

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



**Integrazione di nuove funzionalità ad un  
applicativo web utilizzando Spring e  
Angular**

*Tesi di laurea triennale*

*Relatore*

Prof. Paolo Baldan

*Laureando*

Lorenzo Matterazzo

---

ANNO ACCADEMICO 2020-2021

*Integrazione di nuove funzionalità ad un applicativo web utilizzando 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*<sup>[g]</sup>) di effettuare un'attività sportiva. La piattaforma fa 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<sup>\[g\]</sup>](#) mediante il [framework<sup>\[g\]</sup> Spring<sup>\[g\]</sup>](#) 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 [??](#), 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<sup>\[g\]</sup>](#) 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 affinchè 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<sup>\[g\]</sup>](#) di tutti i microservizi su Docker.

- La seconda attività è stata la modifica del *front end*, mediante il framework [Angular<sup>\[g\]</sup>](#), 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 che mi ha permesso di implementare anche la parte *front end* dell'applicazione.



*“Life is really simple, but we insist on making it complicated”*

— Confucius

# 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.*

*Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.*

*Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.*

*Padova, Dicembre 2021*

Lorenzo Matterazzo



# Indice

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduzione</b>  | <b>1</b>  |
| 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      | Accessibilità . . . . .                                    | 4         |
| 1.7      | Organizzazione del testo . . . . .                         | 4         |
| <b>2</b> | <b>Descrizione dello stage</b>                             | <b>5</b>  |
| 2.1      | Analisi preventiva dei rischi . . . . .                    | 5         |
| 2.2      | Requisiti e obiettivi . . . . .                            | 6         |
| 2.3      | Pianificazione del lavoro . . . . .                        | 7         |
| <b>3</b> | <b>Analisi dei requisiti</b>                               | <b>9</b>  |
| 3.1      | Casi d'uso . . . . .                                       | 9         |
| 3.2      | Tracciamento dei requisiti . . . . .                       | 23        |
| <b>4</b> | <b>Progettazione e codifica</b>                            | <b>25</b> |
| 4.1      | Tecnologie . . . . .                                       | 25        |
| 4.2      | Progettazione . . . . .                                    | 26        |
| 4.2.1    | Back end . . . . .   | 26        |
| 4.2.2    | Front end . . . . .  | 28        |
| 4.3      | Design Pattern utilizzati . . . . .                        | 31        |
| 4.3.1    | Microservizi . . . . .                                     | 31        |
| 4.3.2    | API Gateway . . . . .                                      | 32        |
| 4.3.3    | Client-Side Service Discovery e Service Registry . . . . . | 33        |
| 4.3.4    | Dependency Injection . . . . .                             | 33        |
| 4.3.5    | Singleton . . . . .  | 35        |
| 4.3.6    | Feature Service . . . . .                                  | 35        |
| 4.3.7    | Lazy Loading . . . . .                                     | 35        |
| 4.3.8    | Observer . . . . .   | 36        |
| 4.4      | Codifica . . . . .   | 38        |

|          |  |           |
|----------|--|-----------|
| 4.4.1    | Back end . . . . .                       | 38        |
| 4.4.2    | Eureka Server . . . . .                  | 43        |
| 4.4.3    | Front end . . . . .                      | 45        |
| 4.4.4    | Servizi . . . . .                        | 49        |
| <b>5</b> | <b>Verifica e validazione</b>            | <b>51</b> |
| <b>6</b> | <b>Conclusioni</b>                       | <b>53</b> |
| 6.1      | Consuntivo finale . . . . .              | 53        |
| 6.2      | Raggiungimento degli obiettivi . . . . . | 53        |
| 6.3      | Conoscenze acquisite . . . . .           | 53        |
| 6.4      | Valutazione personale . . . . .          | 53        |
| <b>A</b> | <b>Appendice A</b>                       | <b>55</b> |
|          | <b>Bibliografia</b>                      | <b>59</b> |

# 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  | UC12: Vis. lista <i>will</i> degli utenti appartenenti agli stessi gruppi . . . . .  | 22 |
| 3.10 | UC12.1: Vis. singola <i>will</i> . . . . .   | 22 |
| 4.1  | Architettura a strati di Spring Boot . . . . .   | 27 |
| 4.2  | Spring Boot <i>workflow</i> . . . . .  | 27 |
| 4.3  | <i>Mockup</i> pagina per la visualizzazione dei gruppi creati dall'utente . . . . .  | 29 |
| 4.4  | <i>Mockup</i> pagina per la visualizzazione dei gruppi a cui partecipa l'utente . . . . .  | 30 |
| 4.5  | <i>Mockup</i> pagina per la visualizzazione dei dettagli di un gruppo . . . . .  | 30 |
| 4.6  | <i>Mockup</i> pagina per la modifica dei dettagli di un gruppo . . . . .   | 31 |
| 4.7  | Diagramma concettuale che descrive l' <i>API Gateway</i> . . . . .   | 32 |
| 4.8  | Diagramma concettuale che descrive la registrazione dei microservizi all' <i>Eureka Server</i> . . . . .                           | 33 |
| 4.9  | Esempio di <i>Dependency Injection</i> con Spring . . . . .  | 34 |
| 4.10 | Esempio di creazione di un servizio . . . . .  | 34 |
| 4.11 | Esempio di <i>constructor injection</i> . . . . .  | 35 |
| 4.12 | Implementazione del <i>pattern Lazy loading</i> nel <i>AppRoutingModule</i> . . . . .  | 36 |
| 4.13 | Esempio implementazione <i>pattern Observer</i> . . . . .  | 37 |
| 4.14 | Implementazione della classe <i>AuthFilter</i> . . . . .   | 42 |
| 4.15 | Configurazione dello <i>Spring Cloud Gateway</i> nel file <i>application.yml</i> del microservizio <i>API Gateway</i> . . . . .    | 43 |
| 4.16 | File <i>application.yml</i> del microservizio <i>Eureka Server</i> . . . . .   | 44 |
| 4.17 | Configurazione per la registrazione di un microservizio all' <i>Eureka Server</i> nel file <i>application.properties</i> . . . . . | 44 |
| 4.18 | Pagina di homepage . . . . .   | 45 |
| 4.19 | Pagina dei gruppi creati o a cui partecipa un utente . . . . .   | 46 |
| 4.20 | Pagina dei gruppi . . . . .  | 46 |
| 4.21 | Pagina di creazione di un gruppo . . . . .   | 47 |
| 4.22 | Pagina di modifica di un gruppo . . . . .  | 48 |

# Elenco delle tavelle

|     |  |    |
|-----|--|----|
| 3.1 | Tabella del tracciamento dei requisiti funzionali . . . . .  | 24 |
| 3.2 | Tabella del tracciamento dei requisiti qualitativi . . . . . | 24 |
| 3.3 | Tabella del tracciamento dei requisiti di vincolo . . . . .  | 24 |

# 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<sup>\[g\]</sup>](#);
- 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 clientidiretti 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 gli utenti che hanno intenzione di unirsi in questa attività vi possono partecipare.

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’integrale alla piattaforma già esistente la suddivisione delle visualizzazioni delle *will* solo agli utenti che appartengono agli stessi gruppi.

Gli obiettivi da raggiungere nel corso dello *stage* sono principalmente due:

- sviluppo di un **microservizio** utilizzando il **framework Spring Java** per la creazione dei gruppi;
- modifica dei **microservizi** esistenti affinchè 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**: permette di ampliare le funzionalità di Git integrate in Visual Studio Code;
- **Spring Boot Extension Pack**: raccolta di estensioni per lo sviluppo con Spring Boot Application;
- **W3C Web Validator**: permette di controllare la validità del *markup* dei documenti html e css;
- **Web Accessibility**: permette di verificare l’accessibilità dei documenti html, evidenziando gli elementi che si potrebbe prendere in considerazione di cambiare e dando suggerimenti su come potrebbe modificato;
- **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

Si tratta di uno 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. Grazie alla sua interfaccia chiara e facile da usare, DbVisualizer si è dimostrato uno strumento di database molto conveniente, che funziona su tutti i principali sistemi operativi e supporta 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 Accessibilità

Durante la realizzazione del sito è stata resa la navigazione più efficace attraverso l'uso di accesskey.

Le immagini sono tutte marcate con gli appositi *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.

Ogni *link* è stato reso distinguibile da ogni altro elemento tramite appositi CSS, *hover* e *visited*, in modo da aiutare l'utente ad orientarsi.

Inoltre, per evitare *link* circolari, nella barra di navigazione le pagine non contengono *link* che navigano alla pagina corrente.

I form contengono dei tag label per ogni input.

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 gli attributi *accesskey* con chiave uguale alla prima lettera della parola del *tag label* associato per migliorare l'accessibilità alle *form* da tastiera senza l'uso del *mouse*. Sono presenti degli aiuti contestuali che mostrano gli errori nel caso fossero presenti. Non sono stati aggiunti *tabindex* nei *form* dato che l'ordine di tabulazione è già corretto.

Sono stato evitati i tag ed attributi deprecati.

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* la barra di navigazione, passando direttamente al contenuto, ed è presente all'inizio della navigazione per segnalare che le scorciatoie da tastiera sono attive.

Inoltre per migliorare la navigazione dello *scroll*, viene mostrato un pulsante in basso a destra dello schermo che, se un utente lo clicca, viene effettuato uno *scroll* verso l'alto fino all'inizio della pagina.

Il pulsante è un *link* con testo nascosto e come immagine di *background* una freccia, in modo che lo *screen reader* riesca comunque a leggere il testo. Le parole in lingua straniera sono state racchiuse dentro un *tag* con l'attributo *lang*, in modo da permettere allo *screen reader* di leggere la parola correttamente.

## 1.7 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 che potevano venire riscontrati durante lo svolgimento dello 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 è possibile incominciare. 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 guastarsi, 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 id di ogni estensione utilizzata in Visual Studio Code;
- Il file `launch.json`, che viene usato per configurare il `debugger` in Visual Studio Code.

#### 3. Tempistiche

**Descrizione:** il tempo di apprendimento di nuove tecnologie potrebbe portare a

ritardi sulle scadenze previste. I ritardi verranno individuati nel caso in cui il lavoro da effettuare si scostasse dalla pianificazione presente su [Trello<sup>\[g\]</sup>](#).

**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   |
|-------------------|---|
| <b>160</b>        | <b>Formazione sulle tecnologie</b>  |
| 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   |
| <b>40</b>         | <b>Definizione architettura di riferimento e relativa documentazione</b>  |
| 14                | Analisi del problema e del dominio applicativo  |
| 22                | Progettazione della piattaforma e relativi test   |
| 4                 | Stesura documentazione relativa ad analisi e progettazione  |
| <b>80</b>         | <b>Implementazione del nuovo servizio</b>   |
| <b>40</b>         | <b>Collaudo Finale</b>  |
| 30                | Collaudo  |
| 6                 | Stesura documentazione finale   |
| 2                 | Incontro di presentazione della piattaforma con gli stakeholders  |
| 2                 | Live demo di tutto il lavoro di stage   |
| <b>Totale ore</b> | <b>320</b>  |

# Capitolo 3

# Analisi dei requisiti

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

## 3.1 Casi d'uso

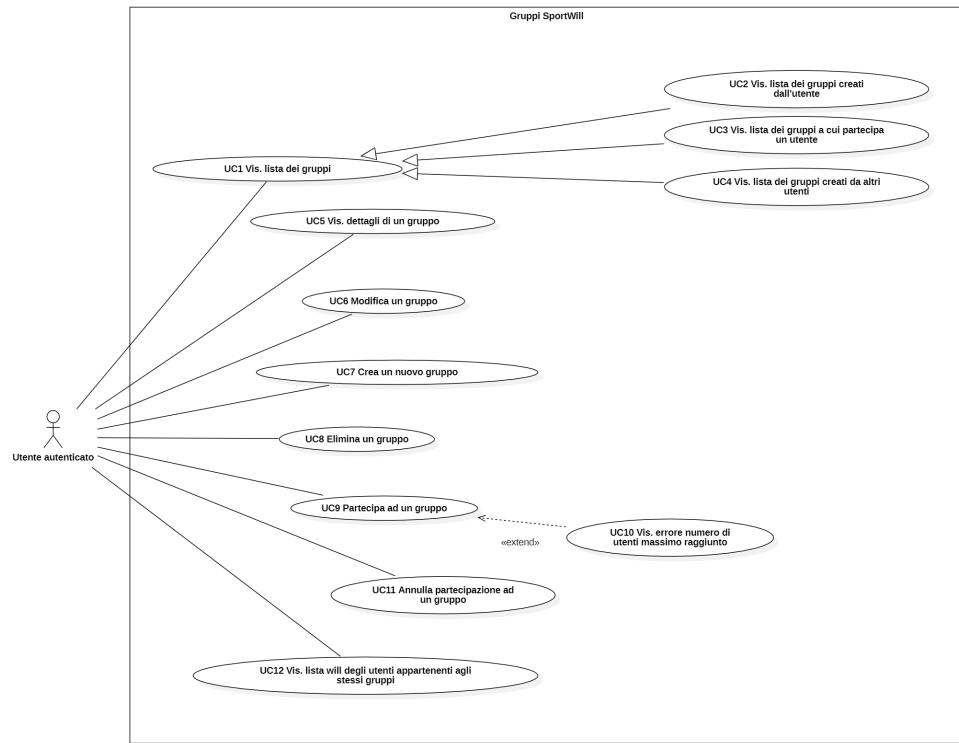
### Attori principali

Nell'analisi del progetto di *stage* è emerso solo un attore, ovvero l'**utente autenticato**, attore che indica un utente che ha effettuato l'autenticazione all'interno dell'applicazione web. Ha quindi la possibilità di vedere tutte le informazioni sui gruppi e sulle *will* che sarebbero altrimenti inaccessibili.

### Elenco dei casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. 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 che 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.

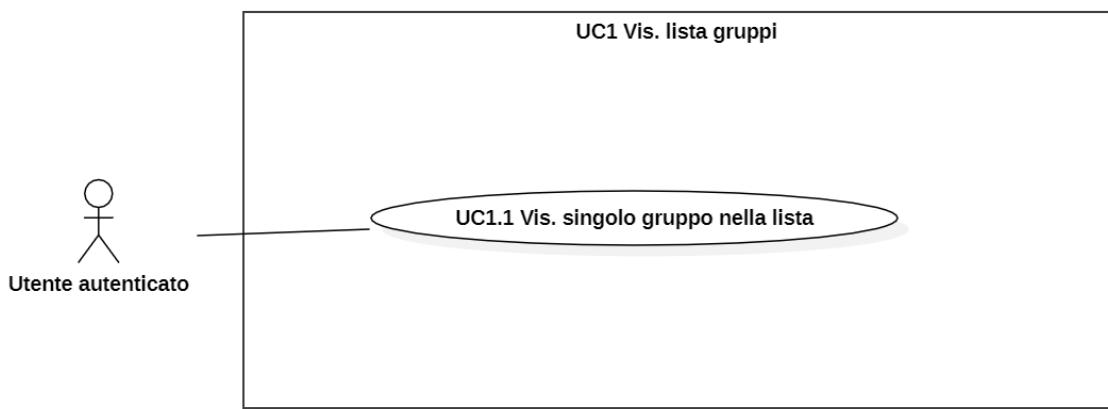
**Precondizioni:** L'utente ha aperto la *web app* ed è autenticato.

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

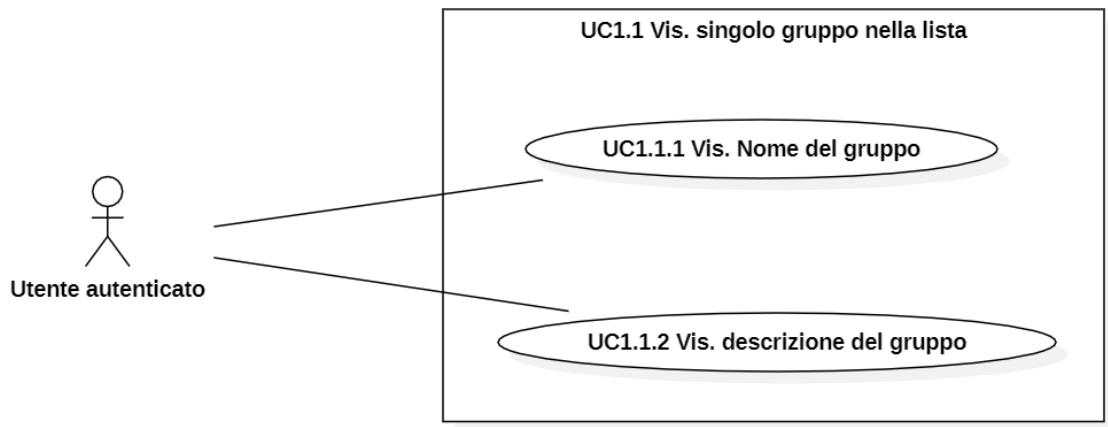
**Postcondizioni:** L'utente ha visualizzato la lista dei gruppi.

**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 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 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 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.

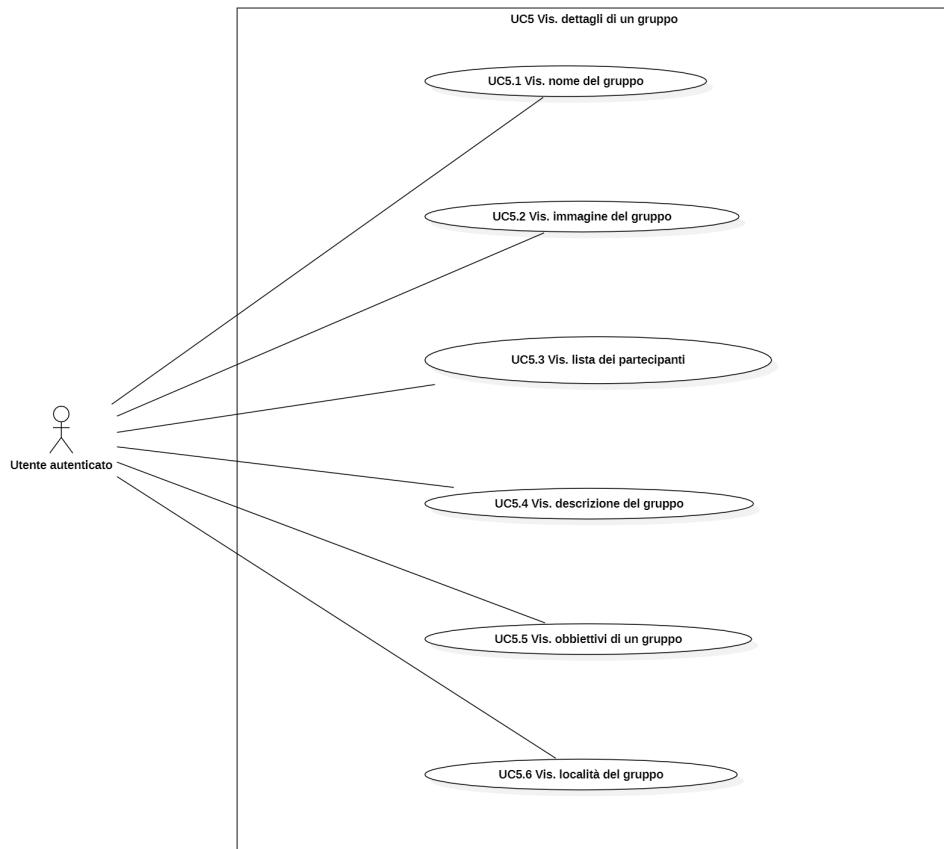
**Precondizioni:** L'utente ha aperto la *web app* ed è autenticato.

**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 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.

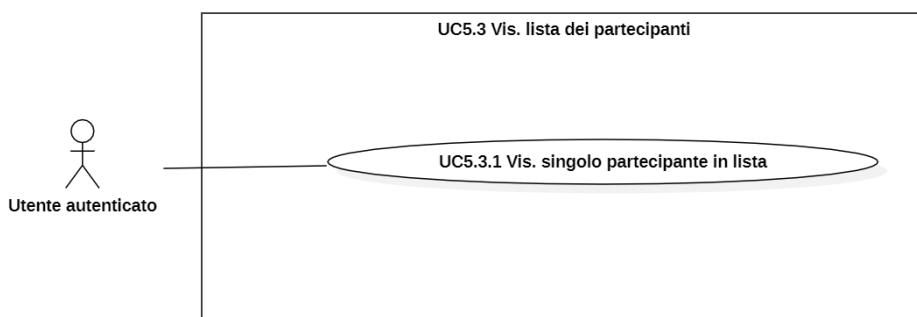
**Precondizioni:** L'utente ha aperto la *web app*, è autenticato e sta visualizzando i dettagli di un gruppo.

**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.

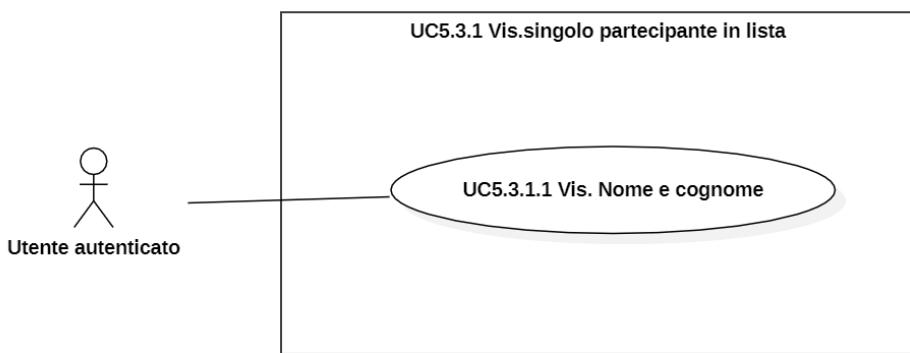
**Precondizioni:** L'utente ha aperto la *web app*, è autenticato e sta visualizzando la lista dei partecipanti ad un gruppo.

**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 ha 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.

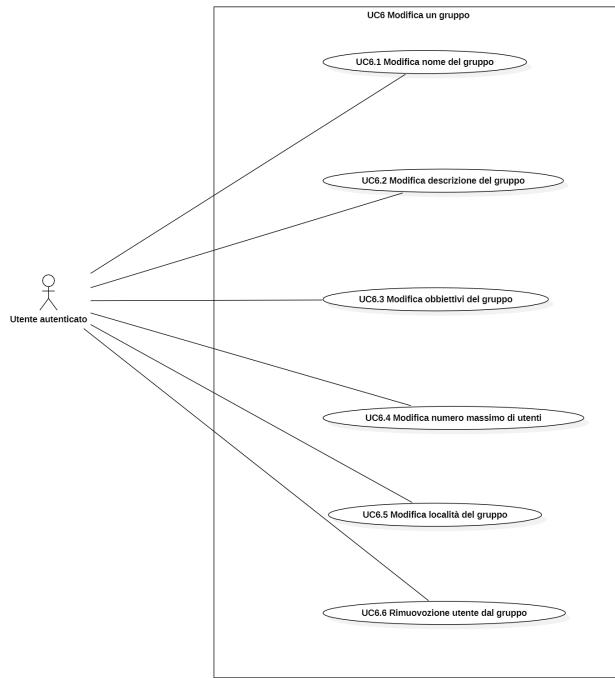
**Precondizioni:** L'utente ha aperto la *web app* ed è autenticato.

**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.

#### UC6.6: Rimozione utente

**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 rimuovere un utente dal gruppo.

**Postcondizioni:** L'utente ha rimosso un utente dal gruppo.

**Scenario principale:**

1. L'utente rimuove un utente dal gruppo.

#### UC7: Crea un nuovo gruppo

**Attori Principali:** Utente autenticato.

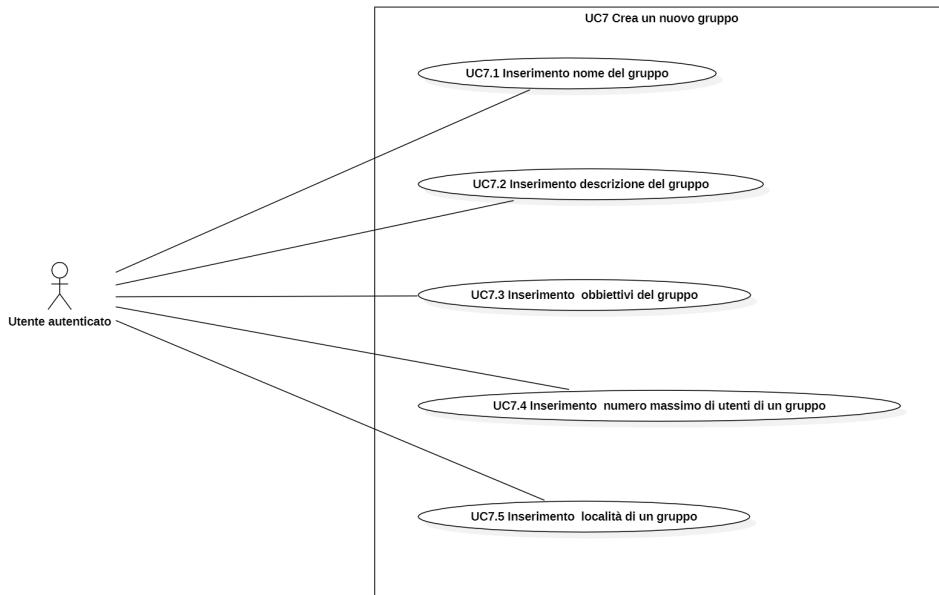
**Precondizioni:** L'utente ha aperto la *web app* ed è autenticato.

**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* ed è autenticato.

**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* ed è autenticato.

**Descrizione:** L'utente vuole partecipare ad un gruppo.

**Postcondizioni:** L'utente ha partecipato ad un gruppo.

**Scenario principale:**

1. L'utente partecipa ad un gruppo.

**Scenario Alternativo:** il gruppo ha raggiunto la capienza massima.

**Estensioni:** viene visualizzato un messaggio di errore a causa della capienza massima raggiunta dal gruppo.

### **UC10: Visualizzazione errore numero di utenti massimo raggiunto**

**Attori Principali:** Utente autenticato.

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

**Descrizione:** L'utente vuole partecipare ad un gruppo.

**Postcondizioni:** L'utente non si unisce al gruppo.

**Scenario principale:**

1. l'utente sceglie di partecipare ad un gruppo;
2. viene visualizzato un messaggio di errore a causa della capienza massima raggiunta dal gruppo.

### **UC11: Annulla partecipazione ad un gruppo**

**Attori Principali:** Utente autenticato.

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

**Descrizione:** L'utente vuole annullare la partecipazione ad un gruppo.

**Postcondizioni:** L'utente ha annullato la partecipazione ad un gruppo.

**Scenario principale:**

1. L'utente annulla la partecipazione ad un gruppo.

### **UC12: Visualizzazione lista delle will degli utenti appartenenti agli stessi gruppi**

**Attori Principali:** Utente autenticato.

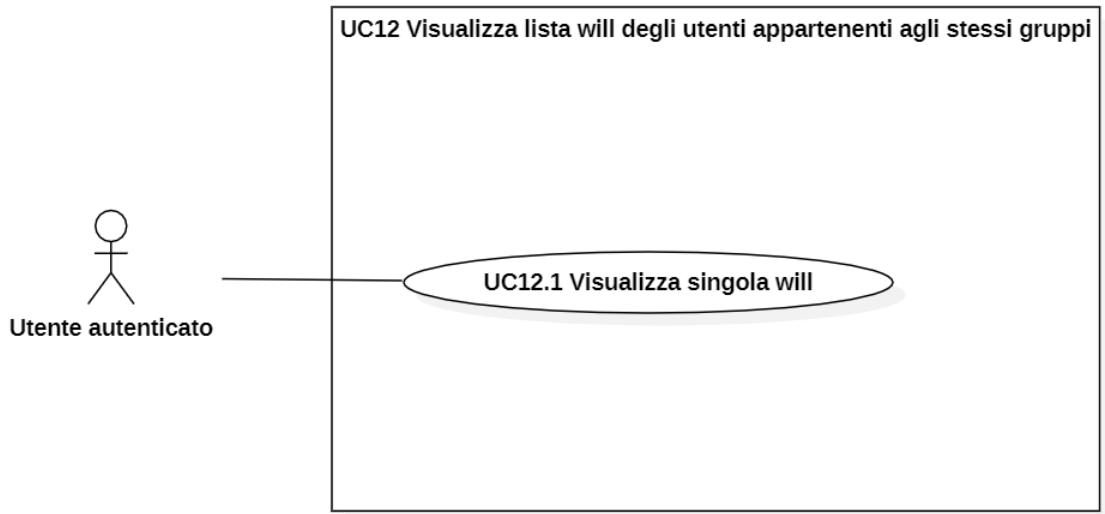
**Precondizioni:** L'utente ha aperto la *web app* ed è autenticato.

**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:** UC12: Vis. lista *will* degli utenti appartenenti agli stessi gruppi

### UC12.1: Visualizzazione singola will in lista

**Attori Principali:** Utente autenticato.

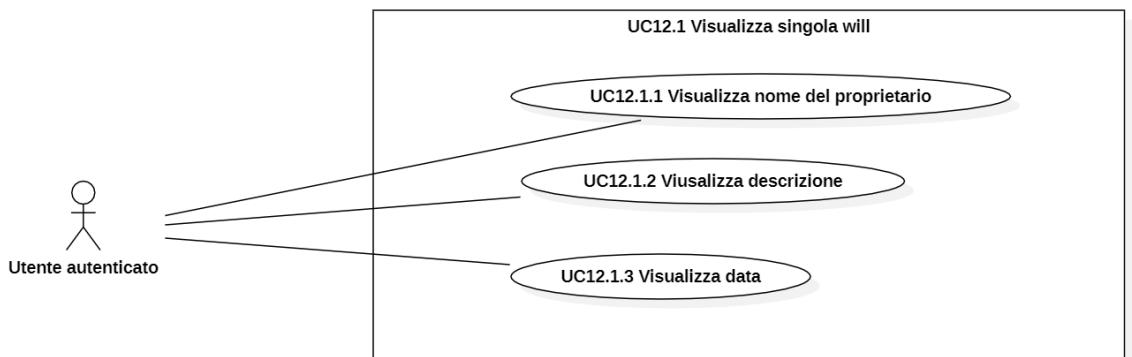
**Precondizioni:** L'utente ha aperto la *web app*, è autenticato e sta visualizzando la lista dei gruppi.

**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:** UC12.1: Vis. singola *will*

### UC12.1.1: 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*.

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

**Scenario principale:**

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

### UC12.1.2: 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*.

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

**Scenario principale:**

1. L'utente visualizza la descrizione della *will*.

### UC12.1.3: 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 distinguerli.

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 3.1, 3.2 e 3.3 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

**Tabella 3.1:** Tabella del tracciamento dei requisiti funzionali

| Requisito | Descrizione   | Use Case |
|-----------|---|----------|
| RFN-1     | L'interfaccia permette di configurare il tipo di sonde del test | UC1      |

**Tabella 3.2:** Tabella del tracciamento dei requisiti qualitativi

| Requisito | Descrizione  | Use Case |
|-----------|--|----------|
| RQD-1     | Le prestazioni del simulatore hardware deve garantire la giusta esecuzione dei test e non la generazione di falsi negativi | -        |

**Tabella 3.3:** Tabella del tracciamento dei requisiti di vincolo

| Requisito | Descrizione   | Use Case |
|-----------|---|----------|
| RVO-1     | La libreria per l'esecuzione dei test automatici deve essere riutilizzabile | -        |

# 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.

#### 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*.

#### TypeScript

TypeScript è un linguaggio di programmazione sviluppato e gestito da Microsoft. È un *superset*<sup>[g]</sup> di JavaScript, che permette di aggiungere la tipizzazione statica opzionale al linguaggio. TypeScript è progettato per lo sviluppo di applicazioni di grandi dimensioni e per la *transcompilazione*<sup>[g]</sup> in JavaScript. Poiché TypeScript è un *superset* di JavaScript, anche i programmi JavaScript esistenti sono validi programmi TypeScript.

## 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.

## 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.

## 4.2 Progettazione

### 4.2.1 Back end

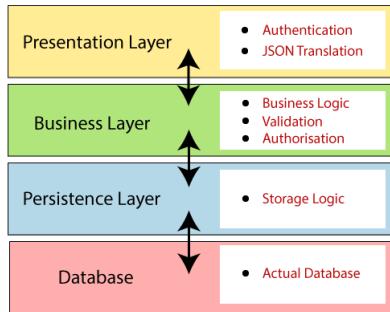
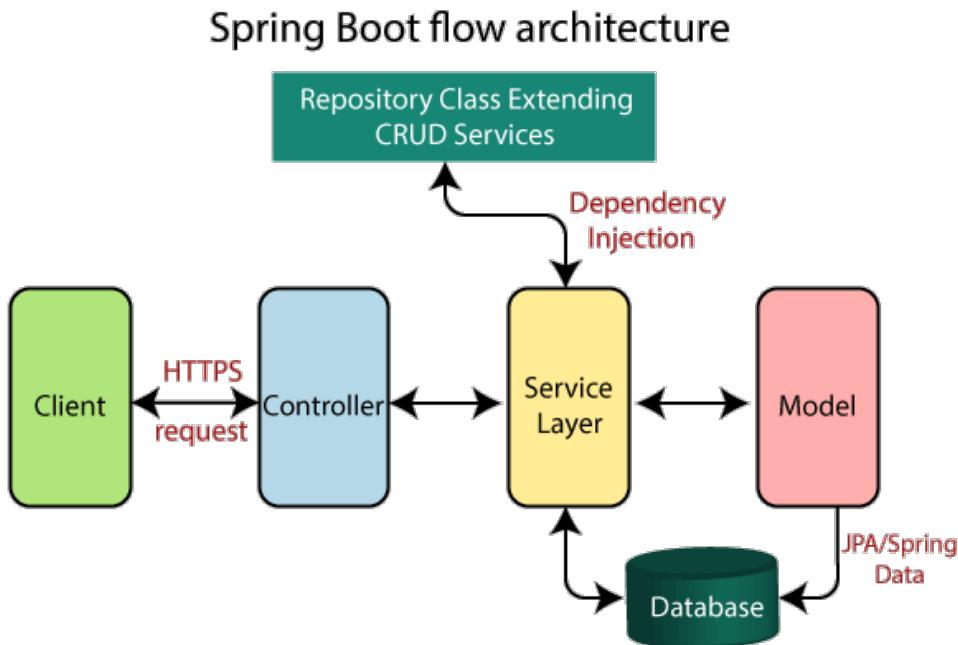
#### 4.2.1.1 Architettura Spring Boot

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*<sup>[g]</sup> 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.

#### 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.

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](#), permettendo una connessione più rapida rispetto all’utilizzo di connessioni diverse per ogni [microservizio](#), essendo tutti 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.

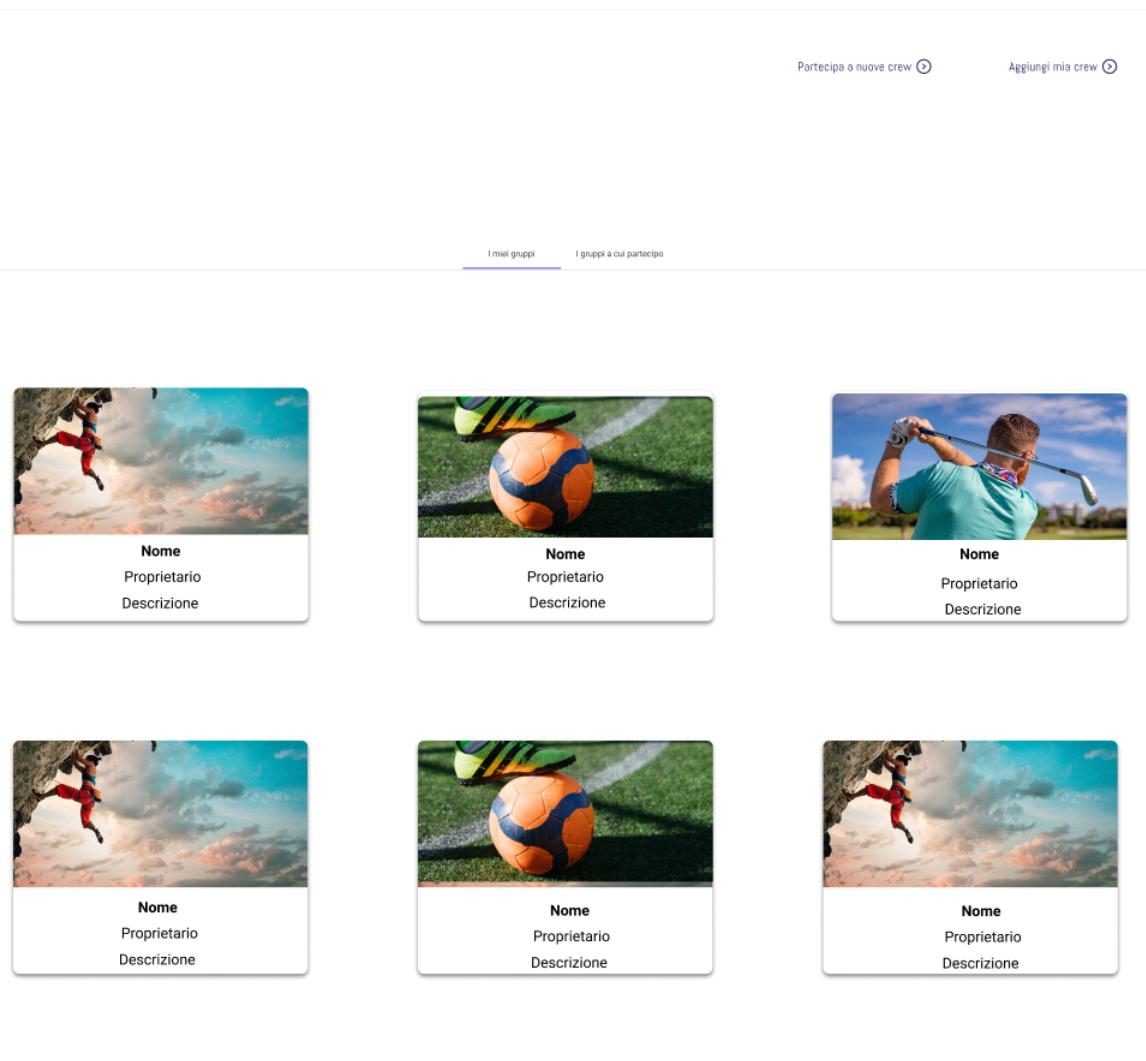
È possibile creare un *component* che rappresenta la pagina completa, in cui inserire diversi *component* per ogni elemento contenuto in quella pagina. Ogni *component* contenuto si occuperà così di gestire la grafica di quella determinata funzionalità e di comunicare con i servizi di cui necessita;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

### 4.2.2.2 Progettazione delle maschere

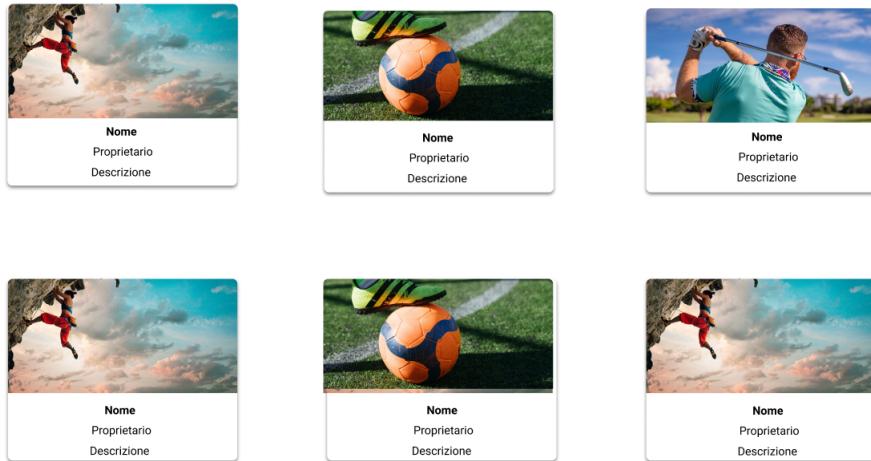
Con il *tutor* aziendale è stato discusso come dovrebbe essere la grafica delle nuove maschere da aggiungere a **SportWill**. Da queste discussioni è stato poi utilizzato **Figma** per fare il *mockup*, in modo da testare come i vari elementi visivi lavorano insieme.

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 sono riportati in seguito.



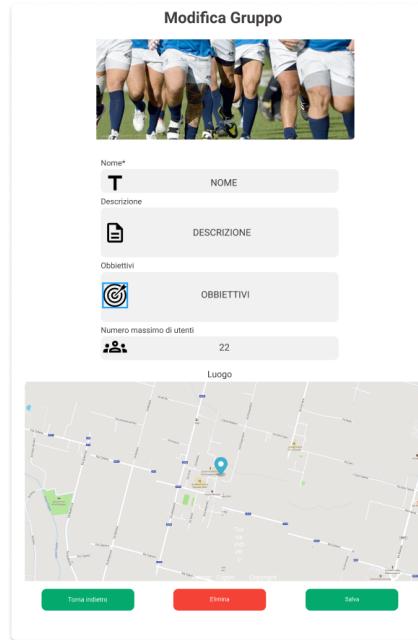
**Figura 4.3:** Mockup pagina per la visualizzazione dei gruppi creati dall'utente



**Figura 4.4:** Mockup pagina per la visualizzazione dei gruppi a cui partecipa l'utente

The image shows a detailed view of a group page. At the top, it says "Nome crew". Below that is a large image of several people with their hands raised against a sunset background. Underneath the image are four input fields: "PARTECIPANTI" (with a user icon), "DESCRIZIONE" (with a text icon), "OBBIETTIVI" (with a target icon), and "LUOGO" (with a location pin icon). Below these fields is a map showing a specific location marked with a blue pin. At the bottom are two buttons: a green "Torna indietro" button and a red "ANNULLA PARTECIPAZIONE" button.

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



**Figura 4.6:** 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

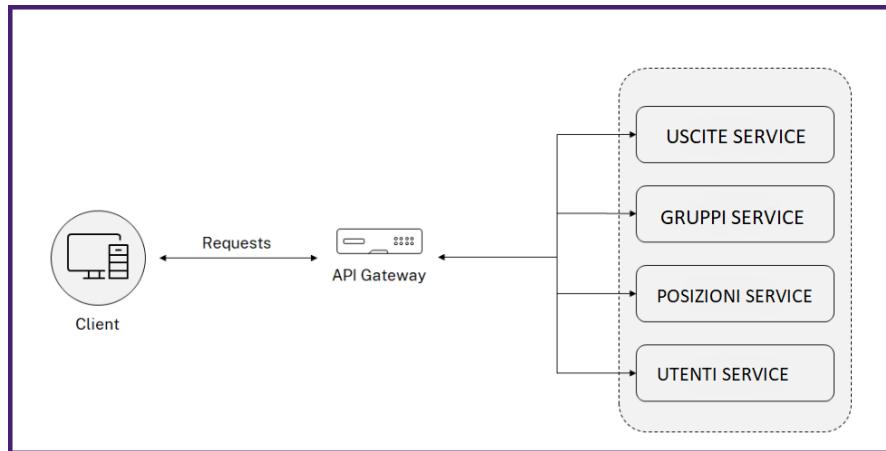
### 4.3.1 Microservizi

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**.

### 4.3.2 API Gateway



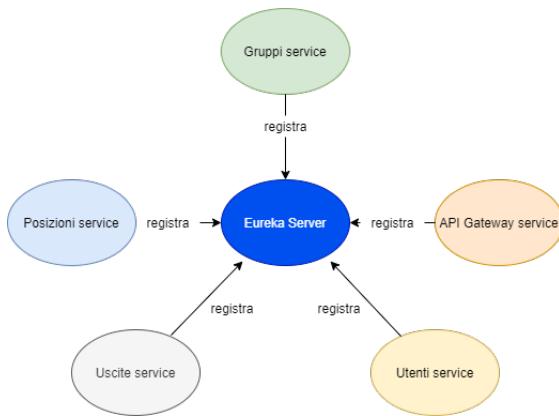
**Figura 4.7:** Diagramma concettuale che descrive l'*API Gateway*

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 4.8:** 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.

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*.

### 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`.



**Figura 4.9:** 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;



**Figura 4.10:** Esempio di creazione di un servizio

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



**Figura 4.11:** 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](#)<sup>[g]</sup>, 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.

### 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.

### 4.3.7 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](#).



```

const routes: Routes = [
  { path: 'homepage', component: HomepageComponent },
  { path: 'detail/:id', component: WillDetailComponent },
  { path: 'edit/:id', component: EditableFormComponent },
  { path: 'editcrew/:id', component: EditableCrewComponent },
  { path: 'detailcrew/:id', component: CrewDetailComponent },
  { path: 'signIn', component: SignInComponent },
  { path: 'myCrew/:type', component: MyCrewpageComponent },
  { path: 'signUp', component: SignUpComponent },
  { path: 'add', component: EditableFormComponent },
  { path: '', redirectTo: '/homepage', pathMatch: 'full' },
  { path: '404', component: NotFoundComponent },
  { path: '**', redirectTo: '404' },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

**Figura 4.12:** 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*.

```
export class MyCrewpageComponent implements OnInit {
  crewCards: Crew[] = []
  constructor(private gruppiService: GruppiService,
    private router: Router,
    private route: ActivatedRoute) {
  }
  getGlobalCrew() {
    this.gruppiService.getTeams().subscribe(teams =>{
      this.crewCards = teams.filter(c=>{
        return c.proprietario === environment.userData.id;
      });
    });
  }
}

@ Injectable({
  providedIn: 'root'
})
export class GruppiService {
  constructor(private http: HttpClient) {}
  getTeams(): Observable<Team[]>{
    return this.http.get<Team[]>(environment.getTeams());
  }
}
```

**Figura 4.13:** Esempio implementazione *pattern Observer*

## 4.4 Codifica

### 4.4.1 Back end

#### 4.4.1.1 Gruppi service

##### **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 uno specifico gruppo.  
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.  
Mappatura: `/gruppi/inserisci`, verbo HTTP: POST;
- ***modifyGruppo***: gestisce la chiamata che modifica un gruppo.  
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`, ed ha lo scopo di fungere 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\*\*<sup>\[g\]</sup>](#), 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* vincolante 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> findByOrderId():* 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.

```

@Component
public class AuthFilter extends AbstractGatewayFilterFactory<AuthFilter.Config> {
    private WebClient.Builder webClientBuilder;
    @Autowired
    EurekaClient eurekaClient;
    public AuthFilter(Builder webClientBuilder) {
        super(Config.class);
        this.webClientBuilder = webClientBuilder;
    }
    public static class Config {
        /**
         * Questa classe rimane vuota perché non c'è bisogno di particolari impostazioni
         * di configurazione
         */
    }
    private Mono<Void> onError(ServerWebExchange exchange, String err, HttpStatus httpStatus) {
        ServerHttpResponse response = exchange.getResponse();
        response.setStatusCode(httpStatus);

        return response.setComplete();
    }
    @Override
    public GatewayFilter apply(Config config) {
        return (exchange, chain) -> {
            if (!exchange.getRequest().getHeaders().containsKey(HttpHeaders.AUTHORIZATION)) {
                return this.onError(exchange, "**** Informazioni di autorizzazione mancanti ****", HttpStatus.UNAUTHORIZED);
            }
            String auth = exchange.getRequest().getHeaders().get(HttpHeaders.AUTHORIZATION).get(0);
            String[] parts = auth.split(" ");
            if (parts.length != 2 || !"Bearer".equals(parts[0])) {
                return this.onError(exchange,
                    "**** Autorizzazione incorretta ****", HttpStatus.UNAUTHORIZED);
            }
            InstanceInfo nextServerFromEureka = eurekaClient.getNextServerFromEureka("UTENTI-SERVICE", false);
            String homeUrl = nextServerFromEureka.getHomePageUrl();
            return webClientBuilder.build().post().uri(homeUrl + "/validateToken?token=" + parts[1]).retrieve()
                .bodyToMono(String.class).map(login -> {
                    exchange.getRequest().mutate().header("X-auth-user-id", String.valueOf(login));
                    return exchange;
                }).flatMap(chain::filter);
        };
    }
}

```

**Figura 4.14:** Implementazione della classe *AuthFilter*

Infine, è stata utilizzata la seguente configurazione dello *Spring Cloud Gateway* nel file `configuration.yml`.



```

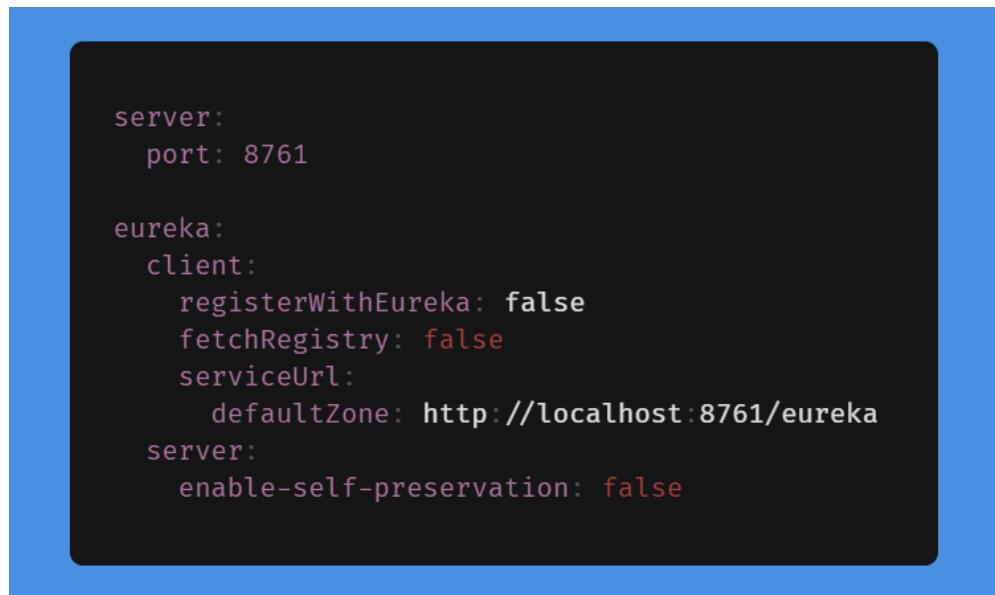
spring:
  application:
    name: API-GATEWAY-SERVICE
  main:
    allow-bean-definition-overriding: true
  cloud:
    gateway:
      discovery.locator.enabled: true
      routes:
        - id: utenti
          uri: lb://UTENTI-SERVICE
          predicates:
            - Path= /utenti/**, /signin, /signup, /validateToken
          filters:
            - AuthFilter
        - id: posizioni
          uri: lb://POSIZIONI-SERVICE
          predicates:
            - Path=/posizioni/**
          filters:
            - AuthFilter
        - id: uscite
          uri: lb://USCITE-SERVICE
          predicates:
            - Path=/uscite/**
          filters:
            - AuthFilter
        - id: gruppi
          uri: lb://GRUPPI-SERVICE
          predicates:
            - Path=/gruppi/**
          filters:
            - AuthFilter
      default-filters:
        - DedupeResponseHeader=Access-Control-Allow-Credentials Access-Control-Allow-Origin
    globalcors:
      corsConfigurations:
        '/**':
          allowedOrigins: "http://localhost:4200/"
          allowedMethods: "*"
          allowedHeaders: "*"

```

**Figura 4.15:** Configurazione dello *Spring Cloud Gateway* nel file `application.yml` del microservizio *API Gateway*

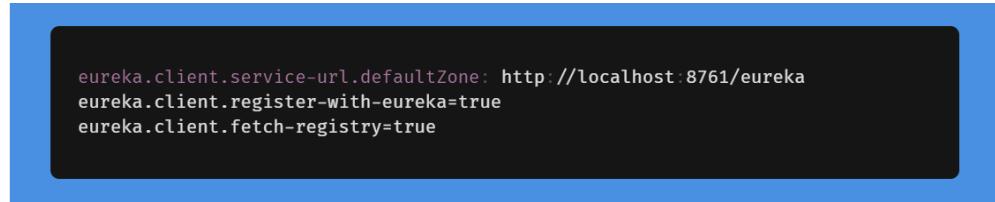
#### 4.4.2 Eureka Server

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`.



**Figura 4.16:** 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`.



**Figura 4.17:** 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:

```

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

```

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

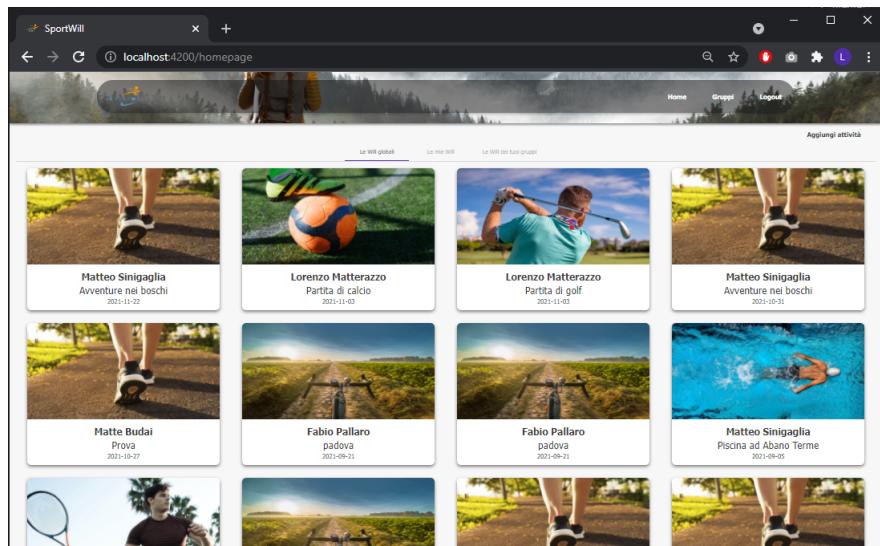
### 4.4.3 Front end

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 4.18:** Pagina di homepage

#### Componenti utilizzati

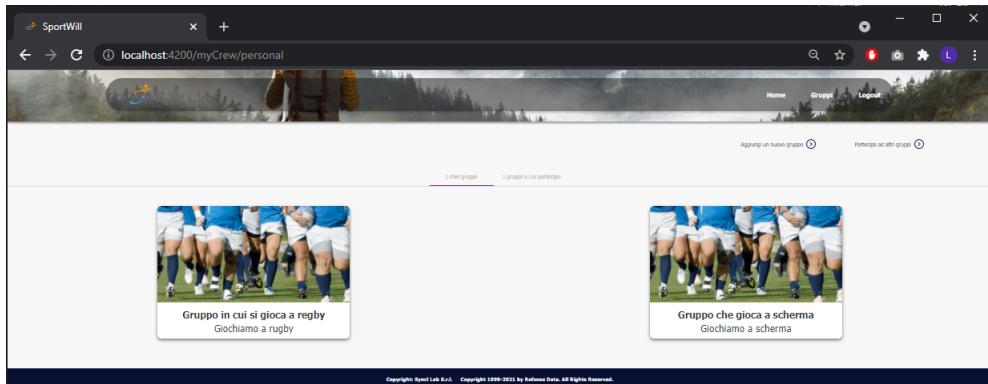
- [Homepage](#)
- [CrewCard](#)

#### Servizi

- [GruppiService](#)
- [WillDataService](#)

#### Gruppi dell'utente

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



**Figura 4.19:** Pagina dei gruppi creati o a cui partecipa un utente

### Componenti utilizzati

- [Crewpage](#)
- [CrewCard](#)

### Servizi

- [GruppiService](#)

### 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 4.20:** Pagina dei gruppi

### Componenti utilizzati

- [Crewpage](#)
- [CrewCard](#)

### Servizi

- GruppiService

#### Crea nuovo gruppo

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

**Figura 4.21:** Pagina di creazione di un gruppo

### Componenti utilizzati

- CrewDetail

### Servizi

- GruppiService

#### Modifica un gruppo

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



**Figura 4.22:** Pagina di modifica di un gruppo

### Componenti utilizzati

- [CrewEditable](#)
- [Dialog](#)

### Servizi

- [GruppiService](#)

#### 4.4.3.2 Componenti

##### Homepage

##### CrewCard

##### Crewpage

**CrewDetail**

**CrewEditable**

**Dialog**

#### **4.4.4 Servizi**

**GruppiService**

**WillDataService**

##### **4.4.4.1 Servizi**



## **Capitolo 5**

### **Verifica e validazione**



# **Capitolo 6**

# **Conclusioni**

- 6.1 Consuntivo finale**
- 6.2 Raggiungimento degli obiettivi**
- 6.3 Conoscenze acquisite**
- 6.4 Valutazione personale**



## **Appendice A**

# **Appendice A**

Citazione

Autore della citazione







# Bibliografia