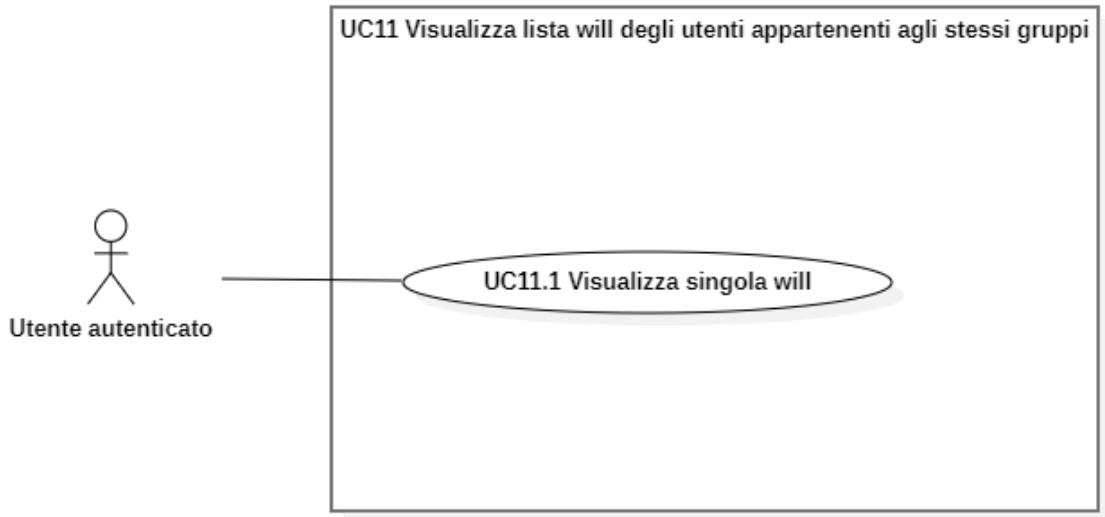


Figura 3.9: UC11: Vis. lista *will* degli utenti appartenenti agli stessi gruppi

UC11.1: Visualizzazione singola will in lista

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando la lista delle *will*.

Descrizione: L'utente vuole visualizzare una singola *will* dalla lista delle *will*.

Postcondizioni: L'utente ha visualizzato una singola *will* dalla lista delle *will*.

Scenario principale:

1. L'utente visualizza una singola *will* dalla lista delle *will*.

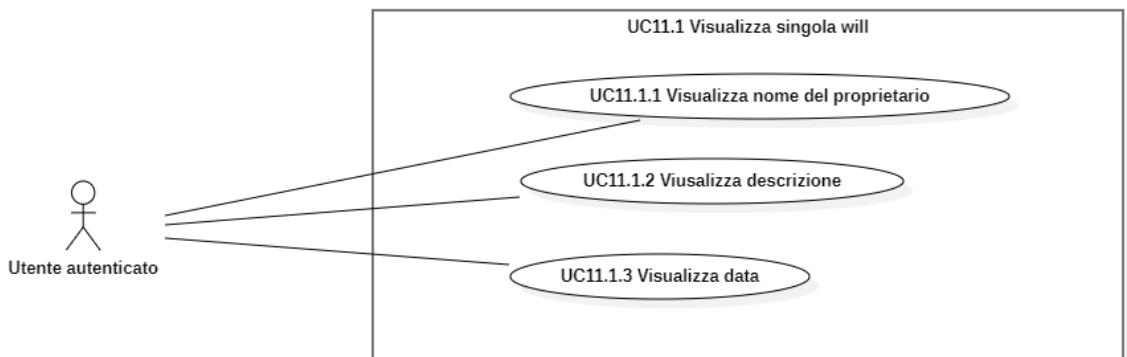


Figura 3.10: UC11.1: Vis. singola *will*

UC11.1.1: Visualizzazione nome del proprietario

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando un *will* dalla lista delle *will*.

Descrizione: L'utente vuole visualizzare il nome del proprietario della *will* dalla lista delle *will*.

Postcondizioni: L'utente ha visualizzato il nome del proprietario della *will* dalla lista delle *will*.

Scenario principale:

1. L'utente visualizza il nome del proprietario della *will* dalla lista delle *will*.

UC11.1.2: Visualizzazione descrizione

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando un *will* dalla lista delle *will*.

Descrizione: L'utente vuole visualizzare la descrizione della *will* dalla lista delle *will*.

Postcondizioni: L'utente ha visualizzato la descrizione della *will* dalla lista delle *will*.

Scenario principale:

1. L'utente visualizza la descrizione della *will* dalla lista delle *will*.

UC11.1.3: Visualizzazione data

Attori Principali: Utente autenticato.

Precondizioni: L'utente ha aperto la *web app*, è autenticato e sta visualizzando un *will* dalla lista delle *will*.

Descrizione: L'utente vuole visualizzare la data in cui verrà effettuata l'uscita.

Postcondizioni: L'utente ha visualizzato la data in cui verrà effettuata l'uscita.

Scenario principale:

1. L'utente visualizza la data in cui verrà effettuata l'uscita.

3.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.

Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguergli.

Il codice dei requisiti è così strutturato R(F/Q/V)(N/D/O) dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

N = obbligatorio (necessario)

D = desiderabile

Z = opzionale

Nelle tabelle 3.2, 3.4 e 3.6 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

Tabella 3.2: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
RFO-1	Il sistema deve permettere di visualizzare la lista dei gruppi	UC1
RFO-1.1	Il sistema deve permettere di visualizzare un singolo gruppo nella lista dei gruppi	UC1.1
RFO-1.1.1	Il sistema deve permettere di visualizzare il nome di un gruppo dalla lista dei gruppi	UC1.1.1
RFO-1.1.2	Il sistema deve permettere di visualizzare la descrizione di un gruppo dalla lista dei gruppi	UC1.1.2
RFO-2	Il sistema deve permettere di visualizzare la lista dei gruppi creati dall'utente	UC2
RFO-3	Il sistema deve permettere di visualizzare la lista dei gruppi a cui partecipa l'utente	UC3
RFO-4	Il sistema deve permettere di visualizzare la lista dei gruppi creati da altri utenti	UC4
RFO-5	Il sistema deve permettere di visualizzare i dettagli di un gruppo	UC5
RFO-5.1	Il sistema deve permettere di visualizzare il nome di un gruppo	UC5.1
RFO-5.2	Il sistema deve permettere di visualizzare l'immagine di un gruppo	UC5.2
RFO-5.3	Il sistema deve permettere di visualizzare la lista dei partecipanti ad un gruppo	UC5.3
RFO-5.3.1	Il sistema deve permettere di visualizzare un partecipante dalla lista dei partecipanti ad un gruppo	UC5.3.1
RFO-5.3.1.1	Il sistema deve permettere di visualizzare nome e cognome di un partecipante dalla lista dei partecipanti ad un gruppo	UC5.3.1.1
RFO-5.4	Il sistema deve permettere di visualizzare la descrizione di un gruppo	UC5.4
RFO-5.5	Il sistema deve permettere di visualizzare gli obiettivi di un gruppo	UC5.5
Requisito	Descrizione	Fonte

Tabella 3.2: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
RFO-5.6	Il sistema deve permettere di visualizzare località di un gruppo	UC5.6
RFO-6	Il sistema deve permettere di modificare i dettagli di un gruppo	UC6
RFO-6.1	Il sistema deve permettere di modificare il nome di un gruppo	UC6.1
RFO-6.2	Il sistema deve permettere di modificare la descrizione di un gruppo	UC6.2
RFO-6.3	Il sistema deve permettere di modificare gli obiettivi di un gruppo	UC6.3
RFO-6.4	Il sistema deve permettere di modificare il numero massimo di utenti di un gruppo	UC6.4
RFO-6.5	Il sistema deve permettere di modificare la località di un gruppo	UC6.5
RFO-7	Il sistema deve permettere di creare un nuovo gruppo	UC7
RFO-7.1	Il sistema deve permettere di inserire il nome di un gruppo	UC7.1
RFO-7.2	Il sistema deve permettere di inserire la descrizione di un gruppo	UC7.2
RFO-7.3	Il sistema deve permettere di inserire gli obiettivi di un gruppo	UC7.3
RFO-7.4	Il sistema deve permettere di inserire il numero massimo di utenti del gruppo	UC7.4
RFO-7.5	Il sistema deve permettere di inserire la località del gruppo	UC7.5
RFO-8	Il sistema deve permettere di eliminare un gruppo	UC8
RFO-9	Il sistema deve permettere di partecipare ad un gruppo	UC9
RFO-10	Il sistema deve permettere di annullare la partecipazione	UC10
RFO-11	Il sistema deve permettere di visualizzare la lista delle <i>will</i> degli utenti appartenenti agli stessi gruppi	UC11
RFO-11.1	Il sistema deve permettere di visualizzare una singola <i>will</i> dalla lista delle <i>will</i>	UC11.1
Requisito	Descrizione	Fonte

Tabella 3.2: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
RFO-11.1.1	Il sistema deve permettere di visualizzare il nome del proprietario della <i>will</i> dalla lista delle <i>will</i>	UC11.1.1
RFO-11.1.2	Il sistema deve permettere di visualizzare la descrizione delle <i>will</i> dalla lista delle <i>will</i>	UC11.1.2
Requisito	Descrizione	Fonte

Tabella 3.4: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Fonte
RQO-1	Deve essere fornito un documento tecnico che descriva quanto sviluppato	Committente
RQO-2	L'applicazione <i>web</i> deve essere accessibile	Interno
RQD-3	L'applicazione <i>web</i> deve essere <i>responsive</i>	Interno
Requisito	Descrizione	Fonte

Tabella 3.6: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Fonte
RVO-1	Le nuove maschere sviluppate devono avere uno stile coerente con quello già presente nell'applicazione <i>web</i>	Interno
Requisito	Descrizione	Fonte

Capitolo 4

Progettazione e codifica

*Il seguente capitolo descrive gli strumenti e la progettazione con cui sono state implementate le integrazioni con la web app **SportWill**.*

4.1 Tecnologie

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

Java

Java è un linguaggio di programmazione e una piattaforma di elaborazione rilasciato per la prima volta da Sun Microsystems nel 1995. Si è evoluto da umili origini per sostenere gran parte del mondo digitale di oggi, fornendo una piattaforma affidabile su cui sono costruiti molti servizi e applicazioni. Anche i nuovi prodotti, innovativi nei servizi digitali progettati per il futuro, continuano a fare affidamento su Java. [18]

Spring

Spring è un *framework* open source per lo sviluppo di applicazioni su piattaforma Java. A questo *framework* sono associati tanti altri progetti, che hanno nomi composti come Spring Boot, Spring Data, Spring Batch, etc. Questi progetti sono stati ideati per fornire funzionalità aggiuntive al *framework*. [15]

TypeScript

TypeScript è un linguaggio di programmazione sviluppato e gestito da Microsoft.

È un *superset*[[g](#)] di JavaScript, permettendo di aggiungere la tipizzazione statica opzionale al linguaggio. TypeScript è progettato per lo sviluppo di applicazioni di grandi dimensioni e per la *transcompilazione*[[g](#)] in JavaScript. Poiché TypeScript è un *superset* di JavaScript, anche i programmi JavaScript esistenti sono validi programmi TypeScript. [16]

Angular

Angular è un *framework* JavaScript per applicazioni *web* dinamiche, utilizzato in particolare per la creazione di [Single-Page Application \(SPA\)](#) e *web app*. Consente di utilizzare HTML come linguaggio *template* e di estenderne la sintassi per esprimere le componenti di un'applicazione in modo chiaro e succinto.

Angular Material

Angular Material è una libreria sviluppata da Google nel 2014 progettata per aiutare a sviluppare pagine *web* in modo strutturato.

I suoi componenti aiutano a creare pagine *web* e applicazioni *web* attraenti, coerenti e funzionali. [1]

Node.js

Node.js è una piattaforma di sviluppo open source per l'esecuzione di codice JavaScript lato *server*. Node è utile per sviluppare applicazioni che richiedono una connessione permanente dal *browser* al *server* ed è spesso utilizzato per applicazioni in tempo reale come chat, feed di notizie e di notifiche.

Node.js è utilizzato da Angular per gestire le dipendenze, permettendo la dichiarazione di due insiemi di dipendenze: per gli sviluppatori e per far funzionare l'applicativo. In questo modo è possibile differenziare quali librerie si possono tralasciare in fase di *deploy* dell'applicazione perché, ad esempio, necessarie solo per effettuare i test. [10]

4.2 Progettazione

4.2.1 Back end

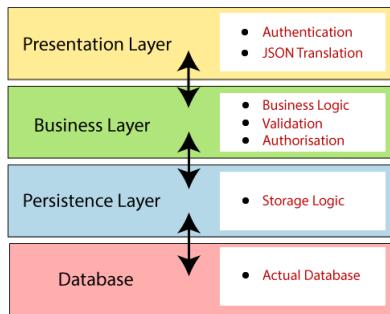
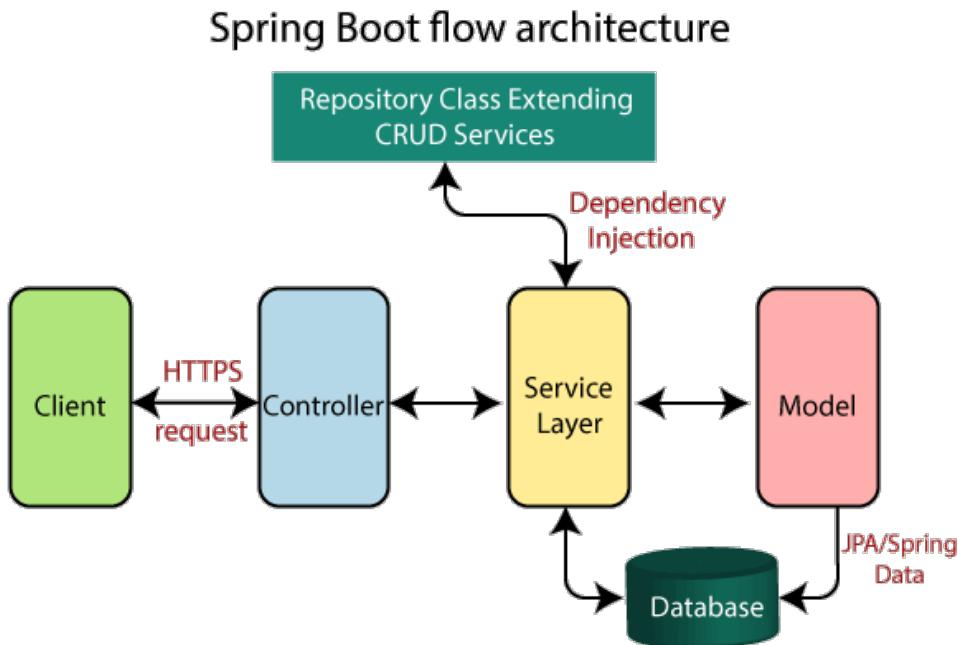
4.2.1.1 Architettura Spring Boot

Spring Boot è un modulo di Spring *framework*. Viene utilizzato per creare applicazioni *stand-alone* di livello produttivo con il minimo sforzo.

Spring Boot segue un'architettura a strati, in cui ogni livello comunica con gli strati vicini.

Ci sono quattro strati in Spring Boot, che sono i seguenti:

- ***Presentation layer***: gestisce le richieste HTTP, trasforma i parametri da formato JSON in classe e autentica la richiesta, per poi trasferirla al *business layer*;
- ***Business Layer***: gestisce tutta la *business logic*. Consiste in classi di servizio e utilizza servizi forniti dagli strati di accesso ai dati;
- ***Persistence Layer***: contiene tutta la *storage logic*, e trasformando gli oggetti provenienti dalla *business logic* in righe del *database*;
- ***Database Layer***: in questo strato vengono effettuate le operazioni [CRUD](#).

**Figura 4.1:** Architettura a strati di Spring Boot**4.2.1.2 Spring Boot Flow Architecture****Figura 4.2:** Spring Boot workflow

Il *workflow* con cui vengono effettuate richieste HTTP utilizzando Spring Boot è il seguente:

- il client esegue una richiesta HTTP (GET, POST, PUT O DELETE) ad un *endpoint*^[g] esposto;
- la richiesta viene ricevuta dal *controller*, che mappa la richiesta e la gestisce. Dopodiché chiama la *service logic* presente nel *service layer*;
- nel *service layer* viene eseguita la *business logic*. Vengono eseguite le operazioni sulle classi mappate nel *database*;

- il *repository JpaRepository* esegue le operazioni sul *database*;
- una pagina [JavaServer Pages \(JSP\)](#) viene restituita all’utente se non si è verificato un errore. [9]

4.2.1.3 Progettazione delle API

Il *back end* del progetto è strutturato a [microservizi](#), ognuno contenente una *business logic* atta a soddisfare un certo tipo di richieste.

Il *front end* comunica con il *back end* attraverso gli [endpoint](#) che ogni [microservizio](#) espone.

Tuttavia, effettuare connessioni dirette fra *front end* e ogni [microservizio](#) presenta alcuni problemi, come:

- numerose connessioni a seconda della quantità dei [microservizi](#);
- i [microservizi](#) devono esporre pubblicamente il proprio [IP](#), causando problemi sia di sicurezza, dovuta all’esposizione degli indirizzi [IP](#) al mondo esterno, sia in fase di latenza, ovvero il tempo che intercorre tra l’invio di una richiesta ed una risposta tenderà ad essere sempre più alto. [11]

Per far fronte a queste problematiche si è deciso di utilizzare un [API Gateway](#).

In questo modo, solamente un [IP](#) sarà visibile pubblicamente, mentre quelli dei [microservizi](#) possono diventare privati.

Per quanto riguarda la latenza, il *front end* comunicherà attraverso l’[API Gateway](#), stabilendo solo le connessioni per la richiesta e la risposta, lasciando all’[API Gateway](#) il compito di smistare le richieste al giusto [microservizio](#), rendendo così la connessione più rapida rispetto all’utilizzo di diverse connessioni per ogni [microservizio](#), essendo tutte le richieste effettuate all’interno dello stesso *network*.

4.2.2 Front end

4.2.2.1 Architettura Angular

Il componente principale di Angular è il **modulo**. Un modulo è un contenitore di funzionalità che sono esposte ad altri moduli. Questa suddivisione in moduli rende la struttura dell’applicazione ordinata e il codice mantenibile.

Un elemento fondamentale di Angular è il **component**, ovvero delle classi che gestiscono le *view* dell’applicazione e la loro logica.

I dati da visualizzare nella *view* vengono forniti dalle classi dette **servizi**. Queste classi svolgono diverse funzioni, come per esempio l’esecuzione delle richieste HTTP.

Ad ogni component è associato un *template*, ovvero del codice HTML in cui si definisce come viene visualizzato il *component*.

È possibile personalizzare il codice HTML utilizzando le **direttive**, ovvero delle classi che aggiungono un comportamento aggiuntivo agli elementi nelle applicazioni Angular. Le direttive integrate di Angular permettono di gestire moduli, elenchi, stili e ciò che gli utenti vedono.

È possibile creare un *component* che rappresenta la pagina completa, in cui inserire diversi *component* per ogni elemento contenuto in quella pagina. Ogni *component* contenuto si occuperà così di gestire la grafica di quella determinata funzionalità e di comunicare con i servizi di cui necessita; sarà poi il *component* “padre” a gestire la disposizione dei *component* utilizzati. [5]

4.2.2.2 Progettazione delle maschere

Con il *tutor* aziendale è stato discusso come dovrebbero essere graficamente le nuove maschere da aggiungere a *Sport Will*. A partire da questi confronti, è stato poi utilizzato **Figma** fare il *mockup*, in modo da testare come i vari elementi visivi siano accattivanti.

Durante la progettazione del *mockup* è stato seguito lo stile presente nella *web app*, in modo da non disorientare l'utente durante la navigazione.

I risultati della progettazione delle maschere sono riportati in seguito.

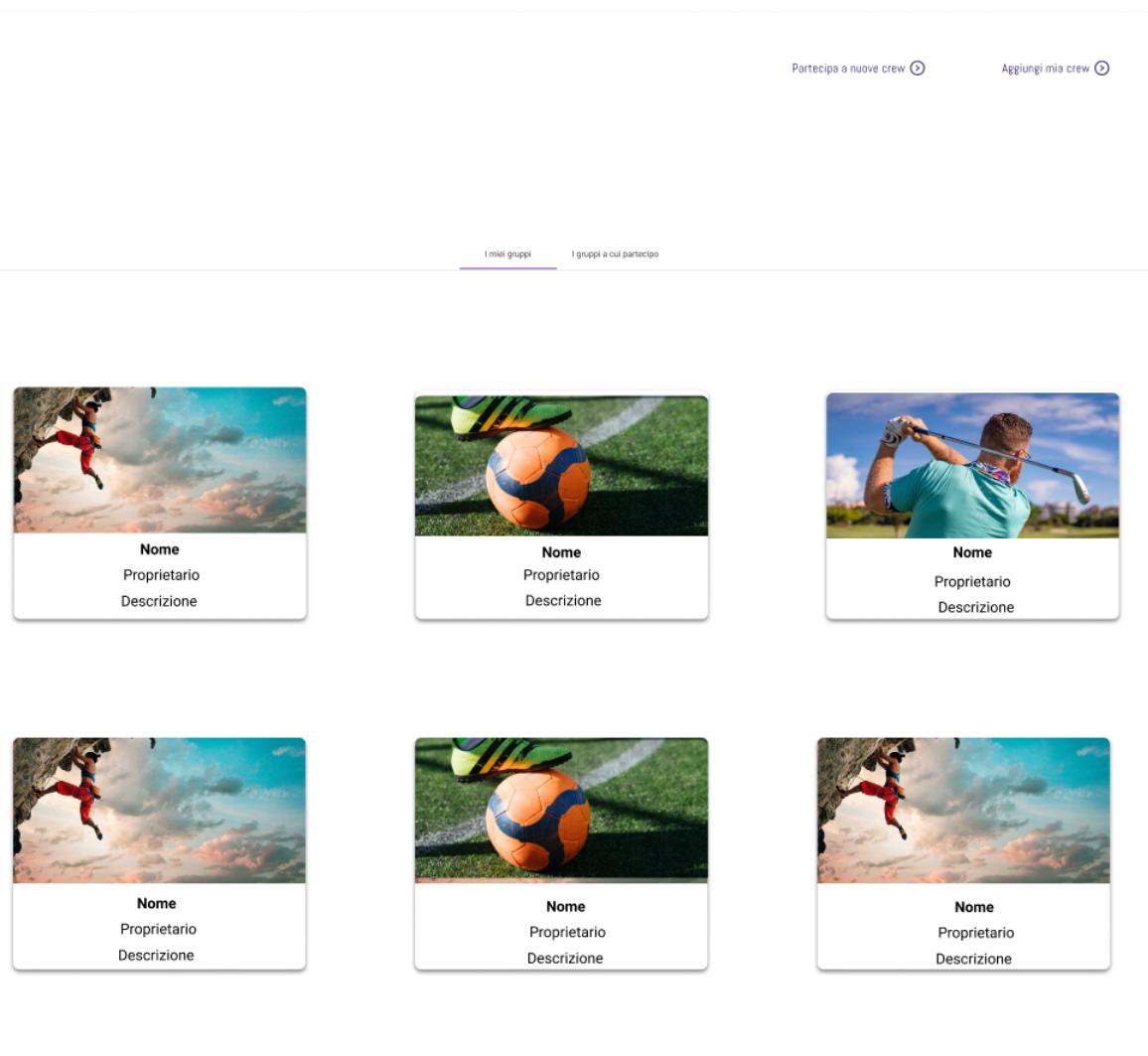


Figura 4.3: Mockup pagina per la visualizzazione dei gruppi creati dall'utente

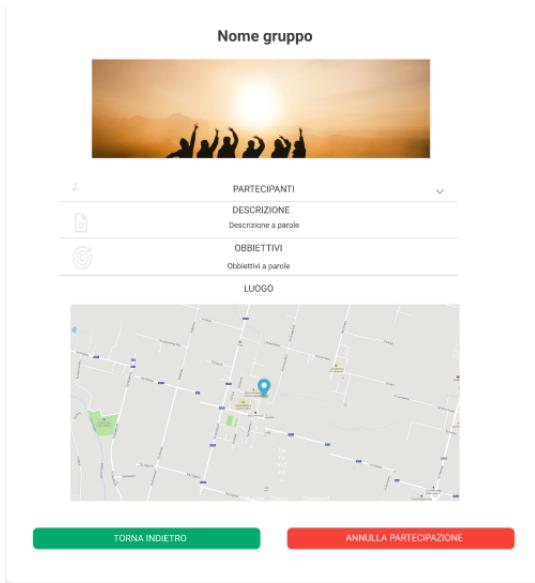


Figura 4.4: Mockup pagina per la visualizzazione dei dettagli di un gruppo

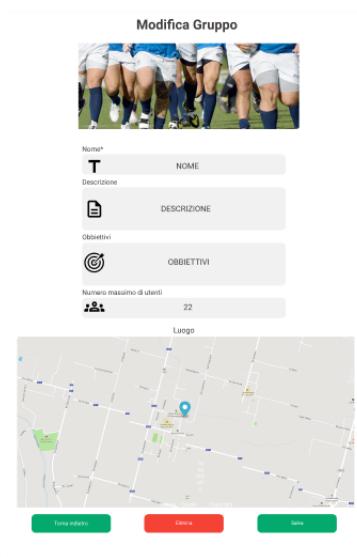


Figura 4.5: Mockup pagina per la modifica dei dettagli di un gruppo

Come si può notare dalle immagini, sono stati omessi dalla progettazione l'*header* e il *footer*, in quanto sono già presenti nella *web app*.

4.3 Design Pattern utilizzati

4.3.1 Microservizi

I [microservizi](#) sono un approccio per lo sviluppo e l'organizzazione dell'architettura *software* secondo cui quest'ultimi sono composti di servizi indipendenti di piccole dimensioni che comunicano tra loro tramite [API](#) ben definite.

Nel contesto di questo progetto, sono stati implementati tre [microservizi](#):

- **SW_Gruppi:** [microservizio](#) responsabile per la gestione delle funzionalità legate ai gruppi;
- **ApiGateway:** [microservizio](#) che implementa il *pattern API Gateway*;
- **EurekaServer:** [microservizio](#) che implementa il *pattern ServiceRegistry*. [2]

4.3.2 API Gateway

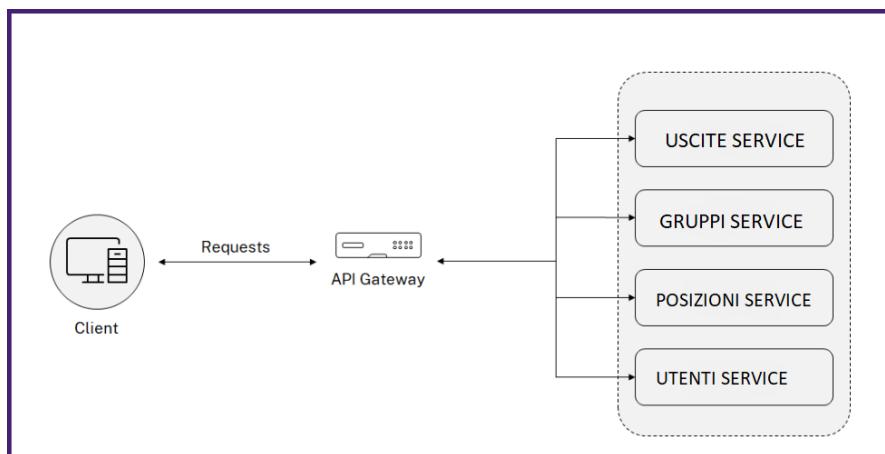


Figura 4.6: Diagramma concettuale che descrive l'*API Gateway*

Un **API Gateway** è uno strumento di gestione delle [API](#) che si situa tra un *client* e una raccolta di servizi *back end*. Un *API Gateway* si comporta come un *proxy* inverso per ricevere tutte le chiamate [API](#), aggregando le chiamate alle [API](#) dei servizi richiesti, gestendo e restituendo i risultati appropriati in base alle richieste ricevute.

Utilizzare un *API Gateway* è vantaggioso perché:

- permette di centralizzare il punto di ingresso per le chiamate;
- permette di monitorare le risorse utilizzate;
- permette di proteggere un servizio che è aperto a tutti;
- ha latenza minore rispetto alla chiamata a diversi servizi.

4.3.3 Client-Side Service Discovery e Service Registry

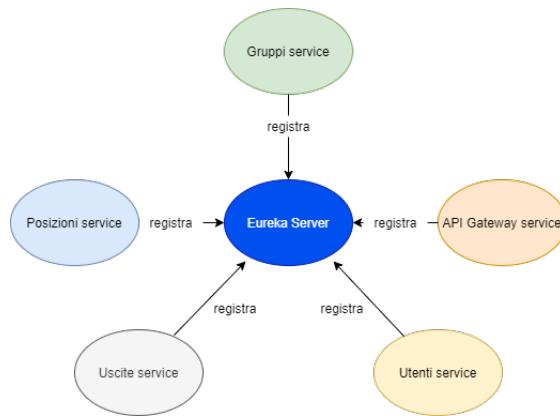


Figura 4.7: Diagramma concettuale che descrive la registrazione dei microservizi all'[Eureka Server](#)

I [microservizi](#) hanno una natura dinamica, in quanto è possibile che più istanze di un singolo [microservizio](#) coesistano, probabilmente esponendo le loro [API](#) in un indirizzo IP diverso o in una porta diversa. Queste evenienze portano all'impossibilità di:

- conoscere la posizione di qualsiasi istanza dei [microservizi](#);
- tenere traccia di tutte le istanze;
- selezionare un'istanza di [microservizio](#).

La soluzione a questi problemi è l'utilizzo del *pattern Client-side Service Discovery*, che fornisce un meccanismo che tiene traccia di tutti i servizi e delle relative istanze. Tutti i [microservizi](#) si registrano ad un *Service registry* e continuano ad aggiornare regolarmente le proprie informazioni di rete.[\[14\]](#)

Il *pattern Service registry* è stato applicato mediante l'implementazione di un [Eureka Server](#). L'[Eureka Server](#) è un *database* di servizi che tiene traccia di ogni [microservizio](#), delle loro istanze e delle loro locazioni. I [microservizi](#) si registrano all'avvio dell'applicazione e vengono rimossi alla sua chiusura.

L'utilizzo di questi *pattern* permette ai servizi di comunicare fra di loro, ottenendo le informazioni degli altri servizi direttamente dall'[Eureka Server](#).[\[12\]](#)

4.3.4 Dependency Injection

Sia Angular sia Spring sono dei [framework](#) che implementano delle convenzioni che permettono l'utilizzo del *pattern Dependency Injection*.

L'ecosistema all'interno del quale le applicazioni Spring vivono viene definito **IoC container**. L'*IoC container* si occupa di istanziare gli oggetti (*beans*) dichiarati nel progetto e di reperire e iniettare tutte le dipendenze ad essi associate. Tali dipendenze possono essere componenti del [framework](#) o altri *bean* dichiarati nel contesto applicativo.

La dichiarazione di una classe come componente nel progetto avviene tramite l'utilizzo dell'annotazione `@Component`, che rappresenta la categoria più generica per la

dichiarazione di un componente. Durante la fase di codifica del progetto sono state utilizzate le annotazioni `@RestController`, `@Service` e `@Repository`, che sono delle specializzazioni dell'annotazione `@Component`.

Per iniettare delle classi da recuperare dal *IoC container* è sufficiente utilizzare l'annotazione `@Autowired`.^[6]

```

1  @RestController
2  @CrossOrigin(origins = "*", maxAge = 3600)
3  @RequestMapping(value = "/gruppi")
4  public class GruppiController {
5
6      @Autowired
7      private GruppiService gruppiService;
8
9      @Autowired
10     EurekaClient eurekaClient;
11 }
```

Codice 4.1: Esempio di *Dependency Injection* con Spring

Angular permette l'utilizzo della *Dependency Injection* di tipo *constructor*, che prevede che una classe dichiari nel costruttore le dipendenze di cui ha bisogno che in fase di inizializzazione gli verranno fornite.

Angular include un meccanismo di *Dependency Injection* davvero solido e molto flessibile, il cui utilizzo si può riassumere in due semplici passi: si crea il servizio e si inietta la dipendenza ove necessario.

Nel dettaglio, le fasi sono le seguenti:

- si crea un servizio, ovvero una classe Angular, che viene annotata come `@Injectable`. Di *default*, questo decoratore ha una proprietà `providedIn`, che crea un *provider* per il servizio. Nel caso, per esempio, di `providedIn: 'root'`, viene specificato che Angular dovrebbe fornire il servizio nella *root injector*, ovvero disponibile a tutte le classi;

```

1  @Injectable({
2      providedIn: 'root',
3  })
4  export class GruppiService {
```

Codice 4.2: Esempio di creazione di un servizio

- si inietta il servizio e lo si utilizza ovunque sia necessario. [3]

```

1  @Component({
2      selector: 'app-crew-card',
3      templateUrl: './crew-card.component.html',
4      styleUrls: ['./crew-card.component.css'],
5  })
6  export class CrewCardComponent implements OnInit {
7
8      constructor(private auth: AuthenticationService, private
9                  router: Router, private gruppiService: GruppiService) {
10 }
```

11 }

Codice 4.3: Esempio di *constructor injection*

4.3.5 Singleton

Angular e Spring integrano fra i loro *pattern* il **Singleton**, che viene implementato mediante l'utilizzo della [Dependency Injection](#).

Utilizzando questo *pattern* è stato possibile creare una sola istanza di una classe, potendola utilizzare fra i vari servizi e componenti.

L'utilizzo del *pattern Singleton*, tuttavia, potrebbe violare il [Single Responsibility Principle](#)[g], rendendolo un *anti-pattern*. Nel contesto del progetto di *stage*, questo *pattern* è stato utilizzato per le classi di tipo *Controller*, *Service* e *Repository*. In questo caso il suo utilizzo è necessario in quanto sarebbe logicamente sbagliato avere più istanze per il tipo di classi sopra citate. [13]

4.3.6 Feature Service

Questo *pattern* è utilizzato da Angular per estrarre da una classe di tipo *Component* la sua relativa logica, che può essere per esempio la visualizzazione dei dettagli di un gruppo.

Mediante l'utilizzo del *pattern Singleton* e [Dependency Injection](#) è possibile utilizzare i vari componenti in tutti i componenti dell'applicazione. [4]

4.3.7 Lazy Loading

Questo *pattern* è utilizzato da Angular per caricare i moduli solo nel momento in cui effettivamente vengono utilizzati, ottenendo così bassa la quantità di dati scaricata dall'utente iniziale e una diminuzione del tempo necessario al caricamento dell'applicazione.

Viene utilizzato questo *pattern* per la gestione del *routing*, associando ad una specifica *view* dell'applicazione un *path* di navigazione presente nella [URL](#).[7]

```

1 const routes: Routes = [
2   { path: 'homepage', component: HomepageComponent },
3   { path: 'detail/:id', component: WillDetailComponent },
4   { path: 'edit/:id', component: EditableFormComponent },
5   { path: 'editcrew/:id', component: EditableCrewComponent },
6   { path: 'detailcrew/:id', component: CrewDetailComponent },
7   { path: 'signIn', component: SignInComponent },
8   { path: 'myCrew/:type', component: MyCrewpageComponent },
9   { path: 'signUp', component: SignUpComponent },
10  { path: 'add', component: EditableFormComponent },
11  { path: '', redirectTo: '/homepage', pathMatch: 'full' },
12  { path: '404', component: NotFoundComponent },
13  { path: '**', redirectTo: '404' },
14];
15
16 @NgModule({
17   imports: [RouterModule.forRoot(routes)],
18   exports: [RouterModule]

```

```

19     })
20     export class AppRoutingModule { }

```

Codice 4.4: Implementazione del pattern *Lazy loading* nel AppRoutingModule

4.3.8 Observer

Gli ***Observable*** forniscono supporto per il passaggio di messaggi tra le parti dell'applicazione. Sono usati frequentemente in Angular e sono una tecnica per la gestione ad eventi e la programmazione asincrona.

Il *pattern Observer* è un *design pattern* in cui un oggetto, chiamato **Subject**, contiene una lista di suoi osservatori, chiamati **Observers**, e li notifica automaticamente ai cambiamenti di stato.

Gli *Observable* sono dichiarativi, ovvero viene definita una funzione che non viene eseguita fino a quando un *Observer* si sottoscrive ad esso. L'*Observer* viene quindi notificato al completamento della funzione, e il tipo di ritorno di un *Observable* può cambiare in base al contesto, che nel caso nel progetto di *stage* è stato prevalentemente di tipo HTTP *response*. [17]

```

1  export class MyCrewpageComponent implements OnInit {
2      crewCards: Crew[] = [];
3      constructor(
4          private gruppiService: GruppiService,
5          private router: Router,
6          private route: ActivatedRoute) {
7      }
8      getGlobalCrew() {
9          this.gruppiService.get CrewsOfUser().subscribe((crews) =>
10             {
11                 this.gruppiService.get Crews().subscribe((crews2) => {
12                     this.crewCards = crews2.filter((c) => {
13                         let toReturn: boolean = true;
14                         for (let crew of crews){
15                             if (crew.id == c.id){
16                                 toReturn = false;
17                                 break;
18                             }
19                         }
20                         return toReturn;
21                     });
22                 });
23             }
24         }
25
26 @Injectable({
27     providedIn: 'root',
28 })
29 export class GruppiService {
30     constructor(private http: HttpClient) {}
31
32     getCrews(): Observable<Crew[]> {

```

```

33     }
34   }
35 }
```

Codice 4.5: Esempio implementazione *pattern Observer*

4.4 Codifica

4.4.1 Back end

4.4.1.1 Gruppi service

Crew

Questa classe rappresenta i gruppi. La tabella che mappa la classe nel *database* è rappresentata nella figura 4.8. Per aggiungere o modificare un gruppo è necessario rappresentare la classe in formato JSON, come in figura 4.6. Per creare un nuovo gruppo è necessario specificare almeno i campi **nome**, **proprietario**, **latitudine** e **longitudine**.

COLUMN_NAME	TYPE_NAME	NULLABLE
id	int4	0
descrizione	varchar	1
latitudine	float4	0
longitudine	float4	0
max_user	int4	1
nome	varchar	0
obiettivi	varchar	1
proprietario	varchar	0

Figura 4.8: Tabella Crew

```

1 {
2   "nome": "Prova",
3   "proprietario": "proprietario",
4   "latitudine": 45.388552,
5   "longitudine": 11.9287197,
6   "descrizione": "descrizione",
7   "obiettivi": "obiettivi",
8   "maxUser": 20
9 }
```

Codice 4.6: Rappresentazione JSON della classe Crew

GruppiController

Classe che permette di esporre le API mappate nel path `/gruppi`. L'annotazione `@RestController` permette di creare un controller Restful, e l'oggetto da restituire viene serializzato automaticamente in JSON e restituito all'oggetto di risposta HTTP.

Metodi principali

- ***getAllGruppi***: gestisce la chiamata che visualizza tutti i gruppi.
Mappatura: /gruppi/, **verbo HTTP:** GET;
- ***getGruppo***: gestisce la chiamata che visualizza un gruppo specifico.
Mappatura: /gruppi/{id}, **verbo HTTP:** GET;
- ***getUtentiByGruppo***: gestisce la chiamata che visualizza gli utenti appartenenti ad un gruppo.
Mappatura: /gruppi/{idCrew}/utenti, **verbo HTTP:** GET;
- ***createGruppo***: gestisce la chiamata che inserisce una nuovo gruppo. È necessario specificare i dati del gruppo da inserire nel *body* della chiamata HTTP, come in figura 4.6.
Mappatura: /gruppi/inserisci, **verbo HTTP:** POST;
- ***modifyGruppo***: gestisce la chiamata che modifica un gruppo. È necessario specificare i dati del gruppo da modificare nel *body* della chiamata HTTP, come in figura 4.6. I campi non specificati non verranno modificati.
Mappatura: /gruppi/modifica/{idCrew}, **verbo HTTP:** PUT;
- ***getUsciteByGruppo***: gestisce la chiamata che visualizza le uscite di una gruppo.
Mappatura: /gruppi/{idCrew}/uscite, **verbo HTTP:** GET;
- ***deleteGruppo***: gestisce la chiamata che elimina un gruppo.
Mappatura: /gruppi/elimina/{idCrew}, **verbo HTTP:** DELETE;
- ***getGruppiUtente***: gestisce la chiamata che richiede tutti i gruppi a cui appartiene un utente.
Mappatura: /gruppi/utente/{idUtente}, **verbo HTTP:** GET;
- ***addUtenteToGruppo***: gestisce la chiamata che permette ad un utente di unirsi ad un gruppo.
Mappatura: /gruppi/{idCrew}/aggiungi/utente/{idUtente}, **verbo HTTP:** POST;
- ***removeUtenteFromGruppo***: gestisce la chiamata che permette ad un utente di rimuoversi da un gruppo.
Mappatura: /gruppi/{idCrew}/elimina/utente/{idUtente}, **verbo HTTP:** DELETE;
- ***addUscitaToGruppo***: gestisce l'aggiunta di un'uscita ad un gruppo.
Mappatura: /gruppi/{idCrew}/aggiungi/uscita/{idUscita}, **verbo HTTP:** POST;
- ***removeUscitaFromGruppo***: gestisce l'eliminazione di un'uscita ad un gruppo.
Mappatura: /gruppi/{idCrew}/elimina/uscita/{idUscita}, **verbo HTTP:** DELETE.

GruppiService

Classe responsabile della *business logic* del [microservizio](#).

Siccome fornisce funzionalità di *business*, questa classe è annotata con l'annotazione `@Service`, e funge da intermediario fra la classe [GruppiController](#) e il [Data Access Object \(DAO\)](#), ovvero le classi [CrewRepository](#), [JointUtentiCrewRepository](#) e [JointUsciteCrewRepository](#).

I metodi di questa classe hanno gli stessi nomi e funzione dei metodi presenti in [GruppiController](#).

Repository

Interfacce con funzionalità [JPA](#) che permettono di mappare una classe in una tabella di un *database* relazionale, svolgendo anche il compito [EntityManager](#)^[g], effettuando l'accesso agli oggetti ed eseguendo operazioni [CRUD](#) sui dati immagazzinati nelle tabelle del *database*.

Tutte queste interfacce estendono l'interfaccia `JpaRepository<T, ID>`, dove T è il tipo della classe da mappare, ed ID è il tipo dell'identificativo della classe mappata.

L'interfaccia `JpaRepository` permette di creare le *query* a partire dal nome del metodo, senza doverlo necessariamente implementare.

Il meccanismo che permette di generare le *query* integrato nell'infrastruttura JPA Spring Data è utile per creare *query* vincolate sulle entità del *repository*. Questo meccanismo rimuove i prefissi `find...By`, `read...By` e `get...By` dal metodo ed effettua il *parsing* della funzione alla ricerca dei nomi delle proprietà dell'entità mappata nel `JpaRepository`, come nel seguente esempio:

```
List<JointUsciteCrew> findAllByIdCrew(int idCrew);
```

in cui `idCrew` è una proprietà dell'entità `JointUsciteCrew`.

Le proprietà inserite nel nome del metodo possono essere concatenate con *And* e *Or*, ma anche con operatori come *Between*, *LessThan*, *GreaterThan*, *Like*.

Un esempio è il seguente:

```
Long deleteByIdUscitaAndIdCrew (int idUscita, int idCrew);
```

È possibile applicare l'ordinamento statico aggiungendo una clausola *OrderBy* al metodo di *query* facendo riferimento ad una proprietà, come nel seguente caso:

```
List<Crew> findByOrderId();
```

Ci sono tre *repository* nel [microservizio](#): `JointUsciteCrewRepository`, `JointUtentiCrewRepository` e `CrewRepository`.

JointUsciteCrewRepository

Classe di *repository* che contiene le uscite di un gruppo. Ogni qualvolta un utente aggiunge un'uscita, essa viene aggiunta alle uscite di tutti i gruppi a cui appartiene. Viceversa quando viene eliminata un'uscita viene eliminata dalle uscite di tutti i gruppi. Un accorgimento che è stato fatto è che quando viene rimosso un gruppo, tutte le uscite associate a quel gruppo vengono rimosse da questa tabella (n.b. le uscite non vengono eliminate dalla tabella `Uscite` presente nel *database*, vengono solo eliminate le voci relative nella tabella `joint_uscrite_crew`).

Metodi

- `List<JointUsciteCrew> findAllByIdCrew(int idCrew)`: restituisce tutte le occorrenze in base all'id del gruppo;
- `Long deleteByIdUscitaAndIdCrew (int idUscita, int idCrew)`: elimina le occorrenze in base all'id del gruppo e all'id dell'uscita. Ritorna il numero di occorrenze eliminate (che sono al più una);

- ***JointUsciteCrew getByIdUscitaAndIdCrew(int idUscita, int idCrew):*** restituisce un'occorrenza in base all'id dell'uscita e all'id del gruppo;
- ***Long deleteByIdCrew(int idCrew):*** elimina tutte le occorrenze in base all'id del gruppo. Ritorna il numero di occorrenze eliminate;
- ***void deleteByIdUscita(int idUscita):*** elimina tutte le occorrenze in base all'id dell'uscita.

JointUtentiCrewRepository

Classe di *repository* che contiene le partecipazioni degli utenti ai gruppi. Come in [JointUsciteCrewRepository](#), quando viene eliminato un utente vengono rimosse le partecipazioni degli utenti a tutti i gruppi a cui partecipava. Tuttavia è stato deciso di non eliminare i gruppi creati da un utente nel caso di eliminazione, in quanto i gruppi non sono strettamente associati al suo creatore, ma sono delle entità indipendenti.

Metodi

- ***List<JointUtentiCrew> findAllByIdUtente(String utenteId):*** restituisce tutte le occorrenze in base all'id dell'utente;
- ***void deleteByIdUtenteAndIdCrew(String idUtente, int idCrew):*** elimina le occorrenze in base all'id dell'utente e all'id del gruppo;
- ***JointUtentiCrew getByIdUtenteAndIdCrew(String idUtente, int idCrew):*** restituisce un'occorrenza in base all'id dell'utente e all'id del gruppo;
- ***Long deleteByIdCrew(int idCrew):*** elimina tutte le occorrenze in base all'id del gruppo. Ritorna il numero di occorrenze eliminate;
- ***List<JointUtentiCrew> findAllByIdCrew(int id):*** restituisce tutte le occorrenze in base all'id del gruppo;
- ***void deleteByIdUtente(String idUtente):*** elimina tutte le occorrenze in base all'id dell'utente.

CrewRepository

Classe di *repository* che contiene i dettagli dei gruppi.

Metodi

- ***List<Crew> findByOrderByid():*** restituisce tutti i gruppi ordinati per id;
- ***Crew findById(int id):*** restituisce un gruppo in base al suo id;
- ***Long deleteById(int id):*** elimina un'occorrenza in base all'id. Ritorna uno se avviene un'eliminazione, zero altrimenti.

4.4.1.2 Uscite Service

In questo [microservizio](#) sono state effettuate le seguenti modifiche:

- è stato aggiunto un campo booleano `visGlobale` alla classe `Uscita` che permette di specificare se l'uscita sarà visibile da tutti gli utenti oppure soltanto dagli utenti appartenenti agli stessi gruppi;
- è stato aggiunto al metodo `createUscita` presente nella classe `UsciteController` la possibilità di inserire nel corpo per la creazione dell'uscita il campo che specifica il tipo di visibilità desiderata.
Nello stesso metodo è stata anche inserita una chiamata HTTP che aggiunge l'uscita appena creata al [repository](#) `JointUsciteCrewRepository`;
- è stato aggiunto al metodo `eliminaUscita` una chiamata HTTP che rimuove l'uscita dal [repository](#) `JointUsciteCrewRepository`;
- è stato aggiunto un metodo `modificaVisibilità` alla classe `UsciteController` che permette di modificare la visibilità dell'uscita. Questo metodo viene mappato in `uscite/modifica/visibilità/{id}`, ed `id` specifica l'`id` del gruppo del quale si vuole modificare la visibilità. Il verbo HTTP della chiamata è di tipo PUT;
- è stato aggiunto un metodo `getAllUsciteGlobali` alla classe `UsciteController` che permette di ricevere tutte le uscite che hanno il campo `visGlobale = true`. Questo metodo viene mappato in `uscite/globali`, ed il verbo HTTP della chiamata è di tipo GET.

4.4.1.3 Api Gateway

È stato implementato un *Spring Cloud Gateway* applicando dei filtri alle chiamate per verificare se è presente e valido il *token* per l'autenticazione presente nell'*header* della richiesta HTTP. Per fare ciò è stato creata una classe `AuthFilter` che estende l'interfaccia `AbstractGatewayFilterFactory` ed implementa il metodo astratto `GatewayFilter apply(C config)` presente nella *superclasse*. In questa funzione viene letta la richiesta in entrata viene verificata l'autenticazione.

```

1  @Component
2  public class AuthFilter extends AbstractGatewayFilterFactory<
3      AuthFilter.Config> {
4
5      private WebClient.Builder webClientBuilder;
6
7      @Autowired
8      EurekaClient eurekaClient;
9
10     public AuthFilter(Builder webClientBuilder) {
11         super(Config.class);
12         this.webClientBuilder = webClientBuilder;
13     }
14
15     public static class Config {
16         /*
17          * Questa classe rimane vuota perché non c'è bisogno
18          * di particolari impostazioni
19     }

```

```

17         * di configurazione
18     */
19 }
20
21     private Mono<Void> onError(ServerWebExchange exchange,
22         String err, HttpStatus httpStatus) {
23         ServerHttpResponse response = exchange.getResponse();
24         response.setStatusCode(httpStatus);
25
26         return response.setComplete();
27     }
28     @Override
29     public GatewayFilter apply(Config config) {
30         return (exchange, chain) -> {
31             if (!exchange.getRequest().getHeaders().
32                 containsKey(HttpHeaders.AUTHORIZATION)) {
33                 return this.onError(exchange, "****"
34                         Informazioni di autorizzazione mancanti
35                         ****", HttpStatus.UNAUTHORIZED);
36             }
37             String auth = exchange.getRequest().getHeaders().
38                 get(HttpHeaders.AUTHORIZATION).get(0);
39             String[] parts = auth.split(" ");
40             if (parts.length != 2 || !"Bearer".equals(parts[0]
41                 )) {
42                 return this.onError(exchange,
43                         "**** Autorizzazione incorrenta ***",
44                         HttpStatus.UNAUTHORIZED);
45             }
46             InstanceInfo nextServerFromEureka = eurekaClient.
47                 getNextServerFromEureka("UTENTI-SERVICE",
48                 false);
49             String homeUrl = nextServerFromEureka.
50                 getHomePageUrl();
51             return webClientBuilder.build().post().uri(
52                 homeUrl + "/validateToken?token=" + parts[1])..
53                 retrieve()
54                     .bodyToMono(String.class).map(login -> {
55                         exchange.getRequest().mutate().header
56                             ("X-auth-user-id", String.valueOf(
57                                 login));
58                         return exchange;
59                     }).flatMap(chain::filter);
60     };
61 }
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
625
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
```

```

3      name: API-GATEWAY-SERVICE
4      main:
5          allow-bean-definition-overriding: true
6      cloud:
7          gateway:
8              discovery.locator.enabled: true
9          routes:
10             - id: utenti
11                 uri: lb://UTENTI-SERVICE
12                 predicates:
13                     - Path= /utenti/**, /signin , /signup , /
14                         validateToken
15                 filters:
16                     - AuthFilter
17             - id: posizioni
18                 uri: lb://POSIZIONI-SERVICE
19                 predicates:
20                     Path=/posizioni/**
21                 filters:
22                     - AuthFilter
23             - id: uscite
24                 uri: lb://USCITE-SERVICE
25                 predicates:
26                     Path=/uscite/**
27                 filters:
28                     - AuthFilter
29             - id: gruppi
30                 uri: lb://GRUPPI-SERVICE
31                 predicates:
32                     Path=/gruppi/**
33                 filters:
34                     - AuthFilter
35             default-filters:
36                 - DedupeResponseHeader=Access-Control-Allow-
37                     Credentials Access-Control-Allow-Origin
38         globalcors:
39             corsConfigurations:
40                 '[/*]':
41                     allowedOrigins: "http://localhost:4200/"
42                     allowedMethods: "*"
43                     allowedHeaders: "*"

```

Codice 4.8: Configurazione dello *Spring Cloud Gateway* nel file `application.yml` del microservizio API *Gateway*

4.4.2 Eureka Server

Questo **microservizio** contiene solo una classe al suo interno, che è la classe che viene generata automaticamente quando si inizializza un progetto con Spring. L'unica cosa

aggiunta alla classe è l'annotazione `@EnableEurekaServer`, che permette di attivare la l'Eureka Server come specificato nel file `application.yml`.

```

1 server:
2   port: 8761
3 eureka:
4   client:
5     registerWithEureka: false
6     fetchRegistry: false
7     serviceUrl:
8       defaultZone: http://localhost:8761/eureka
9   server:
10    enable-self-preservation: false

```

Codice 4.9: File `application.yml` del microservizio Eureka Server

Nella *Main class* di tutti i **microservizi** è stata aggiunta l'annotazione `@EnableEurekaClient`, che permette di registrarsi all'Eureka Server in base a quanto specificato nel file `application.properties`.

```

1 eureka.client.service-url.defaultZone: http://localhost:8761/
  eureka
2 eureka.client.register-with-eureka=true
3 eureka.client.fetch-registry=true

```

Codice 4.10: Configurazione per la registrazione di un microservizio all'Eureka Server nel file `application.properties`

4.4.2.1 Docker

Per ogni **microservizio** è stato creato un file `Dockerfile` contenente le operazioni da effettuare per la creazione di un'immagine Docker. Tutti i `Dockerfile` hanno la seguente struttura:

```

1 FROM openjdk:11
2 ARG JAR_FILE=target *.jar
3 COPY ${JAR_FILE} nome_microservizio.jar
4 ENTRYPOINT ["java","-jar","nome_microservizio.jar"]

```

Per eseguire ogni `container`^[g] nello stesso *network* è stato creato un file `docker-compose.yml` in cui vengono specificate le immagini Docker da avviare.

4.4.3 Front end

In questa sezione verranno descritte solo le maschere, componenti e servizi modificati in maniera rilevante o creati. Tutto ciò che era già presente nella *web app* non verrà descritta.

4.4.3.1 Maschere

Homepage

Questa pagina permette ad un utente autenticato di visualizzare le *will* visibili da tutti gli utenti, le *will* create dall'utente e le *will* dei gruppi a cui partecipa.

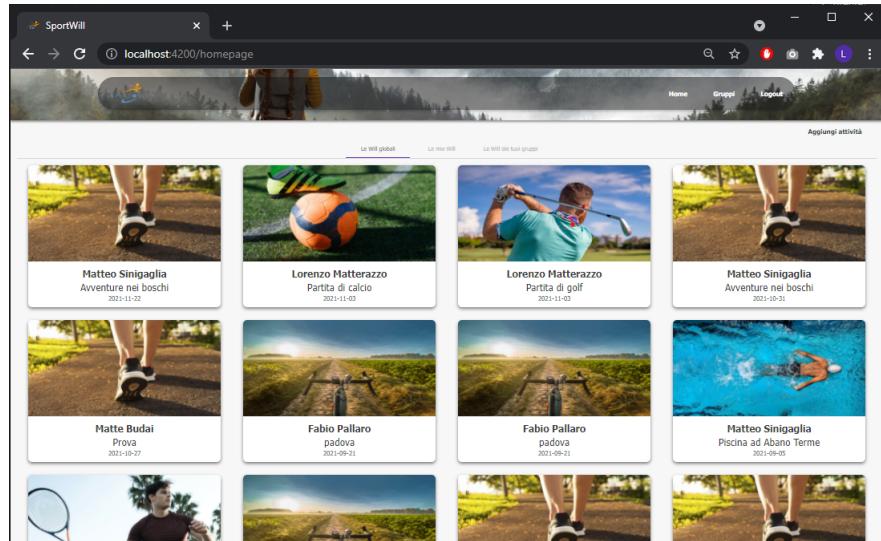


Figura 4.9: Pagina di homepage

Componenti utilizzati

- [HomepageComponent](#)

Gruppi dell'utente

Questa pagina permette ad un utente autenticato di visualizzare i gruppi creati o a cui partecipa un utente.

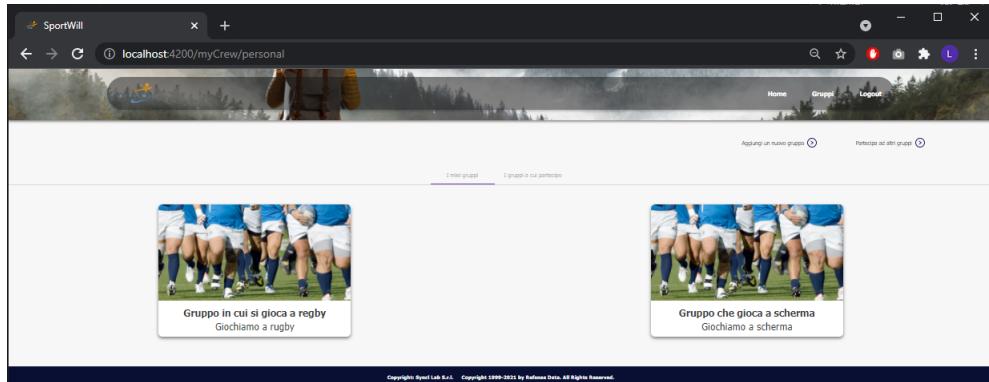


Figura 4.10: Pagina dei gruppi creati o a cui partecipa un utente

Componenti utilizzati

- Crewpage

Gruppi

Questa pagina permette ad un utente autenticato di visualizzare i gruppi a cui può partecipare, senza visualizzare i gruppi a cui già partecipa.



Figura 4.11: Pagina dei gruppi

Componenti utilizzati

- Crewpage

Crea nuovo gruppo

Questa pagina permette ad un utente autenticato di creare un nuovo gruppo.

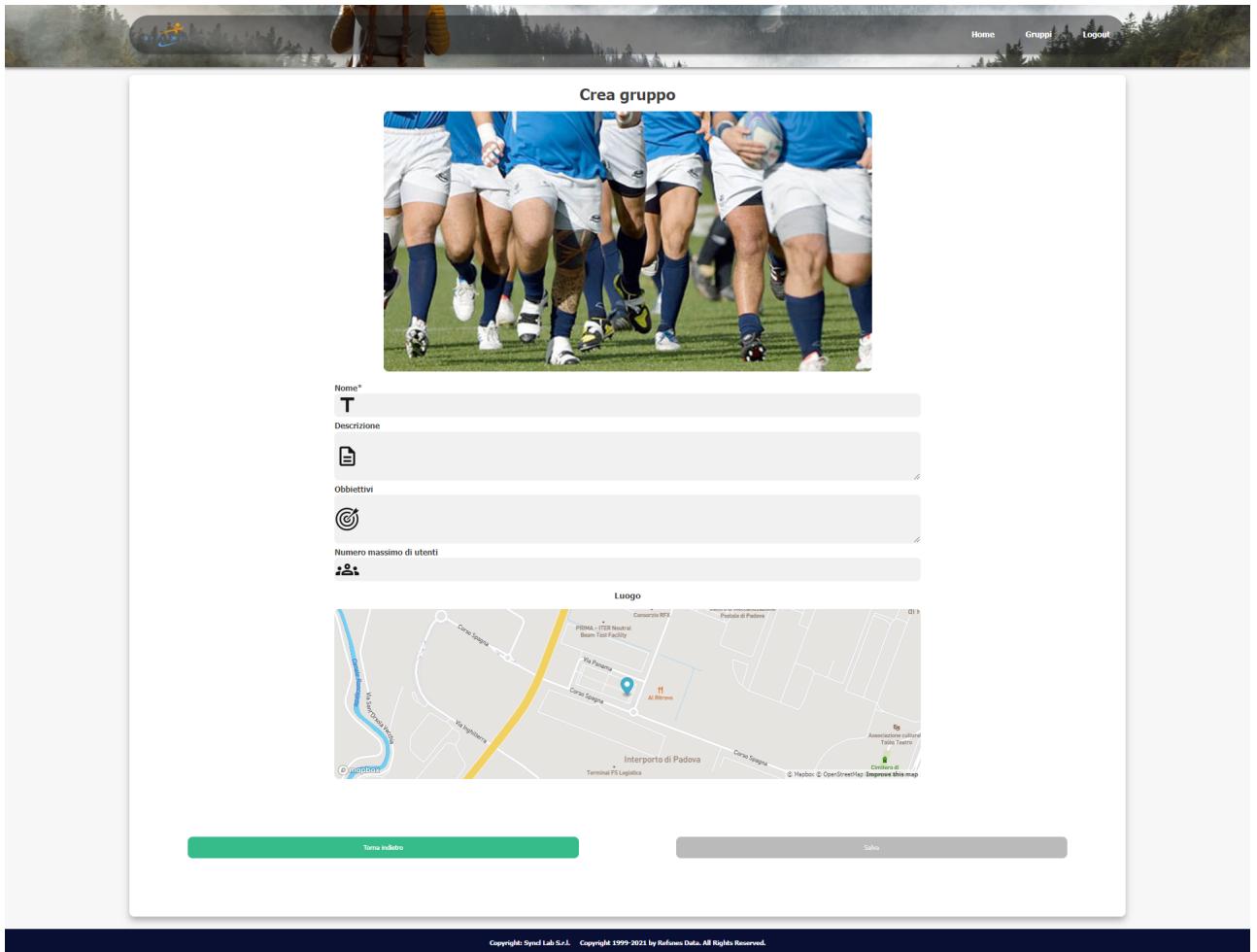


Figura 4.12: Pagina di creazione di un gruppo

Componenti utilizzati

- CrewDetail

Modifica un gruppo

Questa pagina permette ad un utente autenticato di modificare i dettagli di un gruppo.

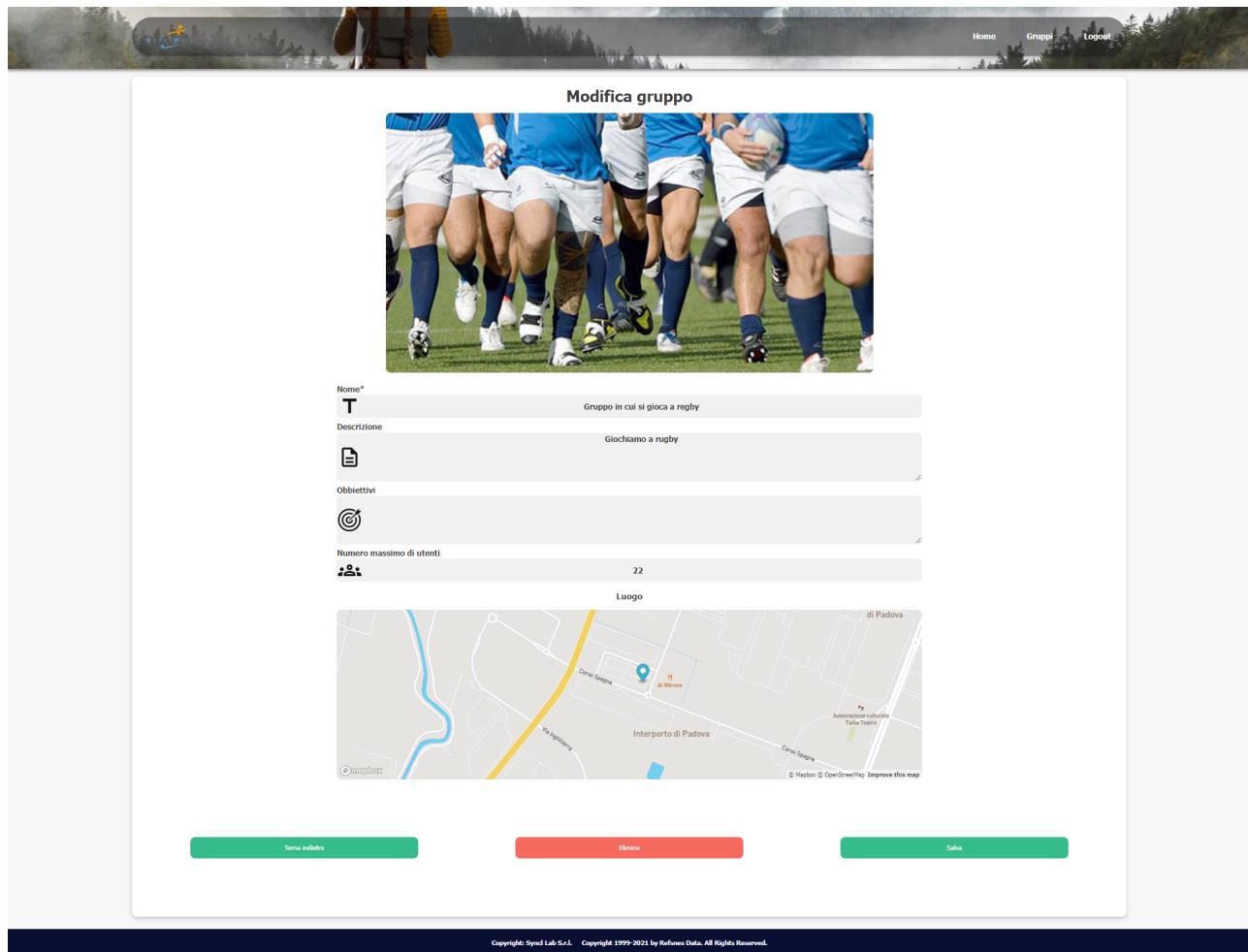


Figura 4.13: Pagina di modifica di un gruppo

Componenti utilizzati

- CrewEditable

Visualizza dettagli di un gruppo

Questa pagina permette ad un utente autenticato di visualizzare i dettagli di un gruppo.

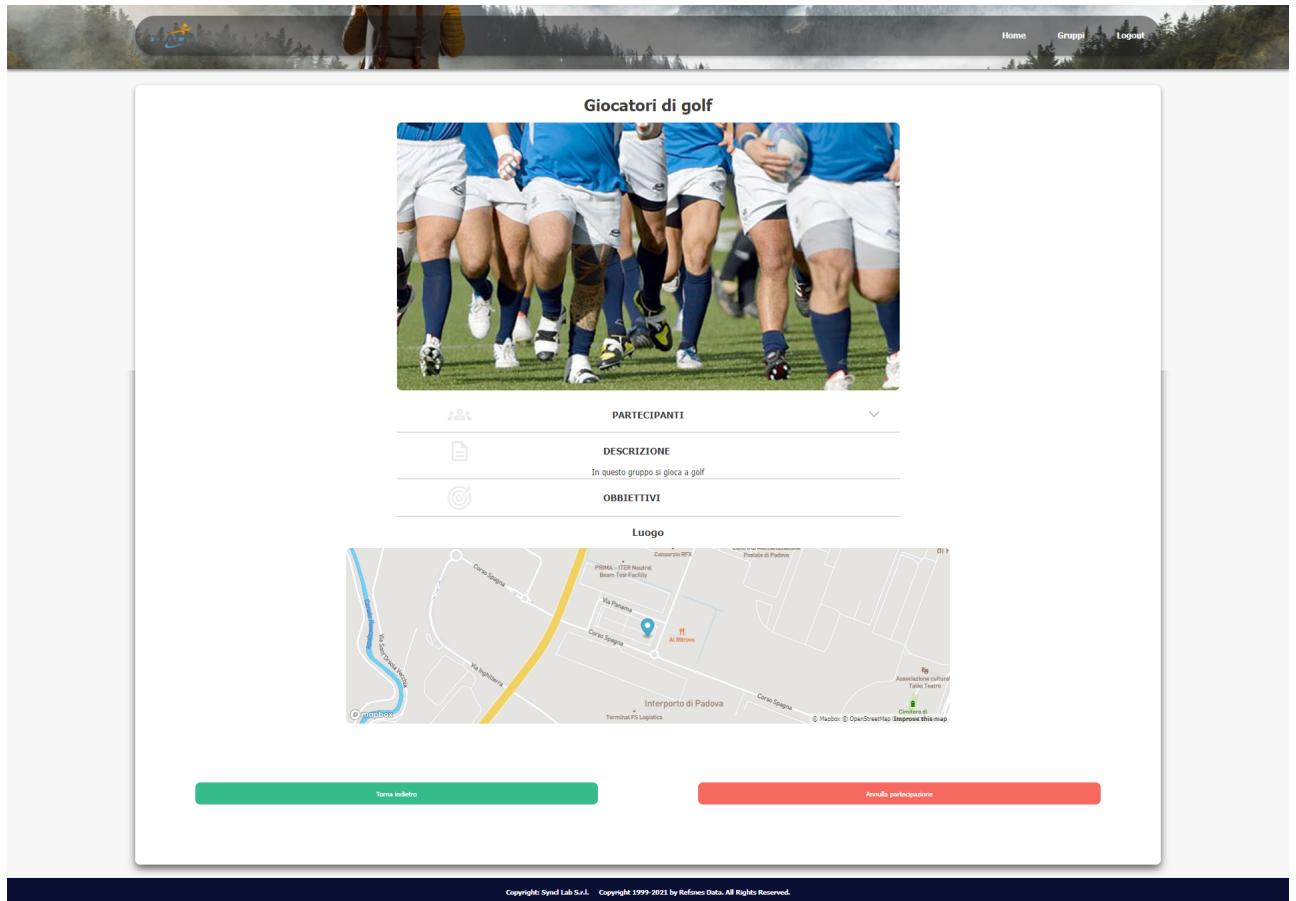


Figura 4.14: Pagina di visualizzazione dei dettagli di un gruppo

Componenti utilizzati

- [CrewDetail](#)

4.4.3.2 Componenti

HomepageComponent

Questo componente è composto da una lista di [WillCard](#), ed ha le seguenti funzionalità:

- visualizzare le *will* visibili a tutti gli utenti (anche quelli non autenticati);
- visualizzare le *will* dell'utente nel caso sia autenticato;
- visualizzare le *will* dei gruppi a cui appartiene l'utente nel caso sia autenticato.

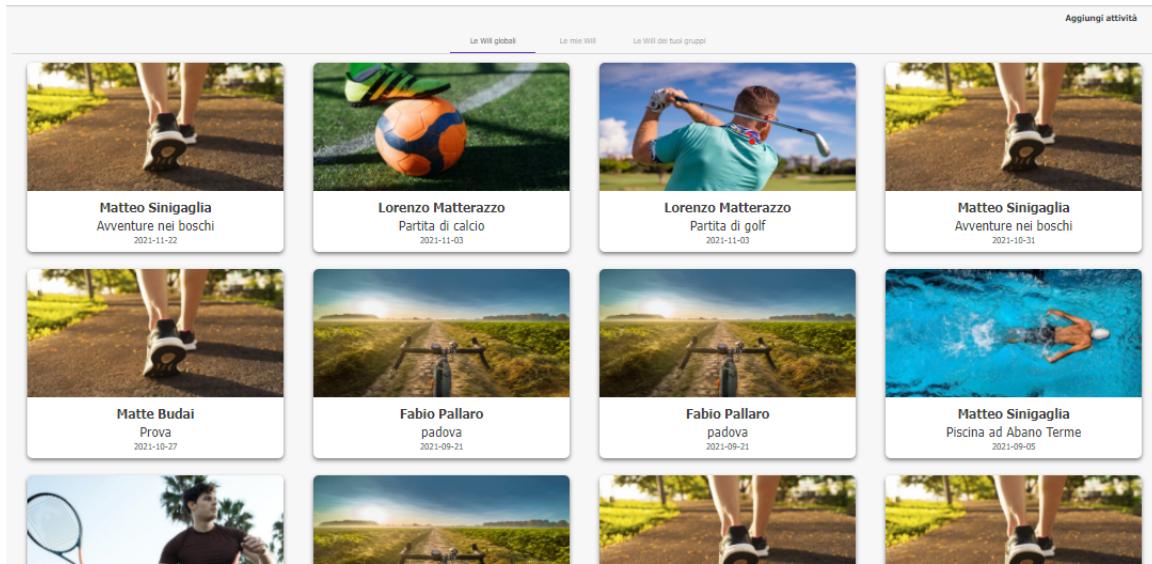


Figura 4.15: Componente Homepage di un utente autenticato

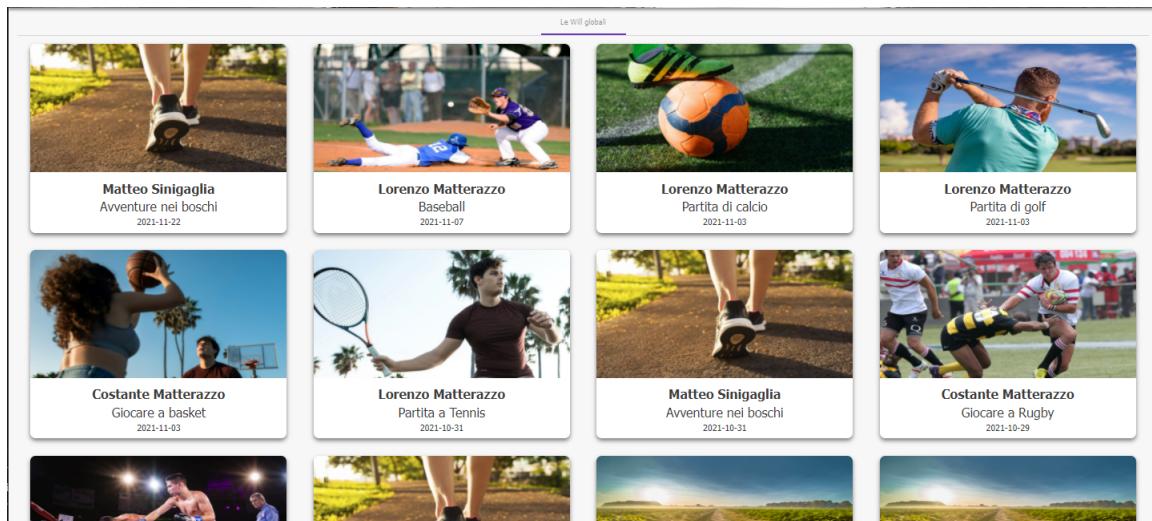


Figura 4.16: Componente Homepage di un utente non autenticato

Metodi

- ***ngOnInit***: all'inizializzazione del componente viene controllato, mediante il servizio `AuthenticationService`(non implementato da me), se l'utente è autenticato, in modo da valutare se visualizzare anche le schermate “le mie Will” e “le Will dei tuoi gruppi” o meno;
- ***getDataWithGlobalVisibility***: ottiene le `will` con visibilità globale;
- ***getDataPersonal***: ottiene le `will` create dall'utente;

- *getDataCrew*: ottiene le *will* dei gruppi a cui appartiene l'utente;
- *orderCards*: ordina le *will* in base alla data d'uscita.

WillCard

Questo componente, implementato da un collega, permette di visualizzare l'anteprima di una *will*, in particolare:

- Il nome dell'organizzatore;
- lo sport relativo all'uscita;
- la data d'uscita.



Figura 4.17: Componente WillCard

CrewCard

Questo componente permette di visualizzare l'anteprima di un gruppo, in particolare:

- il nome del gruppo;
- la descrizione.



Figura 4.18: Componente CrewCard

Input

- *crew*: gruppo da visualizzare.

Metodi

- ***navigate***: permette di navigare alla pagina di visualizzazione dei dettagli del gruppo. Se il gruppo cliccato è stato creato dall’utente allora viene visualizzato il *component* **CrewEditable**, altrimenti **CrewDetail**.

Crewpage

Questo componente è composto da una lista di **CrewCard**, e permette di visualizzare le maschere dei **gruppi dell’utente** e **gruppi a cui è possibile partecipare**.



Figura 4.19: Componente **Crewpage** che visualizza i gruppi in cui partecipare

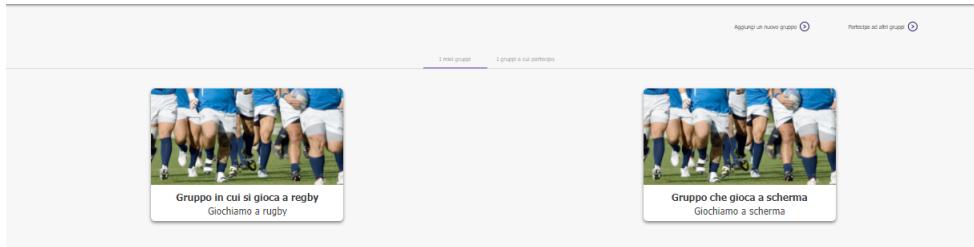


Figura 4.20: Componente **Crewpage** che visualizza la pagina di gestione dei gruppi dell’utente

Metodi

- ***ngOnInit***: all’inizializzazione del componente viene valutato se visualizzare la schermata **Gruppi dell’utente** o **Gruppi**;
- ***getGlobalCrew***: ottiene i gruppi a cui non partecipa un utente;
- ***getUserCrew***: ottiene i gruppi create e a cui partecipa un utente;
- ***navigateToNewCrew***: permette di navigare alla pagina di **creazione di un nuovo gruppo**;
- ***navigateToGlobalCrew***: permette di navigare alla pagina di visualizzazione dei **gruppi**.

CrewDetail

Questo componente permette di visualizzare i dettagli di un gruppo.

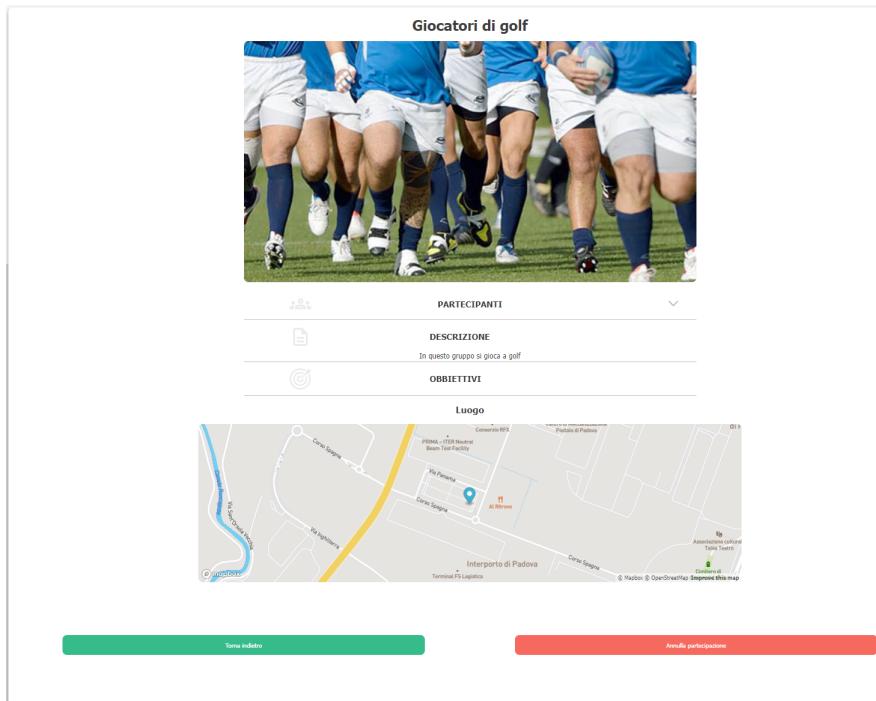


Figura 4.21: Componente che permette di visualizzare i dettagli di un gruppo

Componenti

- **MapComponent**

Metodi

- **getParticipants**: ottiene gli utenti che partecipano al gruppo;
- **addParticipant**: aggiunge l'utente ad un gruppo;
- **removeParticipant**: rimuove l'utente da un gruppo;
- **getCrew**: ottiene i dettagli di un gruppo.

CrewEditable

Questo componente permette di modificare i dettagli di un gruppo.

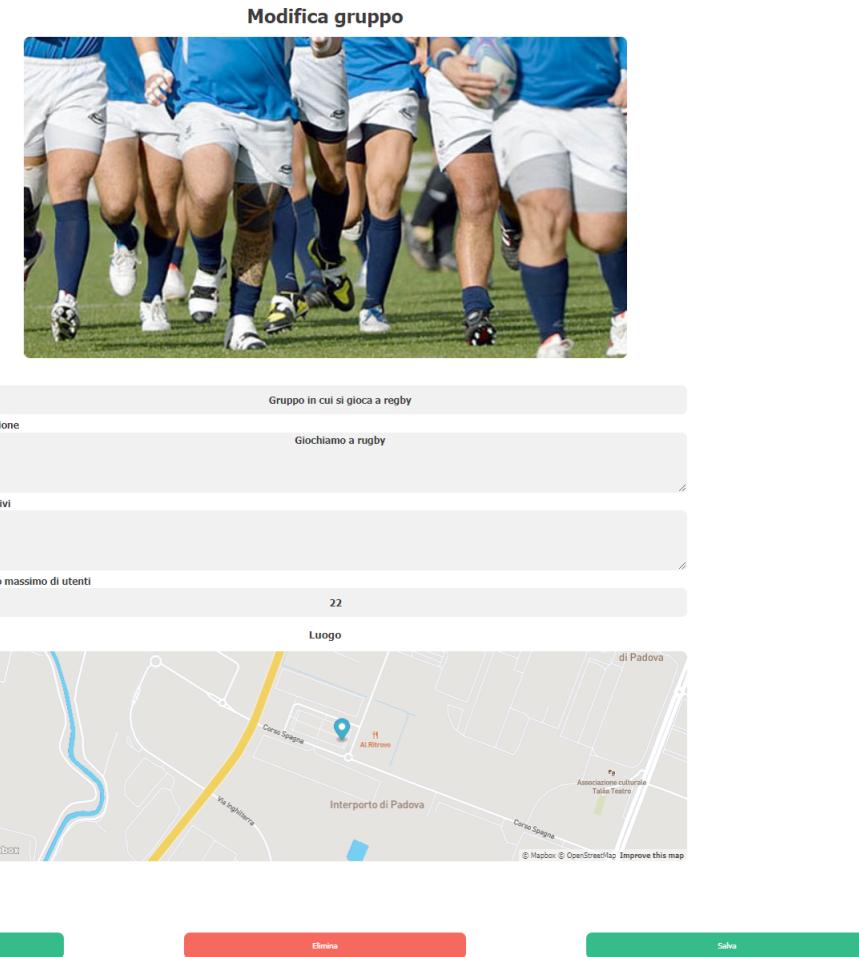


Figura 4.22: Componente CrewEditable

Componenti

- [MapComponent](#)
- [DialogComponent](#)

Metodi

- ***getCrew***: ottiene i dettagli di un gruppo;
- ***deleteCrew***: elimina il gruppo in questione. È richiesta la conferma dell'operazione attraverso il [dialog](#) per confermare l'eliminazione;
- ***saveCrew***: salva le modifiche effettuate ai dettagli del gruppo. È richiesta la conferma dell'operazione attraverso il [dialog](#) per confermare la modifica;

DialogComponent

Questo componente serve per confermare le operazioni effettuate delle modifiche all'applicativo.



Figura 4.23: Componente DialogComponent

Input

- *message*: testo del messaggio da visualizzare.

Output

- *clicked*: segnale emesso quando viene cliccata un'opzione. Il segnale emesso è di tipo *booleano* ed indica se l'azione è stata confermata o meno.

Metodi

- *buttonClicked*: emette l'evento *clicked*.

MapComponent

Questo componente permette di visualizzare una mappa in cui viene segnato con un *marker* nelle coordinate specifiche nei campi *latitudine* e *longitudine*.



Figura 4.24: Componente MapComponent

Input

- *latitudine*;
- *longitudine*;
- *editable*: specifica se è possibile modificare il *marker* nella mappa.

Output

- ***latitudineChanged***: segnale emesso quando viene spostato il *marker*. Emette il nuovo valore per la latitudine;
- ***longitudineChanged***: segnale emesso quando viene spostato il *marker*. Emette il nuovo valore per la longitudine.

Metodi

- ***ngOnInit***: all'inizializzazione del componente viene valutato se la mappa sia modificabile o no; nel caso lo fosse viene effettuato un *binding* fra il *click* sulla mappa e la funzione `modify_marker`;
- ***modify_marker***: modifica le coordinate da visualizzare sulla mappa in caso questa venga modificata. Emette inoltre i segnali `latitudineChanged` e `longitudineChanged`.

4.4.4 Servizi

GruppiService

Servizio che si occupa di effettuare chiamate HTTP al *back end* per richiedere, gestire e modificare i gruppi.

Metodi

- ***getCrews***: ottiene la lista di tutti i gruppi;
- ***addCrew***: inserisce un nuovo gruppo alla lista dei gruppi;
- ***getInfoCrew***: ottiene le informazioni di un gruppo specifico;
- ***modifyCrew***: modifica i dettagli di un gruppo;
- ***getAllUsciteofCrew***: ottiene tutte le uscite di un gruppo;
- ***getUsersOfCrew***: ottiene gli utenti di un gruppo;
- ***deleteCrew***: elimina un gruppo;
- ***getCrewsOfUser***: ottiene tutti i gruppi a cui appartiene un utente;
- ***addUserToCrew***: aggiunge un utente ad un gruppo;
- ***removeUserFromCrew***: rimuove un utente da un gruppo.

Capitolo 5

Verifica e validazione

5.1 Accessibilità

5.1.1 Elementi di accessibilità

Per rendere la navigazione al sito accessibile da tutte le categorie di utenti sono stati fatti alcuni accorgimenti:

- sono presenti gli attributi *accesskey* con chiave uguale alla prima lettera della parola del *tag label* associato, in modo da migliorare l'accessibilità alle *form* da tastiera senza l'uso del *mouse*;
- tutte le immagini contengono i *tag alt*, che sono stati lasciati vuoti nel caso servissero solo per il *layout*;
- è presente una barra di navigazione che aiuta a navigare nel sito;
- i *form* contengono dei *tag label* per ogni *input*, tuttavia non sono stati aggiunti *tag optgroup* o *fieldset* in quanto sono utili nel caso di *form* molto grandi, ma essendo presenti solo *form* di piccole dimensioni è stato ritenuto non necessario;
- sono presenti degli aiuti contestuali che mostrano gli errori nel caso fossero presenti;
- è presente del testo nascosto utile agli utenti con disabilità visive, come il *link* con la funzione di saltare al contenuto, ovvero di permettere di non far leggere allo *screen reader*^[g] la barra di navigazione, passando direttamente al contenuto, ed è presente all'inizio della navigazione per segnalare che le scorciatoie da tastiera sono attive;
- sono stati evitati testi scorrevoli, lampeggianti, barrati ed in generale *font* troppo elaborati per agevolare la lettura.

5.2 Verifica del GruppiService

In parallelo con la fase di codifica del [microservizio GruppiService](#) è stata verificata la correttezza delle funzioni presenti nella classe `GruppiController`.

Sono stati effettuati dei test di unità utilizzando **JUnit** e **Mockito**, *framework* di *testing* per il linguaggio Java.

Per avere una reportistica dei test effettuati è stato utilizzato **JaCoCo**, strumento che fornisce informazioni sulla *lines coverage*, *branches coverage* e *cyclomatic complexity*. L'attività di verifica si è rivelata molto proficua, permettendomi di agevolare lo sviluppo in modo significativo, tanto da produrre un grande quantitativo di test, con un'elevata copertura di casistiche, garantendo così qualità del prodotto.

Come dimostra la *code coverage* (figura 5.1) è stata raggiunta la massima copertura del codice, senza che questo inficiasse sulle tempistiche programmate.

sportwill.Gruppi.controller

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes		
G GruppiController	100%	100%	100%	100%	0	40	0	18	0	1
Total	0 of 1.193	100%	0 of 44	100%	0	40	0	18	0	1

Figura 5.1: Code coverage della classe **GruppiController**

5.2.1 Test effettuati

Tabella 5.2: Test di unità

Codice	Descrizione	Esito
TU1	Il metodo <code>getAllGruppi</code> deve restituire tutti i gruppi presenti	S
TU2	Il metodo <code>getAllGruppi</code> deve restituire un <code>HttpStatus.NO_CONTENT</code> nel caso non siano presenti gruppi	S
TU3	il metodo <code>getGruppo</code> deve restituire il gruppo specificato nel caso sia presente	S
TU4	il metodo <code>getGruppo</code> deve restituire un <code>HttpStatus.NO_CONTENT</code> nel caso non sia presente il gruppo	S
TU5	Il metodo <code>getUtentiByGruppo</code> deve restituire gli utenti di un gruppo	S
TU6	il metodo <code>createGruppo</code> deve creare un gruppo	S
TU7	Il metodo <code>createGruppo</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista il proprietario del gruppo specificato	S
TU8	Il metodo <code>createGruppo</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso che le coordinate geografiche inserite non siano valide	S
TU9	Il metodo <code>modifyGruppo</code> deve modificare un gruppo	S
TU10	Il metodo <code>modifyGruppo</code> deve restituire un <code>HttpStatus.NOT_MODIFIED</code> nel caso il gruppo specificato non esiste	S
Codice	Descrizione	Esito

Tabella 5.2: Test di unità

Codice	Descrizione	Esito
TU11	Il metodo <code>modifyGruppo</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista il proprietario del gruppo specificato	S
TU12	Il metodo <code>deleteGruppo</code> deve eliminare un gruppo	S
TU13	Il metodo <code>deleteGruppo</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista il gruppo specificato	S
TU14	Il metodo <code>getUsciteByGruppo</code> deve restituire le uscite di un gruppo	S
TU15	Il metodo <code>getUsciteByGruppo</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista il gruppo specificato	S
TU16	Il metodo <code>getGruppiOfUtente</code> deve restituire i gruppi a cui partecipa un utente	S
TU17	Il metodo <code>addUtenteToGruppo</code> deve aggiungere un utente ad un gruppo	S
TU18	Il metodo <code>addUtenteToGruppo</code> non deve aggiungere un nuovo utente nel caso il gruppo abbia superato il numero massimo di utenti	S
TU19	Il metodo <code>addUtenteToGruppo</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista il gruppo specificato	S
TU20	Il metodo <code>removeUtenteFromGruppo</code> deve rimuovere un utente da un gruppo	S
TU21	Il metodo <code>removeUtenteFromGruppo</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista il gruppo specificato	S
TU22	Il metodo <code>deleteUtenteFromJointUtenteCrew</code> deve rimuovere un utente dalla tabella <code>joint_utenti_crew</code>	S
TU23	Il metodo <code>addUscitaToGruppoCrew</code> deve aggiungere un'uscita ad un gruppo	S
TU24	Il metodo <code>addUscitaToGruppoCrew</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista l'uscita specificata	S
TU25	Il metodo <code>addUscitaToGruppoCrew</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista il gruppo specificato	S
TU26	Il metodo <code>removeUscitaFromGruppo</code> deve rimuovere un'uscita dal gruppo specificato	S
TU27	Il metodo <code>removeUscitaFromGruppo</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista l'uscita specificata	S
Codice	Descrizione	Esito

Tabella 5.2: Test di unità

Codice	Descrizione	Esito
TU28	Il metodo <code>removeUscitaFromGruppo</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista il gruppo specificato	S
TU29	Il metodo <code>deleteUscitaFromJointUscitaCrew</code> deve rimuovere un'uscita dal gruppo specificato	S
TU30	Il metodo <code>deleteUscitaFromJointUscitaCrew</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista l'uscita specificata	S
TU31	Il metodo <code>deleteUscitaFromJointUscitaCrew</code> deve restituire un <code>HttpStatus.BAD_REQUEST</code> nel caso non esista il gruppo specificato	S
Codice	Descrizione	Esito

Capitolo 6

Conclusioni

6.1 Raggiungimento degli obiettivi

Sono stati raggiunti gli obiettivi fissati dello *stage*: infatti sono stati raggiunti con un anticipo tale che mi ha permesso di realizzare anche delle attività che non erano inizialmente pianificate.

Tabella 6.2: Tabella raggiungimento degli obiettivi

Codice	Descrizione	Esito
O01	Acquisizione competenze sulle tematiche sopra descritte	Soddisfatto
O02	Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma	Soddisfatto
O03	Portare a termine l'implementazione dei <i>microservizi</i> richiesti con una percentuale di superamento pari al 80	Soddisfatto
D01	Portare a termine l'implementazione dei <i>microservizi</i> richiesti con una percentuale di superamento pari al 100	Soddisfatto
D02	Utilizzo della <i>containerizzazione</i> per portare tutti i <i>microservizi</i> su Docker	Soddisfatto
Codice	Descrizione	Esito

6.2 Conoscenze acquisite

Nel corso dello *stage* ho appreso nuove tecnologie per la realizzazione della parte sia *back end* sia *front end*. Tra le prime spiccano l'utilizzo di Spring Boot e Spring Data JPA, oltre che i linguaggio Java. Ho, infatti, avuto modo di conoscere uno dei *framework* più importanti e vasti per quanto concerne la programmazione in Java. La conoscenza del linguaggio Java è sicuramente uno strumento molto utile per la gestione della persistenza dei dati in un *database* relazionale. Combinando queste

conoscenze con il *framework* Spring è stato possibile creare applicazioni *Restful* in maniera semplice e veloce.

Per quanto riguarda il lato *front end*, Angular utilizzato assieme al linguaggio Typescript mi ha permesso di sviluppare, attraverso con le sue numerose funzionalità tra cui *templating*, *two-way binding*, modularizzazione, gestione *API Restful* e *dependency injection*, in maniera efficiente ed efficace.

6.3 Valutazione personale

Le aspettative sia dell'azienda che mie sono state ampiamente superate. Per questo motivo, ritengo i risultati ottenuti nel corso dello *stage* sicuramente un successo, anche in virtù delle capacità acquisite. Quanto prodotto ha migliorato il *software* già in possesso all'azienda, e può essere un punto di partenza per l'implementazione di nuove funzionalità. Basterà infatti prendere spunto da quanto integrato nel corso dello *stage* per avere un'idea chiara su come implementare nuove funzionalità.

L'utilizzo di due dei *framework* più importanti e popolari per lo sviluppo di applicazioni Java e per applicazioni *web* dinamiche concorrono a consolidare le mie conoscenze in Java, ed aggiungere nel mio bagaglio di conoscenze un linguaggio a me nuovo in TypeScript.

Infine, considero l'attività di *stage* molto utile per capire come funziona un'azienda e come ci si relaziona con i colleghi. Questo vale, in particolare, per Sync Lab S.r.l.: infatti, mi sono relazionato con lavoratori (e colleghi) che si occupano di sviluppo di applicazioni usufruendo dei *framework* Spring e Angular.

Quanto vissuto in queste 8 settimane rappresenterà un bagaglio di esperienze prezioso per il mio futuro lavorativo e non.

Glossario

Single Responsibility Principle Nella programmazione orientata agli oggetti, il principio di singola responsabilità afferma che ogni elemento di un programma deve avere una sola responsabilità, e che tale responsabilità debba essere interamente incapsulata dall'elemento stesso. [36](#)

containerizzazione Approccio allo sviluppo del *software* in cui un'applicazione o un servizio, le relative dipendenze e la corrispondente configurazione (astratti come file manifesto della distribuzione) sono inclusi in uno stesso pacchetto sotto forma di immagine del **container**. L'applicazione inclusa nel **container** può essere testata come unità e distribuita come istanza dell'immagine del **container** al sistema operativo *host*. [iii, 7, 63](#)

endpoint Canale da cui le **API** possono accedere alle risorse di cui hanno bisogno per eseguire la loro funzione. [29, 30](#)

framework Piattaforma che funge da strato intermedio tra un sistema operativo e il *software* che lo utilizza. [iii, 2, 7, 27, 28, 34, 60, 63, 64](#)

screen reader *Software* che identifica ed interpreta il testo mostrato sullo schermo di un *computer*, presentandolo tramite sintesi vocale o attraverso un *display braille*. Sono utilizzati prevalentemente da persone con problemi di vista. [59](#)

superset Un linguaggio di programmazione che contiene tutte le funzionalità di un determinato linguaggi, però ampliandolo o migliorandolo per includere anche altre funzionalità. [27](#)

will Intenzione di un utente di fare sport. [iii, 2, 3, 22, 23, 25, 26, 46, 50–52](#)

API Insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. [67](#)

container Formato di creazione dei pacchetti che racchiude tutto il codice e le dipendenze di un'applicazione in un formato *standard* che ne consente l'esecuzione rapida e affidabile in tutti gli ambienti di elaborazione.

Ogni container Docker ha il proprio *file system*, il proprio *stack* di rete (quindi il proprio indirizzo **IP**), il proprio spazio di elaborazione e limitazioni di risorse definite per CPU e memoria. [45, 65](#)

CRUD Operazioni di base che possono essere svolte su un *database*. In particolare, sono creazione (Create), lettura (Read), modifica/aggiornamento (Update) ed eliminazione (Delete). [67](#)

DAO *Pattern* che consente di isolare il *business layer* dal *persistence layer* (di solito un database relazionale, ma potrebbe essere qualsiasi altro meccanismo di persistenza). [67](#)

EntityManager API che gestisce il ciclo di vita delle istanze di entità in un *database*. [40](#)

Eureka Server Applicazione che contiene le informazioni su tutte le applicazioni *client-service*. Ogni **microservizio** si registra nel *server* Eureka e il *server* Eureka conosce tutte le applicazioni in esecuzione su ciascuna porta e indirizzo **IP**. [iii, ix, 34, 45](#)

ICT Insieme dei metodi e delle tecniche utilizzate nella trasmissione, ricezione ed elaborazione di dati e informazioni. [67](#)

IP Codice numerico usato da tutti i dispositivi (*computer, server web, stampanti, modem*) per navigare in Internet e per comunicare in una rete locale. Un indirizzo IP costituisce quindi la base per una trasmissione corretta delle informazioni dal mittente al ricevente. [67](#)

JPA *Framework* che offre delle **API** per aiutare gli sviluppatori nelle operazioni di persistenza dei dati su un *database* relazionale. [67](#)

JSP Documento di testo, scritto con una specifica sintassi, che rappresenta una pagina *web* di contenuto parzialmente o totalmente dinamico. Elaborando la pagina JSP, il motore JSP produce dinamicamente la pagina HTML finale che verrà rappresentata nel *browser* dell'utente. [67](#)

microservizio Approccio per sviluppare e organizzare l'architettura dei *software* secondo cui quest'ultimi sono composti di servizi indipendenti di piccole dimensioni che comunicano tra loro tramite **API** ben definite. [iii, 2, 6–8, 30, 33, 34, 39, 40, 42, 44, 45, 59, 63, 65](#)

SPA Una *single-page application* (SPA) è un'applicazione *web* o un sito *web* che interagisce con l'utente riscrivendo dinamicamente la pagina *web* corrente con nuovi dati dal *server*, invece del metodo predefinito di un *browser web* che carica intere nuove pagine. [67](#)

transcompilazione Tipo di traduzione che prende come *input* il codice sorgente di un programma scritto in un linguaggio di programmazione e produce un codice sorgente equivalente nello stesso o in un linguaggio di programmazione diverso. [27](#)

Trello *Web app* che serve per gestire il/i team, gestire *task*, programmare *macro* e *micro* attività, gestire l'esecuzione dei progetti, controllarne l'andamento e far sì che le varie azioni sui progetti o sul cliente, si intreccino in modo fluido. [6](#)

UML Linguaggio di modellazione basato sul paradigma *object-oriented*. L'*UML* svolge un'importantissima funzione di “lingua franca” nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. [67](#)

Acronimi

API Application Program Interface. [iii](#), [3](#), [33](#), [34](#), [38](#), [64–66](#)

CRUD Create Read Update Delete. [iii](#), [28](#), [40](#)

DAO Data Access Object. [39](#)

ICT Information and Communications Tecnology. [1](#)

IP Internet Protocol address. [30](#), [34](#), [65](#)

JPA Java Persistence API. [40](#)

JSP JavaServer Pages. [30](#)

SPA Single-Page Application. [28](#)

UML Unified Modeling Language. [9](#)

URL Uniform Resource Locator. [36](#)

Bibliografia

Siti web consultati

- [1] *Cos'è Angular?* URL: <https://psicografici.com/angular-js/#:~:text=Angular%20%C3%A8%20un%20framework%20JavaScript> (cit. a p. 28).
- [2] *Cosa sono i microservizi?* URL: <https://aws.amazon.com/it/microservices/> (cit. a p. 33).
- [3] *Dependency injection in Angular.* URL: <https://angular.io/guide/dependency-injection> (cit. a p. 35).
- [4] *Feature modules.* URL: <https://angular.io/guide/feature-modules> (cit. a p. 36).
- [5] *Introduction to Angular concepts.* URL: <https://angular.io/guide/architecture> (cit. a p. 30).
- [6] *La Dependency Injection in Spring.* URL: <https://daviooh.com/blog/2017/08/19/spring-dependency-injection> (cit. a p. 35).
- [7] *Lazy-loading feature modules.* URL: <https://angular.io/guide/lazy-loading-ngmodules> (cit. a p. 36).
- [8] *Manifesto Agile.* URL: <http://agilemanifesto.org/iso/it/>.
- [9] *Microservizi in Java.* URL: <https://codingjam.it/microservizi-in-java-con-spring-boot-e-spring-cloud/> (cit. a p. 30).
- [10] *Node.js.* URL: <https://whatism.techttarget.com/definition/Nodejs#:~:text=James%20Denman-,Node.,feeds%20and%20web%20push%20notifications> (cit. a p. 28).
- [11] *Pattern: API Gateway.* URL: <https://microservices.io/patterns/apigateway.html> (cit. a p. 30).
- [12] *Pattern: Service registry.* URL: <https://microservices.io/patterns/service-registry.html> (cit. a p. 34).
- [13] *Singleton.* URL: <https://refactoring.guru/design-patterns/singleton> (cit. a p. 36).
- [14] *Spring Boot Microservices — Developing Service Discovery.* URL: <https://aws.amazon.com/it/microservices/> (cit. a p. 34).

- [15] *Spring Framework*. URL: https://it.wikipedia.org/wiki/Spring_Framework (cit. a p. 27).
- [16] *TypeScript*. URL: <https://en.wikipedia.org/wiki/TypeScript> (cit. a p. 27).
- [17] *Using observables to pass values*. URL: <https://angular.io/guide/observables> (cit. a p. 37).
- [18] *What is Java*. URL: https://java.com/en/download/help/whatis_java.html (cit. a p. 27).