

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 condividere la sua intenzione (*will*) di effettuare un’attività sportiva. La piattaforma permetteva ad ogni utente l’accesso a tutte le *will* di tutti gli utenti indistintamente, 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” potessero unirsi, vedendo quindi solo le *will* relative al gruppo. Le attività svolte nel corso dello *stage* sono state due:

- l’aggiornamento del *back end* della *web app*, mediante lo sviluppo di tre microservizi mediante il *framework* Spring Java. In particolare:
 1. il primo microservizio gestisce i 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 API dei vari servizi presenti in un unico punto di accesso;
 3. il terzo è l’implementazione di un **Eureka Server** che contiene le informazioni di tutti i servizi che si registrano nel suo *server*.

Inoltre i microservizi 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* di tutti i microservizi su Docker.

- la modifica del *front end*, mediante il *framework* Angular, per adeguarlo alle nuove funzionalità del *back end*.

“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 impressionato e porto con me ricordi positivi delle sue lezioni.

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	L'azienda	1
1.2	Lo stage proposto	1
1.3	Strumenti utilizzati	2
1.4	Prodotto ottenuto	3
1.5	Organizzazione del testo	3
1.6	Convenzioni tipografiche	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	24
4	Progettazione e codifica	29
4.1	Tecnologie	29
4.2	Progettazione	30
4.2.1	Back end	30
4.2.2	Front end	32
4.3	Design Pattern utilizzati	35
4.3.1	Microservizi	35
4.3.2	API Gateway	35
4.3.3	Client-Side Service Discovery e Service Registry	36
4.3.4	Dependency Injection	36
4.3.5	Singleton	38
4.3.6	Feature Service	38
4.3.7	Lazy Loading	38
4.3.8	Observer	39
4.4	Codifica	40
4.4.1	Back end	40
4.4.2	Front end	48
5	Verifica e validazione	61
5.1	Accessibilità	61
5.1.1	Elementi di accessibilità	61
5.2	Verifica del GruppiService	61

5.2.1	Test effettuati	62
6 Conclusioni		65
6.1	Raggiungimento degli obiettivi	65
6.2	Conoscenze acquisite	65
6.3	Valutazione personale	66
Glossario		67
Acronimi		71
Bibliografia		73

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	15
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>	23
4.1	Architettura a strati di Spring Boot	31
4.2	Spring Boot <i>workflow</i>	31
4.3	<i>Mockup</i> pagina per la visualizzazione dei gruppi creati dall'utente	33
4.4	<i>Mockup</i> pagina per la visualizzazione dei dettagli di un gruppo	34
4.5	<i>Mockup</i> pagina per la modifica dei dettagli di un gruppo	34
4.6	Diagramma concettuale che descrive l' <i>API Gateway</i>	35
4.7	Diagramma concettuale che descrive la registrazione dei microservizi all' <i>Eureka Server</i>	36
4.8	Tabella Crew	40
4.9	Pagina homepage	48
4.10	Pagina dei gruppi creati o a cui partecipa un utente	49
4.11	Pagina dei gruppi	49
4.12	Pagina di creazione di un gruppo	50
4.13	Pagina di modifica di un gruppo	51
4.14	Pagina di visualizzazione dei dettagli di un gruppo	52
4.15	Componente Homepage di un utente autenticato	53
4.16	Componente Homepage di un utente non autenticato	53
4.17	Componente WillCard	54
4.18	Componente CrewCard	54
4.19	Componente Crewpage che visualizza i gruppi in cui partecipare	55
4.20	Componente Crewpage che visualizza la pagina di gestione dei gruppi dell'utente	55
4.21	Componente che permette di visualizzare i dettagli di un gruppo	56
4.22	Componente CrewEditable	57
4.23	Componente DialogComponent	58

4.24 Componente MapComponent	58
5.1 <i>Code coverage</i> della classe GruppiController	62

Elenco dei codici

4.1	Esempio di <i>Dependency Injection</i> con Spring	37
4.2	Esempio di creazione di un servizio	37
4.3	Esempio di <i>constructor injection</i>	37
4.4	Implementazione del <i>pattern Lazy loading</i> nel AppRoutingModule	38
4.5	Esempio implementazione <i>pattern Observer</i>	39
4.6	Rappresentazione JSON della classe Crew	40
4.7	Implementazione della classe AuthFilter	44
4.8	Configurazione dello <i>Spring Cloud Gateway</i> nel file application.yml del microservizio API Gateway	46
4.9	File application.yml del microservizio Eureka Server	47
4.10	Configurazione per la registrazione di un microservizio all'Eureka Server nel file application.properties	47

Elenco delle tabelle

3.1	Tabella del tracciamento dei requisiti funzionali	24
3.1	Tabella del tracciamento dei requisiti funzionali	25
3.1	Tabella del tracciamento dei requisiti funzionali	26
3.2	Tabella del tracciamento dei requisiti qualitativi	26
3.3	Tabella del tracciamento dei requisiti di vincolo	27
5.1	Test di unità	62
5.1	Test di unità	63
5.1	Test di unità	64
6.1	Tabella raggiungimento degli obiettivi	65

Capitolo 1

Introduzione

1.1 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

1.2 Lo stage proposto

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

Frank Deford

La funzionalità principale di **Sport Will** è quella di permettere agli utenti di condividere le proprie *will^[g]* e partecipare a queste attività sportive. Tuttavia, allo stato dell’arte attuale, non esiste ancora la suddivisione delle *will* per gruppi, quindi 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.

L’azienda possiede già i dati per la gestione degli utenti e delle *will*, le quali vanno però associate ai gruppi. In particolare, è necessario tenere traccia delle *will* e degli

utenti appartenenti ad ogni gruppo, risultato che si ottiene attraverso la creazione di tabelle che congiungono gli utenti e le *will* con la tabella dei gruppi.

Le implementazioni alla *web app* permettono di visualizzare le *will* appartenenti ad ogni gruppo. Inoltre, è ora possibile per l'utente visualizzare i gruppi a cui partecipa, quelli a cui può unirsi e quelli da lui creati e modificare ed eliminare questi ultimi.

È stata prestata attenzione alla consistenza e alla coerenza dei dati in caso di eliminazione di un gruppo, di un utente o di una *will* dal *database*. Infatti, nel caso di eliminazione occorre aggiornare le voci nelle tabelle di congiunzione corrispondenti.

1.3 Strumenti utilizzati

Nella prima fase del progetto sono state apportate modifiche al *back end*. Gli strumenti utilizzati in questa fase sono:

Visual Studio Code Editor di codice con funzionalità di *debugging*, controllo di versione con Git integrato, *syntax highlighting*, IntelliSense, Snippet e *refactoring* del codice.

Il 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 di questa fase 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;
- **Docker Extension Pack**: raccolta di estensioni per lo gestione dei *container* Docker, Docker images, Dockerfile e file Docker-compose;
- **Extension Pack for Java**: raccolta di estensioni popolari che possono aiutare a scrivere, testare e fare il *debugging* di applicazioni Java.

Postman Strumento che permette di eseguire richieste HTTP ad un *server* di *back end*. È stato utilizzato per testare le richieste HTTP dei servizi Rest sviluppati.

DbVisualizer Strumento *multi-database* intuitivo e ricco di funzionalità per sviluppatori, analisti e amministratori di *database*, che fornisce un'unica 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*.

È stato utilizzato per visualizzare il popolamento delle tabelle di un *database* PostgreSQL, in particolare dopo aver effettuato delle modifiche attraverso le richieste HTTP.

Nella seconda fase dello sviluppo, in cui sono state apportate modifiche al *front end*, sono state progettate le interfacce delle pagine *web* per la visualizzazione, creazione e modifica di un gruppo da implementare con **Figma**, 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.

Per quanto riguarda la codifica, come nella prima fase, è stato utilizzato Visual Studio Code come *editor* di codice, con l'aiuto di nuove estensioni per lo sviluppo di applicazioni *web*, ovvero:

- **Angular Extension Pack:** raccolta di estensioni per lo sviluppo con Angular;
- **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;

1.4 Prodotto ottenuto

Al termine dello *stage* le integrazioni delle funzionalità con la *web app* sono state realizzate con successo. Tutte le chiamate alle **Application Program Interface (API)** sono state testate e 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.

1.5 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.

1.6 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*.

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

In questa fase vengono analizzati i rischi emersi durante la fase di analisi iniziale in cui si potrebbe incombere. Si è proceduto quindi all'elaborazione di un piano di contingenza per ogni rischio individuato.

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 un livello di capacità tecnica elevata 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, il che 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 stringhe identificative di ogni estensione utilizzata 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, considerando il rinvio delle scadenze prefissate solo in caso estremo.

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 nello 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^{\[g\]}](#) 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](#)^[g] 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](#)^[g] 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 della *web app*. Questo attore ha 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*.

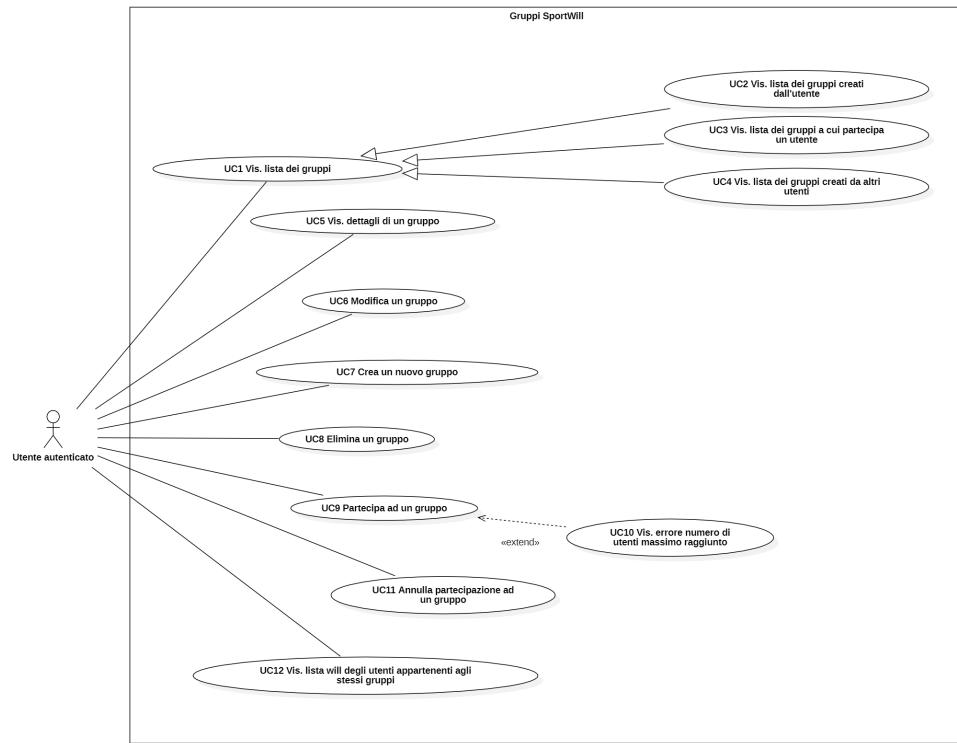


Figura 3.1: Diagramma generale dei casi d'uso

UC1: Visualizzazione lista dei gruppi

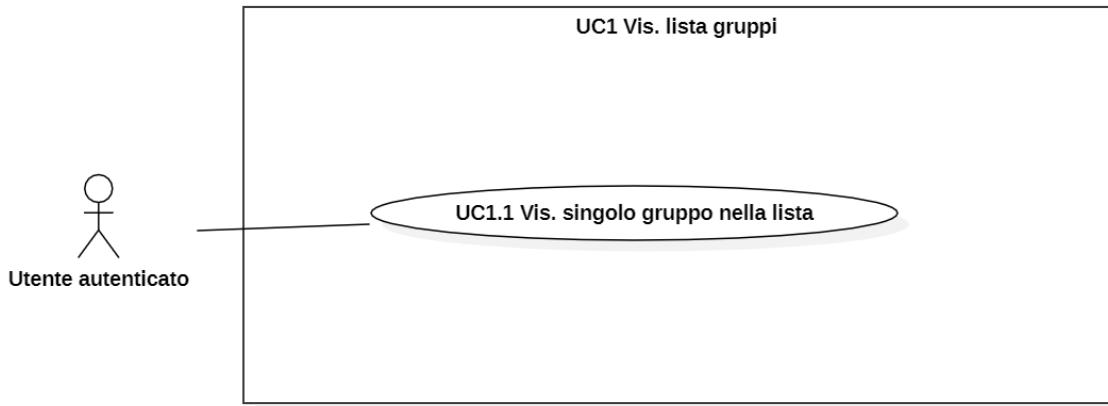
Attori Principali: Utente autenticato.

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: L'utente si trova nella schermata di visualizzazione della lista dei gruppi.

**Figura 3.2:** UC1: Vis. lista dei gruppi**UC1.1: Visualizzazione singolo gruppo in lista**

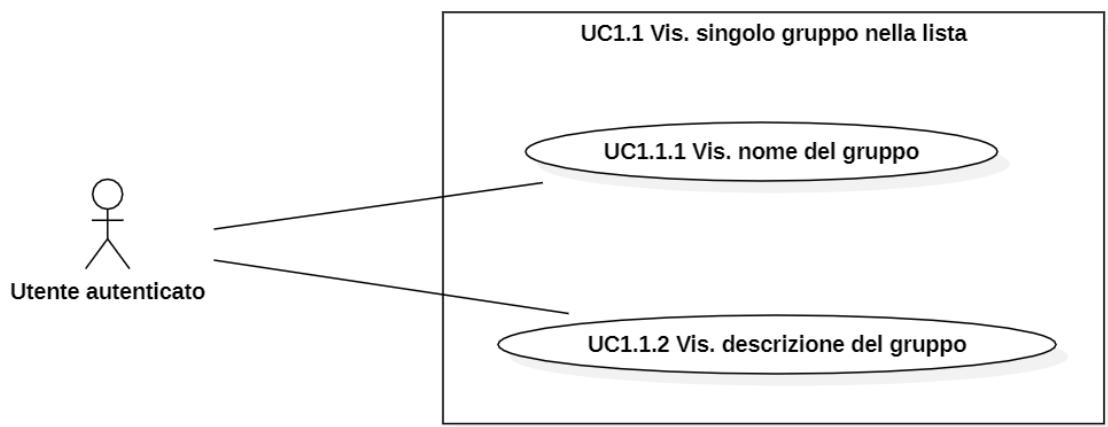
Attori Principali: Utente autenticato.

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: L'utente si trova nella schermata di visualizzazione dei gruppi e visualizza un singolo gruppo dalla 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: L'utente si trova nella schermata di visualizzazione dei gruppi e visualizza dalla lista dei gruppi il nome di uno specifico gruppo.

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: L'utente si trova nella schermata di visualizzazione dei gruppi e visualizza dalla lista dei gruppi la descrizione di uno specifico gruppo.

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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare la schermata dei gruppi creati dall'utente.

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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare la schermata dei gruppi a cui partecipa l'utente. I gruppi creati dall'utente, nonostante vi partecipi, non sono visualizzati in questa schermata.

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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare la schermata dei gruppi creati da altri utenti a cui è possibile partecipare. Non sono quindi presenti in questa schermata i gruppi a cui partecipa già un utente.

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: L'utente, che si trova nella schermata di visualizzazione dei gruppi, decide di visualizzare i dettagli di un gruppo.

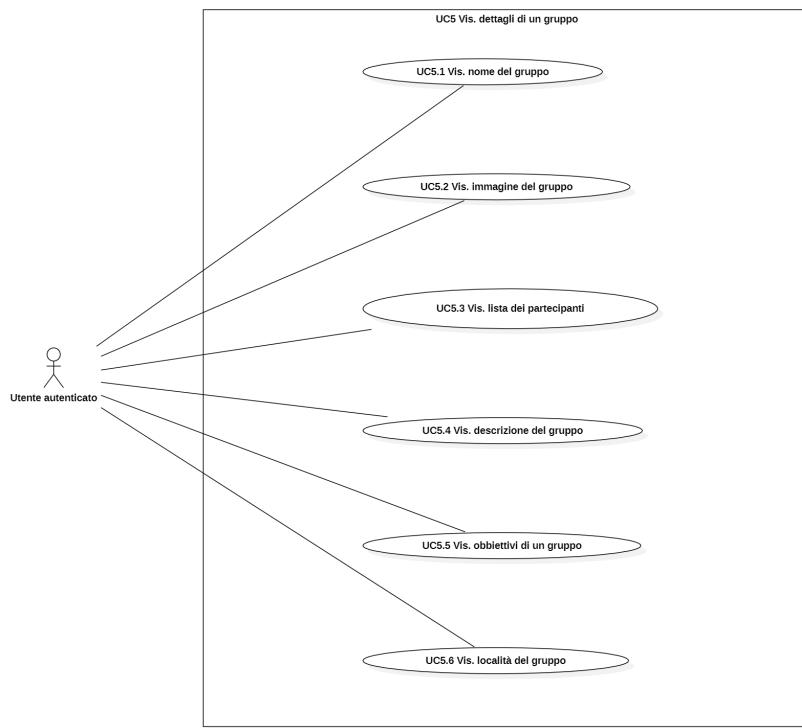


Figura 3.4: UC5: Vis. dettagli di un gruppo

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: L'utente dalla schermata dei gruppi ha deciso di visualizzare i dettagli di un gruppo, e dalla schermata di visualizzazione dei dettagli di un gruppo visualizza il nome del 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: L'utente dalla schermata dei gruppi ha deciso di visualizzare i dettagli di un gruppo, e dalla schermata di visualizzazione dei dettagli di un gruppo visualizza l'immagine del 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: L'utente dalla schermata dei gruppi ha deciso di visualizzare i dettagli di un gruppo, e dalla schermata di visualizzazione dei dettagli di un gruppo 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: L'utente dalla schermata dei gruppi ha deciso di visualizzare i dettagli di un gruppo, e dalla schermata di visualizzazione dei dettagli di un gruppo visualizza un partecipante dalla lista dei partecipanti ad un gruppo.

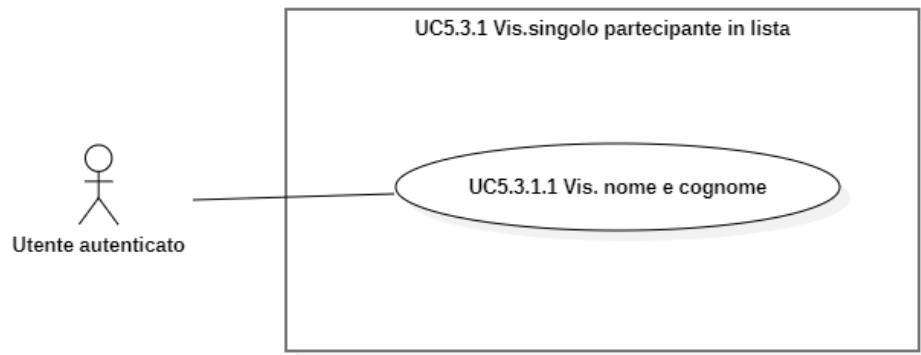


Figura 3.6: UC5.3.1: Vis. singolo partecipante in lista

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: L'utente dalla schermata dei gruppi ha deciso di visualizzare i dettagli di un gruppo, e dalla schermata di visualizzazione dei dettagli di un gruppo visualizza il nome e il cognome di un partecipante dalla lista dei partecipanti ad un gruppo.

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: L'utente dalla schermata dei gruppi ha deciso di visualizzare i dettagli di un gruppo, e dalla schermata di visualizzazione dei dettagli di un gruppo visualizza la descrizione di un gruppo.

UC5.5: Visualizzazione obiettivi 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 obiettivi di un gruppo.

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

Scenario principale: L'utente dalla schermata dei gruppi ha deciso di visualizzare i dettagli di un gruppo, e dalla schermata di visualizzazione dei dettagli di un gruppo visualizza gli obiettivi 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: L'utente dalla schermata dei gruppi ha deciso di visualizzare i

dettagli di un gruppo, e dalla schermata di visualizzazione dei dettagli di un gruppo visualizza la località in cui il gruppo è presente.

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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e desidera modificare i dettagli di un gruppo dalla schermata di modifica dei gruppi.



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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e desidera modificare il nome di un gruppo dalla schermata di modifica dei gruppi.

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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e desidera modificare la descrizione di un gruppo dalla schermata di modifica dei gruppi.

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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e desidera modificare gli obiettivi di un gruppo dalla schermata di modifica dei gruppi.

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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e desidera modificare il numero massimo di partecipanti ad un gruppo dalla schermata di modifica. Nel caso al gruppo vi partecipino più utenti di quanto modificato, non sarà più permesso ad altri utenti unirsi al 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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e desidera modificare la località in cui è presente un gruppo dalla schermata di modifica dei gruppi.

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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e crea un nuovo gruppo dalla schermata di creazione di un gruppo.

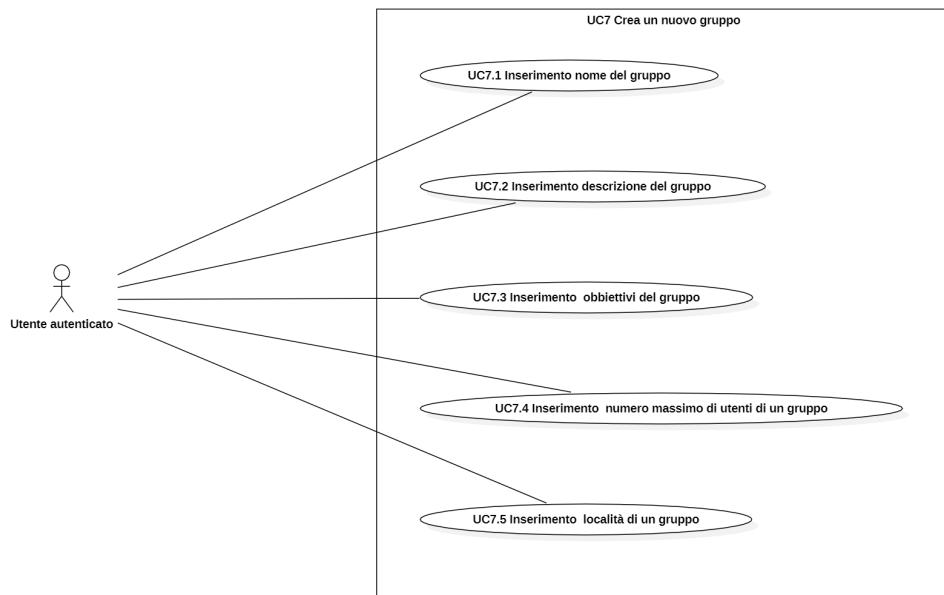


Figura 3.8: UC7: Crea un nuovo gruppo

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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e inserisce il nome del gruppo dalla schermata di creazione di un 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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e inserisce la descrizione del gruppo dalla schermata di creazione di un 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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e inserisce gli obiettivi del gruppo dalla schermata di creazione di un 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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e inserisce il numero massimo di partecipanti al gruppo dalla schermata di creazione di un 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: L'utente si trova nella schermata di visualizzazione dei gruppi, in particolare dei gruppi creati dall'utente e inserisce la località in cui è presente il gruppo dalla schermata di creazione di un 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: L'utente si trova nella schermata di modifica dei gruppi di un gruppo creato dall'utente ed elimina il gruppo in questione.

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: L'utente si trova nella schermata dei dettagli di un gruppo a cui non partecipa, e partecipa al gruppo in questione.

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: L'utente si trova nella schermata dei dettagli di un gruppo a cui già partecipa, e annulla la partecipazione al gruppo in questione.

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: L'utente si trova nella schermata principale della *web app* nella finestra di visualizzazione della lista delle *will* degli utenti appartenenti agli stessi gruppi a cui partecipa l'utente.

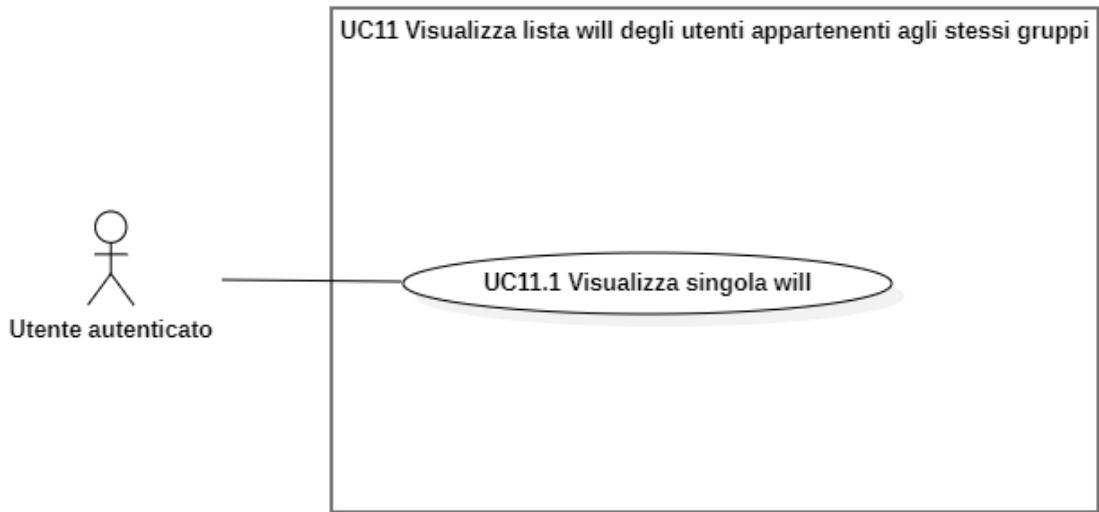


Figura 3.9: UC11: Vis. lista *will* degli utenti appartenenti agli stessi gruppi

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: L'utente si trova nella schermata principale della *web app*, nella finestra di visualizzazione della lista delle *will* degli utenti appartenenti agli stessi gruppi a cui partecipa l'utente e visualizza una *will* da questa lista.

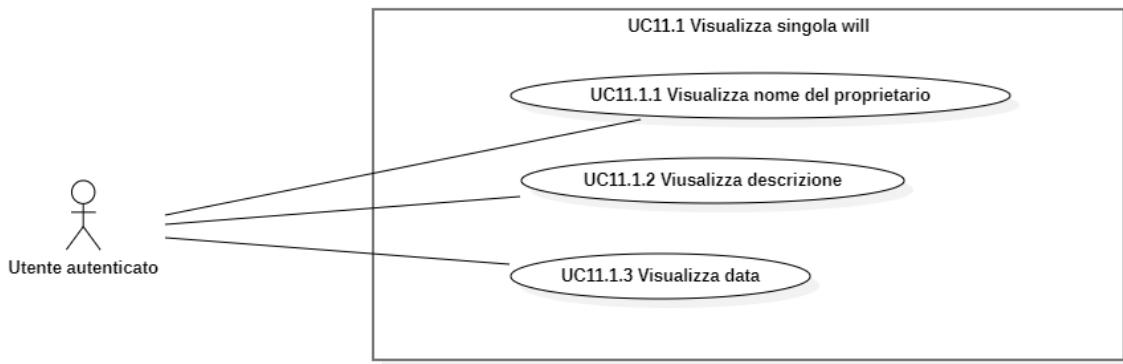


Figura 3.10: UC11.1: Vis. singola will

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: L'utente si trova nella schermata principale della *web app*, nella finestra di visualizzazione dalla lista delle *will* degli utenti appartenenti agli stessi gruppi a cui partecipa l'utente e visualizza il nome del proprietario di una *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: L'utente si trova nella schermata principale della *web app*, nella finestra di visualizzazione dalla lista delle *will* degli utenti appartenenti agli stessi gruppi a cui partecipa l'utente e visualizza la descrizione di una *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: L'utente si trova nella schermata principale della *web app*, nella finestra di visualizzazione dalla lista delle *will* degli utenti appartenenti agli stessi gruppi a cui partecipa l'utente e visualizza la data di uscita di una *will* dalla lista delle *will*.

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	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
Requisito	Descrizione	Fonte

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
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
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
Requisito	Descrizione	Fonte

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
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
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 3.2: 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 3.3: 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 degli 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 a tal punto da sostenere oggi gran parte del mondo digitale, fornendo una piattaforma affidabile su cui costruire molti servizi e applicazioni. Infatti, molti dei nuovi prodotti fanno 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, dal momento 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*^[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 che viene utilizzato 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. [1]

Angular Material

Angular Material è una libreria sviluppata da Google nel 2014 allo scopo di 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: uno per gli sviluppatori e uno 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

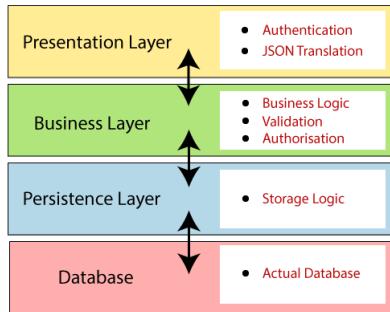
4.2.1.1 Architettura Spring Boot

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*, poiché trasforma gli oggetti provenienti dalla *business logic* in righe del *database*;
- **Database Layer:** in questo strato vengono effettuate le operazioni **Create Read Update Delete (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*, il quale 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 questo *component* si possono inserire diversi altri *component* per ogni elemento della 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 dovrebbe essere la grafica delle nuove maschere da aggiungere a *SportWill*. A partire dai confronti con il *tutor*, è stato poi utilizzato **Figma** per fare il *mockup*, in modo da testare quanto siano accattivanti i vari elementi visivi.

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.

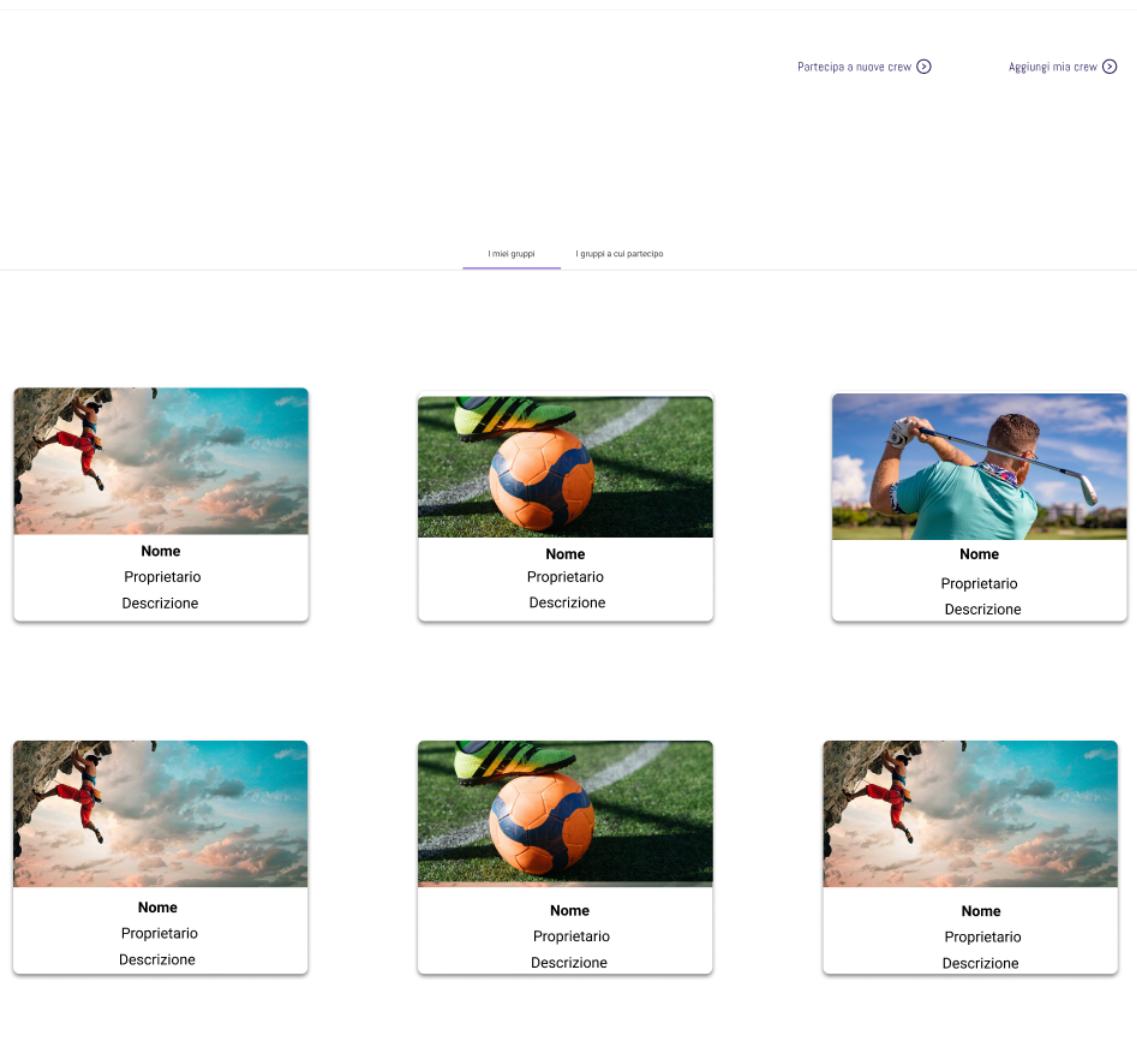


Figura 4.3: Mockup pagina per la visualizzazione dei gruppi creati dall'utente



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 questo approccio, questi ultimi sono composti di servizi indipendenti di piccole dimensioni che comunicano tra loro tramite [API](#).

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 che riceve 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

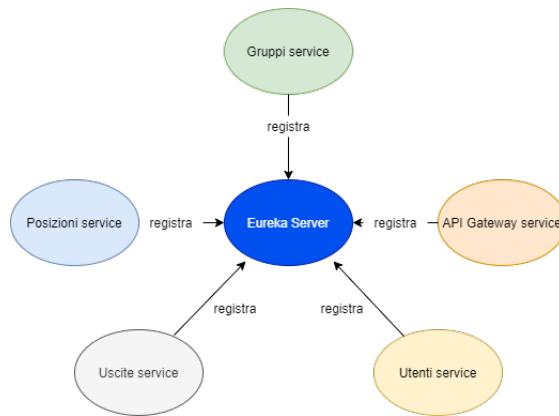


Figura 4.7: Diagramma concettuale che descrive la registrazione dei microservizi all’*Eureka Server*

I **microservizi** hanno una natura dinamica, in quanto è possibile che più istanze di un singolo **microservizio** coesistano, talvolta 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*^[g].^[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* molto solido e 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 di `providedIn: 'root'`, per esempio, viene specificato che Angular dovrebbe fornire il servizio nella *root injector*, vale a dire 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 essi vengono utilizzati, tenendo così bassa la quantità di dati scaricata e una diminuzione del tempo necessario al caricamento dell'applicazione.

Questo *pattern* per la gestione del *routing* viene applicato mediante l'associazione di una specifica *view* a 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. Inoltre, 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((secondCrews)
12                     => {
13                         this.crewCards = secondCrews.filter((c) => {
14                             let toReturn: boolean = true;
15                             for (let crew of crews){
16                                 if (crew.id == c.id){
17                                     toReturn = false;
18                                     break;
19                                 }
20                             }
21                             return toReturn;
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*, mentre 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. Oltre a ciò, le repository svolgono al contempo il compito di [EntityManager](#)^[g], ossia effettuano l'accesso agli oggetti ed eseguono 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, mentre 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. Allo stesso modo, quando viene eliminata un'uscita viene eliminata dalle uscite di tutti i gruppi.

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 `modificaVisibilita` 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 e 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     public AuthFilter(Builder webClientBuilder) {
10         super(Config.class);
11         this.webClientBuilder = webClientBuilder;
12     }
13
14     public static class Config {
15         /*
16          * Questa classe rimane vuota perche non c'e bisogno
17          * di particolari impostazioni
18          * di configurazione
19          */
20     }
21
22     private Mono<Void> onError(ServerWebExchange exchange,
23         String err, HttpStatus httpStatus) {
24         ServerHttpResponse response = exchange.getResponse();
25         response.setStatusCode(httpStatus);
26
27         return response.setComplete();
28     }
29     @Override
30     public GatewayFilter apply(Config config) {
31         return (exchange, chain) -> {
32             if (!exchange.getRequest().getHeaders().
33                 containsKey(HttpHeaders.AUTHORIZATION)) {
34                 return this.onError(exchange, "****"
35                     Informazioni di autorizzazione mancanti
36                     ****", HttpStatus.UNAUTHORIZED);
37             }
38             String auth = exchange.getRequest().getHeaders().
39                 get(HttpHeaders.AUTHORIZATION).get(0);
40             String[] parts = auth.split(" ");
41             if (parts.length != 2 || !"Bearer".equals(parts[0]
42                ])) {
43                 return this.onError(exchange,
44                     "**** Autorizzazione incorretta ***",
45                     HttpStatus.UNAUTHORIZED);
46             }
47             InstanceInfo nextServerFromEureka = eurekaClient.
48                 getNextServerFromEureka("UTENTI-SERVICE",
49                 false);
50             String homeUrl = nextServerFromEureka.
51                 getHomePageUrl();
52             return webClientBuilder.build().post().uri(
53                 homeUrl + "/validateToken?token=" + parts[1])..
54                 retrieve()
55                     .bodyToMono(String.class).map(login -> {
56                         exchange.getRequest().mutate().header
57                             ("X-auth-user-id", String.valueOf(
58                             login));
59                         return exchange;
60                     }).flatMap(chain::filter);
61         };
62     }
63 }
```

```

48
49 }

```

Codice 4.7: Implementazione della classe `AuthFilter`

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

```

1   spring:
2     application:
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*

4.4.1.4 Eureka Server

Questo *microservizio* contiene solo una classe al suo interno, ossia quella classe che viene generata automaticamente quando si inizializza un progetto con Spring. L'unica modifica effettuata alla classe è l'annotazione `@EnableEurekaServer`, che permette di attivare 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`

4.4.1.5 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.

4.4.2 Front end

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

4.4.2.1 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.

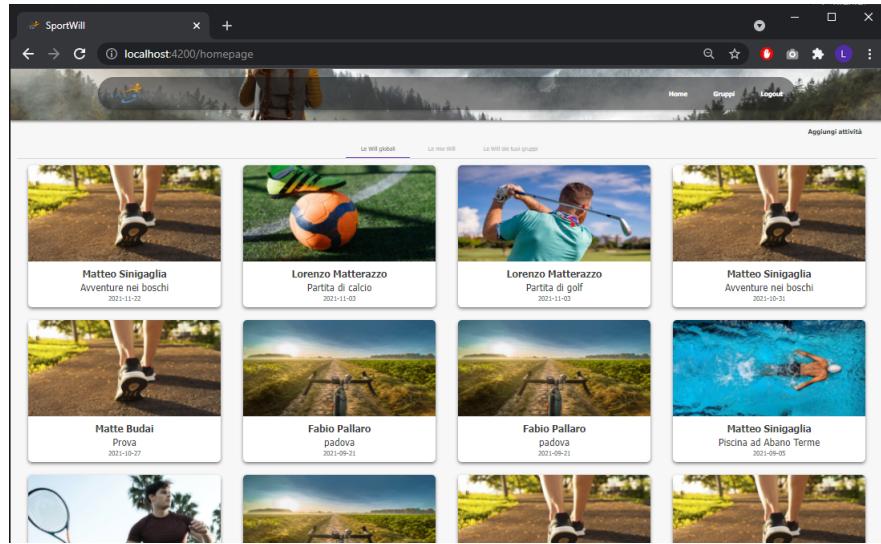


Figura 4.9: Pagina homepage

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.

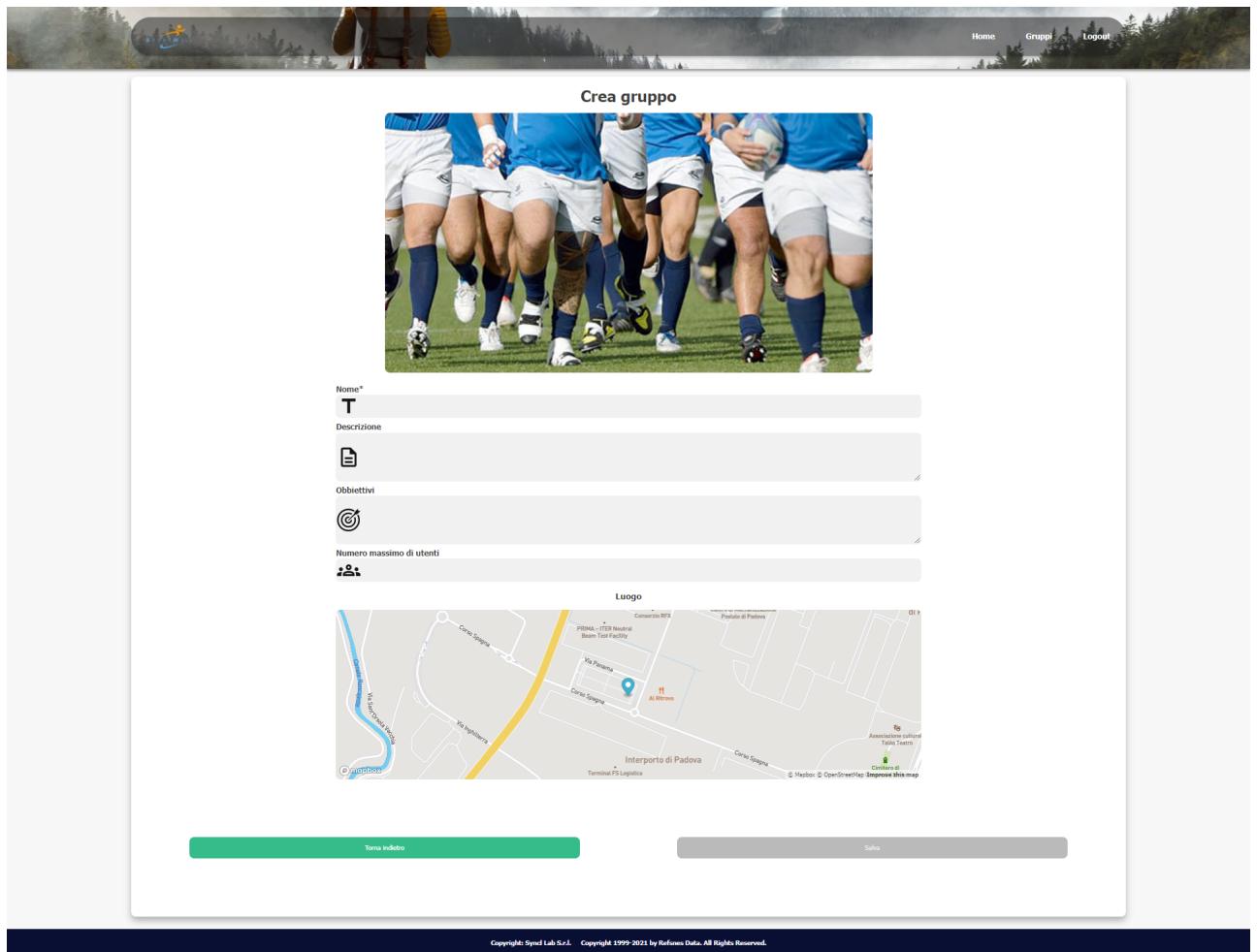


Figura 4.12: Pagina di creazione di un gruppo

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.

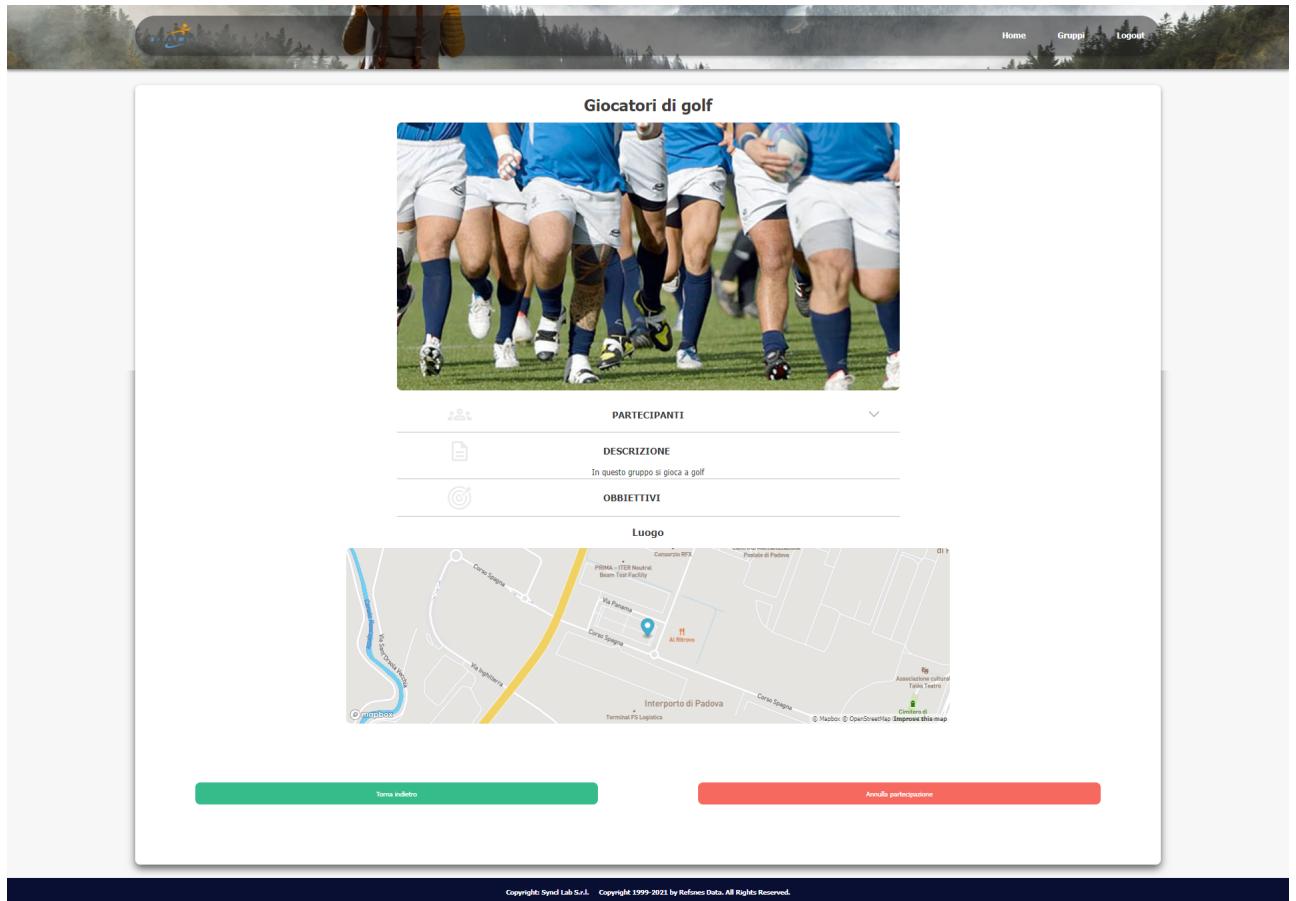


Figura 4.14: Pagina di visualizzazione dei dettagli di un gruppo

Componenti utilizzati

- [CrewDetail](#)

4.4.2.2 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.

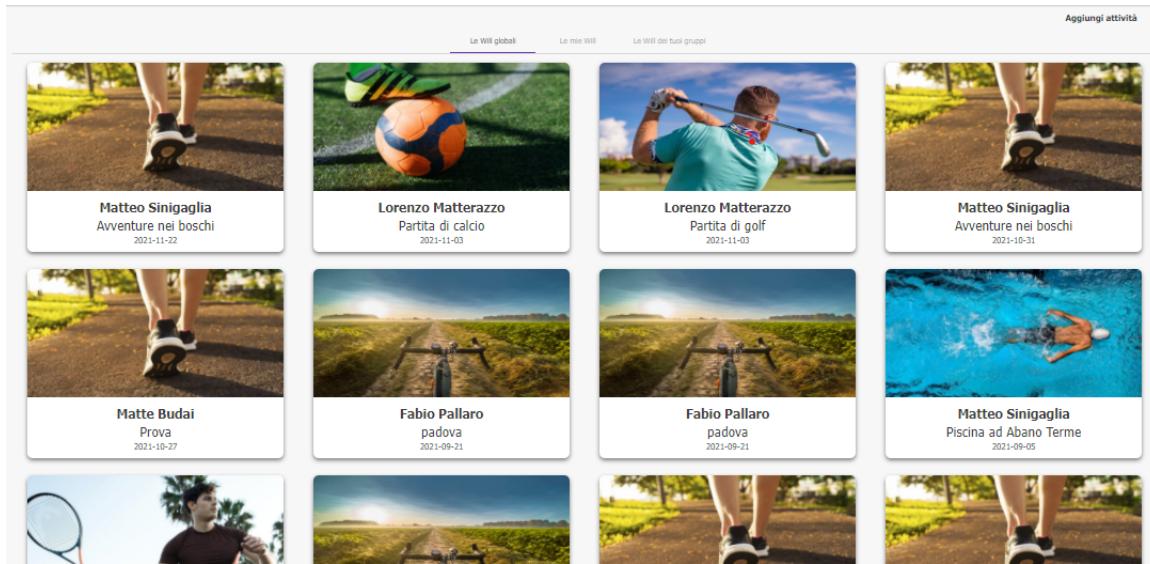


Figura 4.15: Componente Homepage di un utente autenticato

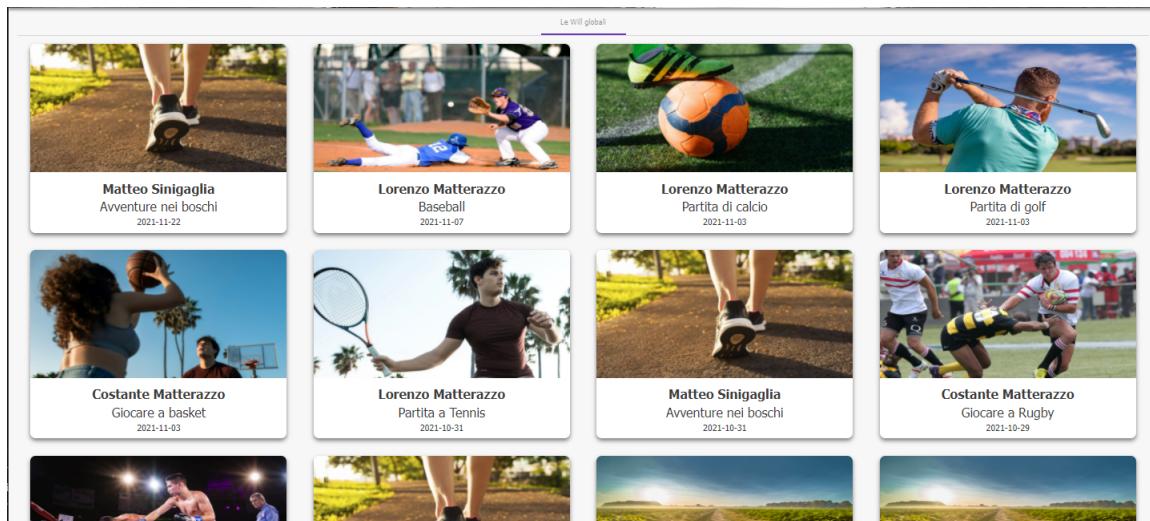


Figura 4.16: Componente Homepage di un utente non autenticato

Metodi

- ***ngOnInit***: all'inizializzazione del componente viene controllato, mediante il servizio `AuthenticationService`(già implementato prima del presente *stage*), 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

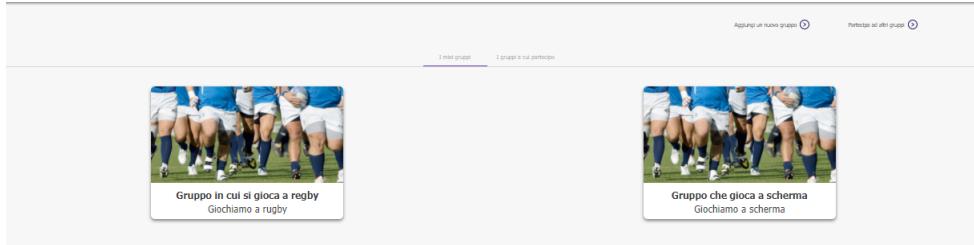


Figura 4.20: Componente **Crewpage** che visualizza la pagina di gestione dei gruppi dell’utente

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 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.

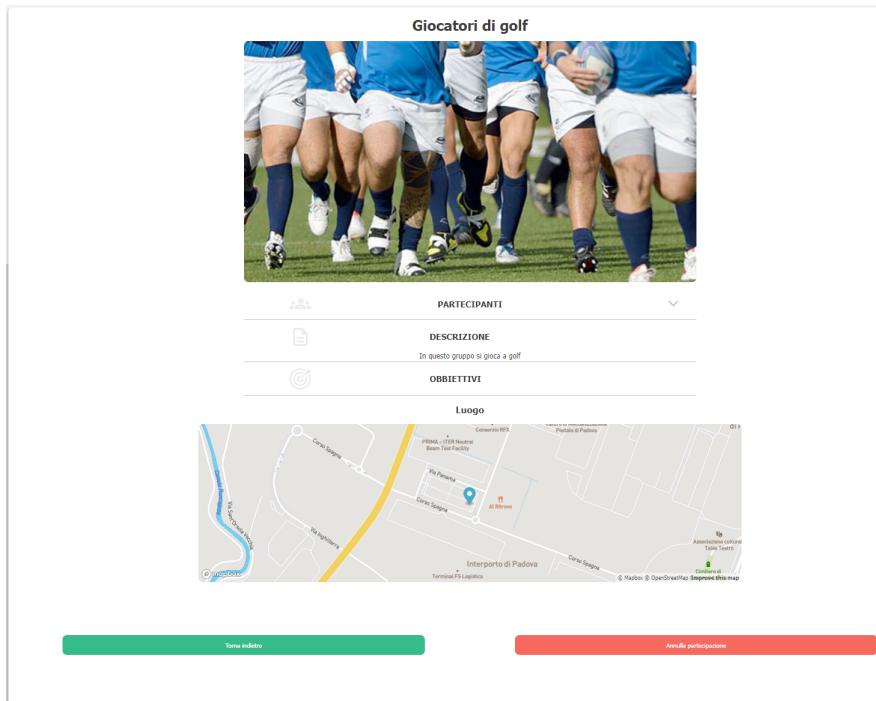


Figura 4.21: Componente che permette di visualizzare i dettagli di un gruppo

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.

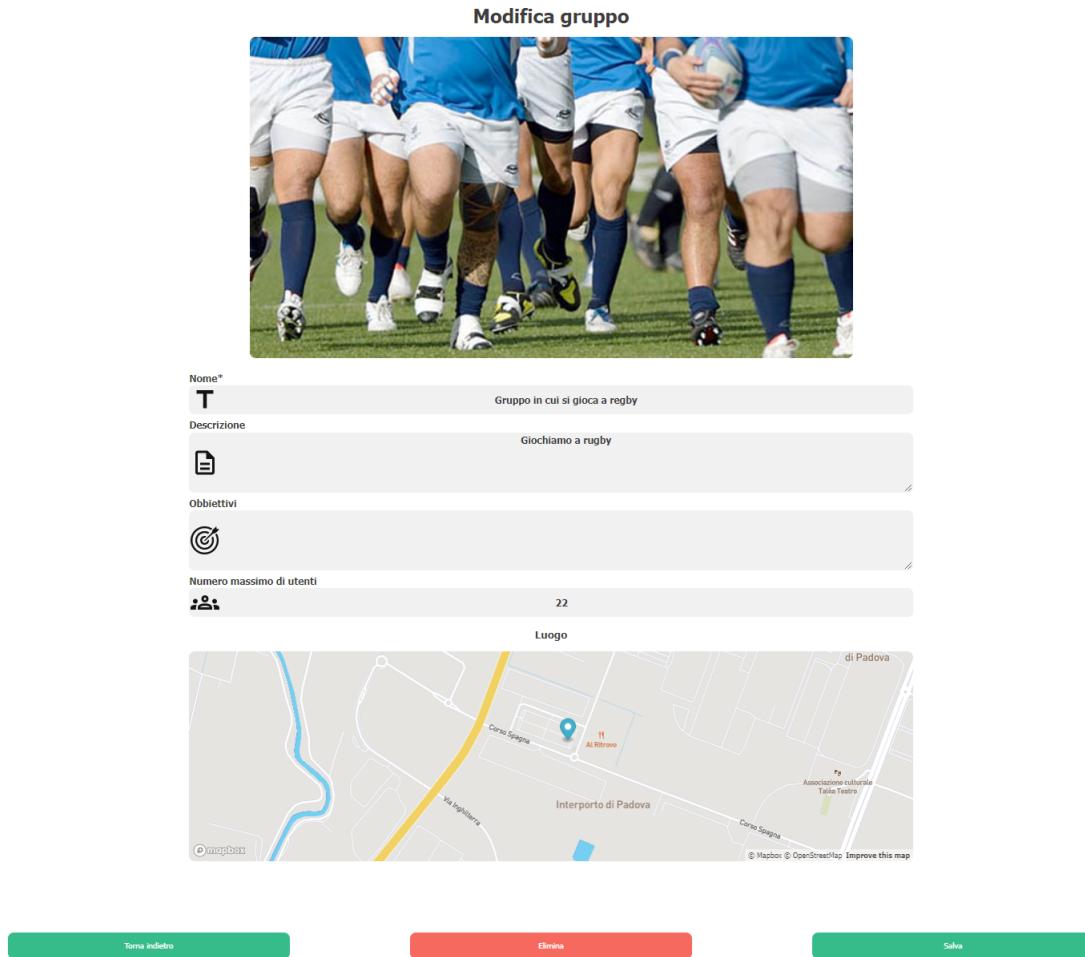


Figura 4.22: Componente CrewEditable

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 specificate 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; in caso lo sia 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`.

4.4.2.3 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*, i quali 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*, dal momento che essi sono utili solamente qualora si abbia a che fare con *form* molto grandi. Di conseguenza, essendo qui 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*^[8] 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, al fine di 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, dal momento che ha agevolato 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 5.1: 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
Codice	Descrizione	Esito

Tabella 5.1: Test di unità

Codice	Descrizione	Esito
TU10	Il metodo <code>modifyGruppo</code> deve restituire un <code>HttpStatus.NOT_MODIFIED</code> nel caso il gruppo specificato non esiste	S
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
Codice	Descrizione	Esito

Tabella 5.1: Test di unità

Codice	Descrizione	Esito
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
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*. Inoltre, sono stati portati a termine con un anticipo tale che ha reso possibile anche la realizzazione di attività che non erano inizialmente pianificate.

Tabella 6.1: 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 microservizi richiesti con una percentuale di superamento pari al 80	Soddisfatto
D01	Portare a termine l'implementazione dei microservizi richiesti con una percentuale di superamento pari al 100	Soddisfatto
F01	Utilizzo della containerizzazione per portare tutti i microservizi 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 con il 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 in maniera efficiente ed efficace attraverso le sue numerose funzionalità, tra cui *templating*, *two-way binding*, modularizzazione, gestione *API Restful* e *dependency injection*.

6.3 Valutazione personale

Le aspettative sia mie che dell'azienda sono state ampiamente superate. Per questo motivo, ritengo i risultati ottenuti nel corso dello *stage* siano sicuramente un successo, anche in virtù delle capacità acquisite. Quanto prodotto ha migliorato il *software* già in possesso all'azienda e costituisce 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 e ad aggiungere al mio bagaglio di conoscenze il linguaggio 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 costituisce un'esperienza preziosa per il mio futuro lavorativo e non.

Glossario

Single Responsibility Principle Nella programmazione orientata agli oggetti, il *single responsibility principle* afferma che ogni elemento di un programma deve avere una sola responsabilità, e che tale responsabilità debba essere interamente incapsulata dall'elemento stesso. [38](#)

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*. [7](#), [65](#)

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. [47](#), [67](#)

endpoint Canale da cui le *API* possono accedere alle risorse di cui hanno bisogno per eseguire la loro funzione. [31](#), [32](#)

framework Piattaforma che funge da strato intermedio tra un sistema operativo e il *software* che lo utilizza. [7](#), [29](#), [30](#), [36](#), [62](#), [65](#), [66](#)

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. [61](#)

superset Un linguaggio di programmazione che contiene tutte le funzionalità di un determinato linguaggi, però ampliandolo o migliorandolo per includere anche altre funzionalità. [29](#)

will Intenzione di un utente di fare sport. [1–3](#), [22–24](#), [26](#), [48](#), [52–54](#)

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. [69](#)

CRUD Operazioni di base che possono essere svolte su un *database*. In particolare, sono creazione (Create), lettura (Read), modifica/aggiornamento (Update) ed eliminazione (Delete). [69](#)

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). [69](#)

EntityManager API che gestisce il ciclo di vita delle istanze di entità in un *database*. [42](#)

Eureka Server Applicazione che contiene le informazioni su tutte le applicazioni *client-service*. Ogni **microservizio** si registra nel *server* Eureka ed esso conosce tutte le applicazioni in esecuzione su ciascuna porta e indirizzo IP. [36, 47](#)

ICT Insieme dei metodi e delle tecniche utilizzate nella trasmissione, ricezione ed elaborazione di dati e informazioni. [69](#)

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. [69](#)

JPA *Framework* che offre delle **API** per aiutare gli sviluppatori nelle operazioni di persistenza dei dati su un *database* relazionale. [69](#)

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. [69](#)

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. [6–8, 32, 35, 36, 42, 44, 47, 61, 65, 67](#)

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. [69](#)

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. [29](#)

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. [69](#)

Acronimi

API Application Program Interface. 3, 35, 36, 40, 66–68

CRUD Create Read Update Delete. 30, 42

DAO Data Access Object. 42

ICT Information and Communications Technology. 1

IP Internet Protocol address. 32, 36, 67

JPA Java Persistence API. 42

JSP JavaServer Pages. 32

SPA Single-Page Application. 30

UML Unified Modeling Language. 9

URL Uniform Resource Locator. 38

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. 30).
- [2] *Cosa sono i microservizi?* URL: <https://aws.amazon.com/it/microservices/> (cit. a p. 35).
- [3] *Dependency injection in Angular.* URL: <https://angular.io/guide/dependency-injection> (cit. a p. 37).
- [4] *Feature modules.* URL: <https://angular.io/guide/feature-modules> (cit. a p. 38).
- [5] *Introduction to Angular concepts.* URL: <https://angular.io/guide/architecture> (cit. a p. 32).
- [6] *La Dependency Injection in Spring.* URL: <https://daviooh.com/blog/2017/08/19/spring-dependency-injection> (cit. a p. 37).
- [7] *Lazy-loading feature modules.* URL: <https://angular.io/guide/lazy-loading-ngmodules> (cit. a p. 38).
- [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. 32).
- [10] *Node.js.* URL: <https://whatism.techttarget.com/definition/Nodejs#:~:text=James%20Denman-,Node.,feeds%20and%20web%20push%20notifications> (cit. a p. 30).
- [11] *Pattern: API Gateway.* URL: <https://microservices.io/patterns/apigateway.html> (cit. a p. 32).
- [12] *Pattern: Service registry.* URL: <https://microservices.io/patterns/service-registry.html> (cit. a p. 36).
- [13] *Singleton.* URL: <https://refactoring.guru/design-patterns/singleton> (cit. a p. 38).
- [14] *Spring Boot Microservices — Developing Service Discovery.* URL: <https://aws.amazon.com/it/microservices/> (cit. a p. 36).

- [15] *Spring Framework*. URL: https://it.wikipedia.org/wiki/Spring_Framework (cit. a p. 29).
- [16] *TypeScript*. URL: <https://en.wikipedia.org/wiki/TypeScript> (cit. a p. 29).
- [17] *Using observables to pass values*. URL: <https://angular.io/guide/observables> (cit. a p. 39).
- [18] *What is Java*. URL: https://java.com/en/download/help/whatis_java.html (cit. a p. 29).