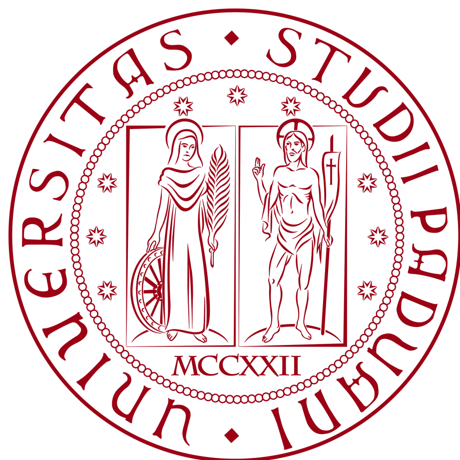
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sviluppo di un'applicazione web per il fitness con Angular e Spring integrata con un chatbot basato su Retrieval-Augmented Generation

Tesi di Laurea Triennale

Relatore

Prof. Luigi De Giovanni

Laureando

Andrea Barutta

Matricola 2042355

Anno Accademico 2023-2024

© Andrea Barutta, Luglio 2024. Tutti i diritti riservati. Tesi di Laurea Triennale: "*Sviluppo di un'applicazione web per il fitness con Angular e Spring integrata con un chatbot basato su Retrieval-Augmented Generation*", Università degli Studi di Padova, Dipartimento di Matematica "Tullio Levi-Civita".

"I can't change the direction of the wind, but I can adjust my sails."

— Jimmy Dean

Ringraziamenti

Desidero esprimere la mia gratitudine al professor Luigi De Giovanni, mio relatore, per l'aiuto e il sostegno che mi ha dato durante la stesura dell'elaborato.

Desidero esprimere un affettuoso ringraziamento ai miei genitori, alla mia famiglia ed in particolare, alla mia nonna, per il loro sostegno incondizionato, la loro costante presenza e la buonissima pasta.

Desidero esprimere la mia gratitudine ai miei cari amici, Alex ed Alessandro, così come al mio allenatore e amico Leandro, e alla mia squadra di pallavolo per avermi supportato e sopportato.

Padova, Luglio 2024 *Andrea Barutta*

Sommario

La tesi descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Barutta Andrea presso l'azienda *Sync Lab S.r.l.* L'attività di stage si è concentrata principalmente sullo studio dei *framework Angular* e *Spring*, accompagnato dallo sviluppo di un'applicazione web *full stack*. Il progetto prevedeva la creazione di una fitness web app, concepita come assistente per il monitoraggio dei progressi e la gestione di programmi sportivi e di allenamento.
La tesi esamina i processi e le attività svolte durante il periodo di stage per il raggiungimento degli obiettivi pianificati. Presenta il contesto operativo e metodologico in cui lo stage è stato condotto, seguito da un'analisi tramite metodologie *Agile* dei requisiti funzionali riguardanti l'applicazione sviluppata. La tesi discute le scelte progettuali più rilevanti del progetto, per poi illustrare le tecniche di verifica e validazione implementate. Infine, fornisce un'analisi consuntiva e una valutazione personale sullo svolgimento e i risultati dello stage.

Convenzioni tipografiche

Riguardo la stesura del testo 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*$_{\boldsymbol{G}}$;
- i termini riportati nel glossario presenti nel testo avranno un collegamento diretto alla loro definizione;
- i termini in lingua straniera o appartenenti al gergo tecnico sono evidenziati con il carattere *corsivo* quando non sono presenti nei titoli;
- i nomi di classi o metodi menzionati nel testo sono evidenziati con il carattere *corsivo*.

Indice

1 Introduzione **1**

1.1 L'azienda 1

1.2 L'idea 1

1.3 Descrizione dello stage 2

1.3.1 Introduzione al progetto 2

1.3.2 Obiettivi 3

1.3.3 Pianificazione del lavoro 4

1.3.3.1 Ripartizione oraria 4

1.3.3.2 Pianificazione settimanale 4

1.3.4 Analisi preventiva dei rischi 6

1.4 Organizzazione del testo 7

2 Processi e metodologie **8**

2.1 Modello di sviluppo 8

2.1.1 Agile 8

2.1.2 Framework SCRUM 9

2.1.3 Strumenti 10

2.1.3.1 Strumenti di comunicazione 10

2.1.3.2 Strumenti organizzativi 10

3 Analisi dei requisiti **12**

3.1 User stories 13

3.1.1 Codice identificativo 13

3.1.2 Tabella delle user stories 13

4 Tecnologie e strumenti **17**
4.1 Strumenti . . . 17
4.1.1 Visual Studio Code . . . 17
4.1.2 GitHub . . . 17
4.1.3 Postman . . . 18
4.1.4 StopLight . . . 18
4.1.5 MailDev . . . 18
4.2 Tecnologie . . . 19
4.2.1 Java . . . 19
4.2.2 TypeScript . . . 19
4.2.3 Maven . . . 19
4.2.4 Spring . . . 19
4.2.5 Spring Boot . . . 20
4.2.6 Spring Data JPA . . . 20
4.2.7 Spring Security . . . 20
4.2.8 PostgreSQL Driver . . . 21
4.2.9 Swagger API . . . 21
4.2.10 Lombok . . . 21
4.2.11 Docker . . . 21
4.2.12 Angular . . . 22

5 Progettazione e codifica **23**
5.1 Architettura . . . 23
5.1.1 Multi-layered architecture . . . 25
5.2 Backend . . . 26
5.2.1 Architettura RESTful . . . 26
5.2.1.1 Componente "Controller" . . . 27
5.2.1.2 Componente "Service" . . . 30
5.2.1.3 Componente "Repository" . . . 32
5.2.1.4 Componente "Model (Entity)" . . . 32
5.2.2 Struttura delle tabelle relazionali . . . 33
5.2.3 Progettazione servizi REST risorse . . . 34

5.2.3.1 Convenzione di configurazione degli endpoint in Spring Boot . . . 35
5.2.3.2 Componenti . . . 36
5.2.3.3 Operazioni CRUD . . . 36
5.2.3.4 Pattern DTO . . . 37
5.2.3.5 Validazione DTO . . . 38
5.2.3.6 Mapper . . . 39
5.2.3.7 Gestione degli errori . . . 40
5.2.3.8 Auditing . . . 40
5.2.3.9 UserExtractor . . . 41
5.2.4 Servizi REST risorsa *Utente* . . . 41
5.2.4.1 Entità *Utente* . . . 41
5.2.4.2 Entità *Ruolo* . . . 42
5.2.4.3 Entità *Token* . . . 43
5.2.4.4 Servizio REST di autenticazione/registrazione . . . 43
5.2.4.5 Autenticazione e autorizzazione . . . 45
5.2.4.6 Servizio REST per operazioni di utilità sul profilo utente 48
5.2.5 Servizi REST risorsa *Esercizio* . . . 49
5.2.6 Servizi REST risorsa *Allenamento* . . . 51
5.2.7 Servizi REST risorsa *AllenamentoEsercizio* . . . 51
5.2.8 Servizi REST risorsa *Progresso* . . . 52
5.2.9 Servizi REST risorsa *Periodo* . . . 53
5.2.10 Servizi REST risorsa *PeriodoAllenamento* . . . 54
5.2.11 Servizi REST risorsa *Notifica* . . . 54
5.3 Frontend . . . 55
5.3.1 Component-Based Architecture . . . 55
5.3.2 Moduli Angular . . . 56
5.3.3 Angular HttpInterceptor . . . 56
5.3.4 Angular Route Guards . . . 56
5.3.5 Generazione dei Servizi per le richieste al backend . . . 57
5.3.6 Gestione feedback . . . 58
5.3.6.1 Gestione degli errori . . . 59

5.3.6.2 Classe *MessageHandler* 59
5.3.7 Standalone e lazy loading 59
5.3.8 Angular Reactive Form 60
5.3.9 Maschere frontend 60
5.3.9.1 Home page, registrazione, conferma email e autenticazione 61
5.3.9.2 Visualizzazione homepage utente autenticato 62
5.3.9.3 Visualizzazione notifiche 63
5.3.9.4 Creazione e modifica esercizio 63
5.3.9.5 Visualizzazione lista esercizi 64
5.3.9.6 Visualizzazione store esercizi 65
5.3.9.7 Creazione e modifica allenamento 65
5.3.9.8 Visualizzazione lista allenamenti e dettagli 66
5.3.9.9 Creazione e modifica periodo 67
5.3.9.10 Visualizzazione lista periodi e dettagli 68
5.3.9.11 Creazione, modifica e visualizzazioni progressi 69
5.3.9.12 Modifica informazioni personali 69
5.3.9.13 Visualizzazione piano alimentare personalizzato . . . 70
5.3.9.14 Maschera per il chatbot 71
5.4 Integrazione con il chatbot 71
5.4.1 Il chatbot 72
5.4.2 Allenamenti e piani alimentari personalizzati 72

6 Verifica e validazione **73**
6.1 Strumenti 73
6.1.1 GitHub Actions 73
6.1.2 SonarCloud 74
6.1.3 JaCoCo 74
6.1.4 Coveralls 74
6.1.5 JUnit 5 75
6.1.6 Mockito 75
6.2 Verifica 75

6.2.1 Test di unità . . . 75
6.2.2 Test di integrazione . . . 76
6.2.3 Analisi statica . . . 76
6.2.4 Processo di integrazione nel *branch* principale . . . 76
6.3 Validazione . . . 77
6.3.1 Stato user stories . . . 78

7 Conclusioni **79**
7.1 Raggiungimento degli obiettivi . . . 79
7.2 Consuntivo orario . . . 81
7.3 Conoscenze acquisite . . . 82
7.4 Valutazione personale . . . 82

Acronimi e abbreviazioni **83**

Glossario **84**

Sitografia **88**

Elenco delle figure

5.1 Architettura di sistema 24
5.2 Multi-layered architecture 25
5.3 Componenti architettura RESTful [22] 26
5.4 Diagramma entità relazione 34
5.5 UML entità utente 42
5.6 Diagramma UML Servizio REST di autenticazione/registrazione . . . 44
5.7 Pipeline di autenticazione/autorizzazzione 45
5.8 Diagramma UML delle classi adibite alle operazioni di utilità sul profilo utente 48
5.9 Diagramma UML servizio REST risorsa *Esercizio* 50
5.10 Diagramma UML servizio REST risorsa *Allenamento* 51
5.11 Diagramma UML servizio REST risorsa *AllenamentoEsercizio* 52
5.12 Diagramma UML servizio REST risorsa *Progresso* 52
5.13 Diagramma UML servizio REST risorsa *Periodo* 53
5.14 Diagramma UML servizio REST risorsa *PeriodoAllenamento* 54
5.15 Diagramma UML servizio REST risorsa *Notifica* 55
5.16 Visualizzazione *feedback* 58
5.17 Maschere di home page, registrazione, conferma email e autenticazione 62
5.18 Home page utente autenticato 62
5.19 Visualizzazione notifiche 63
5.20 Maschera di creazione e modifica esercizi 64
5.21 Visualizzazione lista esercizi 64
5.22 Visualizzazione *store* esercizi 65
5.23 Creazione e modifica allenamento 66

5.24 Visualizzazione lista allenamenti e dettagli 67
5.25 Creazione e modifica periodo 68
5.26 Visualizzazione lista periodi e dettagli 68
5.27 Maschere di creazione,modifica e visualizzazione progressi. 69
5.28 Maschera informazioni personali 70
5.29 Maschera piano alimentare 70
5.30 Maschera *chatbot* 71

6.1 File `README` della *repository* di progetto 74

Elenco delle tabelle

1.1 Ripartizione oraria delle attività 4

3.1 Tabella del tracciamento dei requisiti funzionali tramite *User stories*. 16

6.1 Stato *User stories*. 78

7.1 Stato obiettivi piano di lavoro. 80
7.2 Consuntivo orario delle attività 81

Capitolo 1

Introduzione

Questo capitolo introduttivo fornisce una breve presentazione dell'azienda ospitante e del progetto di stage della durata di otto settimane.

1.1 L'azienda

Sync Lab [2] è un'azienda nata a Napoli nel 2002 come software house, che ha rapidamente esteso la sua presenza nel settore dell'*Information and Communications Technology* (ICT). Grazie a una crescente esperienza tecnologica, metodologica e applicativa nel campo del software, l'azienda si è trasformata in un *System Integrator*, guadagnando importanti quote di mercato nei settori *mobile*, videosorveglianza e sicurezza delle infrastrutture informatiche aziendali.
Attualmente, Sync Lab serve una vasta clientela diretta e finale, con un team di oltre 300 dipendenti distribuiti tra sei sedi in Italia.
La missione di Sync Lab è fornire soluzioni tecnologiche avanzate e di alta qualità, rispondendo alle esigenze di una vasta gamma di settori e clienti.

1.2 L'idea

Il progetto di stage consiste nello studio, utilizzo e comprensione di tecnologie all'avanguardia per lo sviluppo web, quali *Spring* per il *backend* e *Angular* per il *frontend*.
A tal fine, è stato deciso di sviluppare un'applicazione web per il fitness, concepita come un vero e proprio assistente per la gestione degli allenamenti e del benesse-

re fisico. Questo obiettivo viene ulteriormente arricchito grazie all'integrazione con un altro progetto di stage che si focalizza sullo sviluppo di un *chatbot* basato su Retrieval-Augmented Generation (RAG)***G*** . Questo *chatbot* è in grado di rispondere alle domande degli utenti utilizzando le loro informazioni personali e di generare automaticamente programmi di allenamento e piani alimentari personalizzati.

1.3 Descrizione dello stage

In questa sezione vengono descritti gli obiettivi e le attività dello stage, come previste dal piano di lavoro.

1.3.1 Introduzione al progetto

Il progetto di stage consiste nello sviluppo di un'applicazione web utilizzando le tecnologie citate nella Sezione 1.2, con l'obiettivo di consolidare le conoscenze acquisite durante lo studio preliminare delle stesse.
Questa applicazione intende essere un assistente personale per il fitness, permettendo agli utenti di registrare i propri esercizi***G*** , allenamenti***G*** e periodi di allenamento***G*** , oltre che i propri progressi***G*** . L'applicazione consentirà inoltre una facile visualizzazione degli allenamenti su di un calendario, basata sul periodo attivo***G*** , e l'invio di notifiche in-app relative agli allenamenti da svolgere.
Per favorire il supporto e la collaborazione tra gli utenti, sarà possibile condividere i propri esercizi con altri utenti, permettendo loro di importarli e integrarli nei propri programmi di allenamento. A tal fine, saranno sviluppate funzionalità di filtraggio per facilitare la ricerca di nuovi esercizi.
Infine, verrà effettuata l'integrazione con il progetto di stage di un compagno di corso che svilupperà un *chatbot* basato su Retrieval-Augmented Generation (RAG)***G*** .
Questo *chatbot* sarà in grado di rispondere alle domande degli utenti basandosi sulle loro informazioni personali e di generare automaticamente programmi di allenamento e piani alimentari personalizzati.

1.3.2 Obiettivi

Si farà riferimento agli obiettivi secondo le seguenti notazioni:

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

Le sigle appena indicate saranno seguite da una coppia sequenziale di numeri per identificare l'obiettivo.
Si prevede lo svolgimento dei seguenti obiettivi:

- **Obbligatori**:
  - **O-01**: acquisizione competenze approfondite su tecnologie moderne per lo sviluppo di Web App;
  - **O-02**: capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma;
  - **O-03**: nella fitness Web App realizzata, l'utente deve poter effettuare la registrazione e autenticazione nel sistema così da poter definire il proprio stato atletico e di salute con relativi obiettivi di miglioramento, oltre che tracciare i propri progressi (variazione di peso, percentuale di grasso corporeo e della massa muscolare);
  - **O-04**: nella fitness Web App realizzata, l'utente deve poter creare, gestire e pianificare le schede di allenamento;
  - **O-05**: nella fitness Web App realizzata, l'utente deve poter creare e gestire esercizi personalizzati per la progettazione delle schede di allenamento;
  - **O-06**: nella fitness Web App realizzata, l'utente deve poter ricevere promemoria di allenamento (notifiche in-app oppure notifiche *push*).
- **Desiderabili**:

  - **D-01**: inserimento di ulteriori funzionalità alla Web App come la possibilità di ricevere supporto da altri utenti;
  - **D-02**: integrazione nella Web App di un chat-bot per l'ottenimento di consigli personalizzati sviluppato tramite *Retrieval-Augmented Generation* come parte di un progetto di Stage parallelo.

- **Facoltativi**:
  - **F-01**: inserimento di ulteriori funzionalità alla Web App come la gestione del piano alimentare settimanale.

1.3.3 Pianificazione del lavoro

1.3.3.1 Ripartizione oraria

Nella Tabella 1.1 è presente la ripartizione oraria preventivata delle attività.

Durata oraria	Descrizione attività
100	Formazione sulle tecnologie
180	Fase di sviluppo
30	Progettazione dell'architettura
70	Sviluppo del *backend* con *Spring*
60	Sviluppo del *frontend* con *Angular*
20	Testing e debugging
20	Collaudi ed integrazioni finali
300 totali	

Tabella 1.1: Ripartizione oraria delle attività

1.3.3.2 Pianificazione settimanale

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

- **Prima settimana (40 ore)**:
  - incontro con persone coinvolte nel progetto per discutere i requisiti e le richieste relativamente al sistema da sviluppare;

  - presentazione strumenti di lavoro per la condivisione del materiale di studio e per la gestione dell'avanzamento;
  - condivisione scaletta di argomenti;
  - ripasso concetti Metodologia Agile/Scrum;
  - ripasso principi SOLID e *Clean Coding*.

- **Seconda settimana (40 ore)**:
  - studio principi generali di *Spring Core* (IOC (*Inversion of control*), Dependency Injection);
  - studio *SpringBoot*;
  - studio *Spring Data/DataRest*;
  - realizzazione prototipo di servizio REST con metodi GET/POST/PUT/-DELETE;
  - studio generale teorico delle Application Programming Interface (API)$_G$ *Google Gemini* e *OpenAI-ChatGPT*;
  - studio tool di lavoro *StopLight*.

- **Terza settimana (40 ore)**:
  - ripasso linguaggio *Javascript*;
  - studio del linguaggio *TypeScript* e Framework$_G$ *Angular*.

- **Quarta settimana (40 ore)**:
  - studio integrazione *Angular Material*;
  - realizzazione prototipo maschera di login/registrazione con dati cablati sul *frontend*.

- **Quinta settimana (40 ore):**
  - progettazione servizi web dei tracciati REST su *StopLight*;
  - realizzazione maschere di login/registrazione con chiamate a *StopLight*;
  - realizzazione maschere di profilazione con chiamate a *StopLight*.

- **Sesta settimana (40 ore):**
  - realizzazione maschere di avanzamento programma con chiamate a *StopLight*.
- **Settima settimana (40 ore):**
  - realizzazione maschere di gestione delle schede di lavoro con chiamate a *StopLight*;
  - implementazione notifiche di avanzamento;
  - integrazione con il *backend* effettivo.
- **Ottava settimana (20 ore):**
  - collaudi ed integrazioni finali.

1.3.4 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi, con particolare attenzione a quelli derivanti dallo sviluppo collaborativo dell'applicazione, nonostante le funzionalità siano nettamente separate tra i membri. Pertanto, è stata avviata un'elaborazione di soluzioni per gestire efficacemente tali rischi.

1. Conflitti nel lavoro collaborativo

Descrizione: questo rischio si presenta quando diversi membri del team lavorano sullo stesso progetto contemporaneamente, portando a sovrapposizioni e conflitti nel codice sorgente.

Soluzione: adottare il flusso di lavoro offerto dal modello *GitFlow*, che fornisce una struttura e delle linee guida per gestire in modo efficiente il versionamento, il *branching* e il *merging* del codice, facilitando la collaborazione e minimizzando i conflitti.

2. Conflitti ambiente di sviluppo

Descrizione: questo rischio si verifica quando i membri del team utilizzano ambienti di sviluppo diversi, con configurazioni e dipendenze eterogenee, portando a discrepanze nell'esecuzione e nel test del codice.

Soluzione: adottare *Docker* per creare *container* standardizzati che includano tutte le dipendenze e le configurazioni necessarie per eseguire l'applicazione in qualsiasi

ambiente. Questo permette di eliminare le differenze tra i vari ambienti di sviluppo e di garantire coerenza nel processo di sviluppo e distribuzione.

3. Disallineamento tra Sviluppo *Backend* e *Frontend*

Descrizione: questo rischio si verifica quando lo sviluppo del *backend* e del *frontend* non è ben coordinato, portando a discrepanze nei dati e nell'interfaccia utente oltre che ritardi nel piano di lavoro.

Soluzione: adottare l'approccio *API-driven development*, dove, successivamente ad un accurata progettazione delle API, lo sviluppo del *backend* e del *frontend* avviene in parallelo. Questo permette una maggiore coerenza nei dati e nelle funzionalità offerte dall'applicazione.

1.4 Organizzazione del testo

Il secondo capitolo approfondisce i processi e le metodologie adottate durante il percorso.

Il terzo capitolo approfondisce le funzionalità attese dell'applicazione da sviluppare.

Il quarto capitolo approfondisce le tecnologie e gli strumenti utilizzati per la progettazione e la codifica dell'applicazione web.

Il quinto capitolo approfondisce la progettazione e la codifica relativi alla realizzazione del progetto.

Il sesto capitolo approfondisce le attività di verifica e validazione svolte.

Il settimo capitolo descrive le conclusioni e le considerazioni finali.

Capitolo 2

Processi e metodologie

Questo capitolo fornisce il contesto operativo e metodologico in cui è stato condotto lo stage, oltre a offrire una visione chiara delle strategie implementate per raggiungere gli obiettivi prefissati.

2.1 Modello di sviluppo

In questa sezione, analizzeremo il modello di sviluppo impiegato durante il progetto di stage, finalizzato a semplificare la gestione del progetto e garantire il raggiungimento degli obiettivi stabiliti.

2.1.1 Agile

Il metodo *Agile* [25], è un modello di sviluppo software che si basa su iterazioni continue ed incrementali con gli stakeholders$_{G}$, che vengono costantemente coinvolti nello sviluppo del prodotto per garantire che soddisfi le esigenze e le aspettative di chi lo utilizzerà o ne beneficerà. Non consiste in un approccio classico e lineare, ma si basa sulla possibilità di realizzare un progetto per fasi, chiamate "*Sprint*".
Il principio alla base di questo metodo è dare priorità allo sviluppo del software, concentrando le risorse umane nella realizzazione del prodotto piuttosto che sulla documentazione e sulla pianificazione. Per questo motivo, l'obiettivo principale del metodo *Agile* è conseguire in modo incrementale il prodotto desiderato, aggiungen-

do man mano nuove funzionalità e soddisfacendo sempre più requisiti fino al loro completamento.

2.1.2 Framework SCRUM

Nell'ambito del progetto di stage è stato utilizzato *Scrum* [10] come framework *Agile* per la gestione del progetto. *Scrum*, definisce e coordina una serie di eventi, artefatti e ruoli per aiutare i team a strutturare e gestire i progetti.

Ruoli

- ***Product Owner***: responsabile dei profitti o delle perdite generate dal progetto, decide quali attività del *Product Backlog* verranno svolte per prime;
- ***Scrum Master***: facilita il gruppo a risolvere problemi e quindi ad essere efficace nel raggiungere gli obiettivi;
- ***Developer***: sviluppatore, costruisce le funzionalità del *Product Backlog* indicate dal *Product Owner*.

Artefatti

- ***Product Backlog***: elenco delle attività complessive da svolgere;
- ***Sprint Backlog***: elenco di attività da svolgere in un solo *Sprint*;
- ***Increment***: il risultato di uno *Sprint*, ovvero un incremento del prodotto che deve essere utilizzabile e, di norma, potenzialmente rilasciabile.

Eventi

- ***Sprint***: un ciclo di lavoro con durata fissa, solitamente di due-quattro settimane, durante il quale un incremento del prodotto deve essere completato;
- **Sprint Planning**: avviene all'inizio dello *Sprint*, e permette la definizione dello *Sprint backlog*;
- ***Daily Scrum***: una riunione quotidiana di circa 15 minuti dove gli sviluppatori discutono dei progressi;

- ***Sprint Review***: sessione di revisione alla fine dello *Sprint* in cui il team mostra ciò che è stato realizzato e riceve feedback;
- ***Sprint Retrospective***: riunione dopo lo *Sprint Review* in cui il team riflette su ciò che è andato bene, ciò che può essere migliorato e definisce azioni per l'incremento di miglioramento.

Nell'ambito del progetto di stage, il tutor aziendale ha ricoperto i ruoli di *Product Owner* e *Scrum Master*, mentre il team di sviluppo era composto dal sottoscritto e da un compagno di corso, incaricato principalmente della parte di sviluppo del *chatbot* RAG. Regolarmente, il team di sviluppo si riuniva per i *Daily Scrum meeting* con lo scopo di sincronizzare le attività in corso. La durata dello *Sprint*, fissata a **una settimana**, è rimasta costante per l'intera durata dello stage, ed alla fine di ogni *Sprint*, si tenevano le riunioni di *Sprint Review* e *Sprint Retrospective*. Durante queste riunioni, insieme al tutor aziendale, venivano valutati i progressi del prodotto e si rifletteva sui processi di sviluppo adottati oltre che su possibili modifiche del *Product backlog* al fine di individuare possibili miglioramenti. Al termine di questa fase riflessiva, si passava allo *Sprint Planning* per definire le attività da svolgere nello *Sprint* successivo ed impostare il nuovo *Sprint backlog*.

2.1.3 Strumenti

2.1.3.1 Strumenti di comunicazione

Come principale strumento di comunicazione, per svolgere i *Daily scrum meeting* ed eventuali riunioni non programmate, è stato utilizzato **Discord**. *Discord* [9] è una piattaforma di comunicazione online che permette agli utenti di interagire tramite testo, voce e video.

2.1.3.2 Strumenti organizzativi

Per organizzare il lavoro sono stati utilizzati i seguenti strumenti:

- **Trello**[46]: una piattaforma di gestione progetti basata su schede e liste che consente agli utenti di organizzare e monitorare il lavoro in modo visuale. *Trello*

è stato utilizzato per la gestione e la pianificazione settimanale delle attività del progetto. Il piano di lavoro è stato adeguatamente trasferito sulla piattaforma, con le singole attività divise nelle schede settimanali, consentendo una rapida gestione e visualizzazione dello stato di avanzamento;

- ***GitHub* Issue Tracking System$_G$ (ITS)**[45]: una funzionalità integrata nella piattaforma *GitHub* che consente agli utenti di tenere traccia delle problematiche, dei bug, delle richieste di funzionalità e di altri elementi di lavoro all'interno di un progetto software.

  GitHub ITS è stato utilizzato per la gestione collaborativa dei ticket$_G$ riguardanti lo sviluppo del progetto di stage, e per il monitoraggio degli *Sprint e product backlog*. Questo strumento ci ha permesso di creare diverse milestone$_G$ per monitorare lo stato di avanzamento dei ticket associati al soddisfacimento dei requisiti obbligatori, opzionali e desiderabili.

Per prenotare la propria presenza nella sede di Padova, è stato utilizzato *Google Calendar*. In questo modo, è stato possibile notificare la propria presenza per fini organizzativi.

Capitolo 3

Analisi dei requisiti

In questo capitolo verrà effettuata un analisi delle funzionalità richieste all'applicazione sviluppata durante il percorso di stage, facendo uso di un approccio orientato all'utente.
Per lo studio delle aspettative funzionali del progetto sviluppato si è deciso di utilizzare le *user stories* [50]. Le *user stories* sono brevi descrizioni scritte dal punto di vista dell'utente finale che esprimono ciò che l'utente vuole fare con il sistema. Ogni *user story* ha lo scopo di catturare un requisito funzionale in modo semplice e comprensibile, seguendo la struttura:

"Come [tipo di utente], voglio [azione o caratteristica] per [beneficio o valore]".

Si utilizzano principalmente in contesti di sviluppo software *Agile* per mantenere il focus sulle esigenze degli utenti e per facilitare la comunicazione tra i membri del team e gli stakeholders.

3.1 User stories

In questa sezione verrà definita la struttura dei codici identificativi e saranno presentate le diverse *user stories*.

3.1.1 Codice identificativo

Ogni *user stories* sarà associata ad un codice cosi strutturato:

U[*Numero incrementale*] **-** [*Identificativo di priorità*]

Con l' *Identificativo di priorità* che sarà rappresentato con:

O: obbligatorio;

D: desiderabile;

F: facoltativo.

3.1.2 Tabella delle user stories

Le *user stories* e gli identificativi univoci associati sono presentati nella Tabella 3.1.

Codice	User story
U1-O	Come un nuovo utente, voglio registrarmi all'applicazione web inserendo nome, cognome, data di nascita, username, indirizzo email e password, così che possa creare un account e accedere ai servizi.
U2-O	Come un nuovo utente, voglio ricevere un'email di conferma dopo la registrazione contenente un codice di attivazione da inserire nella pagina indicata sulla email, così che possa verificare il mio indirizzo email e attivare il mio account.
U3-O	Come utente registrato, voglio poter effettuare il login inserendo il mio username e password, così che possa accedere al mio account e utilizzare i servizi offerti.
	Continua nella prossima pagina...

Tabella 3.1 – Continua

Codice	User story
U4-O	Come utente autenticato, voglio poter modificare la mia password, così che possa reimpostarla successivamente alla registrazione.
U5-O	Come utente autenticato, voglio poter modificare le mie informazioni personali come nome, cognome e data di nascita, cosi da poterle correggere in caso di errori durante la registrazione.
U6-O	Come utente autenticato, voglio poter registrare, modificare, visualizzare o eliminare con facilità i miei progressi in peso, altezza, massa grassa, massa muscolare e le note associate, per monitorare il mio andamento nel tempo.
U7-O	Come utente autenticato, voglio visualizzare in forma tabellare e in un grafico a linee l'andamento del mio peso confrontato al mio peso ideale e della mia massa grassa e muscolare, per monitorare i miei progressi.
U8-O	Come utente autenticato, voglio avere il pieno controllo sui miei esercizi personalizzati, potendo crearli, modificarli, visualizzarli ed eliminarli a mio piacimento. Ogni esercizio deve includere nome, descrizione, categoria ed immagine rappresentativa, per agevolare la gestione dei miei piani di allenamento e ottimizzare l'esperienza di workout.
U9-D	Come utente autenticato, voglio poter pubblicare i miei esercizi in uno store dedicato all'interno dell'app, in modo da poter condividere le mie idee.
U10-D	Come utente autenticato, voglio poter importare gli esercizi condivisi da altri utenti tra i personali, in modo da poterli usare nei miei allenamenti.
U11-D	Come utente autenticato, voglio poter filtrare gli esercizi per categoria nello store, in modo da rendere rapida la ricerca degli esercizi di mio interesse.

Continua nella prossima pagina...

Tabella 3.1 – Continua

Codice	User story
U12-O	Come utente autenticato, voglio avere il pieno controllo sui miei allenamenti, potendo crearli, modificarli, visualizzarli ed eliminarli a mio piacimento. Ogni allenamento deve includere nome, descrizione, durata ed una lista di esercizi associati con relative ripetizioni, serie e recupero, per agevolare la gestione dei miei piani di allenamento e ottimizzare l'esperienza di workout.
U13-O	Come utente autenticato, voglio avere il pieno controllo sui miei periodi di allenamento, potendo crearli, modificarli, visualizzarli ed eliminarli a mio piacimento. Ogni periodo deve includere nome, obiettivo, durata del ciclo, data di inizio, data di fine e relativi allenamenti. Ogni allenamento del periodo deve specificare il giorno di esecuzione rispetto alla durata del ciclo e il momento della giornata (mattina, pomeriggio o sera), così da pianificare efficacemente il percorso atto al raggiungimento dei miei obiettivi di fitness.
U14-O	Come utente autenticato, voglio poter attivare un periodo di allenamento selezionandolo dalla lista dei periodi disponibili, per avviare il periodo, ricevere notifiche e visualizzare gli allenamenti in un agenda presente nella home page.
U15-O	Come utente autenticato, voglio ricevere notifiche in-app sugli allenamenti da svolgere nella data odierna in base al periodo di allenamento attivo, per ricevere promemoria tempestivi e rimanere aggiornato sulle attività da svolgere.
U16-D	Come utente autenticato, voglio poter interagire con un *chatbot* intelligente che risponda sulla base del mio contesto di fitness per ricevere consigli personalizzati, supporto e motivazione per raggiungere i miei obiettivi.
	Continua nella prossima pagina...

Tabella 3.1 – Continua

Codice	**User story**
U17-D	Come utente autenticato, voglio poter far generare al *chatbot* allenamenti personalizzati e poterli modificare prima di salvarli definitivamente, in modo da rendere più rapida la creazione di allenamenti e ottenere idee.
U18-F	Come utente autenticato, voglio poter far generare al *chatbot* piani alimentari settimanali personalizzati sulla base delle mie esigenze e preferenze alimentari, e poterli stampare per averli sempre a portata di mano.

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali tramite *User stories*.

Capitolo 4

Tecnologie e strumenti

In questo capitolo viene data una panoramica delle tecnologie e degli strumenti utilizzati per lo sviluppo e la progettazione dell'applicazione web.

4.1 Strumenti

In questa sezione vengono descritti gli strumenti utilizzati per i processi di progettazione e codifica dell'applicazione web. Gli strumenti sono software o applicazioni che facilitano il lavoro degli sviluppatori utilizzando le tecnologie prescelte.

4.1.1 Visual Studio Code

Visual Studio Code [51] è un *editor* di codice sorgente leggero, veloce e altamente personalizzabile sviluppato da *Microsoft*. È stato utilizzato per lo sviluppo sia *backend* che *frontend*.

4.1.2 GitHub

GitHub [12] è una piattaforma di hosting per repository *Git*, ampiamente utilizzata dagli sviluppatori per la collaborazione e la gestione del codice sorgente dei progetti software. Offre strumenti per il controllo del codice sorgente, la gestione delle modifiche, il tracciamento dei problemi e la collaborazione tra team di sviluppo. È stata creata un organizzazione *GitHub* per contenere la *repository* di progetto e, come definito alla Sezione 1, si è optato per mantenere *GitFlow* come flusso di lavoro.

GitHub si è inoltre utilizzato come ITS e per la creazione della pipeline di Continuous Integration (CI)$_{G}$.

4.1.3 Postman

Postman [30] è una popolare piattaforma di sviluppo API che offre una suite completa di strumenti per la creazione, il test e la documentazione delle API. Con *Postman*, gli sviluppatori possono facilmente inviare richieste Hypertext Transfer Protocol (HTTP)$_{G}$ a qualsiasi endpoint$_{G}$, visualizzare e analizzare le risposte, automatizzare i test delle API e condividere la documentazione con il team. *Postman* è stato utilizzato per testare il corretto funzionamento degli endpoint sviluppati.

4.1.4 StopLight

Stoplight [40] è uno strumento *open-source* utilizzato per la progettazione e la documentazione delle API REST dell'applicazione. Questo strumento si è rivelato fondamentale nella fase iniziale del progetto, consentendoci di progettare preventivamente le API REST da sviluppare e di implementare componenti del *frontend* prima della realizzazione effettiva delle API stesse. Una volta completato lo sviluppo delle API, *Stoplight* è stato sostituito con uno strumento per la creazione automatica della documentazione, descritto alla Sezione 4.2.9.

4.1.5 MailDev

MailDev [23] è un'applicazione *open-source* che fornisce un server SMTP$_{G}$ e un visualizzatore di posta elettronica locale per lo sviluppo e il testing delle funzionalità di invio di email. È progettato per consentire agli sviluppatori di testare le email inviate dalle proprie applicazioni senza dover effettivamente inviare email reali ai destinatari. *MailDev* cattura le email inviate al server SMTP locale e le visualizza in un'interfaccia web *user-friendly*, consentendo agli sviluppatori di esaminare il contenuto delle email, controllare gli allegati e verificare il corretto funzionamento della logica di invio delle email delle proprie applicazioni. *MailDev* è stato utilizzato per testare l'invio delle mail di conferma successive alla registrazione di un nuovo utente.

4.2 Tecnologie

In questa sezione vengono descritte le tecnologie utilizzate per i processi di progettazione e codifica dell'applicazione web. Le tecnologie rappresentano i principi e le metodologie che costituiscono le fondamenta dello sviluppo software. Definiscono come i software vengono creati, le loro caratteristiche e le loro capacità.

4.2.1 Java

Java [19] è un linguaggio di programmazione versatile e orientato agli oggetti, rinomato per la sua portabilità e indipendenza dalla piattaforma. Ciò significa che il codice *Java* può essere eseguito su diverse macchine senza la necessità di essere riscritto per garantirne la compatibilità. *Java* è il linguaggio utilizzato dal Framework di sviluppo del *backend* (*Spring Boot*).

4.2.2 TypeScript

TypeScript [48] è un linguaggio di programmazione *open-source* sviluppato da *Microsoft*. Estende *JavaScript* aggiungendo tipi statici, migliorando la robustezza e la manutenibilità del codice. *TypeScript* è il linguaggio utilizzato dal framework di sviluppo del *frontend* (*Angular*).

4.2.3 Maven

Maven [24] è uno strumento di gestione delle dipendenze che semplifica la configurazione attraverso un *file* denominato POM (*Project Object Model*). Questo *file* contiene informazioni sulla struttura del progetto, inclusi le librerie e le dipendenze necessarie per la compilazione dell'applicazione.

4.2.4 Spring

Spring [37] è un framework utilizzato per lo sviluppo di applicazioni *Java*. Offre vari strumenti che semplificano la gestione degli oggetti e delle dipendenze, rendendo

la struttura del codice più modulare e manutenibile. La sua flessibilità ha reso la creazione di microservizi più agile e veloce.

4.2.5 Spring Boot

Spring Boot [35] è un framework basato su *Spring* che semplifica il processo di avvio di un progetto, eliminando la necessità di configurazioni complesse. Consente l'inclusione e il collegamento automatico di diverse librerie, consentendo di sfruttarle simultaneamente senza la necessità di configurarle separatamente. *Spring Boot* include un server integrato basato su *Tomcat*, applica automaticamente varie tecniche di sicurezza e gestisce la filtrazione dei dati provenienti dall'esterno, garantendone l'integrità. *Spring Boot* è stato utilizzato come framework per lo sviluppo del *backend* dell'applicazione.

4.2.6 Spring Data JPA

Spring Data JPA [36] è una dipendenza di *Spring* che semplifica l'accesso ai dati persistenti di un'applicazione, facendo uso della *Java Persistence API* (JPA). Questa libreria automatizza la generazione di *query* SQL, permettendo agli sviluppatori di interagire con il database attraverso oggetti *Java*. Ciò semplifica notevolmente lo sviluppo di applicazioni basate su dati persistenti e su database relazionali, migliorando l'efficienza nella gestione delle transazioni e delle operazioni di accesso ai dati come le classiche operazioni CRUD (*Create, Read, Update, Delete*). Inoltre permette di non avere strette dipendenze con il database utilizzato facilitando quindi il cambiamento quando necessario.

4.2.7 Spring Security

Spring Security [38] è un framework potente e altamente personalizzabile per gestire la sicurezza delle applicazioni *Java*. *Spring Security* è stato utilizzato per gestire l'autenticazione degli utenti tramite token JWT$_{\boldsymbol{G}}$, verificando le credenziali di accesso e autorizzando l'accesso alle diverse risorse dell'applicazione basandosi sui ruoli e permessi assegnati.

4.2.8 PostgreSQL Driver

PostgreSQL Driver [29] è una dipendenza di Spring che permette ad un'applicazione *Java* di connettersi ad un database *PostgreSQL* utilizzando codice standard ed indipendente.

4.2.9 Swagger API

Swagger API [41] è uno strumento che consente di progettare, documentare e testare le API in modo efficiente. Con *Swagger*, è possibile creare una specifica API in formato JSON o YAML, che descrive tutte le operazioni, i parametri e le risposte supportate dall'API oltre agli indirizzi per raggiungere gli endpoint sviluppati. Questa specifica può poi essere visualizzata in modo interattivo attraverso un'interfaccia utente *Swagger UI*, che permette agli sviluppatori di esplorare e testare l'API direttamente dal browser.

Per automatizzare la generazione della documentazione Swagger, è stata utilizzata una dipendenza *Maven* chiamata *"swagger-maven-plugin"*. Questo plugin può essere configurato nel *file pom.xml* del progetto *Maven* e si occupa di analizzare il codice sorgente dell'applicazione per generare automaticamente la specifica API in base alle annotazioni *Swagger* presenti nel codice. In questo modo, la documentazione *Swagger* viene aggiornata automaticamente ogni volta che il codice sorgente viene modificato, mantenendo sempre allineata la documentazione con l'implementazione effettiva dell'API.

4.2.10 Lombok

Lombok [32] è una libreria *Java* che offre un set di annotazioni per semplificare lo sviluppo di codice riducendo la necessità di scrivere *boilerplate code*, ovvero codice ripetitivo e di supporto.

4.2.11 Docker

Docker [13] è una piattaforma che semplifica la distribuzione delle applicazioni tramite contenitori software, fornendo un ambiente isolato e portatile per eseguire le

applicazioni su diverse infrastrutture.

4.2.12 Angular

Angular [28] è un framework *open-source* per lo sviluppo di applicazioni web *frontend*, sviluppato e mantenuto da *Google*. È scritto interamente in *TypeScript* e offre un approccio basato sui componenti per la costruzione di interfacce utente dinamiche e reattive. Grazie al concetto di "*two-way data binding*", *Angular* sincronizza automaticamente i dati tra il modello e la vista, garantendo un'esperienza utente fluida e reattiva. Inoltre, fornisce un'ampia gamma di funzionalità, tra cui *routing*, gestione degli stati dell'applicazione, chiamate API HTTP e molto altro ancora, facilitando lo sviluppo di applicazioni web complesse e scalabili. Nell'ambito dello sviluppo *frontend* *Angular* sono state utilizzate le seguenti librerie:

- **Bootstrap:** [15] utilizzato per semplificare lo sviluppo del *frontend* dell'applicazione, offrendo un set di componenti e stili predefiniti che consentono di creare un'interfaccia utente moderna, *responsive*;
- **PrimeNG:** [31] offre una vasta libreria di componenti UI predefiniti e altamente personalizzabili;
- **ng-openapi-gen** [27] questo strumento automatizza la generazione del codice necessario per interagire con l'API a partire dalla specifica *OPENAPI*, inclusi servizi per le chiamate HTTP e modelli di dati corrispondenti. Ciò riduce notevolmente il lavoro manuale, migliorando la coerenza e la precisione del codice generato e accelerando il processo di sviluppo complessivo;
- **RxJS (Reactive Extensions for JavaScript):** [16] fornisce un insieme di operatori e funzioni per la gestione di flussi di dati asincroni, come eventi, timer e richieste di rete;
- **Syncfusion:** [42] facilità la generazione di un componente che permette la visualizzazione su calendario di eventi.

Capitolo 5

Progettazione e codifica

In questo capitolo sarà presentata l'architettura complessiva del sistema e saranno illustrati gli approcci comuni per la progettazione e lo sviluppo delle varie funzionalità necessarie a soddisfare i requisiti del progetto. L'obiettivo è quello di fornire una panoramica generale, trattando in dettaglio i casi particolari quando necessario.
Si precisa che il capitolo offre un'analisi dettagliata dell'architettura dell'applicazione. Qui si esamina il funzionamento e l'interazione delle varie componenti, senza entrare nei dettagli tecnici e spiegazioni delle singole tecnologie utilizzate. L'attenzione è focalizzata sulle decisioni progettuali, sui modelli architetturali adottati e sulla distribuzione delle responsabilità all'interno dell'applicazione. Verranno presentati esempi di codice che non saranno analizzati in dettaglio riga per riga, evitando spiegazioni sintattiche e strutturali già disponibili nella documentazione. Saranno forniti commenti esclusivamente sugli aspetti rilevanti e contestualmente significativi.

5.1 Architettura

L'architettura di sistema proposta si compone di tre macro-componenti principali (vedi Figura 5.1):

- ***frontend***:
  - realizzato con il framework *Angular*, è responsabile della presentazione dell'interfaccia grafica e dell'interazione con l'utente;

  - comunica con il *backend* tramite richieste HTTP per recuperare dati, inviare input e gestire le azioni degli utenti.

- ***backend***:
  - implementa il *layer* logico dell'applicazione e gestisce le operazioni sui dati;
  - espone API RESTful che il *frontend* può consumare;
  - si collega al database *PostgreSQL* per persistere e recuperare i dati;
  - gestisce le chiamate al servizio di *chatbot* RAG dopo un'elaborazione che consente l'iniezione delle informazioni di contesto dell'utente che ha richiesto il servizio.

- **chatbot RAG**:
  - un sistema di *chatbot* specializzato nell'ambito fitness basato su RAG che fornisce funzionalità di assistenza all'interno dell'app web;
  - si integra con il *backend Spring* per ricevere richieste e generare risposte.

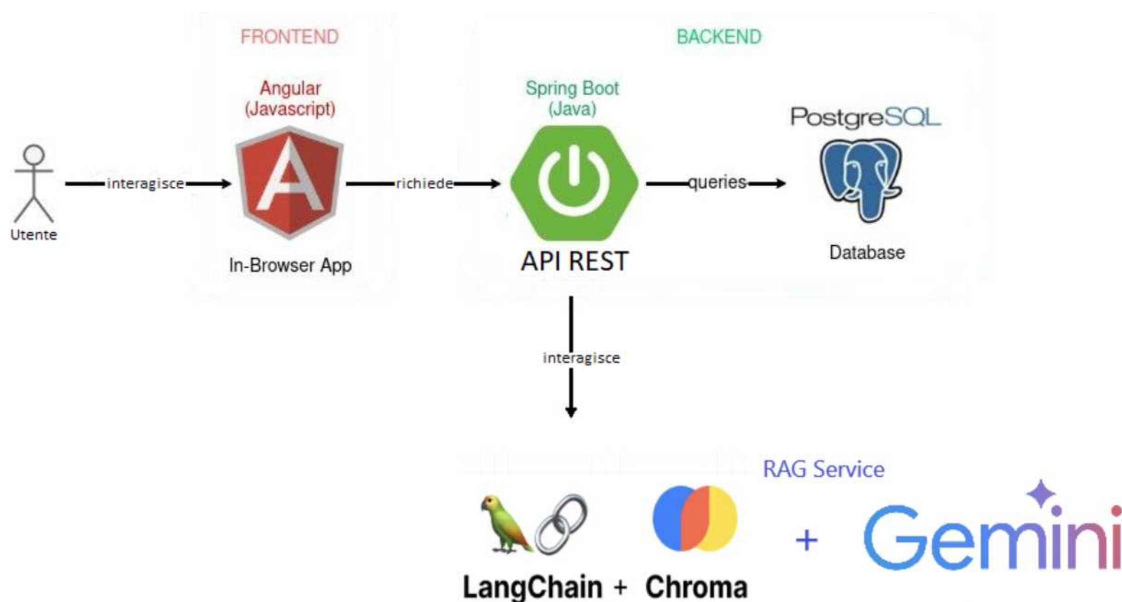


Figura 5.1: Architettura di sistema

Il percorso di stage si è concentrato sullo sviluppo e progettazione dei componenti di *frontend* e *backend*, sia in ambito funzionale che architetturale. La *pipeline* di

comunicazione e le funzionalità del *chatbot* sono state sviluppate dal secondo membro del team, mentre l'integrazione finale si è svolta in collaborazione. L'interesse per l'argomento ha nonostante portato ad uno studio teorico del lavoro svolto dal secondo membro, anche se non vi è stato un coinvolgimento pratico nello sviluppo dell'applicazione correlata. Pertanto, verranno trattate solo le componenti e i servizi relativi al percorso di stage personale, senza approfondire l'ambito di progettazione e sviluppo del *chatbot* RAG.

5.1.1 Multi-layered architecture

L'architettura adottata per lo sviluppo dell'applicazione è la *Multi-layered architecture*. Questa separa l'applicazione in strati distinti (vedi Figura 5.2) ciascuno con responsabilità specifiche, migliorando la manutenibilità, la scalabilità, la testabilità e la sicurezza dell'applicazione. La *Multi-layered architecture* viene comunemente utilizzata nello sviluppo di applicazioni web moderne.

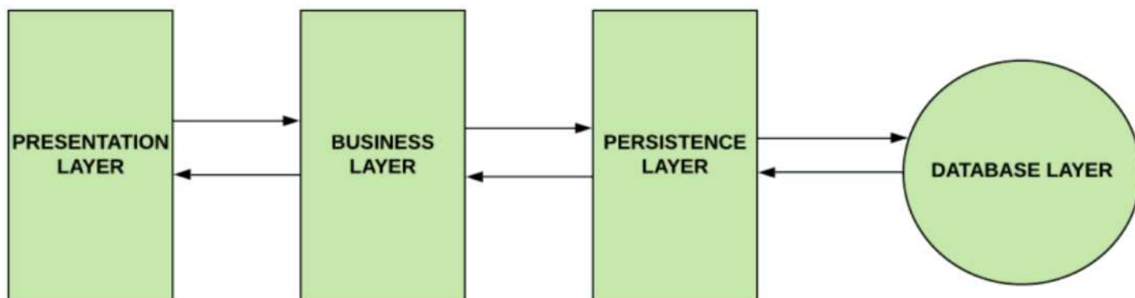


Figura 5.2: Multi-layered architecture

- ***Presentation Layer***: questo strato riguarda l'interfaccia di presentazione per l'utente, inclusa la gestione delle interazioni con l'utente. In questo progetto, è stato implementato utilizzando il framework *Angular*;

- ***Business Layer***: questo strato si occupa della gestione dei servizi offerti al livello di presentazione, includendo tutte le classi e i componenti necessari per eseguire la logica di business. È stato implementato utilizzando il framework *Spring*;

- ***Persistence Layer***: questo strato gestisce le operazioni e il mapping degli oggetti sul database. È stato implementato utilizzando *Spring Data JPA*;

- ***Database Layer***: questo strato rappresenta il database dell'applicazione. In questo progetto, viene utilizzato un database *PostgreSQL* gestito tramite *Spring Data JPA*.

5.2 Backend

Data la vasta gamma di funzionalità offerte dall'applicazione, verrà descritto l'approccio comune utilizzato per lo sviluppo di ogni servizio del progetto. Nei casi particolari che richiedono un'attenzione specifica, sarà fornita un'analisi più dettagliata per affrontare le peculiarità di ciascuno.

5.2.1 Architettura RESTful

Per lo sviluppo dei servizi *backend* è stata adottata un'architettura *RESTful* (*Representational State Transfer*) progettata per creare servizi web scalabili e manutenibili.

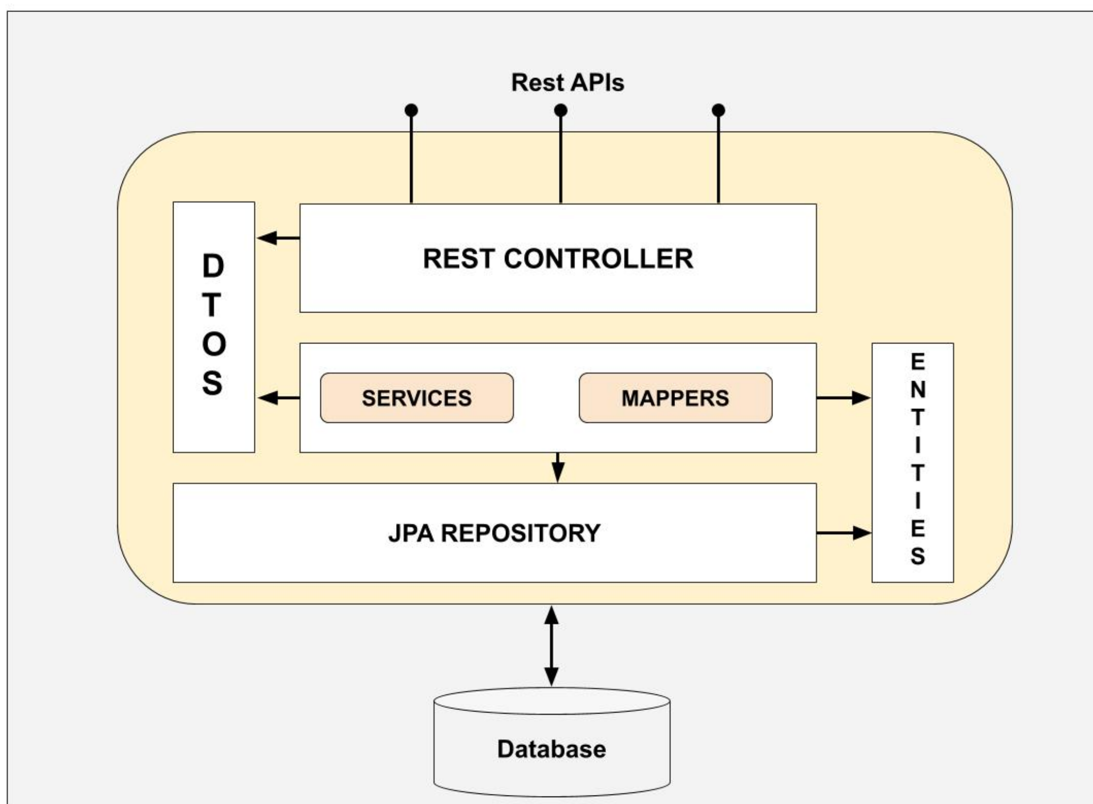


Figura 5.3: Componenti architettura RESTful [22]

Spring Boot facilita l'implementazione di questa architettura grazie alla sua semplicità e al suo robusto ecosistema. Di seguito, viene fornita una spiegazione dettagliata delle principali caratteristiche e componenti di un'architettura *RESTful* implementata con *Spring Boot*.
Nelle architetture *RESTful* si possono individuare i componenti di seguito descritti (vedi Figura 5.3).

5.2.1.1 Componente "Controller"

I *controller* in *Spring Boot* sono responsabili della gestione delle richieste e delle risposte HTTP. Utilizzano l'annotazione `@RestController` per definire le classi che espongono i punti di accesso delle API. Ogni metodo del *controller* è mappato ad un endpoint specifico utilizzando annotazioni come `@GetMapping`, `@PostMapping`, `@PutMapping` e `@DeleteMapping`. Ad ogni *controller* tramite *Dependecy Injection* viene iniettata la corrispondente classe del *layer* di servizio per gestire le richieste dei *client*.
Per lo sviluppo degli endpoint verranno utilizzate le seguenti *best-practice* per i codici di risposta HTTP:

- **Richieste GET:**
  - **200 OK**: la richiesta è stata completata con successo. La risposta include la risorsa richiesta;
  - **404 Not Found**: la risorsa richiesta non esiste.

  Il Codice 5.1 presenta un esempio di metodo che gestisce una richiesta GET per la risorsa *Allenamento*.

- **Richieste POST:**
  - **201 Created**: la risorsa è stata creata con successo. L'intestazione della risposta inlcude un campo `Location` che contiene l'*Uniform Resource Identifier* (URI) della nuova risorsa;
  - **400 Bad Request**: la richiesta è malformata o i dati forniti non sono validi;

  - **409 Conflict**: la richiesta non può essere completata a causa di un conflitto con lo stato corrente della risorsa (ad esempio, un duplicato di una risorsa che dovrebbe essere unica).

  Il Codice 5.2 presenta un esempio di metodo che gestisce una richiesta POST per la risorsa *Allenamento*.

- **Richieste PUT:**
  - **200 OK**: la risorsa è stata aggiornata con successo.
  - **400 Bad Request**: la richiesta è malformata o i dati forniti non sono validi;
  - **404 Not Found**: la risorsa da aggiornare non esiste.

  Il Codice 5.3 presenta esempio di metodo che gestisce una richiesta PUT per la risorsa *Utente*.

- **Richieste DELETE:**
  - **204 No Content**: la risorsa è stata eliminata con successo, senza alcun contenuto nella risposta;
  - **404 Not Found**: la risorsa da eliminare non esiste.

  Il Codice 5.4 presenta un esempio di metodo che gestisce una richiesta DELETE per la risorsa *Allenamento*.

```
@GetMapping("/{allenamento-id}")
public ResponseEntity<AllenamentoResponse> findAllenamentoById(
        @PathVariable("allenamento-id") Long allenamento_id,
        Authentication connectedUser) {
    return ResponseEntity.ok(
    service.findByAuthUserAndAllenamentoId(
        allenamento_id, connectedUser));
}
```

Codice 5.1: Esempio di metodo di un controller che gestisce richieste GET per la risorsa *Allenamento*.

```
    @PostMapping
    public ResponseEntity<AllenamentoResponse> saveAllenamento(
            @Valid @RequestBody AllenamentoRequest request,
            Authentication connectedUser
    ) {
        AllenamentoResponse createdResource =
                            service.save(request, connectedUser);
        URI location = ServletUriComponentsBuilder
                .fromCurrentRequest()
                .path("/{allenamento-id}")
                .buildAndExpand(createdResource.getId())
                .toUri();
        return ResponseEntity.created(location)
            .body(createdResource);
    }
```

Codice 5.2: Esempio di metodo di un controller che gestisce richieste POST per la risorsa *Allenamento*.

```
@PutMapping
    public ResponseEntity<?> updateUser(
            @Valid @RequestBody UserRequest request,
            Authentication connectedUser
    ) {
        service.updateUser(request, connectedUser);
        return ResponseEntity.ok().build();
    }
```

Codice 5.3: Esempio di metodo di un controller che gestisce richieste PUT per la risorsa *Utente*.

```
    @DeleteMapping("/{allenamento-id}")
    public ResponseEntity<?> deleteAllenamento(
            @PathVariable("allenamento-id") Long allenamento_id,
            Authentication connectedUser) {
        service.deleteById(allenamento_id, connectedUser);
        return ResponseEntity.noContent().build();
    }
```

Codice 5.4: Esempio di metodo di un controller che gestisce richieste DELETE per la risorsa *Allenamento*.

5.2.1.2 Componente "Service"

Il *Service Layer* contiene la logica di business dell'applicazione. È separato dal livello del *controller* per promuovere una chiara separazione delle responsabilità. Utilizza l'annotazione `@Service` per indicare che una classe fornisce funzionalità di servizio e viene utilizzato dai *controller* per gestire le richieste dei *client*. Ogni classe annotata con `@Service` utilizza la *Dependency Injection* per ricevere l'istanza della corrispondente interfaccia *repository* di *Spring Data JPA*, consentendo così l'interazione con il database. Inoltre, ogni classe del *layer* di servizio sarà annotata con `@Transactional` per garantire che tutte le operazioni di business all'interno di tali classi vengano ese-

guite in un contesto transazionale. Questo assicura che le operazioni su database siano atomiche, consistenti, isolate e durevoli **(ACID)**, migliorando l'affidabilità e la coerenza dei dati nell'applicazione.

Il Codice 5.5 presenta un esempio di classe del *layer* di servizio relativo alla risorsa *Allenamento*. Non verrà esposta l'implementazione dei metodi per mantenere la chiarezza e la concisione del documento.

```
@Service @RequiredArgsConstructor @Transactional
public class AllenamentoService{
        private final AllenamentoRepository allenamentoRepository;
        private final AllenamentoMapper allenamentoMapper;
        private final UserExtractor userExtractor;

        public AllenamentoResponse save(AllenamentoRequest request,
            Authentication connectedUser);
        public AllenamentoResponse findByAuthUserAndAllenamentoId(
            Long allenamento_id, Authentication connectedUser);
        public ArrayList<AllenamentoResponse>
            findAllAuthUserAllenamento_noPagination(
                Authentication connectedUser);
        public PageResponse<AllenamentoResponse>
            findAllAuthUserAllenamento_paginated(int page,
                                                 int size,
                                                 Authentication
                                                    connectedUser);
        public Allenamento findByIdAndCreator(Long allenamento_id,
                                              User creator);
        public void deleteById(Long allenamento_id,
                               Authentication connectedUser);
}
```

Codice 5.5: Classe del *layer* di servizio della risorsa *Allenamento*

5.2.1.3 Componente "Repository"

Il *Repository Layer* si occupa della persistenza dei dati e del loro recupero. Utilizza *Spring Data JPA* per interagire con il database in modo semplice ed efficace. Le interfacce del *layer* di *repository* implementando `JpaRepository` (interfaccia offerta da *Spring Data JPA*), forniscono metodi CRUD pronti all'uso e la possibilità di definire query personalizzate. Le query personalizzate possono essere create con l'uso di annotazioni come `@Query` all'interno delle interfacce di *repository* oppure tramite la costruzione di metodi con il *query derivation*, che permette di generare automaticamente query basate sul nome del metodo stesso. È importante specificare che nelle rappresentazioni Unified Modeling Language (UML) incluse nel documento, le interfacce del livello *repository* indicheranno esclusivamente i metodi esplicitamente definiti non disponibili tra quelli forniti di default dall'interfaccia `JpaRepository`.
Il Codice 5.6 presenta un esempio di classe del layer di *repository* relativo alla risorsa *Allenamento*.

```java
@Repository
public interface AllenamentoRepository
    extends JpaRepository<Allenamento, Long> {
    Page<Allenamento> findByCreator(Pageable pageable, User user);
    Optional<Allenamento> findByNameAndCreator(String name,
        User user);
    Optional<Allenamento> findByIdAndCreator(Long id,User user);
    Optional<ArrayList<Allenamento>> findByCreator(User user);
    void deleteByIdAndCreator(Long id, User user);
}
```

Codice 5.6: Classe del layer di *repository* della risorsa *Allenamento*

5.2.1.4 Componente "Model (Entity)"

Le entità rappresentano gli oggetti del dominio dell'applicazione e sono mappate alle tabelle del database. Utilizzano annotazioni come `@Entity`, `@Table`, `@Id`, e `@GeneratedValue` per definire la struttura delle tabelle e le relazioni tra di esse.

Il Codice 5.7 presenta un esempio di entità della risorsa *Allenamento*.

```
@Getter @Setter @SuperBuilder @AllArgsConstructor
@NoArgsConstructor @Entity
@Table(name = "_allenamento",uniqueConstraints =
    {@UniqueConstraint(columnNames = {"name", "created_by_user"})})
public class Allenamento extends BaseEntity{
        @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(updatable = false)
        private Long id;
        @Column(nullable = false, length = 100)
        private String name;
        private String description;
        private float durata_in_ore;
        @ManyToOne @JoinColumn(name = "created_by_user")
        private User creator;
        @OneToMany(mappedBy = "allenamento",fetch = LAZY,
            cascade = CascadeType.ALL)
        private List<AllenamentoEsercizio> allenamenti_esercizi;
        @OneToMany(mappedBy = "allenamento", fetch = LAZY,
            cascade = CascadeType.ALL)
        private List<PeriodoAllenamento> periodo_allenamenti;
}
```

Codice 5.7: Esempio entità della risorsa *Allenamento*

5.2.2 Struttura delle tabelle relazionali

In fase di progettazione, è stata definita la struttura relazionale del database (vedi Figura 5.4) progettata in modo da assicurare la capacità di soddisfare tutti i requisiti. Come già detto, però, utilizzando *Spring Data JPA*, non sarà necessario implementare tale schema direttamente nel database, ma questo diagramma fungerà da guida per la costruzione delle classi del modello di business (entità) in *Spring*.

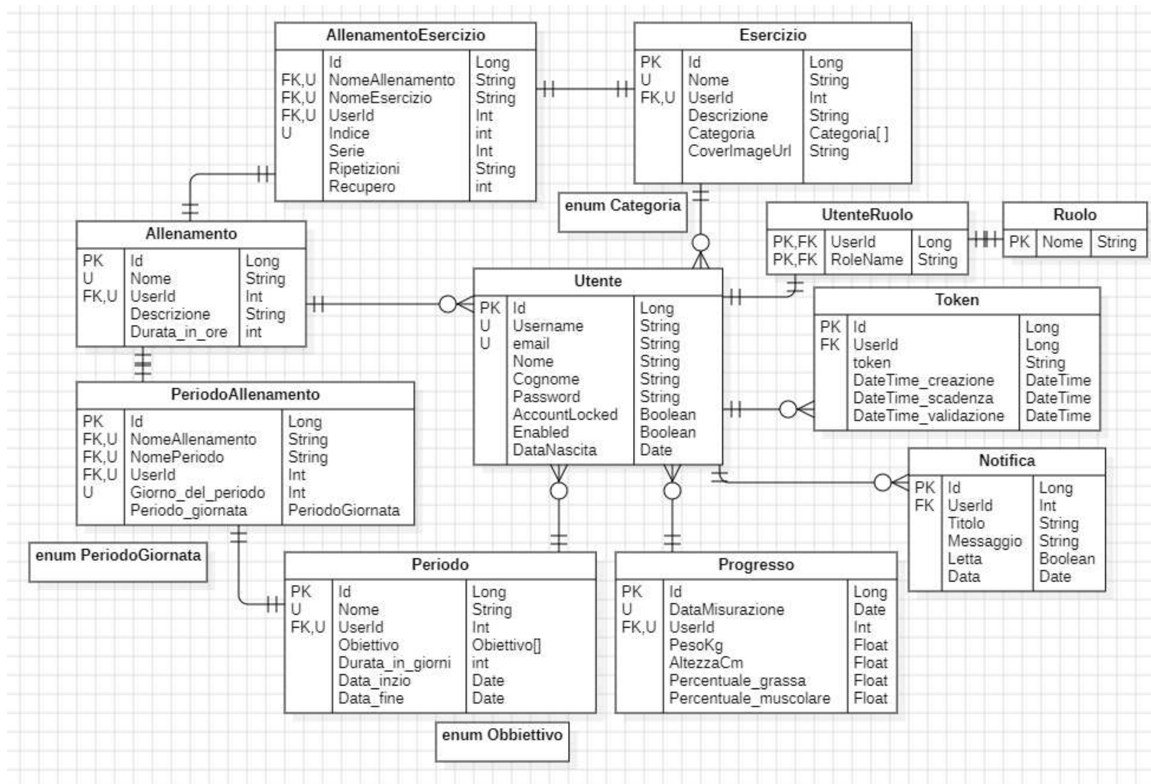


Figura 5.4: Diagramma entità relazione

Ogni entità è caratterizzata da una chiave primaria di tipo *Long*, anziché fare affidamento sui vincoli di unicità forniti. Ad esempio, l'entità *Esercizio* avrebbe potuto avere come chiave primaria una combinazione del nome e dell'identificativo dell'utente che ha creato l'esercizio. Questa scelta è stata compiuta per semplificare la gestione dei dati, migliorare le prestazioni (specialmente nelle operazioni di `JOIN`, che potrebbero risultare meno efficienti con chiavi composte), ma soprattutto per garantire l'immutabilità della chiave primaria.

5.2.3 Progettazione servizi REST risorse

In questa sezione, verranno descritte le scelte progettuali relative ai servizi REST sviluppati. Inizieremo fornendo una panoramica delle pratiche comuni a tutti i servizi, per poi concentrarci, senza entrare nel dettaglio, sui servizi REST delle risorse corrispondenti alle entità del diagramma entità-relazione appena discusso. Verranno implementati dei servizi REST per le risorse di seguito elencate.

- **Utenti**: verranno implementati due servizi REST, uno per la gestione della registrazione, conferma indirizzo mail e autenticazione degli utenti e uno

per effettuare operazioni sull'utente come il cambio password o modifica delle informazioni personali;

- **Esercizi**;
- **Allenamenti**;
- **Periodi**;
- **Progressi**;
- **Notifiche**;
- **AllenamentoEsercizio**: entità nata dalla relazione molti a molti tra le entità *Esercizio* e *Allenamento*. Permette la gestione della lista di esercizi che dovranno essere svolti all'interno di un allenamento;
- **PeriodoAllenamento**: entità nata dalla relazione molti a molti tra le entità *Periodo* e *Allenamento*. Permette la gestione della lista di allenamenti che dovranno essere svolti all'interno di un periodo.

Le risorse rappresentate nel diagramma entità-relazione, ovvero *Token* (oggetto utilizzato per la conferma della mail di registrazione), *Ruolo* e *UtenteRuolo* (per la gestione delle autorizzazzioni), saranno utilizzate dal servizio REST responsabile dell'autenticazione e della registrazione degli utenti. Queste risorse non richiedono servizi REST dedicati da esporre. Di conseguenza, le considerazioni comuni non si applicano a tali entità.

5.2.3.1 Convenzione di configurazione degli endpoint in Spring Boot

Ogni endpoint *Spring* è raggiungibile tramite l'indirizzo del server alla porta configurata nel *file YAML* di configurazione dell'applicazione *Spring Boot*, seguito dalla *base Uniform Resource Locator (URL) "api/v1"*. Questa convenzione di indirizzamento consente la coesistenza di più versioni e l'introduzione di nuove funzionalità o modifiche significative, dove gli endpoint associati alla nuova versione avranno una base URL aggiornata: *"api/v2"*.

Questo approccio assicura che le applicazioni che utilizzano la versione precedente (v1) continueranno a funzionare senza problemi, mentre quelle che necessitano delle nuove funzionalità possono migrare alla nuova versione (v2).

5.2.3.2 Componenti

Ogni servizio, al fine di gestire le operazioni necessarie per soddisfare le richieste dei *client*, presenterà i componenti esposti nella Sezione 5.2.1, ciascuno specifico per la risorsa ad esso associata. In particolare, ogni servizio sarà costituito dai seguenti elementi:

- una classe annotata con `@RestController`, responsabile della pubblicazione degli endpoint API;
- una classe annotata con `@Service` che gestisce la logica di business relativa alla risorsa;
- una classe *mapper*, anch'essa annotata con `@Service`, che si occupa della mappatura dei Data Transfer Object (DTO)$_G$ alle entità del modello dati;
- un'interfaccia annotata con `@Repository` per l'interazione con il livello di persistenza dei dati;
- i propri DTO, che rappresentano gli oggetti trasferiti tra il *client* e il servizio REST per la manipolazione dei dati.

5.2.3.3 Operazioni CRUD

Ogni servizio associato a ciascuna risorsa dell'applicazione offre le operazioni CRUD, ovvero *Create* (Creazione), *Read* (Lettura), *Update* (Aggiornamento) e *Delete* (Cancellazione). Queste operazioni costituiscono le funzionalità basilari per la gestione delle risorse all'interno del sistema.
Tra le operazioni di lettura, si includono sia la richiesta di un singolo elemento identificato, sia l'elenco completo delle risorse relative all'utente che effettua la richiesta. Inoltre, in alcuni casi, la risposta a tali richieste è fornita con paginazione. Questo

significa che anziché restituire l'intero insieme di dati in una sola volta, vengono forniti blocchi di dati, o pagine, rendendo più efficiente il processo di recupero dei dati e riducendo il carico di lavoro del sistema.
L'attenzione nel *backend* si è concentrata sulla gestione dei dati e sulla loro integrità costante, mirando a esporre solo le operazioni necessarie per svolgere tutte le funzionalità richieste e ad assicurare la loro correttezza formale. Questa scelta di design, caratterizzata da una semplicità mirata, ha consentito di minimizzare i casi eccezionali da gestire e di mantenere un maggiore controllo su ogni situazione.

5.2.3.4 Pattern DTO

Ogni servizio REST ha implementato il *Design Pattern* DTO (Data Transfer Object) per incapsulare i dati in modo efficiente e sicuro e consentirne il trasferimento tra diversi strati dell'applicazione. Per ogni risorsa, è stato realizzato un DTO di "richiesta" (vedi Codice 5.8) utilizzato per operazioni di salvataggio o aggiornamento della risorsa, e uno di "risposta" (vedi Codice 5.9). Ogni classe DTO espone metodi *setter* e *getter* per ogni attributo e implementa il pattern ***Builder*** per semplificare le operazioni di *mapping* svolte dalla corrispondente classe di servizio *Mapper*.

```
public record AllenamentoRequest(
    Long id,
    @NotNull(message="Il nome dell'esercizio è necessario")
    @NotEmpty(message="Il nome dell'esercizio è necessario")
    String name,
    @NotNull(message="La descrizione dell'esercizio è necessaria")
    String description,
    @Min(value=0, message="La durata deve essere maggiore di zero")
    float durata_in_ore
) {}
```

Codice 5.8: DTO di "richiesta" della risorsa *Allenamento*

```
@Getter @Setter @AllArgsConstructor
@NoArgsConstructor @Builder
public class AllenamentoResponse {
@NotNull(message = "L'id dell'allenamento è necessario")
private Long id;
@NotNull(message = "Il nome dell'allenamento è necessario")
private String name;
@NotNull(message = "La descrizione dell'allenamento è necessaria")
private String description;
@NotNull(message = "La durata dell'allenamento è necessaria")
private float durata_in_ore;
@NotNull(message = "L'id dell'utente creatore è necessario")
private String creator_username;
}
```

Codice 5.9: DTO di "risposta" della risorsa *Allenamento*

5.2.3.5 Validazione DTO

La validazione dei DTO è una pratica essenziale nello sviluppo di applicazioni web per garantire che i dati ricevuti dai *client* siano conformi ai requisiti previsti. Questa validazione viene implementata tramite annotazioni nel codice, che offrono un modo dichiarativo e centralizzato per definire le regole di validazione al momento dell'implementazione degli attributi delle classi di DTO.
Per abilitare la validazione, è sufficiente annotare il parametro del metodo del controller che riceve il DTO con `@Valid` e gestire eventuali errori di validazione (vedi Sezione 5.2.3.7).
Nella Sezione 5.2.3.4 sono forniti esempi di codifica che illustrano l'utilizzo delle annotazioni di validazione. Introdurre annotazioni di validazione nei DTO di risposta consente agli strumenti che generano automaticamente le specifiche delle API di documentare con informazioni più dettagliate le risposte attese dagli endpoint.

5.2.3.6 Mapper

Per eseguire le operazioni di conversione tra entità di business e DTO per ogni risorsa, viene implementata una classe di servizio (*Mapper*), annotata con `@Service` per permettere la *Dependency Injection*. Questa classe espone due metodi che consentono la conversione da DTO di richiesta a entità del modello di business e da entità del modello di business a DTO di risposta.
Il Codice 5.10 presenta un esempio di classe *Mapper* della risorsa *Allenamento*.

```
@Service
public class AllenamentoMapper {
    public Allenamento toAllenamento(
        AllenamentoRequest allenamentoRequest, User creator) {
        return Allenamento.builder()
                .id(allenamentoRequest.id())
                .name(allenamentoRequest.name())
                .description(allenamentoRequest.description())
                .durata_in_ore(allenamentoRequest.durata_in_ore())
                .creator(creator).build();
    }
    public AllenamentoResponse toAllenamentoResponse(
        Allenamento allenamento) {
        return AllenamentoResponse.builder()
                .id(allenamento.getId())
                .name(allenamento.getName())
                .description(allenamento.getDescription())
                .durata_in_ore(allenamento.getDurata_in_ore())
                .creator_username(allenamento.getCreator()
                    .getUsername()).build();
    }}
```

Codice 5.10: Classe *Mapper* della risorsa *Allenamento*

5.2.3.7 Gestione degli errori

Per consentire al *client* di ricevere messaggi di errore strutturati, sono state implementate le seguenti soluzioni:

- è stato creato un DTO appositamente per restituire risposte di errore strutturate. Questo oggetto contiene un codice di errore, una descrizione dell'errore e una lista di messaggi di errore, suddivisi tra errori di validazione e altri tipi di errori;
- è stata implementata una classe annotata con `@RestControllerAdvice`, che funge da gestore globale degli errori per tutta l'applicazione. Questa classe intercetta le eccezioni lanciate dai *controller* durante l'esecuzione delle richieste HTTP e personalizza la risposta di errore in base al tipo di eccezione. Ad esempio, se viene sollevata un'eccezione di validazione dei dati, la classe può preparare una risposta strutturata con i dettagli degli errori di validazione. Inoltre, questa classe può gestire errori generici o non previsti, garantendo che il *client* riceva una risposta coerente e comprensibile in caso di problemi durante l'esecuzione delle richieste.

5.2.3.8 Auditing

È stata creata la classe `BaseEntity` che rappresenta una superclasse utilizzata come base per altre classi entità nell'applicazione *Spring Boot*. La classe contiene attributi comuni a tutte le entità, come le data di creazione (annotata con `@CreatedDate`) e l'ultima modifica (annotata con `@LastModifiedDate`), che verranno popolati automaticamente. Inoltre la superclasse utilizzerà le seguenti annotazioni:

- `@MappedSuperclass`: indica che questa classe è una superclasse mappata. Gli attributi di questa classe saranno mappati alle colonne del database nelle sottoclassi, ma la classe stessa non avrà una tabella corrispondente nel database;
- `@EntityListeners(AuditingEntityListener.class)`: abilita l'*auditing* automatico dei campi annotati con `@CreatedDate` e `@LastModifiedDate`.

La classe `BaseEntity` utilizza l'*auditing* di *Spring Data JPA* per mantenere automaticamente le informazioni sulle date di creazione e ultima modifica delle entità. Quando un'entità che estende `BaseEntity` viene salvata o aggiornata, `AuditingEntityListener` popola automaticamente i campi `createdDate` e `lastModifiedDate`.
Ogni entità dell'applicazione, ad eccezione delle entità relative alle notifiche, ai ruoli e ai token per la conferma email, estenderà questa classe.

5.2.3.9 UserExtractor

Ogni classe di servizio che supporta i *controller* con endpoint protetti da autenticazione utilizza la classe *UserExtractor*, che viene iniettata per estrarre l'entità *Utente* dall'oggetto *Authentication*. L'oggetto *Authentication* è fornito come parametro nei metodi dei *controller* attraverso il processo di autenticazione gestito da *Spring Security* descritto alla Sezione 5.2.4.5. In questo modo, è possibile ottenere le informazioni necessarie per identificare l'utente che ha effettuato la richiesta.

5.2.4 Servizi REST risorsa *Utente*

5.2.4.1 Entità *Utente*

L'utente rappresenta l'entità centrale dell'applicazione (vedi Figura 5.5), poiché ogni altra entità è collegata ad essa tramite una relazione di appartenenza.
Le operazioni relative all'entità *Utente* si dividono in due ambiti di utilizzo distinti e sono gestite da due servizi REST separati.

- **Primo Servizio**: si occupa della gestione della registrazione, della conferma dell'email e dell'autenticazione dell'utente;
- **Secondo Servizio**: gestisce operazioni come il cambio della password e la modifica delle informazioni personali dell'utente.

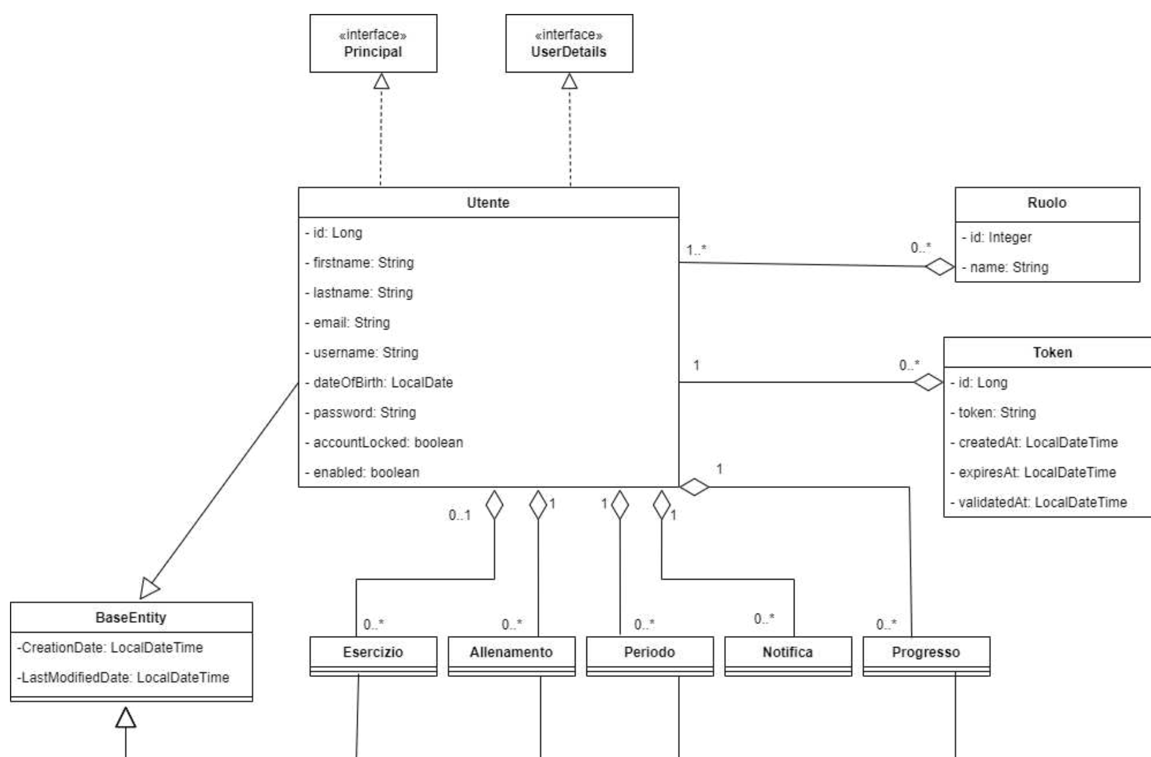


Figura 5.5: UML entità utente

Per rendere disponibile la sua integrazione nel sistema *Spring Security* di autenticazione e autorizzazione l'entità deve implementare *UserDetails* e *Principal*:

- ***UserDetails***:
  - un'interfaccia che rappresenta un utente del sistema;
  - contiene informazioni sull'utente, come username, password, ruoli e autorizzazioni.
- ***Principal***:
  - un oggetto che rappresenta l'identità dell'utente correntemente autenticato nel contesto di *Spring Security*.

5.2.4.2 Entità *Ruolo*

Nonostante attualmente non siano state implementate funzionalità che richiedano ruoli diversi dal semplice *USER*, si è deciso di configurare il sistema in modo da supportare la gestione di più ruoli per un accesso controllato alle risorse. I ruoli sono

entità tracciate nel database e hanno una relazione molti-a-molti con gli utenti. I ruoli associati a ciascun utente saranno utilizzati dall'implementazione di *UserDetails* per comunicare le autorizzazioni.

5.2.4.3 Entità *Token*

Il token è il codice di conferma inviato via email al momento della registrazione. Durante la sua creazione viene anche stabilita la data di scadenza. Una volta abilitato l'utente, ogni token a lui associato viene eliminato.

5.2.4.4 Servizio REST di autenticazione/registrazione

Le operazioni che l'utente svolge per poter accedere all'applicazione sono le seguenti:

- l'utente effettua la richiesta di registrazione inserendo le proprie informazioni personali tra cui username, email e password;
- l'utente inserisce il codice ricevuto via email per per confermare l'indirizzo inserito;
- l'utente effettua l'autenticazione inserendo l'username e la password.

Per svolgere queste operazioni vengono esposti i relativi **servizi REST raggiungibili senza bisogno di autenticazione**. Ovvero per raggiungere tali endpoint non sarà necessario includere nell'header della richiesta il token jwt. Andando più in dettaglio verranno esposti tre endpoint:

- **Registrazione:** il primo endpoint riguarda la richiesta di registrazione di un nuovo utente. Richiede nel corpo della richiesta un DTO valido contenente le informazioni dell'utente. Se l'operazione ha successo grazie ad un servizio dedicato verrà inviata una mail contenente un codice di conferma all'indirizzo inserito dall'utente. Il campo *enable* dell'entità *Utente* è ora *false*;
- **Conferma mail:** il secondo endpoint richiede il codice inviato via mail per la conferma dell'indirizzo, se l'operazione ha successo il campo *enable* dell'entità *Utente* viene impostato a *true*;

- **Autenticazione:** l'ultimo endpoint richiede nel corpo della richiesta un DTO valido con l'username e la password dell'utente. Nel caso di successo verrà restituito un DTO di risposta con il token JWT per effettuare richieste a tutti gli altri endpoint protetti da autenticazione.

Il *controller* che espone gli endpoint per le operazioni appena descritte, utilizza una propria classe di servizio, la quale a sua volta si appoggia a diverse altre classi di servizio per svolgere le operazioni necessarie.
In particolare, è importante notare che, per poter salvare la password dell'utente in modo sicuro, questa viene immediatamente criptata tramite un `@Bean` iniettato che fornisce il servizio di criptazione. Le successive azioni di autenticazione controlleranno che la password inserita dall'utente, una volta criptata, corrisponda a quella salvata nel database.
Inoltre, per consentire l'invio della mail di conferma, viene utilizzato il servizio *EmailService* che, tramite *JavaMailSender* (un'interfaccia parte del modulo *Spring Framework Mail*), crea ed invia la mail di conferma.

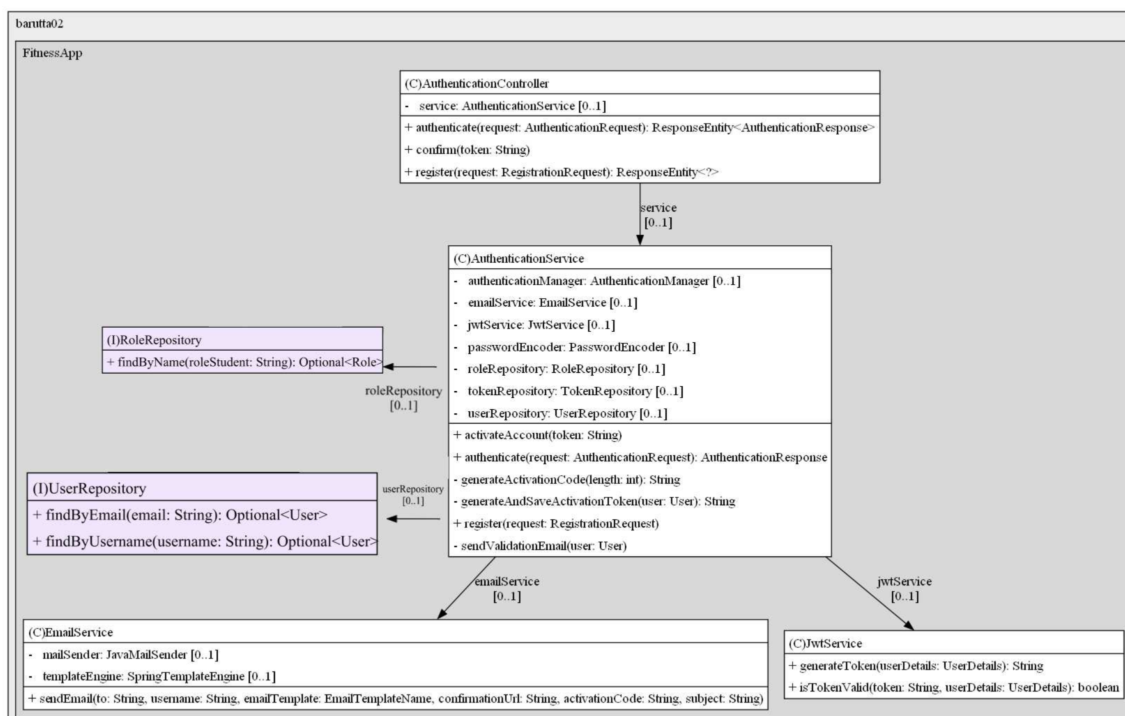


Figura 5.6: Diagramma UML Servizio REST di autenticazione/registrazione

Infine, una volta completata l'autenticazione, il *controller* utilizza il servizio *JwtService* per creare il token jwt da restituire al *client*.

Le operazioni e i metodi del servizio descritto sono esposti nell'UML presente nella Figura 5.6.
Come già detto il servizio REST appena descritto è l'unico a non richiedere autenticazione per essere raggiunto. Tutti gli altri però necessitano di autenticazione tramite token JWT.
Di seguito viene quindi esposto il processo svolto ad ogni richiesta fatta dall'applicazione.

5.2.4.5 Autenticazione e autorizzazione

Per svolgere le operazioni di autenticazione e autorizzazione dell'applicazione si è deciso di utilzzare *Spring security* tramite token jwt cosi da consentire richieste *stateless* scalabili e sicure.

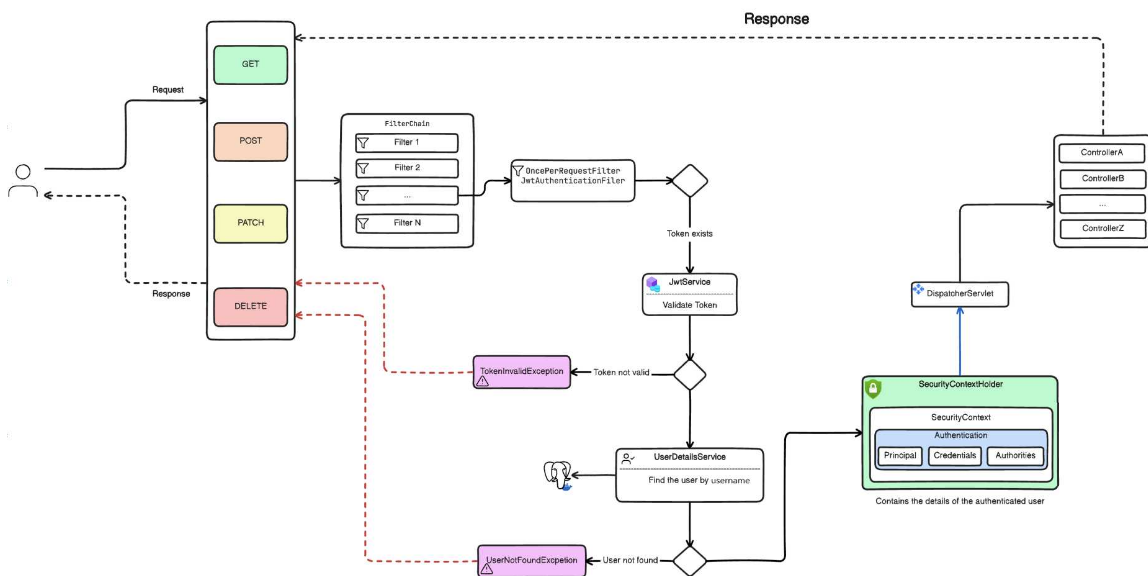


Figura 5.7: Pipeline di autenticazione/autorizzazzione

Il flusso seguito da *Spring Security* per gestire l'autenticazione e l'autorizzazione di ogni richiesta, descritto dalla Figura 5.7, è un processo ben strutturato che coinvolge diversi fasi.

1. **Intercettazione della richiesta**: quando un utente invia una richiesta al server, questa viene intercettata dalla catena di filtri di sicurezza (*Security Filter Chain*) di *Spring Security*. Questa catena è una sequenza di filtri configurati per elaborare e proteggere le richieste HTTP in ingresso. La *Security Filter Chain*

definisce anche quali rotte devono essere protette da autenticazione e l'autorizzazione necessaria. Nella nostra applicazione tutti gli endpoint devono essere protetti da autenticazione e autorizzazione ad utenti con ruolo *USER* tranne quelli dedicati alla registrazione, conferma email e autenticazione dell'utente;

2. **Aggiunta del filtro *JwtAuthenticationFilter***: in *Spring*, esiste già una catena predefinita di filtri di sicurezza. Tuttavia, si è arricchita questa catena aggiungendo un nostro filtro personalizzato chiamato *JwtAuthenticationFilter*. Questo filtro, estendendo *OncePerRequestFilter*, viene eseguito una sola volta per ogni richiesta;

3. **Validazione del token JWT**: il *JwtAuthenticationFilter* verifica inizialmente se è presente un token JWT nella richiesta. Se il token non è presente, il filtro passa la richiesta al successivo filtro nella catena senza eseguire ulteriori operazioni di autenticazione. Altrimenti utilizza il servizio *JwtService* per verificare la sua validità;

4. **Gestione dei casi di token non valido**: se il token è presente ma non è valido (per esempio, è scaduto o non è firmato correttamente), il filtro genera un'eccezione chiamata *TokenInvalidException*. Questo avvisa l'applicazione che la richiesta non può essere autenticata a causa di un token non valido;

5. **Estrazione delle informazioni dell'utente**: se il token è valido, il *JwtAuthenticationFilter* estrae lo username dell'utente dal token. Successivamente, utilizzando il servizio *UserDetailService*, viene effettuata una chiamata al database per recuperare le informazioni complete dell'utente basate sullo username estratto dal token. Queste informazioni includono anche i ruoli dell'utente per valutare le autorizzazioni;

6. **Gestione dell'utente non trovato**: se l'utente non viene trovato nel database, viene lanciata un'eccezione chiamata *UserNotFoundException*. Questo scenario potrebbe verificarsi se lo username estratto dal token non corrisponde a nessun utente nel sistema;

7. **Aggiornamento del *SecurityContextHolder***: se invece l'utente viene trovato nel database, le sue informazioni vengono utilizzate per aggiornare il *SecurityContextHolder*. Il *SecurityContextHolder* è una classe fondamentale nel framework *Spring Security* che gestisce il contesto di sicurezza durante l'esecuzione delle richieste all'interno di un'applicazione. Esso mantiene le informazioni di sicurezza relative all'utente autenticato per tutta la durata della richiesta HTTP.

   All'interno del *SecurityContextHolder*, viene memorizzato il *SecurityContext*. Quest'ultimo rappresenta il contesto di sicurezza corrente e contiene le informazioni principali sull'utente autenticato, le sue credenziali e i suoi ruoli/autorizzazioni.

   Le informazioni principali conservate nel *SecurityContext* sono:

   (a) **Principal**: rappresenta l'utente autenticato. È un oggetto che implementa l'interfaccia *Principal*. Nel nostro caso è l'entità *Utente*, istanza della classe *UserDetails* fornita da *Spring Security* che contiene informazioni come lo username, la password (criptata) e una lista di autorizzazioni associate all'utente;

   (b) **Credentials**: rappresenta le credenziali dell'utente. In genere, sono le credenziali fornite durante il processo di autenticazione, come ad esempio una password o un token JWT;

   (c) **Authorities**: questo campo contiene la lista dei ruoli e quindi delle autorizzazioni associate all'utente. Le autorizzazioni indicano a quali azioni o risorse l'utente è consentito accedere.

8. **Reindirizzamento della richiesta al controller appropriato**: infine, il *DispatcherServlet*, dopo il completamento del processo di autenticazione, reindirizza la richiesta al *controller* ed all'endpoint appropriati per l'elaborazione. Questo passaggio permette di proseguire con la logica di business dell'applicazione una volta che l'utente è stato autenticato con successo e ne viene confermata l'autorizzazzione. Se viene definito come parametro dell'endpoint un oggetto di tipo *Authentication* questo verrà automaticamente iniettato da *Spring security*.

Questo oggetto contiene informazioni sull'utente attualmente autenticato nel sistema.

Da questo momento in poi, tutti i servizi REST esposti richiederanno l'autenticazione dell'utente che effettua la richiesta con ruolo `"USER"`.

5.2.4.6 Servizio REST per operazioni di utilità sul profilo utente

Gli utenti hanno la possibilità di eseguire operazioni di utilità come il cambio password o la modifica delle informazioni personali inserite durante la registrazione, grazie ad un servizio REST dedicato. L'implementazione di tali funzionalità è standard. Qui è importante evidenziare l'utilità dei DTO, distinti da quelli usati per autenticazione e registrazione, poiché consentono di non trasmettere direttamente l'oggetto di business al *client*. Questo è particolarmente cruciale poiché l'oggetto di business potrebbe contenere dati sensibili, come la password, anche se crittografati.

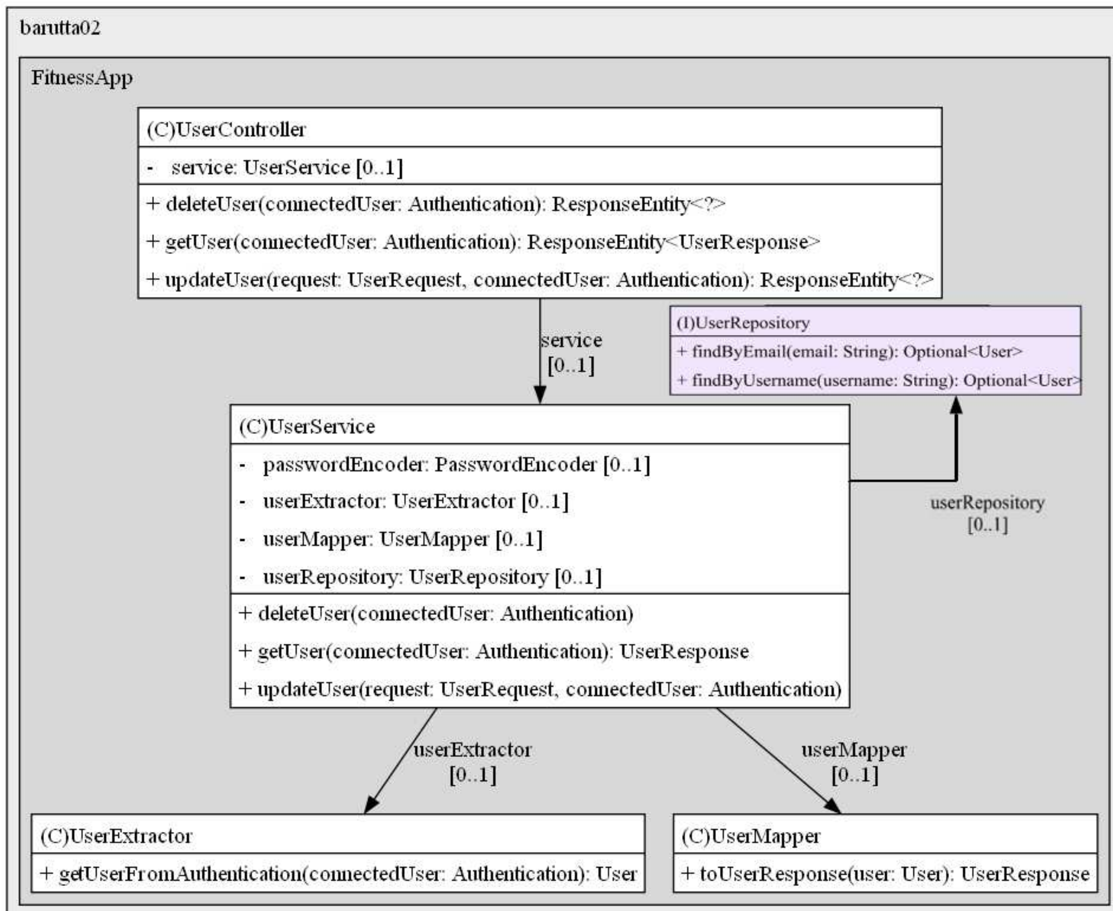


Figura 5.8: Diagramma UML delle classi adibite alle operazioni di utilità sul profilo utente

La progettazione di tale servizio è stata effettuata in collaborazione con il secondo membro del team, mentre il processo di codifica è stato eseguito interamente da lui. Le operazioni offerte dal servizio RESTful appena descritto sono illustrate nell'UML in Figura 5.8.

5.2.5 Servizi REST risorsa *Esercizio*

Nell'attuazione dei servizi disponibili per la risorsa *Esercizio*, cui attributi sono esposti nella Figura 5.4, si sono introdotti solamente due nuove componenti, ovvero due classi di servizio dedicate alla gestione del salvataggio e alla lettura di *file* nel server, ossia le classi *FileUtils* e *FileStorageService*.
Infatti ogni volta che un utente salva un esercizio, ha la possibilità di allegare anche una foto illustrativa. La scelta è stata quella di memorizzare nel database solo il percorso del file, consentendo così di recuperare l'immagine dal server. Questo approccio è motivato dalla volontà di ottimizzare le prestazioni del database, evitando di sovraccaricarlo con dati di grandi dimensioni come le immagini, e semplificare la gestione dei *file* sul server. In questo modo si migliorano le prestazioni del sistema durante le operazioni di accesso e aggiornamento dei dati.
Le classi di servizio utilizzano *FileUtils* per recuperare il *file* dal percorso nel server e trasferirlo all'interno del DTO di risposta come array di byte mentre *FileStorageService* gestisce il salvataggio dell'immagine nella posizione corretta.
Per consentire agli utenti di caricare le immagini, il *controller* espone un endpoint dedicato che richiede l'identificativo dell'esercizio come parametro nell'URL e un oggetto *MultipartFile* nel corpo della richiesta. Questo oggetto contiene i dati effettivi del file, come il nome, il tipo di contenuto e i byte del contenuto del *file* stesso.
L'annotazione del metodo sarà *@PostMapping(value = "/cover/exercise-id", consumes = "multipart/form-data")* per indicare che il tipo di contenuto consumato dalla richiesta sarà *multipart/form-data*, ovvero il tipo di contenuto comunemente usato per l'upload di *file*.
Infine, la risorsa *Esercizio* è coinvolta anche in un'operazione di supporto tra utenti. Al momento della creazione, gli utenti possono scegliere di rendere visibile l'esercizio in uno "*store*", permettendo ad altri di importarlo tra i propri.

Per questo motivo, la risorsa "esercizio" espone due endpoint aggiuntivi non presenti nelle altre risorse: uno per importare l'esercizio utilizzando il suo identificativo, e uno per effettuare operazioni di filtraggio per categoria di esercizio data una lista di categorie. Questo consente all'utente di velocizzare la ricerca di un esercizio, restringendo la visualizzazione alle sole categorie di interesse.

Le operazioni e i metodi del servizio descritto sono disponibili nella Figura 5.9 che espone l'UML dell'architettura *RESTful* della risorsa *Esercizio*.

Gli attributi della risorsa sono esposti nella Figura 5.4. L'attributo "`Categoria`" dell'entità *Esercizio* viene trattato come enumerazione di stringhe nel modello di business e rappresenta le diverse tipologie di esercizi disponibili, come ad esempio "Cardio", "Forza", "Flessibilità" ed "Equilibrio". Inoltre, per rispettare la progettazione illustrata nella stessa figura, è stato implementato un vincolo di *Unique Constraint*. Questo vincolo garantisce che ciascun utente possa salvare un solo esercizio con lo stesso nome.

Nei casi in cui non vengano rispetti i vincoli di *Unique Constraint*, viene generato l'errore HTTP con codice 409, indicante un conflitto di dati. Questo vale per ogni risorsa gestita dall'applicazione.

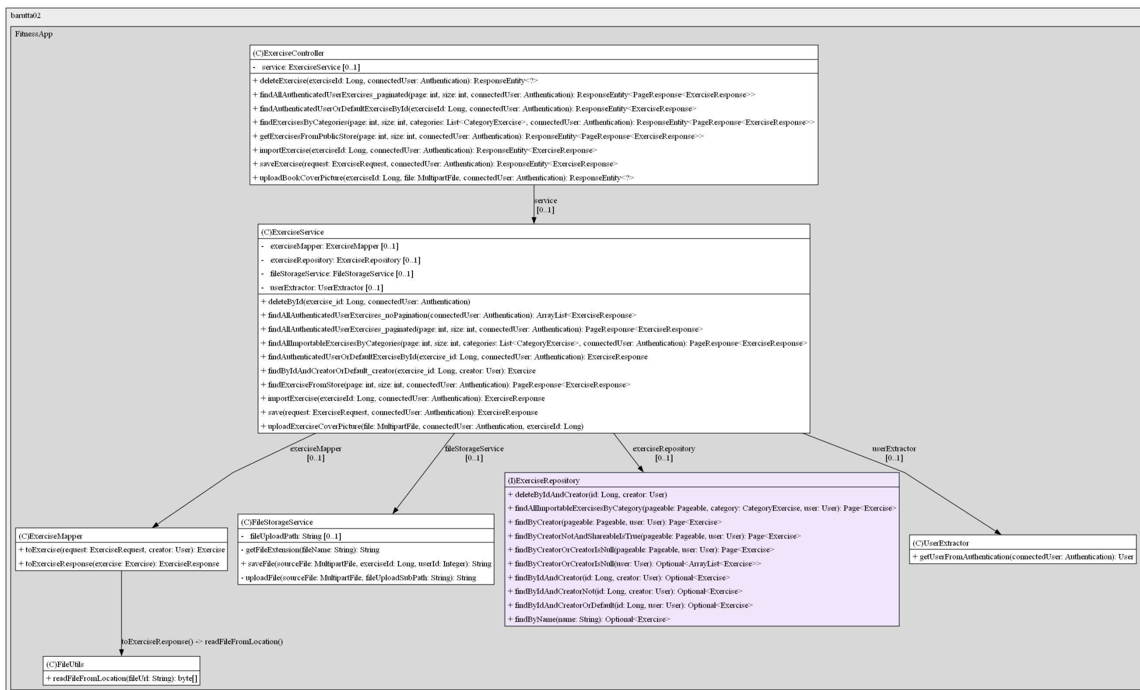


Figura 5.9: Diagramma UML servizio REST risorsa *Esercizio*

5.2.6 Servizi REST risorsa *Allenamento*

Le operazioni offerte dal servizio RESTful per la risorsa *Allenamento*, illustrate nell'UML in Figura 5.10, sono le classiche CRUD e non includono nuove operazioni aggiuntive.
Gli attributi della risorsa sono esposti nella Figura 5.4. Come per l'esercizio viene implementato un vincolo di *Unique Constraint* che garantisce che ciascun utente possa salvare un solo allenamento con lo stesso nome.

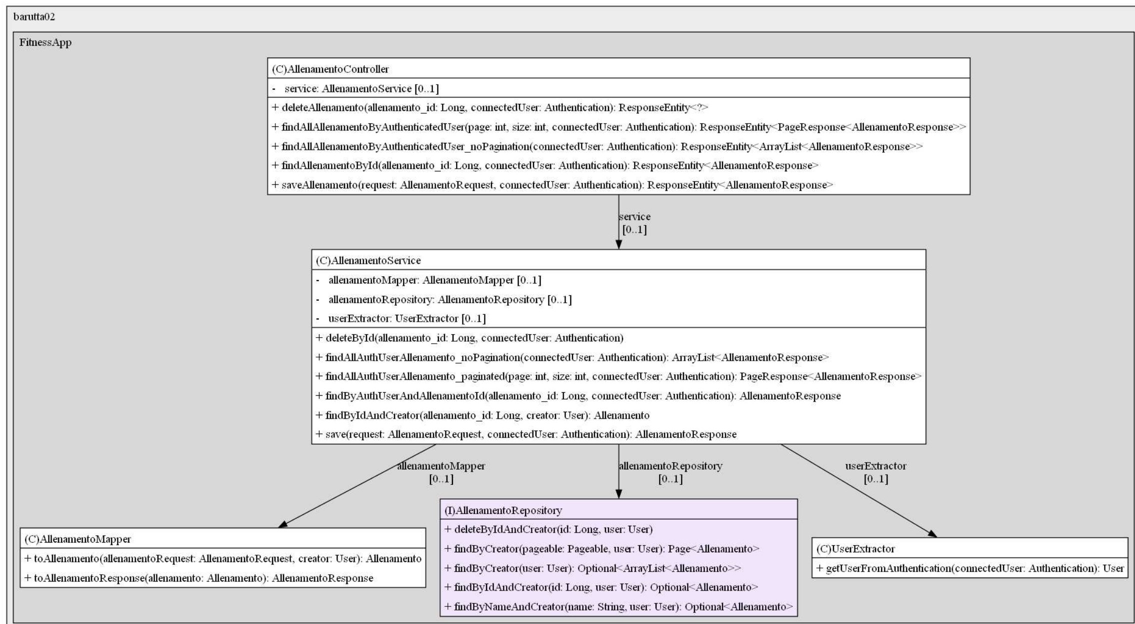


Figura 5.10: Diagramma UML servizio REST risorsa *Allenamento*

5.2.7 Servizi REST risorsa *AllenamentoEsercizio*

La risorsa *AllenamentoEsercizio* è il risultato della relazione molti a molti tra le entità *Allenamento* e *Esercizio*. In questa entità vengono aggiunti quattro attributi aggiuntivi: il numero di serie da completare, il numero di ripetizioni per ogni serie, il tempo di recupero tra le serie e l'indice di esecuzione all'interno dell'allenamento.
Le operazioni CRUD su questa risorsa seguono lo schema classico. Tuttavia, non esiste un endpoint che restituisca tutte le entità di questa risorsa relative a un utente, poiché non è ritenuto utile. Invece, viene fornita una lista degli esercizi relativi ad un unico allenamento senza paginazione, previa verifica che l'utente che effettua la richiesta sia il proprietario dell'allenamento.

Inoltre è stato implementato un vincolo di *Unique Constraint* che garantisce che ciascun utente possa salvare un solo esercizio per allenamento con lo stesso indice.
La Figura 5.11 presenta il diagramma UML delle classi relative al servizio REST appena esposto.

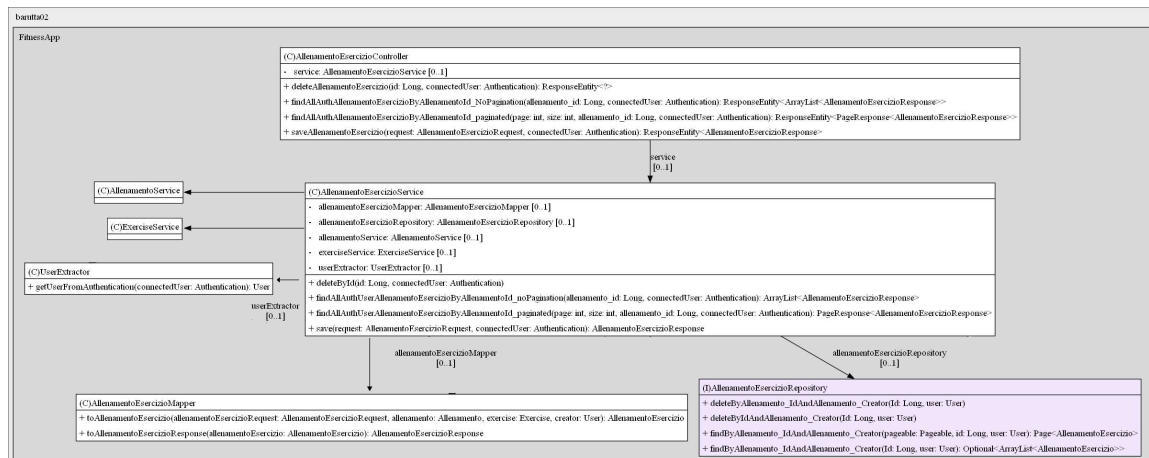


Figura 5.11: Diagramma UML servizio REST risorsa *AllenamentoEsercizio*

5.2.8 Servizi REST risorsa *Progresso*

Per quanto riguarda i servizi relativi alla risorsa `Progresso`, la progettazione è stata effettuata in collaborazione con il secondo membro del team, mentre il processo di codifica è stato eseguito interamente da lui.
Le operazioni offerte dal servizio RESTful per la risorsa *Progresso*, illustrate nel diagramma UML in Figura 5.12, includono le classiche operazioni CRUD.

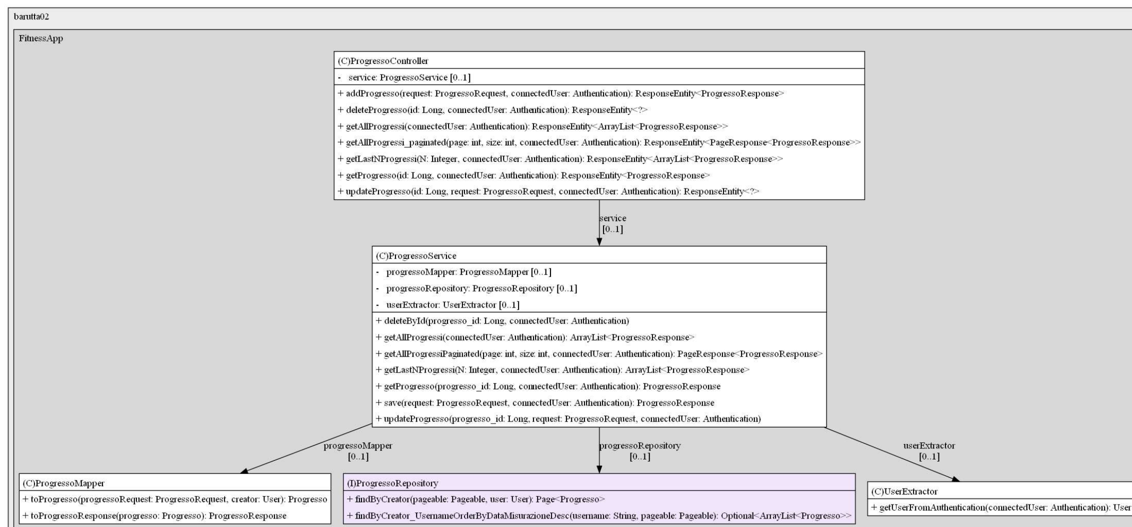


Figura 5.12: Diagramma UML servizio REST risorsa *Progresso*

Inoltre, viene esposto un endpoint che permette di ottenere, senza paginazione, gli ultimi `N` progressi inseriti dall'utente. Gli attributi della risorsa sono esposti nella Figura 5.4. Inoltre viene implementato un vincolo di *Unique Constraint* che garantisce che ciascun utente possa salvare un solo progresso in una stessa data.

5.2.9 Servizi REST risorsa *Periodo*

Le operazioni offerte dal servizio RESTful per la risorsa *Periodo*, illustrate dall'UML in Figura 5.13, sono le classiche CRUD, inoltre vengono esposti due endpoint che permettono l'ottenimento del periodo attivo, se presente, e la sua disattivazione.

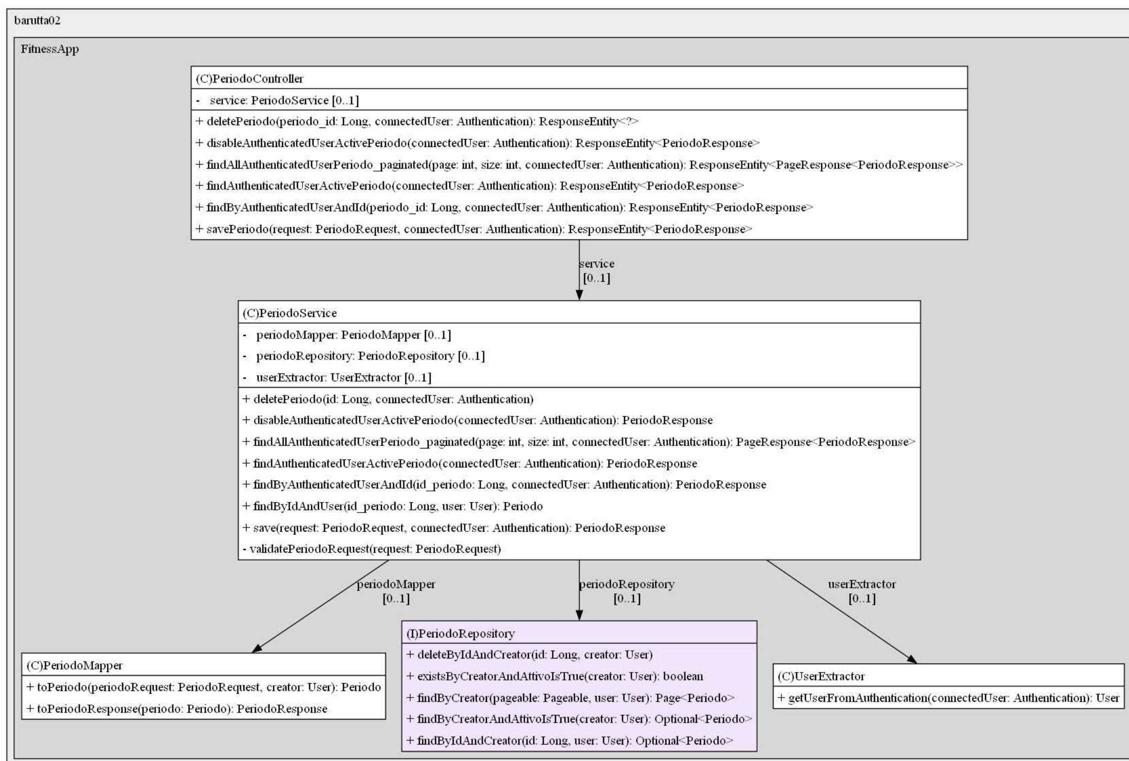


Figura 5.13: Diagramma UML servizio REST risorsa *Periodo*

Gli attributi della risorsa sono esposti nella Figura 5.4. Come per l'esercizio viene implementato un vincolo di *Unique Constraint* che garantisce che ciascun utente possa salvare un solo periodo con lo stesso nome. L'attributo "`Obiettivo`" dell'entità *Periodo* viene trattato come enumerazione di stringhe nel modello di business e rappresenta i diversi scopi del periodo, come ad esempio "massa", "definizione", o "dimagrimento".

5.2.10 Servizi REST risorsa *PeriodoAllenamento*

La risorsa *PeriodoAllenamento* è il risultato della relazione molti a molti tra le entità *Allenamento* e *Periodo*. In questa entità vengono aggiunti due attributi aggiuntivi: il giorno del periodo e il momento della giornata (mattina, pomeriggio o sera) in cui viene effettuato l'allenamento, quest'ultimo trattato come enumerazione di stringhe nel modello di business.

Le operazioni CRUD su questa risorsa seguono lo schema classico. Tuttavia, non esiste un endpoint che restituisca tutte le entità di questa risorsa relative a un utente, poiché non è ritenuto utile. Invece, viene fornita una lista degli allenamenti relativi ad un unico periodo senza paginazione, previa verifica che l'utente che effettua la richiesta sia il proprietario del periodo.

La Figura 5.14 presenta il diagramma UML delle classi relative al servizio REST appena esposto.

Inoltre è stato implementato un vincolo di *Unique Constraint* che garantisce che ciascun utente possa salvare un solo allenamento per periodo nello stesso giorno e momento della giornata.

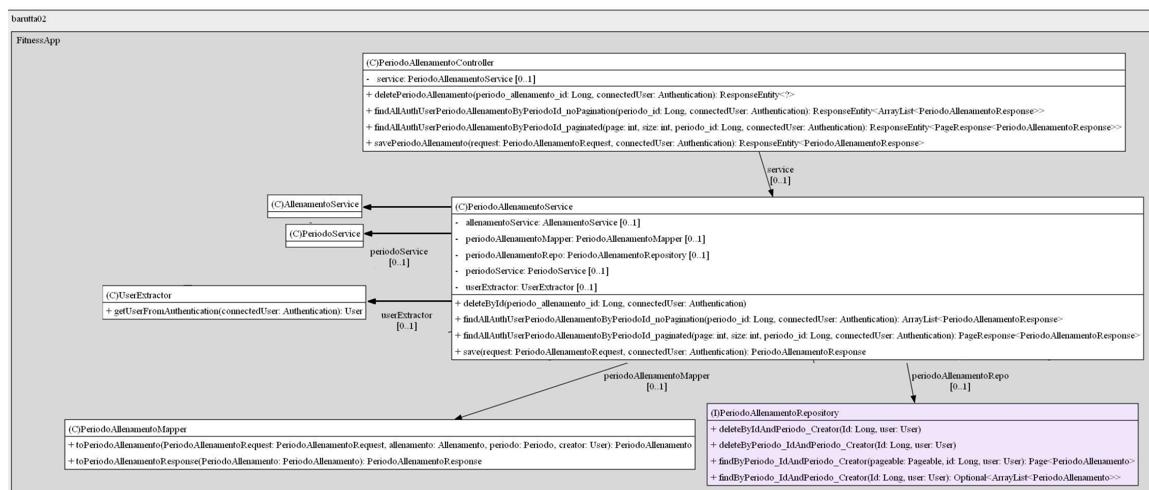


Figura 5.14: Diagramma UML servizio REST risorsa *PeriodoAllenamento*

5.2.11 Servizi REST risorsa *Notifica*

I servizi relativi alle notifiche dell'applicazione differiscono da quelli delle altre risorse.

Per fornire un contesto maggiore, le notifiche riguardano gli allenamenti da svolgere nella giornata odierna, calcolati sulla base del periodo attivo dell'utente.
Gli endpoint esposti sono solamente due:

- Il primo consente di ottenere le notifiche giornaliere non lette, se presenti, dell'utente che effettua la richiesta;
- Il secondo consente di impostare una notifica come letta.

La gestione delle notifiche lato *backend* ha permesso di mantenere uno stato coerente anche se l'utente ha vari dispositivi connessi. Inoltre, poiché le notifiche precedenti alla data odierna non sono più utili, il layer di servizio della risorsa si occupa di eliminarle ad ogni nuova richiesta delle notifiche giornaliere. Gli attributi della risorsa sono esposti nella Figura 5.4 mentre il diagramma UML delle classi relative al servizio REST appena esposto è presentata nella Figura 5.15.

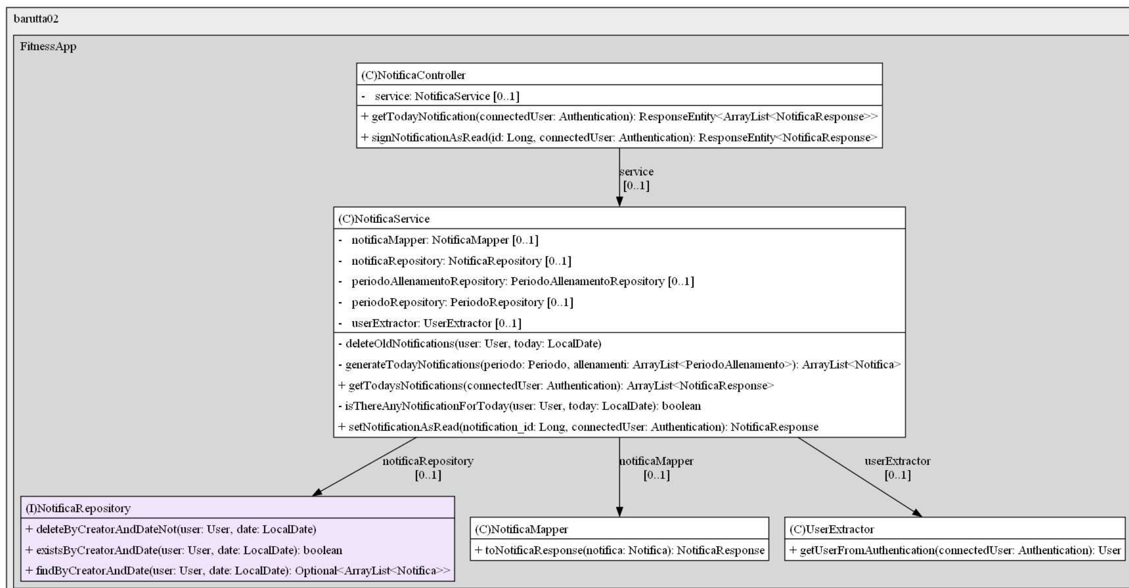


Figura 5.15: Diagramma UML servizio REST risorsa *Notifica*

5.3 Frontend

5.3.1 Component-Based Architecture

Per il *frontend* dell'applicazione, è stata utilizzata l'architettura basata su componenti offerta da *Angular* che permette di suddividere l'applicazione in parti più piccole e

gestibili, chiamate appunto componenti. Ogni componente è responsabile di una specifica parte dell'interfaccia utente e include il proprio template *HTML*, *CSS* e logica *TypeScript*. Questa separazione dei compiti consente di:

- **riutilizzare i componenti:** un componente può essere utilizzato in diverse parti dell'applicazione, riducendo la duplicazione del codice e facilitando la manutenzione;
- **isolare le funzionalità:** ogni componente può essere sviluppato e testato indipendentemente, migliorando la qualità del codice e facilitando il debugging;
- **facilitare la collaborazione:** in un team di sviluppo, i membri possono lavorare su componenti diversi senza interferire tra loro, migliorando l'efficienza del lavoro.

5.3.2 Moduli Angular

Per garantire una gestione efficiente delle pagine e dei componenti utilizzati nell'area personale degli utenti, raggiungibile dopo l'autenticazione, è stato creato un Modulo Angular$_G$ separato. Ciò favorisce la chiarezza architetturale, il riutilizzo del codice e semplifica la gestione delle autorizzazioni, la manutenzione e la scalabilità dell'applicazione.

5.3.3 Angular HttpInterceptor

È stato implementato un *interceptor Angular* per gestire l'inserimento del Token JWT, restituito al momento dell'autenticazione, in ogni chiamata al *backend* che richieda l'autenticazione. Questo *interceptor* si attiva automaticamente ad ogni richiesta HTTP e aggiunge il Token JWT nell'header *Authorization*, consentendo al *backend* di autenticare l'utente e fornire l'accesso alle risorse protette.

5.3.4 Angular Route Guards

Una guardia *Angular* è un meccanismo utilizzato per proteggere le rotte dell'applicazione, che verifica se l'utente ha l'autorizzazione necessaria per accedere a una

specifica rotta. Nel nostro caso, è stata utilizzata una guardia per verificare la validità del Token JWT, salvato nel *local storage* quando ricevuto al termine del processo di autenticazione, prima di permettere l'accesso alle rotte dell'area personale. Il *local storage* è una funzionalità del web storage che permette alle applicazioni web di memorizzare dati direttamente nel browser dell'utente in modo persistente. A differenza dei cookie, il *local storage* ha una capacità di memorizzazione maggiore e i dati non vengono inviati al server con ogni richiesta HTTP. Se il token non è valido, l'utente viene reindirizzato alla pagina di autenticazione.
L'utilizzo di un modulo dedicato per le rotte dell'area personale ha semplificato questa operazione, garantendo una gestione più pulita delle guardie di autenticazione. Le rotte dell'area personale sono state raggruppate in un unico modulo, consentendo l'applicazione di guardie di rotta in modo più efficiente e coerente. Ciò ha semplificato il controllo dell'accesso alle risorse protette e ha reso il codice più ordinato e manutenibile.

5.3.5 Generazione dei Servizi per le richieste al backend

Per generare i servizi necessari ad effettuare le richieste al *backend* è stato utilizzato lo strumento `ng-openapi-gen`. Questo strumento automatizza la generazione del codice dei servizi a partire dalla specifica *OpenAPI* (precedentemente nota come *Swagger*), che descrive le API del *backend*. La specifica è stata progettata in *StopLight* e, successivamente alla codifica, aggiornata automaticamente dallo strumento descritto alla Sezione 4.2.9.
L'uso di `ng-openapi-gen` presenta diversi vantaggi di seguito esposti.

1. **Automatizzazione del codice**: automatizzando la generazione dei servizi, si riduce notevolmente il tempo necessario per scrivere manualmente il codice delle richieste al *backend*;

2. **Riduzione degli errori**: la generazione automatica del codice riduce la probabilità di errori umani, garantendo che le richieste siano conformi alla specifica *OpenAPI*. Questo assicura una maggiore affidabilità e coerenza nelle comunicazioni tra *frontend* e *backend*;

3. **Aggiornamento semplificato**: quando la specifica *OpenAPI* viene aggiornata, `ng-openapi-gen` può rigenerare i servizi, assicurando che il codice del *frontend* sia sempre sincronizzato con le API del *backend*. Questo facilita la manutenzione e l'aggiornamento dell'applicazione;

4. **Consistenza**: l'uso di un tool standardizzato per generare il codice dei servizi garantisce che tutte le richieste al *backend* seguano un pattern coerente, migliorando la leggibilità e la manutenibilità del codice;

5. **Efficienza**: generare automaticamente il codice a partire da una specifica ben definita rende il processo di sviluppo più efficiente, riducendo il carico di lavoro manuale e velocizzando i tempi di sviluppo.

I componenti *Angular* creati, per acquisire informazioni e interagire con il *backend*, dovranno semplicemente sottoscriversi agli Observable restituiti dalle funzioni dei servizi generati, i quali eseguono le richieste agli endpoint desiderati.

5.3.6 Gestione feedback

Per fornire all'utente informazioni sullo stato del sistema e sugli esiti delle sue operazioni, è stato implementato un sistema di *feedback* che consente la visualizzazione di messaggi simili a quelli illustrati nella Figura 5.16. Questi messaggi possono includere notifiche di successo, avvisi o errori, consentendo all'utente di comprendere meglio il risultato delle sue azioni.

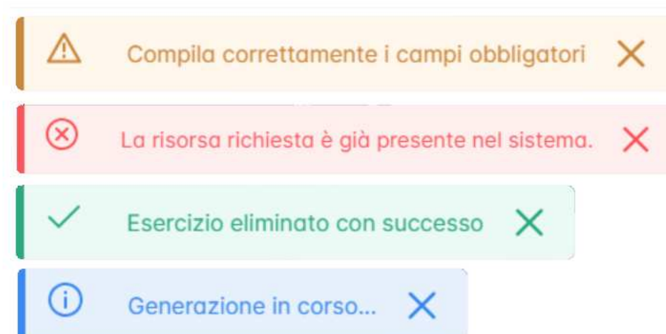


Figura 5.16: Visualizzazione *feedback*

5.3.6.1 Gestione degli errori

Per garantire una gestione centralizzata degli errori, è stato implementato il servizio *Angular ErrorHandlerService*, il quale implementa l'interfaccia *ErrorHandler*. Questa interfaccia definisce un singolo metodo chiamato *handleError*, che accetta un parametro di tipo *any* rappresentante l'errore che è stato rilevato.
Quando viene implementata, l'interfaccia *ErrorHandler* consente di personalizzare la gestione degli errori a livello globale all'interno di un'applicazione *Angular*. Questo servizio è responsabile della gestione degli errori che si verificano nell'applicazione *Angular*.

5.3.6.2 Classe *MessageHandler*

La classe *MessageHandler* è stata sviluppata con l'obiettivo di fornire un meccanismo centralizzato per la gestione dei *feedback* agli utenti all'interno dell'applicazione. È progettata per essere estesa dai componenti che necessitano di gestire o visualizzare messaggi di successo, avvisi, informazioni o errori.
Questa classe espone operazioni semplici per manipolare una lista di messaggi che possono includere lo stato (come errore, avvertenza o informazione), una descrizione testuale e, opzionalmente, un codice di errore.
Inoltre, la classe fornisce un metodo per gestire gli errori, incluso quelli provenienti dal *backend*. Utilizzando il servizio *ErrorHandlerService* precedentemente creato, gli errori vengono catturati e aggiunti automaticamente alla lista dei messaggi. Questo approccio consente ai componenti che estendono la classe di accedere ai messaggi di errore pronti per essere mostrati agli utenti, riducendo la duplicazione del codice e centralizzando la gestione degli errori.
In particolare, grazie alla memorizzazione dei codici di errore, come quelli HTTP del *backend*, ogni componente può personalizzare i messaggi di errore in base ai codici presenti nella lista, fornendo un'esperienza utente più ricca e informativa.

5.3.7 Standalone e lazy loading

Le componenti sono state sviluppate utilizzando un approccio *standalone*, il che significa che ciascuna componente è stata creata come un'entità senza dipendenze implicite

da altre parti dell'applicazione.
Questo approccio è stato adottato per diverse motivazioni di seguito descritte.

1. ***Lazy Loading***: utilizzando componenti *standalone*, è possibile implementare il *lazy loading*, un'ottima pratica per migliorare le prestazioni dell'applicazione. Il *lazy loading* consente di caricare solo le componenti necessarie all'avvio dell'applicazione, riducendo il tempo di caricamento iniziale e migliorando l'esperienza utente complessiva;

2. **Modularità**: il design delle componenti *standalone* promuove la modularità dell'applicazione. Ogni componente è autosufficiente e può essere sviluppato, testato e mantenuto indipendentemente dalle altre parti dell'applicazione. Ciò semplifica la gestione del codice, facilita la collaborazione nello sviluppo e favorisce una migliore organizzazione del progetto;

3. **Riusabilità**: le componenti *standalone* possono essere facilmente riutilizzate in diverse parti dell'applicazione o anche in progetti futuri. Questo permette di massimizzare l'efficienza dello sviluppo, riducendo la duplicazione del codice e promuovendo la coerenza e l'uniformità nell'interfaccia utente.

5.3.8 Angular Reactive Form

I *Reactive Forms* di *Angular* sono uno strumento per la gestione dei form all'interno delle applicazioni web. Basati su un modello reattivo, consentono una gestione dinamica e reattiva dei dati attraverso l'uso di Observable. La decisione di utilizzare i *Reactive Forms* per l'intera applicazione è stata guidata principalmente dalla necessità di gestire efficacemente la validazione dei dati. Questi form integrano funzionalità per implementare regole di validazione complesse e gestire i messaggi di errore in modo intuitivo, semplificando lo sviluppo e migliorando l'affidabilità complessiva dell'applicazione.

5.3.9 Maschere frontend

Le interfacce *frontend* per utenti autenticati sono basate su un componente padre che definisce una struttura fissa per ogni vista. Essa include:

- ***l'header***: dove sono presenti:
  - il Logo dell'applicazione;
  - un Avatar con menu secondario per:
    - visualizzare le informazioni dell'account;
    - effettuare il *logout*.
- **il menu di navigazione principale**: un componente che permette di accedere alle diverse sezioni dell'applicazione;
- **il menu delle notifiche:** un componente dedicato alla visualizzazione delle notifiche;
- **la breadcrumb:** un componente che indica il percorso di navigazione all'interno dell'applicazione;
- **il *chatbot*:** un componente che permette l'interazione con un assistente personale di fitness.

5.3.9.1 Home page, registrazione, conferma email e autenticazione

Per le viste relative alle schermate di autenticazione, conferma dell'email, registrazione e alla home page di presentazione, sono stati sviluppati quattro componenti separati che **non condividono** i componenti appena descritti delle viste destinate agli utenti autenticati (vedi Figura 5.17). In particolare i componenti sono i seguenti:

- **componente homepage**: presenta le informazioni e le funzionalità principali dell'applicazione, permette di accedere alla vista di registrazione e autenticazione;
- **componente di registrazione**: gestisce l'inserimento delle credenziali per la registrazione;
- **componente di conferma email**: consente di inserire il codice di attivazione inviato all'email di registrazione;
- **componente di login**: gestisce l'inserimento delle credenziali per l'accesso.

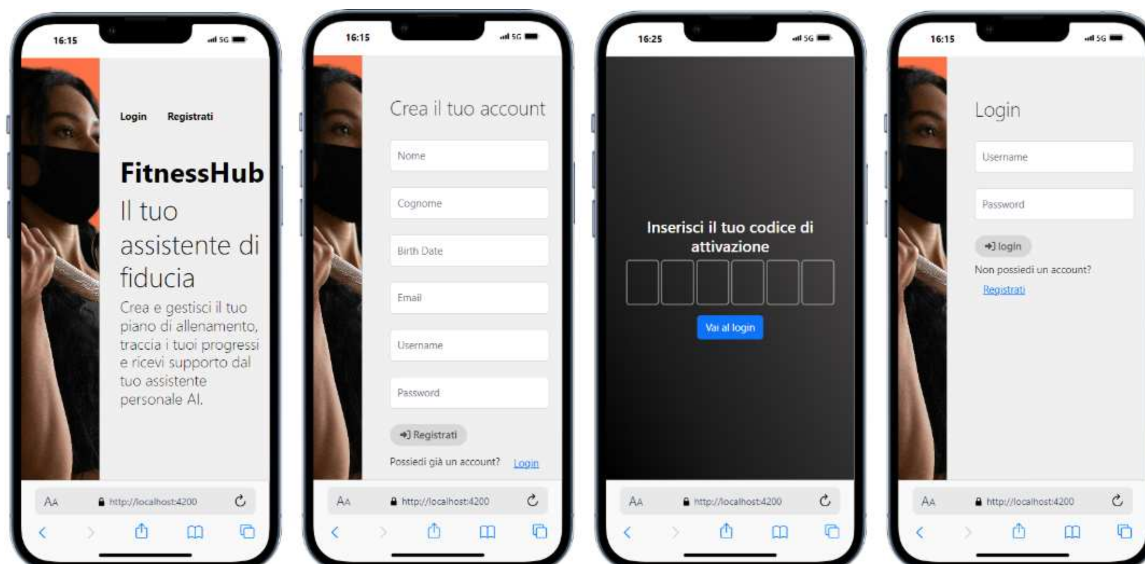


Figura 5.17: Maschere di home page, registrazione, conferma email e autenticazione

5.3.9.2 Visualizzazione homepage utente autenticato

La homepage per gli utenti autenticati offre una panoramica completa dei progressi recenti, visualizzabili tramite un grafico a linee. Inoltre, consente di vedere le attività giornaliere in un'agenda.

In particolare, il grafico permette di monitorare l'andamento del proprio peso in confronto al peso ideale. È anche possibile passare a un grafico che mostra la percentuale di massa grassa e massa muscolare nel tempo.

Per una visione più dettagliata dei progressi, è disponibile una tabella paginata che include tutti i dati pertinenti (vedi Figura 5.18).

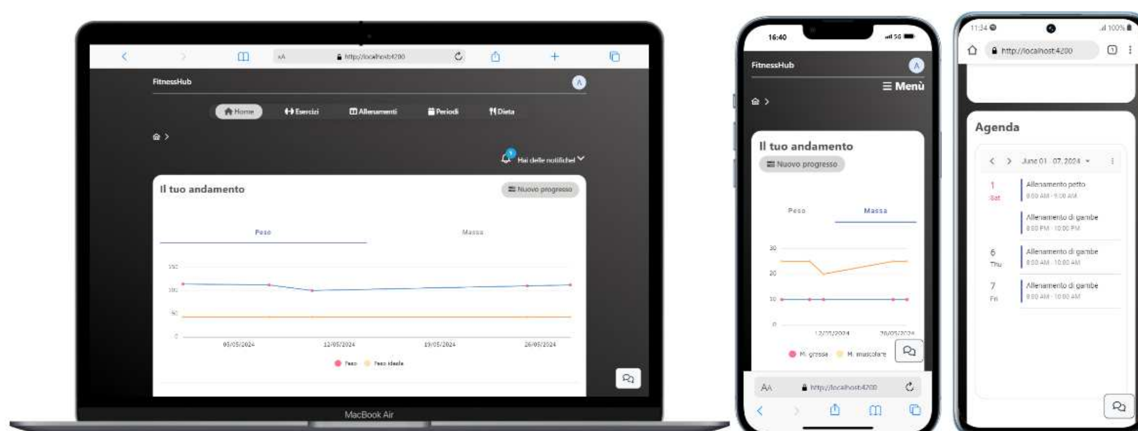


Figura 5.18: Home page utente autenticato

5.3.9.3 Visualizzazione notifiche

Come già descritto nella Sezione 5.3.9, ogni pagina presenta la possibilità di visualizzare le notifiche relative agli allenamenti da svolgere nella giornata. Una volta contrassegnate come lette, queste notifiche non vengono più mostrate. Nella Figura 5.19 è presentata la modalità di interazione con la lista delle notifiche. Espandendo il menu dedicato vengono visualizzati gli allenamenti della giornata, mentre cliccando sulla "X", le notifiche verranno segnate come lette e non saranno più mostrate.

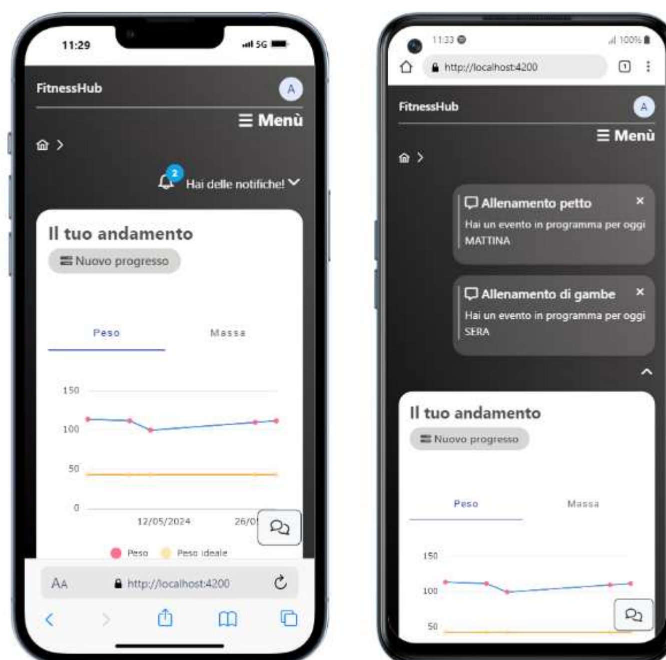


Figura 5.19: Visualizzazione notifiche

5.3.9.4 Creazione e modifica esercizio

Per l'inserimento e la modifica di un esercizio all'interno dell'applicazione è stata creata una vista presentata nella Figura 5.20. Viene permesso l'inserimento di più categorie per ogni esercizio e il caricamento di un immagine rappresentativa è opzionale.

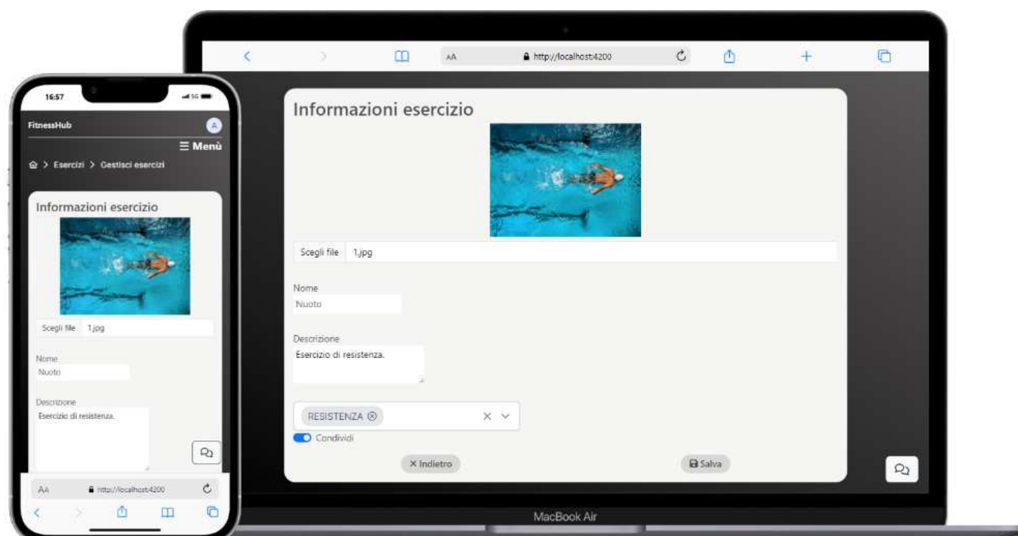


Figura 5.20: Maschera di creazione e modifica esercizi

5.3.9.5 Visualizzazione lista esercizi

Per la visualizzazione della lista degli esercizi disponibili all'utente (ovvero quelli disponibili di default, quelli creati dall'utente stesso e quelli importati dallo *store*), è stata implementata una vista paginata appositamente progettata per mostrare in modo chiaro e ordinato tutti gli esercizi presenti nel sistema (vedi Figura 5.21). La vista, rende disponbile un pulsante per modificare o eliminare ogni esercizio.

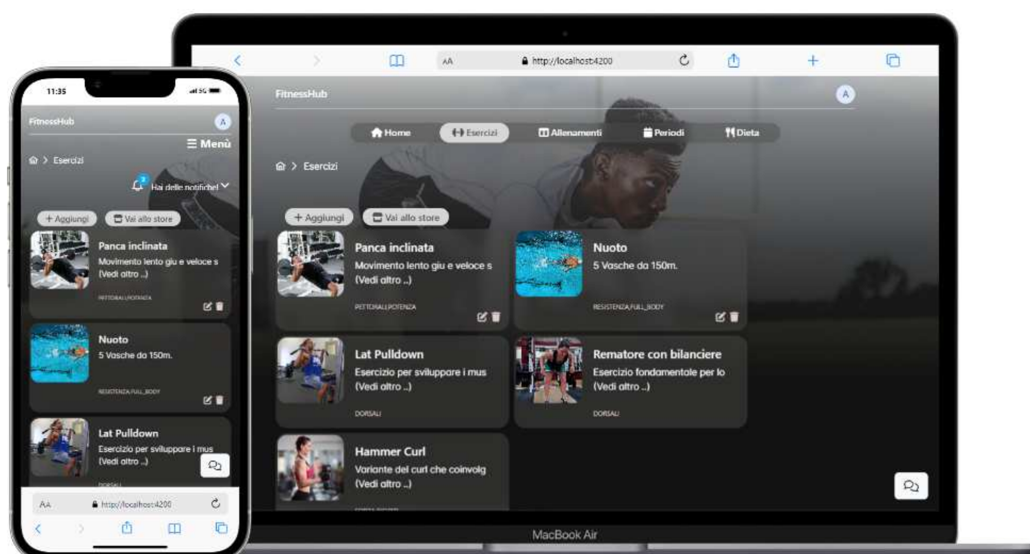


Figura 5.21: Visualizzazione lista esercizi

5.3.9.6 Visualizzazione store esercizi

La visualizzazione dello *store* degli esercizi consente agli utenti di esplorare e cercare gli esercizi che gli altri utenti hanno creato e reso disponbili. Inoltre viene presentato un form per aggiungere filtri sulla categoria degli esercizi e rendere piu rapida la ricerca (vedi Figura 5.22).

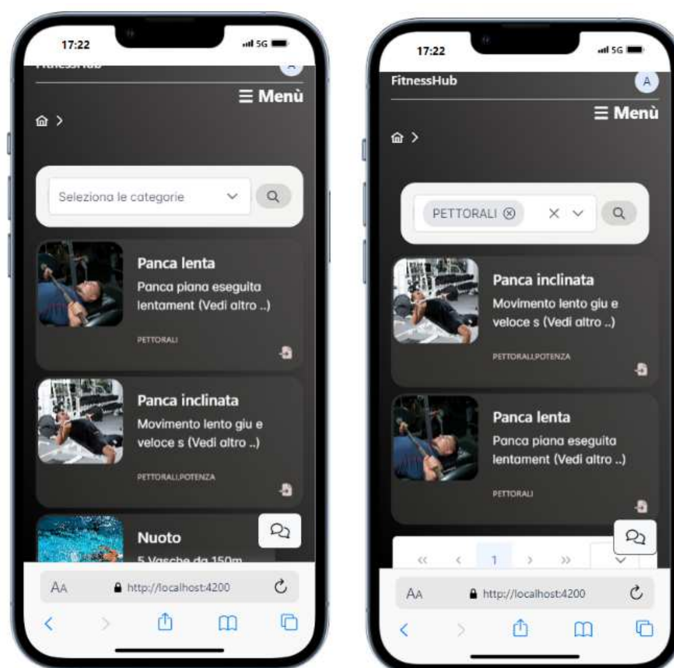


Figura 5.22: Visualizzazione *store* esercizi

5.3.9.7 Creazione e modifica allenamento

Per consentire agli utenti di creare e modificare i propri allenamenti, è stata sviluppata una vista dedicata che offre un'interfaccia intuitiva e facile da usare.
La creazione di un allenamento e divisa in due parti:

1. nella prima vengono inserite informazioni generali sull'allenamento come: nome, descrizione e durata;

2. la seconda parte puo' essere eseguita in modalità differenti:

    - l'utente puo decidere di inserire manualmente i propri esercizi nell'allenamento dalla propria lista e riordinarli a piacere;

- l'utente tramite il pulsante "*Genera con l'IA*" puo' farsi generare l'allenamento dall'intelligenza artificiale che utilizzerà i dati dell'utente (come i progressi) oltre alle informazioni di contesto inserite precedentemente per creare un allenamento personalizzato che puo' essere comunque modificato successivamente dall'utente.

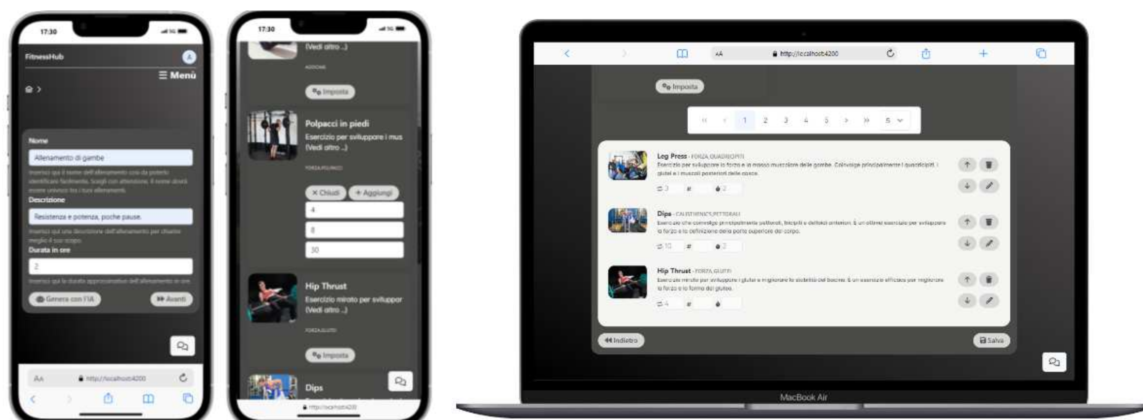


Figura 5.23: Creazione e modifica allenamento

Nella Figura 5.23 sono illustrati i passaggi necessari per la creazione di un allenamento. In primo luogo, si inseriscono le informazioni generali. Successivamente, si aggiungono i vari esercizi, impostando per ciascuno serie, ripetizioni e tempi di recupero. Se necessario, è possibile modificare l'ordine degli esercizi, eliminarli o apportare modifiche a quelli già inseriti.

5.3.9.8 Visualizzazione lista allenamenti e dettagli

La visualizzazione della lista degli allenamenti permette agli utenti di accedere rapidamente ai propri programmi di allenamento. Nella stessa vista, sono presenti pulsanti che consentono l'eliminazione, la modifica e visualizzazione dei dettagli degli allenamenti. La vista dedicata alla visualizzazione dei dettagli di un singolo allenamento offre agli utenti una panoramica completa, includendo anche gli esercizi relativi.

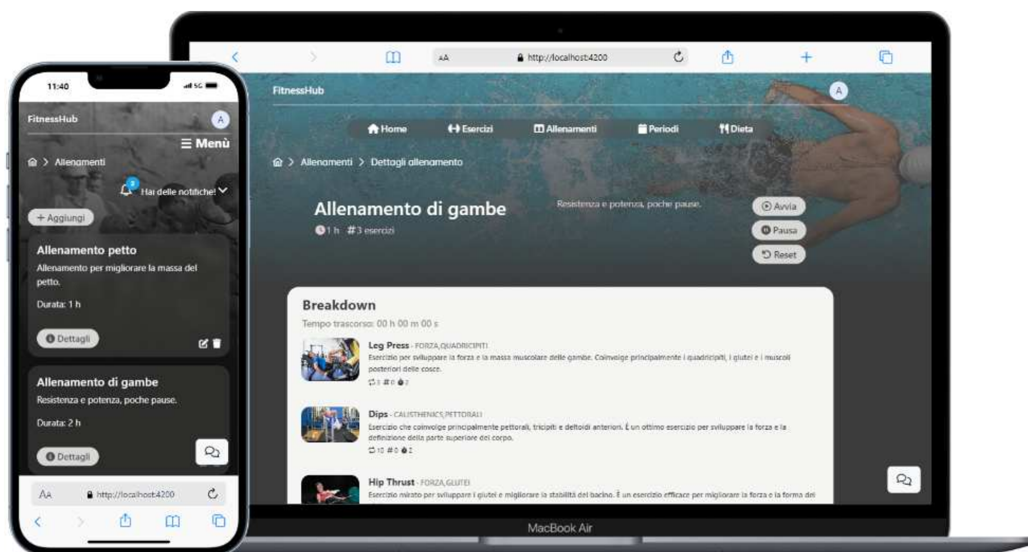


Figura 5.24: Visualizzazione lista allenamenti e dettagli

Nella Figura 5.24 viene presentata la visualizzazione *desktop* del dettaglio di un allenamento mentre sull'interfaccia *mobile* è presente la lista degli allenamenti.

5.3.9.9 Creazione e modifica periodo

Per consentire agli utenti di creare e modificare i propri periodi, è stata sviluppata una vista dedicata che offre un'interfaccia intuitiva e facile da usare.
La creazione di un periodo è suddivisa in due fasi principali:

1. nella prima fase, vengono inserite le informazioni generali sul periodo, tra cui: nome, obiettivo, durata in giorni, data di inizio, data di fine, l'obiettivo ed infine viene chiesto di specificare se il periodo è attivo (vedi interfaccia *desktop* nella Figura 5.25).

2. nella seconda fase, viene richiesto all'utente di inserire i propri allenamenti nel periodo, specificando il giorno e il momento desiderato (vedi interfaccia *mobile* nella Figura 5.25).

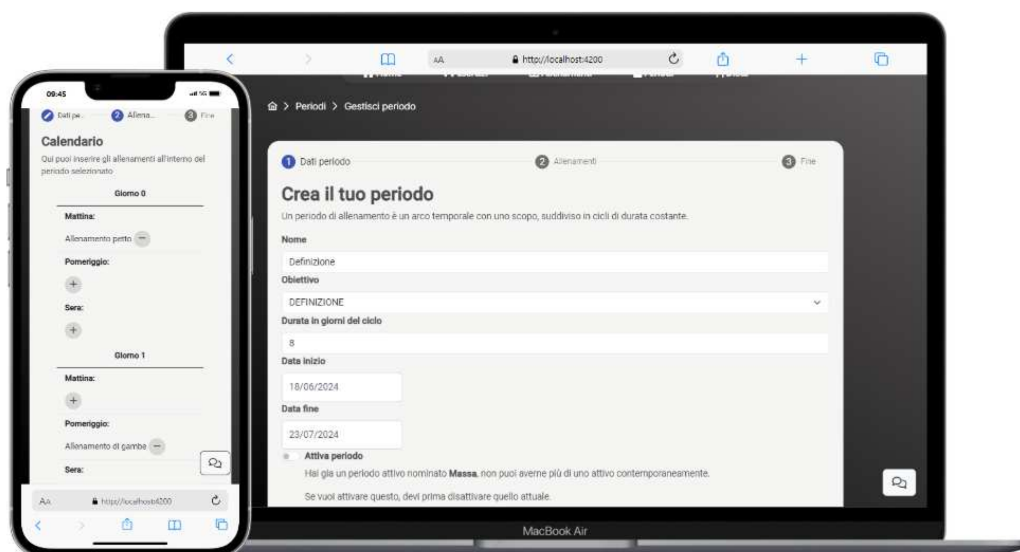


Figura 5.25: Creazione e modifica periodo

5.3.9.10 Visualizzazione lista periodi e dettagli

La visualizzazione della lista dei periodi consente agli utenti di accedere rapidamente ai propri periodi di allenamento. La vista espone dei pulsanti per permettere l'eliminazione, la modifica e la visualizzazione dei dettagli di ogni periodo. La vista dedicata alla visualizzazione dei dettagli del periodo fornisce agli utenti una panoramica approfondita di un singolo periodo inclusi gli allenamenti su di un calendario.

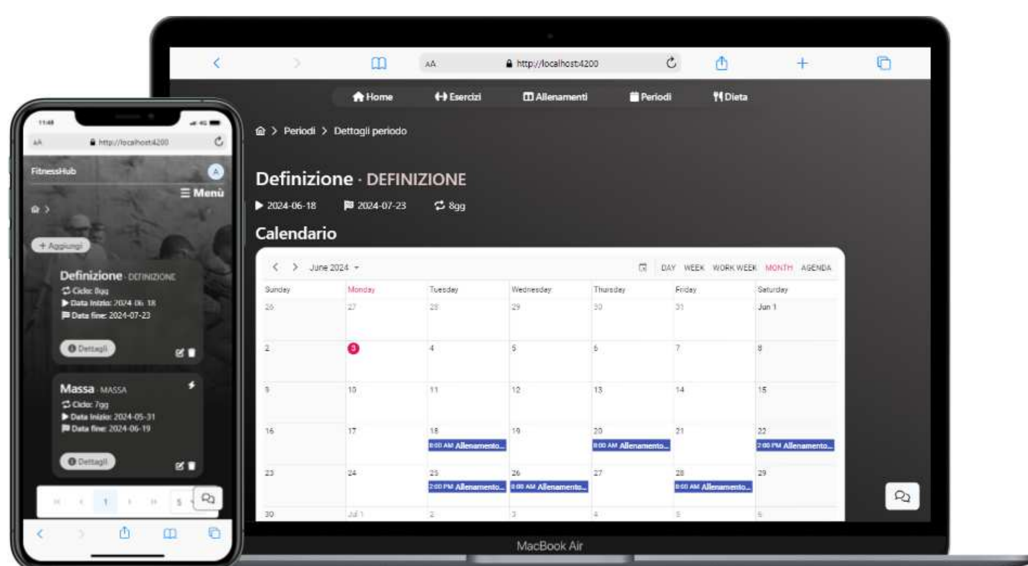


Figura 5.26: Visualizzazione lista periodi e dettagli

Nella Figura 5.24 viene presentata la visualizzazione *desktop* del dettaglio di un periodo mentre sull'interfaccia *mobile* è presente la lista dei periodi.

5.3.9.11 Creazione, modifica e visualizzazioni progressi

Per consentire agli utenti di registrare e monitorare i loro progressi nel tempo, sono state create delle viste che permettono l'inserimento, la modifica e la visualizzazione di tali informazioni. Queste viste sono accessibili dalla *home page* dove oltre alla visualizzazione su grafico a linee descritta nella Sezione 5.3.9.2, è presente un grafico a torta che espone la percentuale di massa grassa e magra dell'ultima misurazione registrata. Inoltre è disponibile una tabella paginata che permette di esaminare i progressi in modo più dettagliato e di accedere alle funzionalità di modifica o eliminazione di quest'ultimi.
Nella Figura 5.27 è presente la visualizzazione *mobile* dell'interfaccia utente per la creazione e la modifica dei progressi, mentre nella visualizzazione *desktop* viene esposta la tabella dei progressi e il grafico a torta sopra citati.

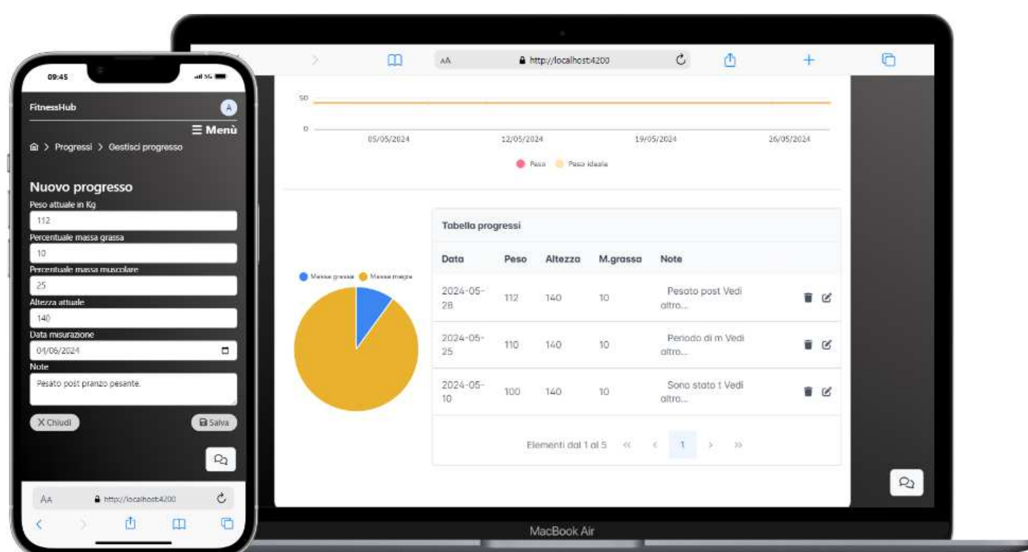


Figura 5.27: Maschere di creazione,modifica e visualizzazione progressi.

5.3.9.12 Modifica informazioni personali

Per consentire agli utenti di gestire e aggiornare le proprie informazioni personali o di modificare la propria password è stata sviluppata una vista apposita (vedi Figura 5.28).

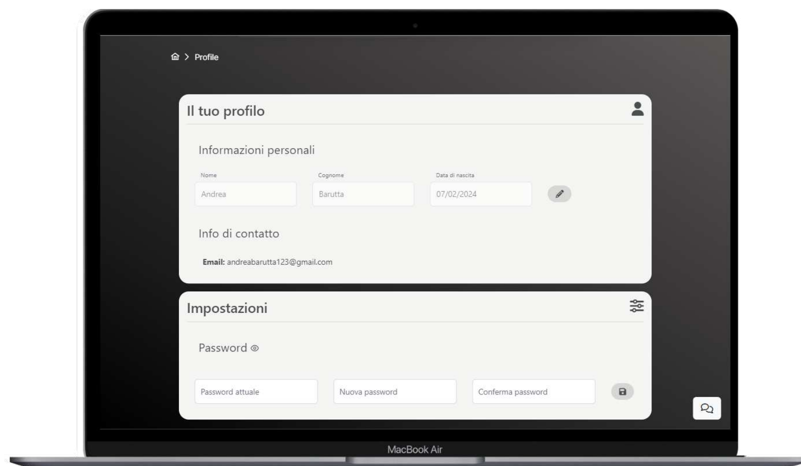


Figura 5.28: Maschera informazioni personali

5.3.9.13 Visualizzazione piano alimentare personalizzato

L'applicazione consente di generare, tramite intelligenza artificale, un piano alimentare personalizzato inserendo alcune informazioni di base, come il titolo, una descrizione e una o più categorie di dieta (ad esempio, "vegetariana", "vegana", "senza lattosio", ecc.). Questo piano viene creato grazie ad un servizio sviluppato dal secondo membro del team.

Il servizio restituisce un oggetto strutturato che contiene il piano settimanale personalizzato dell'utente con una breve motivazione, basato su informazioni personali come i progressi, gli allenamenti del periodo attivo e i dati forniti nel modulo iniziale (titolo, descrizione, ecc.). Il piano alimentare per ciascun giorno della settimana include gli alimenti con le relative calorie e quantità in grammi, suddivisi per colazione, pranzo e cena.

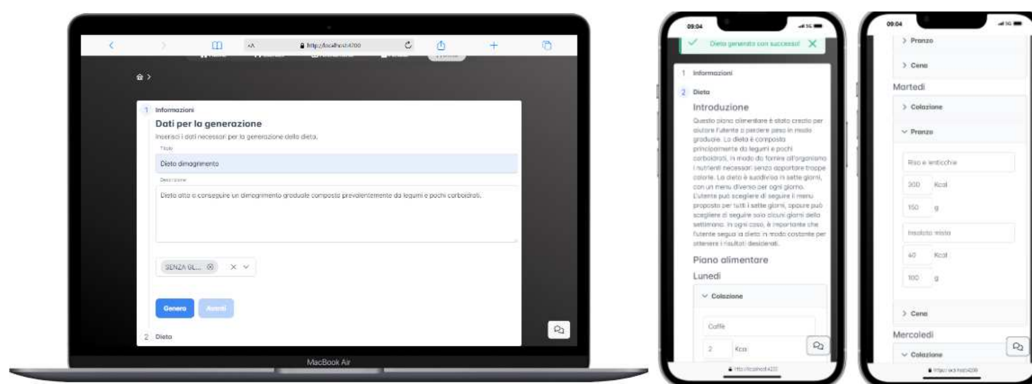


Figura 5.29: Maschera piano alimentare

Nella Figura 5.29 viene presentato il processo di creazione del piano alimentare. Una volta creato tale piano viene data all'utente la possibilità di modificarne i campi (come gli alimenti e la quantità associata) e stampare il risultato.

5.3.9.14 Maschera per il chatbot

Per consentire agli utenti di interagire con il *chatbot* da qualsiasi pagina, è stato creato un componente che si interfaccia con gli endpoint sviluppati dal secondo membro del team. Questo componente è accessibile da ogni pagina una volta che l'utente si è autenticato.

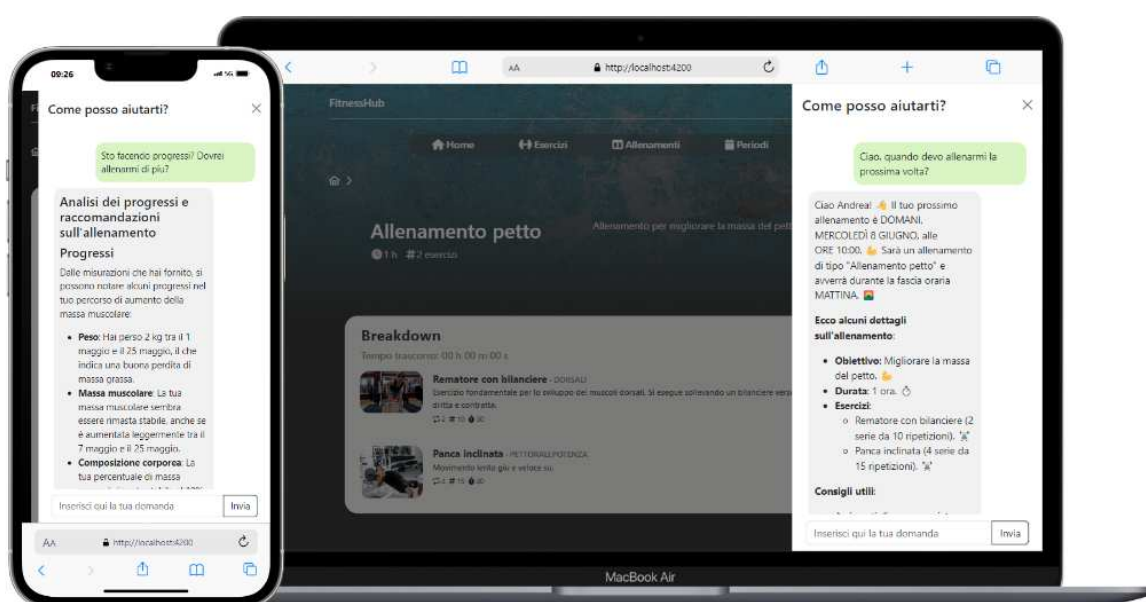


Figura 5.30: Maschera *chatbot*

Nella Figura 5.30 si può notare come il *chatbot* fornisca risposte personalizzate basate sul contesto dell'utente che interagisce con esso.

5.4 Integrazione con il chatbot

In questa sezione verrà illustrata l'integrazione del *chatbot* con l'applicazione, descrivendo come è stata resa possibile una collaborazione efficace e immediata tra i vari componenti del sistema.

5.4.1 Il chatbot

Il secondo membro del team, il cui piano di stage prevedeva la creazione di un *chatbot* basato su RAG, ha semplificato e reso immediata l'integrazione collaborativa nell'applicazione. È stato creato un endpoint nell'applicazione *Spring* utilizzata come *backend*, che permette di porre domande e ottenere risposte sempre basate sul contesto specifico dell'utente (ultimi progressi, allenamenti programmati e informazioni personali).

Come descritto nella Sezione 5.1, l'applicazione *Spring*, dopo aver ottenuto le informazioni contestuali dell'utente dal database, inoltra la richiesta ad un servizio dedicato accessibile tramite API (vedi Figura 5.1). La risposta ricevuta viene poi restituita al *frontend* e presentata all'utente in pochi secondi.

Il *chatbot* è stato potenziato tramite RAG con documenti specifici per il fitness, creando così un assistente personale specializzato nelle attività di allenamento e benessere.

5.4.2 Allenamenti e piani alimentari personalizzati

Come descritto nella Sezione 5.3.9.7 e nella Sezione 5.3.9.13, il secondo membro del team ha creato e reso disponibili due ulteriori endpoint per la creazione di allenamenti e piani alimentari personalizzati. Tali endpoint seguono lo stesso percorso di quello relativo al *chatbot* ma sono progettati per restituire una risposta JSON strutturata, trasferibile e automaticamente convertibile in interfaccia *TypeScript* utilizzabile dal *frontend*. Questo ha reso l'integrazione collaborativa e l'utilizzo di tali endpoint immediati.

La specifica delle API progettata in *StopLight* ha permesso un lavoro asincrono efficace e ha reso l'integrazione del lavoro rapida e priva di imprevisti.

Capitolo 6

Verifica e validazione

In questo capitolo verranno descritti gli strumenti e le pratiche adottate per i processi di verifica e validazione.

6.1 Strumenti

Di seguito, presentiamo gli strumenti utilizzati per la verifica e la validazione.

6.1.1 GitHub Actions

GitHub Actions [49] è una funzionalità offerta da *GitHub* che consente di automatizzare i flussi di lavoro del software direttamente nel *repository* di progetto. Grazie a questo strumento, ogni volta che viene eseguito un Push$_G$ nel *repository*, viene avviato automaticamente un *workflow* che esegue i test *Maven*, calcola la Code coverage$_G$ tramite il plugin *JaCoCo* e svolge l'analisi statica del codice.
L'esito di questi processi è visibile attraverso dei badge nel file `README` del *repository* di progetto (vedi Figura 6.1). Questi badge forniscono una visualizzazione immediata dello stato dei test, del livello di Code coverage e dei risultati dell'analisi statica, permettendo di monitorare costantemente la qualità del codice e di individuare eventuali problemi rapidamente.

Figura 6.1: File README della *repository* di progetto

6.1.2 SonarCloud

SonarCloud [34] è una piattaforma di analisi statica del codice, utilizzata principalmente per valutare la qualità del codice sorgente. Questo strumento ha permesso di rilevare problemi di qualità, vulnerabilità di sicurezza e Code smells. *SonarCloud* offre una visione dettagliata della qualità del codice, permettendo di identificare e correggere problemi prima che possano causare malfunzionamenti o compromettere la sicurezza del software. L'integrazione con *GitHub Actions* permette di eseguire l'analisi automaticamente e di visualizzare i risultati direttamente nel file `README` del *repository* di progetto (vedi sezione *"Backend Static analysis"* nella Figura 6.1).
Per visualizzare informazioni piu dettagliate questo strumento offre anche una propria Dashboard in *cloud*.

6.1.3 JaCoCo

JaCoCo [18] è uno strumento di Code coverage per *Java*, utilizzato per misurare la copertura del codice durante l'esecuzione dei test. Questo strumento consente di identificare quali parti del codice sono state eseguite durante l'esecuzione dei test e quali sono rimaste non testate.

6.1.4 Coveralls

Coveralls [8] è un servizio di integrazione continua che fornisce analisi della copertura del codice per progetti basati su *Git*. *Coveralls* è stato utilizzato per fornire un'analisi dettagliata della Code coverage. Questo strumento si integra con *GitHub Actions*

e raccoglie i dati di *coverage* generati da *JaCoCo*, presentandoli in un'interfaccia intuitiva che permette di visualizzare quali parti del codice sono state coperte dai test e quali no. *Coveralls* offre anche dei badge che possono essere inclusi nel `README` del *repository GitHub*, fornendo un feedback visivo sulla percentuale di codice coperto dai test (vedi badge "*codecov*" in Figura 6.1).

6.1.5 JUnit 5

JUnit 5 [20] è un Framework di testing per il linguaggio di programmazione *Java* ed è stato utilizzato per scrivere ed eseguire i test unitari. L'integrazione con *Maven* e *GitHub Actions* permette di eseguire automaticamente i test ogni volta che viene apportata una modifica al codice del *repository* di progetto, assicurando che eventuali errori vengano individuati e corretti tempestivamente. L'esito dell'esecuzionde di tali test è visibile nel badge "*Java CI with Maven*" del file `README` del *repository* di progetto (vedi Figura 6.1).

6.1.6 Mockito

Mockito [43] è un Framework di *mocking* che consente di creare Mock$_{G}$ di oggetti per sostituire le loro dipendenze durante i test. *Mockito* è stato utizzato nell'implementazione dei test unitari per isolare il codice in test da altre parti del sistema e concentrarsi solo sulla logica specifica che si desidera testare.

6.2 Verifica

Nel contesto dello sviluppo del progetto, la limitata disponibilità di ore preventivate per le attività di test ha richiesto un'attenta pianificazione.

6.2.1 Test di unità

Data l'importanza di mantenere il *backend* del sistema sempre privo di errori e di comportamenti non controllati, si è deciso di concentrare gli sforzi principalmente sui test di unità per questa parte del progetto, ponendo come obiettivo almeno l'**80%** di Code coverage. Il *backend* rappresenta il cuore del sistema, e garantire la sua

affidabilità e correttezza è fondamentale per il corretto funzionamento dell'intero applicativo. Inoltre, considerato il suo contatto diretto con lo *storage layer*, un percorso non controllato potrebbe risultare particolarmente grave.
I test di unità sono parte integrante della *suite di test* eseguita automaticamente ad ogni Push nel repository.

6.2.2 Test di integrazione

Per quanto riguarda i test di integrazione svolti, ci siamo focalizzati esclusivamente sull'interazione tra il *backend Spring* e il servizio RAG. Il *backend Spring* è incaricato di recuperare i dati di contesto dell'utente, i quali vengono quindi utilizzati dal servizio RAG per generare risposte per il chatbot, nonché per creare allenamenti e piani alimentari basati sul contesto fornito dal *backend*.
Questi test sono stati condotti in collaborazione con il secondo membro del team e sono parte integrante della *suite di test* eseguita automaticamente ad ogni Push nel repository.

6.2.3 Analisi statica

Per garantire la robustezza e la qualità del codice del *backend* dell'applicazione, è stata condotta un'analisi statica approfondita utilizzando *Sonarcloud* (vedi Sezione 6.1.2). L'obiettivo principale di questo processo è quello di individuare potenziali difetti nel codice sorgente, come bug, vulnerabilità e violazioni delle best practice di programmazione. L'utilizzo di questo strumento ha giocato un ruolo cruciale nell'individuare e risolvere una grave vulnerabilità di sicurezza legata agli attacchi di ***Path traversal*** nel processo di salvataggio delle immagini relative agli esercizi personali creati dagli utenti.

6.2.4 Processo di integrazione nel *branch* principale

Ogni modifica al codice seguiva il *workflow Gitflow*, che prevede l'uso di Branch$_G$ specifici per lo sviluppo delle nuove funzionalità e la risoluzione dei bug. Prima che una modifica potesse essere integrata nel Branch principale, doveva attraversare le seguente *pipeline:*

1. **creazione di una *Pull Request (PR)*$_G$**: ogni modifica veniva proposta tramite una Pull request nel *repository Git*. Questo permetteva di isolare le modifiche e di facilitare la revisione e il testing;

2. **revisione della *Pull request***: un altro membro del team eseguiva una revisione approfondita della Pull request, analizzando anche le metriche calcolate automaticamente dagli strumenti di analisi integrati;

3. **integrazione delle modifiche nel Branch principale**: solo dopo aver superato tutte le verifiche automatiche e la revisione manuale, le modifiche venivano integrate nel Branch principale del *repository* progetto tramite accettazione della Pull request.

L'integrazione continua e la verifica automatizzata hanno permesso di individuare e risolvere tempestivamente eventuali problemi, migliorando l'affidabilità e la manutenibilità del software.

6.3 Validazione

La validazione del codice e del prodotto è stata effettuata dal tutor aziendale durante le revisioni settimanali degli *Sprint*. In queste sessioni di *Sprint review*, il codice veniva illustrato al tutor ad alto livello che forniva critiche costruttive e suggerimenti per migliorare il lavoro nelle fasi successive, reindirizzando il progetto quando necessario. Questo processo di validazione mirava a garantire che il codice e il prodotto rispettassero gli standard desiderati in termini di leggibilità, correttezza e qualità complessiva.

Inoltre, questo processo periodico ha consentito di valutare se le attività svolte durante lo *Sprint* fossero state completate con successo o meno, fornendo una chiara indicazione sullo stato di avanzamento del lavoro. La validazione di un'attività indicava che non era più necessario intervenire su di essa, semplificando il processo decisionale e consentendo di concentrare gli sforzi sulle aree che richiedevano ancora attenzione o miglioramenti.

L'ultima attività di validazione svolta, è stata condotta per garantire che il prodotto soddisfacesse tutte le richieste iniziali. Durante questa attività il tutor aziendale ha

fornito la conferma formale del soddisfacimento dei requisiti, come descritto nella Sezione 6.3.1.

6.3.1 Stato user stories

Tutte le funzionalità richieste dalle *user stories* esposte nella Tabella 3.1 sono state soddisfatte (vedi Tabella 6.1).

Codice	Tipologia	Stato
U1-O	Obbligatorio	Soddisfatto
U2-O	Obbligatorio	Soddisfatto
U3-O	Obbligatorio	Soddisfatto
U4-O	Obbligatorio	Soddisfatto
U5-O	Obbligatorio	Soddisfatto
U6-O	Obbligatorio	Soddisfatto
U6-O	Obbligatorio	Soddisfatto
U8-O	Obbligatorio	Soddisfatto
U12-O	Obbligatorio	Soddisfatto
U13-O	Obbligatorio	Soddisfatto
U14-O	Obbligatorio	Soddisfatto
U15-O	Obbligatorio	Soddisfatto
U9-D	Desiderabile	Soddisfatto
U10-D	Desiderabile	Soddisfatto
U16-D	Desiderabile	Soddisfatto
U17-D	Desiderabile	Soddisfatto
U18-F	Facoltativo	Soddisfatto

Tabella 6.1: Stato *User stories*.

Capitolo 7

Conclusioni

In questo capitolo verrà effettuata una breve analisi sul raggiungimento degli obiettivi fissati nel piano di lavoro e sulle conoscenze acquisite. Si concluderà con una valutazione finale del percorso di stage svolto.

7.1 Raggiungimento degli obiettivi

Tutti gli obiettivi previsti dal piano di lavoro, come esposti nella Sezione 1.3.2, sono stati raggiunti nei tempi previsti (vedi Tabella 7.1).

Codice	**Obiettivo**	**Tipologia**	**Stato**
O-01	Acquisizione competenze approfondite su tecnologie moderne per lo sviluppo di Web App.	Obbligatorio	Soddisfatto
O-02	Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma.	Obbligatorio	Soddisfatto

Continua nella prossima pagina...

Tabella 7.1 – Continua

Codice	Obiettivo	Tipologia	Stato
O-03	Nella fitness Web App realizzata, l'utente deve poter effettuare la registrazione e autenticazione, definire il proprio stato atletico e di salute con relativi obiettivi di miglioramento, oltre che tracciare i propri progressi.	Obbligatorio.	Soddisfatto
O-04	Nella fitness Web App realizzata, l'utente deve poter creare, gestire e pianificare le schede di allenamento.	Obbligatorio	Soddisfatto
O-05	Nella fitness Web App realizzata, l'utente deve poter creare e gestire esercizi personalizzati per la progettazione delle schede di allenamento.	Obbligatorio	Soddisfatto
O-06	Nella fitness Web App realizzata, l'utente deve poter ricevere promemoria di allenamento.	Obbligatorio	Soddisfatto
D-01	Inserimento di ulteriori funzionalità alla Web App come la possibilità di ricevere supporto da altri utenti	Desiderabile.	Soddisfatto
D-02	Integrazione nella Web App di un chat-bot.	Desiderabile.	Soddisfatto
F-01	Inserimento di ulteriori funzionalità alla Web App come la gestione del piano alimentare settimanale.	Facoltativo	Soddisfatto

Tabella 7.1: Stato obiettivi piano di lavoro.

7.2 Consuntivo orario

La Tabella 7.2 riassume la differenza tra le ore preventivate per ciascuna attività e le ore a consuntivo. Si possono notare leggere variazioni rispetto agli orari preventivati. In particolare, lo studio preliminare e l'integrazione con il lavoro del secondo membro del team hanno richiesto meno tempo del previsto. L'integrazione del chatbot e delle sue funzionalità è stata semplificata grazie a una progettazione di qualità che ha facilitato una collaborazione efficace e ha ridotto la necessità di modifiche e aggiustamenti successivi. Tuttavia, lo sviluppo delle funzionalità dell'applicazione ha richiesto un po' più di tempo del previsto. Questo aumento è stato dovuto alla complessità delle funzionalità implementate e all'attenzione dedicata per garantire che fossero robuste e *user-friendly*.

Attività	**Ore Previste**	**Ore Consuntivate**
Formazione sulle tecnologie	100	95
Fase di sviluppo	180	190
Progettazione dell'architettura	30	30
Sviluppo del *backend* con *Spring*	70	65
Sviluppo del *frontend* con *Angular*	60	65
Testing e debugging	20	20
Collaudi ed integrazioni finali	20	15
Totale	**300**	**300**

Tabella 7.2: Consuntivo orario delle attività

7.3 Conoscenze acquisite

Il percorso di stage mi ha permesso di acquisire nuove conoscenze approfondite nello sviluppo di applicazioni *full stack*, utilizzando Framework all'avanguardia come *Spring* e *Angular*. Inoltre, ho avuto l'opportunità di sviluppare un sistema di autenticazione e autorizzazione tramite Token JWT, un'esperienza per me nuova che mi ha consentito di approfondire anche l'ambiente *Spring Security*. Ho potuto consolidare le mie competenze in *Docker*, sullo sviluppo di *pipeline* di testing automatiche e ho condotto uno studio teorico sui sistemi di *chatbot* basati su RAG. Infine, ho imparato ad utilizzare nuove librerie di User Interface (UI) per il *frontend* e strumenti per la progettazione, documentazione e test delle API.

7.4 Valutazione personale

Sono estremamente soddisfatto del percorso di stage svolto. È stato davvero gratificante osservare l'applicazione prendere forma giorno dopo giorno, progettandone le funzionalità, passo dopo passo, insieme al mio compagno di progetto, nonché amico. L'ambiente aziendale si è dimostrato altamente collaborativo, facilitando lo scambio di idee e il supporto reciproco tra i membri del team.
Durante il periodo di stage, ho avuto l'opportunità di mettere in pratica le competenze acquisite nel corso degli anni di studio universitario al fine di sviluppare un'applicazione robusta e scalabile, con tecnologie mai usate prima e pronta per l'uso immediato. Nessun dettaglio è stato trascurato; ogni aspetto del progetto è stato realizzato con l'obiettivo di garantire un'applicazione professionale.
L'applicazione sviluppata rappresentava un progetto nel cassetto per me e il mio compagno. Grazie a questo stage, abbiamo avuto l'opportunità di svilupparla e perfezionarla. Il nostro obiettivo è portarla in produzione in futuro, e questo stage ci ha fornito le competenze e la motivazione necessarie per raggiungere questo traguardo.

Acronimi e abbreviazioni

API Application Programming Interface. 5

DTO Data Transfer Object. 36

HTTP Hypertext Transfer Protocol. 18

RAG Retrieval-Augmented Generation. 24

SMTP Simple Mail Transfer Protocol. 18

UI User Interface. 82

UML Unified Modeling Language. 32

Glossario

Allenamento Insieme di esercizi svolti in sequenza. 2, 15, 35, 66

API RESTful [6] Un'API REST, nota anche come API RESTful, è un'interfaccia di programmazione delle applicazioni (API o API web) conforme ai vincoli dello stile architetturale REST, che consente l'interazione con servizi web RESTful. Il termine REST, coniato dall'informatico Roy Fielding, è l'acronimo di REpresentational State Transfer. 24

Branch [11] Un branch è una copia indipendente di una linea di sviluppo all'interno di un sistema di controllo versione come Git. Ogni branch permette di lavorare su modifiche specifiche senza influenzare il branch principale o altre linee di sviluppo. 76, 77

Code coverage Il code coverage, o copertura del codice, è una metrica utilizzata per misurare quanto del codice sorgente di un programma viene eseguito durante l'esecuzione dei test. 73–75

Code smells Il termine code smell viene usato quando il codice in esame è stato scritto senza seguire buone pratiche di programmazione . 74

CI [4] La Continuous Integration (CI) è una pratica di sviluppo del software in cui i membri del team integrano frequentemente il proprio codice nel repository condiviso. Ogni integrazione attiva automaticamente una serie di build e test automatizzati per verificare se il codice introdotto ha introdotto errori o problemi di integrazione con il codice esistente. 18

DTO [44] I DTO (Data Transfer Object) sono oggetti utilizzati per trasferire dati tra diverse parti di un'applicazione, in particolare tra il livello di persistenza

(database) e il livello di presentazione (UI). Un DTO è una semplice classe che non contiene logica di business ma solo attributi e metodi getter e setter. La sua funzione principale è quella di incapsulare i dati in modo da poterli trasferire in modo efficiente e sicuro tra diversi strati dell'applicazione. 36–40, 43, 44, 48, 49

Endpoint URL specifico che permette di accedere ad una risorsa. 18, 21, 27, 35, 38, 41, 43, 44, 46, 47, 49–51, 53–55, 58, 71, 72

Esercizio Attività fisica svolta durante un allenamento. 2, 14, 35, 64

Framework Un framework è un insieme di strumenti, librerie e linee guida che forniscono una struttura predefinita per lo sviluppo di software. Essenzialmente, un framework offre un ambiente di lavoro che permette agli sviluppatori di concentrarsi sulla logica specifica dell'applicazione, riducendo la necessità di scrivere codice da zero per funzionalità comuni. 5, 9, 19, 20, 22, 23, 25, 47, 75, 82

ITS [17] Un Issue Tracking System è un sistema software progettato per gestire e tenere traccia di problemi, bug, richieste di funzionalità e altre attività correlate al processo di sviluppo di software o di gestione dei progetti. Fornisce un meccanismo centralizzato per la registrazione, l'assegnazione, il monitoraggio e la risoluzione degli issue nel corso del tempo, consentendo ai team di lavoro di collaborare efficacemente per affrontare e risolvere i problemi. 11, 18

Milestone [7] Una "milestone" nello sviluppo di un progetto software è un punto di riferimento significativo che rappresenta il completamento di una fase importante o il raggiungimento di un obiettivo cruciale all'interno del progetto. Le milestone sono spesso utilizzate per suddividere il percorso di sviluppo in tappe gestibili e per tenere traccia dei progressi compiuti lungo il cammino. 11

Mock [26] In programmazione, un "mock" è un oggetto simulato che replica il comportamento di un oggetto reale durante i test. I mock sono utilizzati per sostituire le dipendenze degli oggetti reali durante i test di unità, consentendo di isolare il codice in test e concentrarsi sulla logica specifica che si desidera testare. 75

Modulo Angular [1] In Angular, un modulo è un blocco fondamentale per la strutturazione di un'applicazione. Permette di organizzare il codice in unità logiche e coese, raggruppando componenti, servizi, direttive e pipe correlati. I moduli offrono diversi vantaggi che rendono lo sviluppo più strutturato, scalabile e manutenibile. 56

Observable [33] Observable è una struttura essenziale nel paradigma della programmazione reattiva. Essenzialmente, è una fonte di dati che può emettere valori in modo asincrono nel tempo e questi valori possono essere osservati e manipolati da altre parti del codice che sono iscritte all'Observable. 58, 60

Periodo Intervallo di tempo definito, caratterizzato da una data di inizio e una data di fine, durante il quale si esegue un programma strutturato di esercizi. Questo programma è composto da una serie di allenamenti che si ripetono ciclicamente tra le due date, con l'obiettivo di raggiungere specifici obiettivi di fitness o prestazione. 2, 15, 35, 68, 86

Periodo attivo Un periodo di allenamento attivo è un periodo che è in corso e durante il quale l'utente è impegnato in attività fisiche regolari e pianificate, al fine di raggiungere determinati obiettivi di fitness o salute. L'applicazione invierà le notifiche di allenamento solamente del periodo attivo. Un solo periodo puo' essere attivo nello stesso momento. 2, 53, 67

Progresso Dato strutturato che permette il tracciamento della propria composizione corporea. 2, 14, 35, 66, 69

Pull request [3] Una Pull Request (PR) è una funzionalità offerta dai sistemi di controllo versione, come Git, e dalle piattaforme di gestione del codice sorgente, come GitHub, GitLab e Bitbucket. La Pull Request consente agli sviluppatori di notificare agli altri membri del team che una branch di sviluppo è pronta per essere integrata nel branch principale o in un altro branch di destinazione. 77

Push [14] Un "push" in Git è un'operazione mediante la quale si caricano i cambiamenti locali effettuati sul proprio computer sul repository remoto. Questi cambiamenti possono includere aggiunte, modifiche o eliminazioni di file. Il

"push" consente agli altri membri del team di vedere e accedere ai cambiamenti che hai effettuato sul codice. 73, 76

RAG [5] tecnica avanzata di Natural Language Processing (NLP) che combina due componenti fondamentali: il recupero di informazioni (retrieval) e la generazione di testo (generation). Questa metodologia è utilizzata per migliorare la qualità e la precisione delle risposte generate da modelli di linguaggio, come i modelli di deep learning. 2, 10, 24, 25

Stakeholders [39] Gli stakeholder nello sviluppo software sono tutte le parti interessate o coinvolte nel progetto di sviluppo software. Queste possono essere persone, gruppi o entità che hanno un interesse diretto o indiretto nel prodotto software che viene sviluppato. 8, 12

Ticket [47] Un "ticket" in un sistema di tracciamento degli issue è una singola richiesta di lavoro o problema che deve essere affrontato o risolto. Questa richiesta è solitamente documentata in dettaglio e contiene informazioni pertinenti come la descrizione del problema o del lavoro da svolgere, le scadenze, lo stato corrente del ticket e altre informazioni rilevanti. 11

Token JWT [21] Un token JWT (JSON Web Token) è uno standard aperto per creare token di accesso che affermano delle informazioni tra due parti, tipicamente utilizzato per autenticazione e autorizzazione. È costituito da tre parti: intestazione (header), payload (che contiene le informazioni o affermazioni, come l'identità dell'utente e i permessi), e una firma crittografica che garantisce l'integrità del token. JWT è compatto, sicuro e facilmente trasmissibile tramite URL, header HTTP o nei corpi delle richieste POST. 20, 43–47, 56, 57, 82

Sitografia

[1] *AngularJS Modules*. URL: https://www.w3schools.com/angular/angular_modules.asp (cit. a p. 86).

[2] *Azienda SyncLab*. URL: https://www.synclab.it/home (cit. a p. 1).

[3] *Come effettuare una pull request*. URL: https://www.atlassian.com/it/git/tutorials/making-a-pull-request (cit. a p. 86).

[4] *Cos'è la Continuous Integration (CI)*. URL: https://www.geekandjob.com/wiki/continuous-integration-ci (cit. a p. 84).

[5] *Cos'è la RAG (Retrieval-Augmented Generation)?* URL: https://aws.amazon.com/it/what-is/retrieval-augmented-generation/ (cit. a p. 87).

[6] *Cos'è un'API REST?* URL: https://www.redhat.com/it/topics/api/what-is-a-rest-api (cit. a p. 84).

[7] *Cosa sono e a che cosa servono le milestones di progetto?* URL: https://www.t-ppm.it/milestones-di-progetto-cosa-sono (cit. a p. 85).

[8] *CoverAll Docs*. URL: https://docs.coveralls.io/ (cit. a p. 74).

[9] *Discord*. URL: https://discord.com/ (cit. a p. 10).

[10] *Framework SCRUM*. URL: https://www.atlassian.com/it/agile/scrum#:~:text=Il%20framework%20Scrum%20delinea%20una,guidano%20il%20team%20nel%20lavoro. (cit. a p. 9).

[11] *Git - git-branch Documentation*. URL: https://git-scm.com/docs/git-branch (cit. a p. 84).

[12] *GitHub docs*. URL: https://docs.github.com/en (cit. a p. 17).

[13] *Guides | Docker Docs*. URL: https://docs.docker.com/guides/ (cit. a p. 21).

[14] *Il comando Git push in Git*. URL: https://aulab.it/guide/94/il-comando-git-push-in-git (cit. a p. 86).

[15] *Introduction | Bootstrap*. URL: https://getbootstrap.com/docs/4.1/getting-started/introduction/ (cit. a p. 22).

[16] *Introduction | RxJS*. URL: https://rxjs.dev/guide/overview (cit. a p. 22).

[17] *Issue tracking system*. URL: https://en.wikipedia.org/wiki/Issue_tracking_system (cit. a p. 85).

[18] *JaCoCo Java Code Coverage Library*. URL: https://www.eclemma.org/jacoco/ (cit. a p. 74).

[19] *Java Documentation*. URL: https://docs.oracle.com/en/java/ (cit. a p. 19).

[20] *JUnit 5 User Guide*. URL: https://junit.org/junit5/docs/current/user-guide/ (cit. a p. 75).

[21] *JWT: cosa è il JSON Web Token?* URL: https://psicografici.com/jwt-json-web-token/ (cit. a p. 87).

[22] *Let's Develop an E-Commerce Application From Scratch Using Java*. URL: https://dev.to/webtutsplus/let-s-develop-an-e-commerce-application-from-scratch-using-java-1gap (cit. a p. 26).

[23] *MailDev*. URL: https://github.com/maildev/maildev (cit. a p. 18).

[24] *Maven Documentation*. URL: https://maven.apache.org/guides/ (cit. a p. 19).

[25] *Metodologia Agile*. URL: https://www.atlassian.com/it/agile (cit. a p. 8).

[26] *Mock object*. URL: https://en.wikipedia.org/wiki/Mock_object (cit. a p. 85).

[27] *ng-openapi-gen: An OpenAPI 3 code generator for Angular*. URL: https://www.npmjs.com/package/ng-openapi-gen (cit. a p. 22).

[28] *Official Angular documentation*. URL: https://v17.angular.io/docs (cit. a p. 22).

[29] *PostgreSQL JDBC Driver*. URL: https://jdbc.postgresql.org/ (cit. a p. 21).

[30] *Postman documentation overview*. URL: https://learning.postman.com/docs/introduction/overview/ (cit. a p. 18).

[31] *PrimeNG - Angular UI Component Library*. URL: https://primeng.org/ (cit. a p. 22).

[32] *Project Lombok*. URL: https://projectlombok.org/ (cit. a p. 21).

[33] *Quali sono i vantaggi nell'utilizzo degli Observable?* URL: https://www.freewebsolution.it/cosa-sono-gli-observable-e-come-utilizzarli-in-angular/#:~:text=In%20programmazione%20un%E2%80%99Observable%20%C3%A8%20paragonabile%20ad%20una%20funzione,gli%20elementi%20arrivano%20in%20maniera%20asincrona%20nel%20tempo. (cit. a p. 86).

[34] *SonarCloud documentation*. URL: https://docs.sonarsource.com/sonarcloud/ (cit. a p. 74).

[35] *Spring Boot Overview*. URL: https://spring.io/projects/spring-boot (cit. a p. 20).

[36] *Spring data JPA Overview*. URL: https://spring.io/projects/spring-data-jpa (cit. a p. 20).

[37] *Spring Framework Documentation*. URL: https://docs.spring.io/spring-framework/reference/index.html (cit. a p. 19).

[38] *Spring security*. URL: https://spring.io/projects/spring-security (cit. a p. 20).

[39] *Stakeholder*. URL: https://it.wikipedia.org/wiki/Stakeholder (cit. a p. 87).

[40] *Stoplight Documentation*. URL: https://docs.stoplight.io/ (cit. a p. 18).

[41] *Swagger Documentation*. URL: https://swagger.io/docs/ (cit. a p. 21).

[42] *Syncfusion Docs*. URL: https://blazor.syncfusion.com/documentation/introduction (cit. a p. 22).

[43] *Tasty mocking framework for unit tests in Java*. URL: https://site.mockito.org/ (cit. a p. 75).

[44] *The DTO Pattern (Data Transfer Object)*. URL: https://www.baeldung.com/java-dto-pattern (cit. a p. 84).

[45] *Tracking your work with issues*. URL: https://docs.github.com/en/issues/tracking-your-work-with-issues (cit. a p. 11).

[46] *Trello*. URL: https://trello.com/it/tour (cit. a p. 10).

[47] *Tutto quello che c'è da sapere sui sistemi di ticketing*. URL: https://www.deepser.com/it/vantaggi-software-gestione-ticket/ (cit. a p. 87).

[48] *TypeScript Documentation*. URL: https://www.typescriptlang.org/docs/ (cit. a p. 19).

[49] *Understanding GitHub Actions*. URL: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions (cit. a p. 73).

[50] *User stories with examples and a template*. URL: https://www.atlassian.com/agile/project-management/user-stories#:~:text=Summary%3A%20A%20user%20story%20is,But%20they're%20not. (cit. a p. 12).

[51] *Visual studio code docs*. URL: https://code.visualstudio.com/docs (cit. a p. 17).