Università degli Studi di Padova
Laurea: Informatica
Corso: Ingegneria del Software
Anno Accademico: 2025/2026

Gruppo: NullPointers Group
Email: nullpointersg@gmail.com

Specifica Tecnica

Stato	In Approvazione
Versione	0.5.0
Data ultima modifica	25/03/2026
Redattori	Matteo Mazzaretto Lisa Casagrande Tommaso Ceron Marco Brunello Luca Marcuzzo
Verificatori	Matteo Mazzaretto Lisa Casagrande Marco Brunello L. Pieripolli
Destinatari	NullPointers Group Ergon Informatica Srl Prof. Tullio Vardanega Prof. Riccardo Cardin

Registro delle modifiche

Vers	Data	Autore	Verificatore	Descrizione
0.5.0	25-03	M. Mazzaretto	L. Casagrande	Scrittura sezione 8.1.2
0.4.0	25-03	L. Casagrande	M. Mazzaretto	Inizio sezione 4, scrittura sezione 8.1.1
0.3.0	21-03	M. Brunello	M. Mazzaretto	Scrittura sezione 6
0.2.0	19-03	T. Ceron	M. Brunello	Scrittura sezione 3 e scrittura sezione 5
0.1.0	11-03	L. Marcuzzo	L. Pieripolli	Creazione e stesura documento. Redazione sezione 1 e inizio sezione 2

Table 1: Registro delle modifiche

Indice

1 Introduzione **9**
1.1 Scopo del documento 9
1.2 Scopo del progetto 9
1.3 Glossario 9
1.4 Riferimenti 10
1.4.1 Riferimenti normativi 10
1.4.2 Riferimenti informativi 10

2 Tecnologie **12**
2.1 Linguaggi di programmazione 12
2.1.1 Python 12
2.1.2 TypeScript 12
2.1.3 PostgreSQL 12
2.1.4 Infrastruttura e servizi 13
2.1.5 FastAPI 13
2.1.6 React 13
2.1.7 TailwindCSS 13
2.1.8 Vite 14
2.1.9 LangChain 14
2.1.10 FAISS 14
2.1.11 GPT-5 14
2.1.12 Docker 15
2.2 Tecnologie di analisi 15
2.2.1 Tecnologie di analisi statica 15
2.2.1.1 SonarQube 15
2.2.1.2 MyPy 15
2.2.2 Tecnologie di analisi dinamica 16
2.2.2.1 Pytest 16
2.2.2.2 Vitest 16

3 Architettura Logica **17**
3.1 Pattern di architettura esagonale 17

4 Servizi principali **17**

5 Architettura di Deployment **20**

6 Database Relazionale **21**

7 Design Pattern **23**
7.1 MVVM: ModelViewViewModel 23
7.1.1 Descrizione del pattern 23
7.1.2 Motivazioni dell'utilizzo del pattern 23
7.1.3 Utilizzo del pattern nel progetto 23
7.2 Dependency Injection 23

7.2.1 Descrizione del pattern . . . 23
7.2.2 Motivazioni dell'utilizzo del pattern . . . 23
7.2.3 Utilizzo del pattern nel progetto . . . 23
7.3 Adapter . . . 23
7.3.1 Descrizione del pattern . . . 23
7.3.2 Motivazioni dell'utilizzo del pattern . . . 23
7.3.3 Utilizzo del pattern nel progetto . . . 23

8 Diagrammi delle classi . . . 24
8.1 Autenticazione e Registrazione . . . 24
8.1.1 Frontend . . . 24
8.1.1.1 FormView . . . 25
8.1.1.1.1 Descrizione . . . 25
8.1.1.1.2 Attributi . . . 25
8.1.1.1.3 Metodi e funzioni . . . 25
8.1.1.2 FormErrors . . . 25
8.1.1.2.1 Descrizione . . . 26
8.1.1.2.2 Attributi . . . 26
8.1.1.2.3 Metodi e funzioni . . . 26
8.1.1.3 FormErrorProps . . . 26
8.1.1.3.1 Descrizione . . . 26
8.1.1.3.2 Attributi . . . 26
8.1.1.3.3 Metodi e funzioni . . . 26
8.1.1.4 FormProps . . . 27
8.1.1.4.1 Descrizione . . . 27
8.1.1.4.2 Attributi . . . 27
8.1.1.4.3 Metodi e funzioni . . . 27
8.1.1.5 FormViewModel . . . 28
8.1.1.5.1 Descrizione . . . 28
8.1.1.5.2 Attributi . . . 28
8.1.1.5.3 Metodi e funzioni . . . 28
8.1.1.6 FormModel . . . 29
8.1.1.6.1 Descrizione . . . 29
8.1.1.6.2 Attributi . . . 29
8.1.1.6.3 Metodi e funzioni . . . 30
8.1.1.7 useFormViewModel . . . 30
8.1.1.7.1 Descrizione . . . 30
8.1.1.7.2 Attributi . . . 30
8.1.1.7.3 Metodi e funzioni . . . 30
8.1.1.8 AuthAPI . . . 31
8.1.1.8.1 Descrizione . . . 31
8.1.1.8.2 Attributi . . . 31
8.1.1.8.3 Metodi e funzioni . . . 31
8.1.1.9 LoginDTO . . . 32
8.1.1.9.1 Descrizione . . . 32
8.1.1.9.2 Attributi . . . 32

8.1.1.9.3 Metodi e funzioni 32
8.1.1.10 RegisterDTO 33
8.1.1.10.1 Descrizione 33
8.1.1.10.2 Attributi 33
8.1.1.10.3 Metodi e funzioni 33
8.1.1.11 AuthResponse 34
8.1.1.11.1 Descrizione 34
8.1.1.11.2 Attributi 34
8.1.1.11.3 Metodi e funzioni 34
8.1.1.12 FieldConfig 35
8.1.1.12.1 Descrizione 35
8.1.1.12.2 Attributi 35
8.1.1.12.3 Metodi e funzioni 35
8.1.1.13 SubmitResult 36
8.1.1.13.1 Descrizione 36
8.1.1.13.2 Attributi 36
8.1.1.13.3 Metodi e funzioni 36
8.1.1.14 RegisterModel 36
8.1.1.14.1 Descrizione 37
8.1.1.14.2 Attributi 37
8.1.1.14.3 Metodi e funzioni 37
8.1.1.15 LoginModel 37
8.1.1.15.1 Descrizione 37
8.1.1.15.2 Attributi 37
8.1.1.15.3 Metodi e funzioni 37
8.1.2 Backend 38
8.1.2.1 UserService 38
8.1.2.1.1 Descrizione 38
8.1.2.1.2 Attributi 39
8.1.2.1.3 Metodi e funzioni 39
8.1.2.2 AuthRouter 39
8.1.2.2.1 Descrizione 40
8.1.2.2.2 Attributi 40
8.1.2.2.3 Metodi e funzioni 40
8.1.2.3 TokenUtility 41
8.1.2.3.1 Descrizione 41
8.1.2.3.2 Attributi 41
8.1.2.3.3 Metodi e funzioni 41
8.1.2.4 IEmailValidationPort 42
8.1.2.4.1 Descrizione 42
8.1.2.4.2 Attributi 42
8.1.2.4.3 Metodi e funzioni 42
8.1.2.5 EmailValidationAdapter 42
8.1.2.5.1 Descrizione 42
8.1.2.5.2 Attributi 42
8.1.2.5.3 Metodi e funzioni 43

8.1.2.6 IUserRepoPort 43
8.1.2.6.1 Descrizione 43
8.1.2.6.2 Attributi 43
8.1.2.6.3 Metodi e funzioni 43
8.1.2.7 UserRepoAdapter 44
8.1.2.7.1 Descrizione 44
8.1.2.7.2 Attributi 44
8.1.2.7.3 Metodi e funzioni 44
8.1.2.8 UserRepository 45
8.1.2.8.1 Descrizione 45
8.1.2.8.2 Attributi 45
8.1.2.8.3 Metodi e funzioni 45
8.1.2.9 PasswordUtility 46
8.1.2.9.1 Descrizione 46
8.1.2.9.2 Attributi 46
8.1.2.9.3 Metodi e funzioni 46

9 Stato dei requisiti funzionali 47
9.1 Riepilogo dei requisiti 47
9.2 Tabella dei requisiti funzionali 47

Elenco delle Tabelle

1 Registro delle modifiche 2
2 Stato dei Requisiti Funzionali 64

Elenco delle Immagini

1 Registrazione e autenticazione 18
2 Schema ER della base di dati 21
3 Diagramma delle classi per Autenticazione e Registrazione: Frontend 24
4 Autenticazione e Registrazione: FormView 25
5 Autenticazione e Registrazione: FormErrors 25
6 Autenticazione e Registrazione: FormErrorProps 26
7 Autenticazione e Registrazione: FormProps 27
8 Autenticazione e Registrazione: FormViewModel 28
9 Autenticazione e Registrazione: FormModel 29
10 Autenticazione e Registrazione: useFormViewModel 30
11 Autenticazione e Registrazione: AuthAPI 31
12 Autenticazione e Registrazione: LoginDTO 32
13 Autenticazione e Registrazione: RegisterDTO 33
14 Autenticazione e Registrazione: AuthResponse 34
15 Autenticazione e Registrazione: FieldConfig 35
16 Autenticazione e Registrazione: SubmitResult 36
17 Autenticazione e Registrazione: RegisterModel 36
18 Autenticazione e Registrazione: LoginModel 37
19 Diagramma delle classi per Autenticazione e Registrazione: Backend 38
20 Autenticazione e Registrazione: UserService 38
21 Autenticazione e Registrazione: AuthRouter 39
22 Autenticazione e Registrazione: TokenUtility 41
23 Autenticazione e Registrazione: IEmailValidationPort 42
24 Autenticazione e Registrazione: EmailValidationAdapter 42
25 Autenticazione e Registrazione: IUserRepoPort 43
26 Autenticazione e Registrazione: UserRepoAdapter 44
27 Autenticazione e Registrazione: UserRepository 45
28 Autenticazione e Registrazione: PasswordUtility 46

1 Introduzione

1.1 Scopo del documento

Il presente documento ha l'obiettivo di definire e descrivere l'architettura adottata per lo sviluppo^G del sistema SmartOrder.
Il documento delinea le scelte architetturali, tecnologiche e di design adottate dal gruppo per la realizzazione del prodotto.
In particolare il documento si propone di definire:

- L'architettura del sistema, comprensiva delle componenti principali e delle loro interazioni;
- Le tecnologie e i framework^G selezionati per lo sviluppo^G, con particolare attenzione alle motivazioni tecniche che ne hanno guidato l'adozione;
- I design pattern applicati e le ragioni tecnologiche che ne hanno guidato l'adozione;
- Il tracciamento dei requisiti, che attesta la copertura di quanto stabilito nel documento di Analisi dei Requisiti.

Il documento è soggetto ad aggiornamenti durante tutto il ciclo di vita del progetto^G, al fine di recepire nuove decisioni architetturali o eventuali modifiche legate all'evoluzione del progetto.

1.2 Scopo del progetto

SmartOrder è un prodotto ideato dall'azienda Ergon Informatica per automatizzare il processo^G di gestione degli ordini di acquisto aziendali tramite l'analisi di input multimodali quali testo, immagini o audio. Attraverso l'utilizzo del Natural Language Processing^G (NLP^G), SmartOrder interpreta queste richieste spesso non strutturate, estraendo le informazioni rilevanti quali prodotti e quantità, trasformandole in ordini strutturati pronti per l'integrazione nei sistemi ERP^G aziendali.

1.3 Glossario

Per assicurare una comprensione univoca dei termini usati in questo Piano di Qualifica^G, si fornisce un glossario dedicato.
La nomenclatura utilizzata per segnalare che la definizione di una parola è contenuta nel glossario è la seguente:

termine^G

I termini sono ordinati alfabeticamente per facilitarne la consultazione e vengono inclusi sia termini tecnici che acronimi significativi.
Il gruppo si impegna a visionare il Glossario periodicamente, per permettere la più completa comprensione di ogni tipo di documento pubblicato e per mantenere un allineamento semantico costante tra tutti i partecipanti al progetto.

1.4 Riferimenti

1.4.1 Riferimenti normativi

- **Norme di Progetto^G, versione 1.1.0**
  https://nullpointersgroup.github.io/Documentazione/output/PB/Documenti%20Interni/Norme_di_Progetto.pdf
  Ultima consultazione: 12 gennaio 2026

- **Capitolato^G C8 - Ergon Informatica Srl - SmartOrder**
  https://www.math.unipd.it/~tullio/IS-1/2025/Progetto/C8.pdf
  Ultima consultazione: 3 dicembre 2025

1.4.2 Riferimenti informativi

- **Glossario, versione 1.0.0**
  https://nullpointersgroup.github.io/Documentazione/output/PB/Documenti%20Interni/Glossario.pdf
  Ultima consultazione: 11 marzo 2026

- **Analisi dei requisiti^G, versione 1.1.2**
  https://nullpointersgroup.github.io/Documentazione/output/PB/Documenti%20Interni/Analisi_dei_Requisiti.pdf
  Ultima consultazione: 11 marzo 2026

- **Analisi dei requisiti^G, SWE 2025-2026**
  https://www.math.unipd.it/~tullio/IS-1/2025/Dispense/T05.pdf
  Ultima consultazione: 11 marzo 2026

- **Software Architecture Patterns, SWE 2025-2026**
  https://www.math.unipd.it/~rcardin/swea/2022/Software%20Architecture%20Patterns.pdf
  Ultima consultazione: 11 marzo 2026

- **Dependency Injection, SWE 2025-2026**
  https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Architetturali%20-%20Dependency%20Injection.pdf
  Ultima consultazione: 11 marzo 2026

- **Design Pattern Creazionali, SWE 2025-2026**
  https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Creazionali.pdf
  Ultima consultazione: 11 marzo 2026

- **Design Pattern Strutturali, SWE 2025-2026**
  https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%

20Strutturali.pdf

Ultima consultazione: 11 marzo 2026

2 Tecnologie

In questa sezione vengono descritte le tecnologie adottate per lo sviluppo[G] del progetto[G] **SmartOrder**. Per ciascuna tecnologia vengono riportate la versione di riferimento, la documentazione ufficiale e il ruolo ricoperto all'interno del sistema.

2.1 Linguaggi di programmazione

2.1.1 Python

- **Versione:** 3.14.3
- **Documentazione:** https://docs.python.org/3/

Python[G] è un linguaggio di programmazione ad alto livello, interpretato e flessibile, progettato per rendere il codice facile da leggere e veloce da sviluppare. Nel progetto[G] SmartOrder, Python[G] viene usato principalmente per sviluppare il backend[G] e per collegarsi ai modelli di intelligenza artificiale. In particolare, gestisce la logica degli ordini, realizza le API[G] con FastAPI e si occupa delle interazioni con il database[G] e con i sistemi di memorizzazione vettoriale. Grazie alla grande quantità di librerie disponibili, Python[G] è particolarmente adatto per applicazioni che usano machine learning[G] e intelligenza artificiale.

2.1.2 TypeScript

- **Versione:** 5.9
- **Documentazione:** https://www.typescriptlang.org/docs/

TypeScript è un linguaggio di programmazione che estende JavaScript introducendo la tipizzazione statica, interfacce, tipi complessi e generics. I controlli sui tipi a compile-time riducono gli errori comuni legati all'assenza di tipizzazione e migliorano la leggibilità, la manutenibilità e la robustezza del codice. Nel progetto[G] viene utilizzato per gestire dati strutturati complessi come ordini, prodotti, quantità e validazioni, contribuendo a mantenere coerenza tra frontend[G] e backend[G] e a ridurre il rischio di errori nella comunicazione tramite API.

2.1.3 PostgreSQL

- **Versione:**
- **Documentazione:** https://www.postgresql.org/docs/

PostgreSQL[G] è un sistema di gestione di basi di dati relazionali conforme agli standard SQL[G], noto per la sua affidabilità e robustezza nella gestione dei dati.

Nel progetto[G] **SmartOrder** viene utilizzato come database[G] principale per la memorizzazione persistente delle informazioni dell'applicazione, tra cui il catalogo dei prodotti, gli ordini generati dal sistema e i dati associati agli input elaborati. L'utilizzo di PostgreSQL[G] consente di garantire integrità, consistenza e affidabilità nella gestione dei dati e si integra facilmente con il backend[G] sviluppato in Python.

2.1.4 Infrastruttura e servizi

2.1.5 FastAPI

- **Versione:**
- **Documentazione:** https://fastapi.tiangolo.com

FastAPI è un framework[G] web per Python[G] utilizzato per sviluppare API[G] REST[G] ad alte prestazioni. Si basa sullo standard ASGI e utilizza librerie come Pydantic e Starlette per la gestione delle richieste e la validazione[G] dei dati.
Nel progetto[G] **SmartOrder** viene utilizzato per implementare il backend[G] dell'applicazione e gestire la comunicazione tra frontend[G], moduli di elaborazione e database. Inoltre, consente di integrare facilmente i modelli di intelligenza artificiale[G] e di generare automaticamente la documentazione delle API[G] tramite strumenti basati su OpenAPI.

2.1.6 React

- **Versione:** 19.2
- **Documentazione:** https://it.react.dev/reference/react

React[G] è una libreria JavaScript open source per la costruzione di interfacce utente basate su un modello dichiarativo e su componenti indipendenti e riutilizzabili. Utilizza un DOM virtuale per aggiornare selettivamente solo le parti dell'interfaccia effettivamente modificate, ottimizzando le prestazioni di rendering.
Non essendo opinionata nella scelta delle librerie di routing o di gestione dello stato, non impone vincoli architetturali, permettendo di strutturare il frontend[G] in modo modulare e adattarlo alle esigenze dell'applicazione.

2.1.7 TailwindCSS

- **Versione:** 4.2
- **Documentazione:** https://tailwindcss.com/docs/installation/using-vite

TailwindCSS è un framework[G] CSS utility-first che mette a disposizione un ampio insieme di classi di utilità che applicano singole proprietà CSS direttamente nel markup. Questo approccio consente di costruire interfacce in modo rapido e coerente, evitando la scrittura di fogli di stile personalizzati e complessi, e favorendo una maggiore manutenibilità del codice di presentazione.

2.1.8 Vite

- **Versione:** 7.3.1
- **Documentazione:** https://vite.dev/guide/#scaffolding-your-first-vite-project

Vite è uno strumento di build^G per lo sviluppo^G di applicazioni web moderne. Sfrutta i moduli ES nativi del browser per avviare l'ambiente di sviluppo^G con tempi di avvio ridotti, offrendo hot module replacement per aggiornamenti immediati durante lo sviluppo. Supporta la configurazione tramite plugin, l'ottimizzazione automatica delle dipendenze e la generazione di bundle di codice ottimizzati tramite Rollup.

2.1.9 LangChain

- **Versione:**
- **Documentazione:** https://python.langchain.com

LangChain è un framework^G open source progettato per facilitare lo sviluppo^G di applicazioni basate su Large Language Models (LLM^G). Fornisce strumenti per gestire prompt, orchestrare le chiamate ai modelli e integrare i modelli di linguaggio con fonti di dati esterne.
Nel progetto^G **SmartOrder** viene utilizzato per gestire l'interazione con il modello di linguaggio e costruire le pipeline di elaborazione necessarie per interpretare gli input degli utenti ed estrarre le informazioni rilevanti dagli ordini, trasformandole in dati strutturati utilizzabili dal sistema.

2.1.10 FAISS

- **Versione:**
- **Documentazione:** https://faiss.ai/index.html

FAISS (Facebook AI Similarity Search) è una libreria open source sviluppata da Meta per l'indicizzazione e la ricerca efficiente di vettori ad alta dimensionalità, utilizzata principalmente per operazioni di similarity search tra embedding.
Nel progetto^G **SmartOrder** viene utilizzato come database^G vettoriale per memorizzare e confrontare gli embedding generati dai modelli di linguaggio. Questo consente di individuare prodotti simili a partire da descrizioni testuali e migliorare l'accuratezza del matching tra le richieste dell'utente e il catalogo aziendale.

2.1.11 GPT-5

- **Versione:**
- **Documentazione:** https://platform.openai.com/docs

GPT-5 è un Large Language Model basato sull'architettura Transformer, progettato per comprendere e generare testo in linguaggio naturale. Grazie all'addestramento su grandi quantità di dati testuali, è in grado di interpretare il significato delle richieste degli utenti e di estrarre informazioni rilevanti dal testo.
Nel progetto^G **SmartOrder** viene utilizzato per analizzare gli input degli utenti espressi in linguaggio naturale e trasformarli in dati strutturati utilizzabili dal sistema, individuando informazioni come prodotti, quantità e altri dettagli necessari alla generazione degli ordini.

2.1.12 Docker

- **Versione:**
- **Documentazione:** https://docs.docker.com

Docker^G è una piattaforma open source che consente di sviluppare ed eseguire applicazioni all'interno di container, ambienti isolati che includono tutte le dipendenze necessarie al funzionamento del software.
Nel progetto^G **SmartOrder** viene utilizzato per containerizzare i diversi componenti dell'applicazione, come backend^G, frontend^G e database^G, garantendo un ambiente di esecuzione coerente tra sviluppo^G, testing e deployment e semplificando il processo^G di distribuzione^G del sistema.

2.2 Tecnologie di analisi

2.2.1 Tecnologie di analisi statica

2.2.1.1 SonarQube

- **Versione:**
- **Documentazione:** https://docs.sonarsource.com/sonarqube

SonarQube è una piattaforma di analisi statica del codice che consente di valutare la qualità del software individuando potenziali errori, vulnerabilità di sicurezza e problemi di manutenibilità.
Nel progetto^G **SmartOrder** viene utilizzato per monitorare automaticamente la qualità del codice prodotto durante lo sviluppo^G, permettendo di individuare precocemente eventuali criticità e mantenere elevati standard qualitativi nel progetto.

2.2.1.2 MyPy

- **Versione:**
- **Documentazione:** https://mypy.readthedocs.io

MyPy è uno strumento di analisi statica per Python^G che verifica^G la correttezza delle annotazioni di tipo presenti nel codice sorgente.
Nel progetto^G **SmartOrder** viene utilizzato per controllare la coerenza dei tipi nel backend^G sviluppato in Python^G, contribuendo a individuare errori prima dell'esecuzione del programma e migliorando la robustezza del codice.

2.2.2 Tecnologie di analisi dinamica

2.2.2.1 Pytest

- **Versione:**
- **Documentazione:** https://docs.pytest.org

Pytest è un framework^G di testing per Python^G che consente di scrivere ed eseguire test^G automatici per verificare il corretto funzionamento del codice.
Nel progetto^G **SmartOrder** viene utilizzato per testare le componenti backend^G sviluppate in Python^G, verificando il comportamento delle API^G e delle funzioni di elaborazione per garantire l'affidabilità del sistema.

2.2.2.2 Vitest

- **Versione:**
- **Documentazione:** https://vitest.dev

Vitest è un framework^G di testing per applicazioni JavaScript e TypeScript integrato con l'ecosistema Vite, progettato per eseguire test^G in modo rapido ed efficiente.
Nel progetto^G **SmartOrder** viene utilizzato per testare il codice frontend^G sviluppato con React^G e TypeScript, verificando il corretto funzionamento delle componenti dell'interfaccia utente e delle logiche di interazione con il backend.

3 Architettura Logica

L'architettura logica del progetto^G SmartOrder adotta un approccio esagonale. L'obiettivo è quello di mantenere separata la logica di business dalle componenti esterne, come l'interfaccia utente e il sistema di persistenza dei dati.
La comunicazione tra la parte di presentazione e la logica applicativa avviene tramite interfacce, che permettono ai controller (ad esempio quelli esposti tramite API^G HTTP) di invocare le funzionalità^G del sistema senza dipendere direttamente dalla loro implementazione.
Allo stesso modo, l'accesso al database^G e ad altri servizi esterni è gestito tramite componenti dedicati, che permettono alla logica di business di rimanere indipendente dalle tecnologie utilizzate.

3.1 Pattern di architettura esagonale

Il pattern esagonale è stato scelto per organizzare il sistema in modo chiaro e modulare, separando le responsabilità tra le diverse componenti.
Questo approccio rende il sistema più facile da mantenere, testare ed estendere nel tempo.
I principali elementi dell'architettura esagonale sono:

- **Core domain**: contiene la logica di business dell'applicazione ed è indipendente dalle tecnologie esterne.
- **Porte**: rappresentano i punti di contatto tra il dominio e l'esterno, e definiscono le operazioni disponibili. Si distinguono in:
  - **Inbound ports**: interfacce che espongono le funzionalità^G del dominio verso l'esterno.
  - **Outbound ports**: interfacce che definiscono come il dominio può utilizzare servizi esterni.
- **Adapters**: sono implementazioni concrete delle porte che collegano il dominio alle tecnologie utilizzate.
  - **Inbound adapters**: adattatori che convertono le richieste esterne nel formato atteso del dominio.
  - **Outbound adapters**: adattori che implementano le interfacce di uscita collegandole a tecnologie specifiche.

4 Servizi principali

- Autenticazione e registrazione

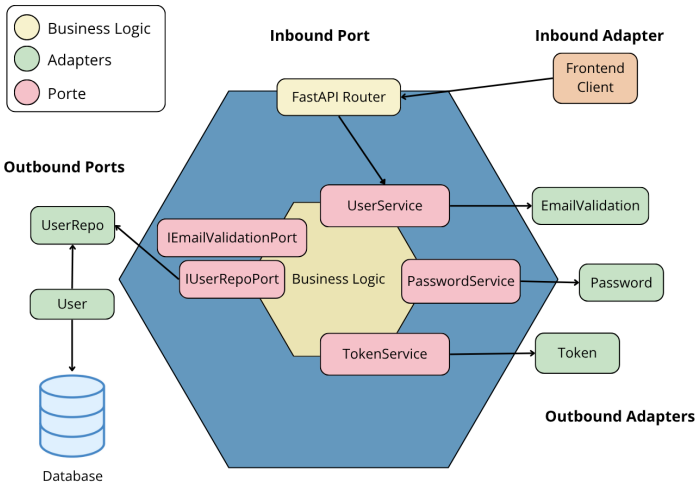

Figure 1: Registrazione e autenticazione

  - **Core domain**: contiene la logica dedicata alla registrazione di un utente nuovo oppure all'autenticazione di un utente già esistente. Permette di elaborare i dati ricevuti tramite l'inserimento dei dati dell'utente.
  - **Inbound ports**: l'applicazione espone un unico punto di ingresso tramite FastAPI Router, che funge da inbound port. Non sono stati definiti ulteriori livelli di astrazione (es. interfacce dedicate), poiché l'HTTP driver è già sufficiente.
  - **Outbound ports**: definiscono i contratti verso l'infrastruttura esterna. `IUserRepoPort` espone le operazioni di persistenza sugli utenti, mentre `IEmailValidationPort` espone la verifica[G] del dominio email tramite DNS. Entrambe le interfacce sono definite nel core domain e implementate da adapter dedicati.
  - **Adapters**: implementano l'integrazione con servizi esterni, tra cui la persistenza su database[G] (UserRepo), la validazione[G] email, la gestione delle password e la generazione dei token.

- Chatbot e gestione ordine
  - **Core domain**:
  - **Inbound ports**:
  - **Outbound ports**:
  - **Adapters**:
- Visualizzazione storico ordini

- **Core domain**:
- **Inbound ports**:
- **Outbound ports**:
- **Adapters**:

5 Architettura di Deployment

Il sistema adotta un'architettura **monolitica**, in cui tutti i componenti applicativi (logica di dominio, layer HTTP e accesso al database^G) sono sviluppati, rilasciati ed eseguiti come un'unica unità.
Rispetto a un'architettura a microservizi, questa scelta offre i seguenti vantaggi nel contesto del progetto^G:

- **Semplicità operativa**: non è necessario gestire orchestrazione di container, service discovery o bilanciamento del carico tra servizi distribuiti.
- **Latenza ridotta**: le comunicazioni tra i componenti avvengono in-process, eliminando l'overhead di chiamate HTTP o message broker tra servizi separati.
- **Transazioni coerenti**: le operazioni sul database^G avvengono all'interno di un'unica connessione, senza la necessità di gestire la consistenza distribuita tra servizi indipendenti.
- **Adeguatezza alla scala**: considerata la natura del progetto^G e il numero limitato di utenti previsti, i benefici introdotti dai microservizi non giustificherebbero la complessità aggiuntiva che comportano.
- **Comunicazione con la proponente^G**: a seguito di confronto con la proponente^G, è emersa la preferenza per un'architettura monolitica rispetto a una soluzione a microservizi. Tale decisione^G è documentata nel Verbale esterno del 11 marzo.

6 Database Relazionale

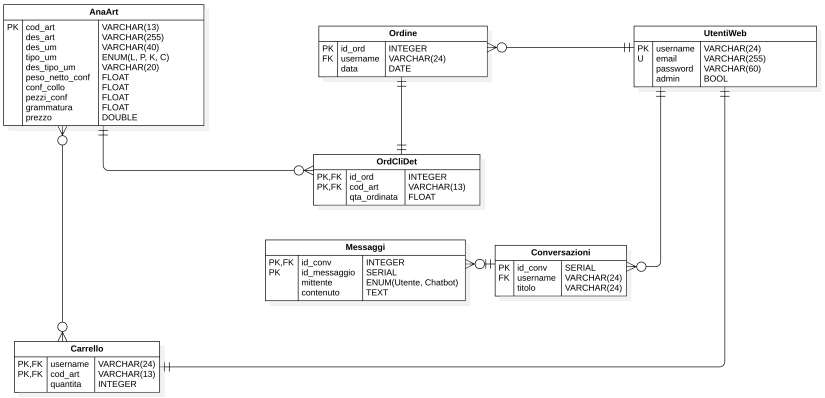

Figure 2: Schema ER della base di dati

Il database^G relazionale utilizzato per il progetto^G SmartOrder non è molto complesso.
In esso vengono salvati:

- **anaart**: l'anagrafica degli articoli. È composto da:
  - **cod_art**: il codice articolo, è la primary key della tabella
  - **des_art**: rappresenta la descrizione dell'articolo
  - **des_um**: rappresenta la modalità di vendita dell'articolo
  - **tipo_um**: rappresenta il tipo della confezione. È descritto da un enum in cui L sta per colli, P per pezzi, K per chilogrammo e C per confezioni
  - **peso_netto_conf**: rappresenta il peso netto della confezione
  - **conf_collo**: rappresenta quante confezioni sono presenti all'interno di un collo
  - **pezzi_conf**: rappresenta quanti pezzi sono presenti all'interno di una confezione
  - **grammatura**: rappresenta il peso
  - **prezzo**: rappresenta il pezzo in euro del prodotto
- **UtentiWeb**: la tabella che tiene salvati gli utenti di SmartOrder. Contiene:
  - **username**: rappresenta il nome dell'utente con cui si può effettuare il login. È la primary key della tabella

  - **email**: l'email associata all'utente. Deve essere univoca
  - **password**: la password dell'utente. È salvata utilizzando l'algoritmo bcrypt
  - **admin**: booleano che identifica se un utente è admin oppure no

- **Ordine**: rappresenta un ordine di un utente. È composto da:
  - **id_ord**: numero univoco per l'id dell'ordine. È la primary key della tabella
  - **username**: rappresenta lo username dell'utente associato all'ordine. È una chiave esterna
  - **data**: rappresenta la data in cui è stato effettuato l'ordine.

- **OrdCliDet**: rappresenta la tabella per collegare gli articoli ordinati dal cliente per un determinato ordine. È composto da:
  - **id_ord**: rappresenta l'id dell'ordine associato. Viene utilizzato come primary key assieme all'attributo **cod_art**. È una chiave esterna
  - **cod_art**: rappresenta il codice dell'articolo in oggetto. Viene utilizzato come primary key assieme all'attributo **id_ord**. è una chiave esterna

- **Conversazione**: rappresenta una chat di un utente. È composta da:
  - **id_conv**: rappresenta l'id della conversazione. È la primary key della tabella
  - **username**: rappresenta lo username associato a quella conversazione. È una chiave esterna
  - **titolo**: rappresenta il titolo della conversazione

- **Messaggi**: rappresenta i messaggi all'interno di una conversazione. È composta da:
  - **id_conv**: rappresenta l'id della conversazione a cui il messaggio è associato. È una primary key assieme all'attributo **id_messaggio** ed è una chiave esterna
  - **id_messaggio**: rappresenta l'id di un messaggio all'interno di una conversazione. È una primary key assieme all'attributo **id_conv**
  - **mittente**: rappresenta chi ha inviato il messaggio. È rappresentato da un enum e può essere solo utente oppure chatbot
  - **contenuto**: rappresenta il testo del messaggio

7 Design Pattern

7.1 MVVM: ModelViewViewModel

7.1.1 Descrizione del pattern

7.1.2 Motivazioni dell'utilizzo del pattern

7.1.3 Utilizzo del pattern nel progetto

7.2 Dependency Injection

7.2.1 Descrizione del pattern

7.2.2 Motivazioni dell'utilizzo del pattern

7.2.3 Utilizzo del pattern nel progetto

7.3 Adapter

7.3.1 Descrizione del pattern

7.3.2 Motivazioni dell'utilizzo del pattern

7.3.3 Utilizzo del pattern nel progetto

8 Diagrammi delle classi

8.1 Autenticazione e Registrazione

Questa parte si occupa di ricevere le richieste di ottenimento Token^G, controllarne i valori e restituire un Token^G valido e temporaneo (24 ore di validità), affinché il Client possa utilizzare il Sistema.
I Token^G sono firmati tramite l'algoritmo **HS256** (HMAC-SHA256), che utilizza una chiave segreta condivisa sia per la firma che per la verifica.
Le password degli utenti sono invece cifrate tramite **bcrypt**, algoritmo appositamente progettato per l'hashing.

8.1.1 Frontend

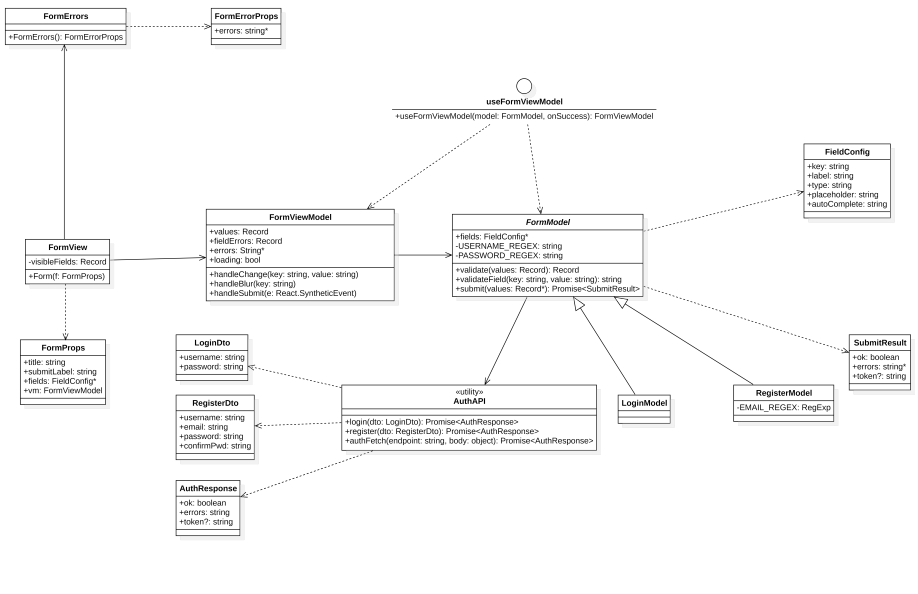

Figure 3: Diagramma delle classi per Autenticazione e Registrazione: Frontend

8.1.1.1 FormView

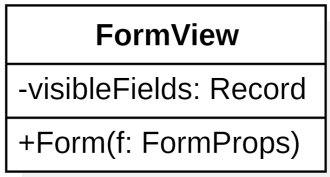

Figure 4: Autenticazione e Registrazione: FormView

8.1.1.1.1 Descrizione

`FormView` è il componente React^G che costituisce il livello di *View* nel pattern MVVM.
Si occupa esclusivamente della presentazione grafica del form di autenticazione o registrazione, delegando tutta la logica al `FormViewModel` ricevuto tramite `FormProps`.

8.1.1.1.2 Attributi

- `visibleFields: Record` - attributo privato di tipo `Record<string, boolean>`. Gestisce la visibilità dei campi password: per ciascun campo di tipo `password`, traccia se il testo è attualmente visibile o nascosto. Viene aggiornato dalla funzione `toggleVisibility` al click sull'icona dell'occhio.

8.1.1.1.3 Metodi e funzioni

- `Form(f: FormProps)` - funzione componente React^G (funzione pubblica). Riceve le props di tipo `FormProps` e restituisce il markup JSX del form. Itera sulla lista `fields` per renderizzare dinamicamente i campi di input, utilizza `vm.values` e `vm.fieldErrors` per il binding dei dati, e collega gli handler `vm.handleChange`, `vm.handleBlur` e `vm.handleSubmit` agli eventi del form. Per i campi di tipo password gestisce la visibilità tramite `visibleFields` e delega la visualizzazione degli errori globali al componente `FormErrors`.

8.1.1.2 FormErrors

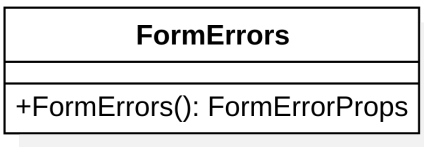

Figure 5: Autenticazione e Registrazione: FormErrors

8.1.1.2.1 Descrizione

`FormErrors` è un componente React^G di presentazione dedicato alla visualizzazione degli errori globali del form.
Riceve un array di messaggi di errore e li renderizza come lista.
Viene invocato da `FormView` e non contiene logica applicativa.

8.1.1.2.2 Attributi

La classe non presenta attributi propri: riceve i dati da visualizzare tramite le props di tipo `FormErrorProps`.

8.1.1.2.3 Metodi e funzioni

- `FormErrors(): FormErrorProps` - funzione pubblica che legge la prop `errors` di tipo `string*` e restituisce il markup JSX contenente la lista degli errori globali da mostrare all'utente. Se l'array è vuoto, il componente non renderizza nulla.

8.1.1.3 FormErrorProps

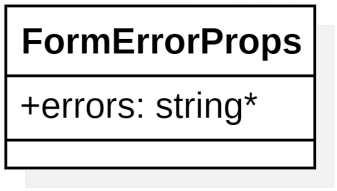

Figure 6: Autenticazione e Registrazione: FormErrorProps

8.1.1.3.1 Descrizione

`FormErrorProps` è un'interfaccia TypeScript che definisce le proprietà accettate dal componente `FormErrors`.
Rappresenta il contratto tra il componente padre `FormView` e il componente `FormErrors`, garantendo il corretto passaggio dei messaggi di errore globali.

8.1.1.3.2 Attributi

- `errors: string*` - Array pubblico di stringhe. Contiene i messaggi di errore globali da visualizzare all'utente, ad esempio errori provenienti dal backend. Può essere un array vuoto se non vi^G sono errori da mostrare.

8.1.1.3.3 Metodi e funzioni

`FormErrorProps` non definisce metodi o funzioni.

8.1.1.4 FormProps

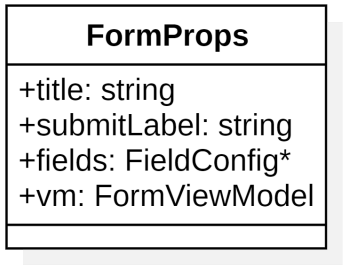

Figure 7: Autenticazione e Registrazione: FormProps

8.1.1.4.1 Descrizione

`FormProps` è un'interfaccia TypeScript che definisce le proprietà passate al componente `FormView`.
Aggrega tutte le informazioni necessarie alla renderizzazione del form: titolo, etichetta del pulsante di invio, configurazione dei campi e riferimento al ViewModel.
Disaccoppia la View dalla logica, seguendo il pattern MVVM.

8.1.1.4.2 Attributi

- `title: string` - attributo pubblico di tipo stringa. Contiene il titolo del form visualizzato nell'intestazione del componente.
- `submitLabel: string` - attributo pubblico di tipo stringa. Etichetta testuale del pulsante di invio del form.
- `fields: FieldConfig*` - attributo pubblico di tipo array di `FieldConfig`. Contiene la configurazione di ciascun campo del form (chiave, etichetta, tipo, placeholder, autocompletamento). Permette la renderizzazione dinamica dei campi senza modificare il componente View.
- `vm: FormViewModel` - attributo pubblico di tipo `FormViewModel`. Riferimento al ViewModel del form, contenente lo stato reattivo e gli handler degli eventi. Costituisce il punto di collegamento tra View e logica applicativa.

8.1.1.4.3 Metodi e funzioni

`FormProps` non definisce metodi o funzioni.

8.1.1.5 FormViewModel

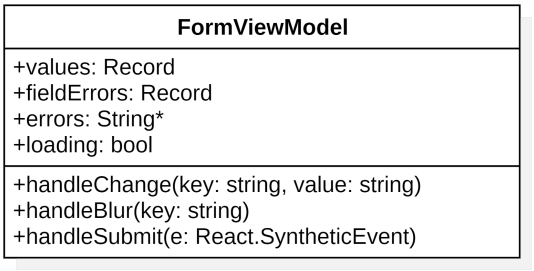

Figure 8: Autenticazione e Registrazione: FormViewModel

8.1.1.5.1 Descrizione

`FormViewModel` è un'interfaccia TypeScript che definisce il contratto del ViewModel nel pattern MVVM.
Espone lo stato del form e gli handler degli eventi che la View deve utilizzare.
La sua implementazione concreta è prodotta dall'hook `useFormViewModel`, che gestisce lo stato React^G e la comunicazione con il Model.

8.1.1.5.2 Attributi

- `values: Record` - attributo pubblico di tipo `Record<string, string>`. Mappa chiave-valore contenente i valori correnti di ciascun campo del form, viene aggiornata ad ogni modifica^G dell'utente.
- `fieldErrors: Record` - attributo pubblico di tipo `Record<string, string>`. Mappa chiave-valore contenente i messaggi di errore di validazione^G per ciascun campo, popolata durante la validazione^G on-blur e al momento del submit.
- `errors: string*` - attributo pubblico di tipo array di stringhe. Contiene i messaggi di errore globali restituiti dal backend^G o generati durante l'invio del form.
- `loading: bool` - attributo pubblico booleano. Indica se è in corso una richiesta asincrona verso il backend. Viene utilizzato dalla View per disabilitare il pulsante di submit e mostrare un indicatore di caricamento.

8.1.1.5.3 Metodi e funzioni

- `handleChange(key: string, value: string): void` - metodo pubblico che aggiorna il valore del campo identificato da `key` all'interno di `values`. Viene invocato dalla View ad ogni evento `onChange` degli input.
- `handleBlur(key: string): void` - metodo pubblico che innesca la validazione^G del singolo campo identificato da `key` nel momento in cui

l'utente abbandona il campo. Aggiorna `fieldErrors` con l'eventuale errore riscontrato.

- `handleSubmit(e: React.SyntheticEvent): void` - metodo pubblico che gestisce la sottomissione del form intercettando l'evento del browser. Innesca la validazione^G completa di tutti i campi e, se non vi^G sono errori, invoca il metodo `submit` del Model delegando la chiamata API^G al layer sottostante.

8.1.1.6 FormModel

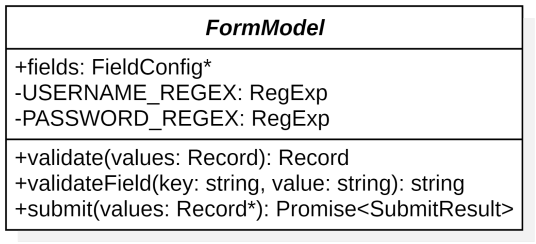

Figure 9: Autenticazione e Registrazione: FormModel

8.1.1.6.1 Descrizione

`FormModel` è una classe astratta TypeScript che costituisce il livello di *Model* nel pattern MVVM.
Incapsula la logica di validazione^G dei campi del form e la logica di invio dei dati al backend^G tramite il modulo `AuthAPI`.
Le classi concrete `LoginModel` e `RegisterModel` la estendono specializzandone il comportamento.
Solo `fields` e `submit` sono astratti: `validate` e `validateField` hanno invece un'implementazione concreta nella classe base.

8.1.1.6.2 Attributi

- `fields: FieldConfig*` - attributo pubblico astratto di tipo array di `FieldConfig`. Contiene la configurazione dei campi del form. Viene definito dalle sottoclassi concrete con i campi specifici per login o registrazione.

- `USERNAME_REGEX: RegExp` - costante privata protetta di tipo `RegExp`. Espressione regolare utilizzata per la validazione^G del formato del nome utente (tra 4 e 24 caratteri alfanumerici o underscore).

- `PASSWORD_REGEX: RegExp` - costante privata protetta di tipo `RegExp`. Espressione regolare utilizzata per la validazione^G della complessità della password (tra 8 e 24 caratteri, con almeno una maiuscola, una minuscola, un numero e un carattere speciale).

8.1.1.6.3 Metodi e funzioni

- `validate(values: Record): Record` - metodo pubblico concreto. Riceve la mappa dei valori correnti del form e restituisce una mappa di errori (`Record<string, string>`) per tutti i campi. Verifica^G che nessun campo sia vuoto; le sottoclassi possono sovrascriverlo per aggiungere regole di validazione^G specifiche.
- `validateField(key: string, value: string): string` - metodo pubblico concreto. Valida il singolo campo identificato da `key` con il valore `value` invocando `validate` sull'intero set di valori correnti. Restituisce il messaggio di errore per il campo richiesto, oppure una stringa vuota se la validazione^G ha successo.
- `submit(values: Record): Promise<SubmitResult>` - metodo pubblico astratto asincrono. Riceve la mappa dei valori del form, costruisce il DTO appropriato e delega la chiamata HTTP al modulo `AuthAPI`. Restituisce una `Promise<SubmitResult>` che il ViewModel utilizza per aggiornare lo stato della View in base all'esito dell'operazione.

8.1.1.7 useFormViewModel

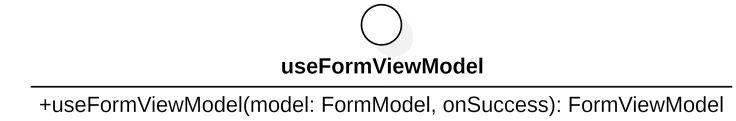

Figure 10: Autenticazione e Registrazione: useFormViewModel

8.1.1.7.1 Descrizione

`useFormViewModel` è un custom React^G Hook che implementa concretamente l'interfaccia `FormViewModel`.
Gestisce lo stato del form tramite i primitivi React^G (`useState`, `useCallback`) e funge da collegamento tra la View e il Model.
Riceve un'istanza di `FormModel` e una callback `onSuccess` da invocare al termine di un'operazione andata a buon fine, passando l'eventuale token^G JWT ricevuto dal backend.

8.1.1.7.2 Attributi

`useFormViewModel` non possiede attributi.

8.1.1.7.3 Metodi e funzioni

- `useFormViewModel(model: FormModel, onSuccess: (token?: string) => void): FormViewModel` - Funzione hook pubblica. Inizializza lo stato del form sulla base dei campi definiti in `model.fields`, implementa gli

handler `handleChange`, `handleBlur` e `handleSubmit` delegando rispettivamente l'aggiornamento dello stato, la validazione^G puntuale e la sottomissione asincrona a `model`. In caso di successo invoca `onSuccess`. Restituisce un oggetto conforme all'interfaccia `FormViewModel`.

8.1.1.8 AuthAPI

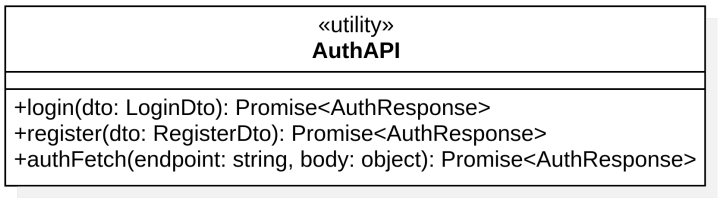

Figure 11: Autenticazione e Registrazione: AuthAPI

8.1.1.8.1 Descrizione

`AuthAPI` è un modulo TypeScript con stereotipo *«utility»* che funge da unico punto di accesso alle API^G REST^G di autenticazione del backend.
Espone funzioni pure (non metodi di istanza) che incapsulano la logica delle chiamate HTTP, costruiscono le richieste nel formato atteso dal server e deserializzano le risposte in oggetti `AuthResponse`.
Viene utilizzato esclusivamente dalle classi Model.

8.1.1.8.2 Attributi

`AuthAPI` non presenta attributi.

8.1.1.8.3 Metodi e funzioni

- `login(dto: LoginDto): Promise<AuthResponse>` - funzione pubblica asincrona. Invia una richiesta HTTP POST all'endpoint `/auth/login` del backend^G con il payload `LoginDto`. Restituisce una `Promise<AuthResponse>` contenente l'esito dell'operazione e l'eventuale token^G in caso di successo.

- `register(dto: RegisterDto): Promise<AuthResponse>` - funzione pubblica asincrona. Invia una richiesta HTTP POST all'endpoint `/auth/register` con il payload `RegisterDto`. Restituisce una `Promise<AuthResponse>` con l'esito dell'operazione.

- `authFetch(endpoint: string, body: object): Promise<AuthResponse>` - funzione helper interna asincrona. Centralizza la logica comune delle chiamate HTTP: costruisce l'header, gestisce gli errori di rete e deserializza la risposta JSON^G in `AuthResponse`.

8.1.1.9 LoginDTO

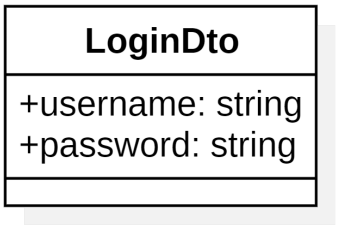

Figure 12: Autenticazione e Registrazione: LoginDTO

8.1.1.9.1 Descrizione

`LoginDto` è un'interfaccia TypeScript che rappresenta il Data Transfer Object utilizzato per la richiesta di autenticazione.
Definisce la struttura del payload JSON^G inviato al backend^G nell'endpoint di login.
Garantisce la correttezza dei tipi dei dati trasmessi e disaccoppia la struttura della richiesta HTTP dalla logica del Model.

8.1.1.9.2 Attributi

- `username: string` - attributo pubblico di tipo stringa, corrisponde al nome utente inserito dall'utente nel campo corrispondente del form di login.
- `password: string` - attributo pubblico di tipo stringa, corrisponde alla password inserita dall'utente, che viene trasmessa in chiaro tramite HTTPS al backend^G, dove verrà verificata contro l'hash bcrypt memorizzato.

8.1.1.9.3 Metodi e funzioni

`LoginDto` non definisce metodi o funzioni.

8.1.1.10 RegisterDTO

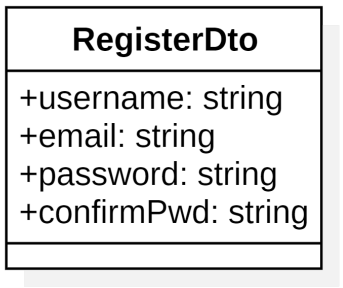

Figure 13: Autenticazione e Registrazione: RegisterDTO

8.1.1.10.1 Descrizione

`RegisterDto` è un'interfaccia TypeScript che rappresenta il Data Transfer Object utilizzato per la richiesta di registrazione di un nuovo utente.
Definisce la struttura del payload JSON^G inviato al backend^G nell'endpoint di registrazione.

8.1.1.10.2 Attributi

- `username: string` - attributo pubblico di tipo stringa. Nome utente scelto dall'utente in fase di registrazione. Deve rispettare il formato definito da `USERNAME_REGEX` in `FormModel`.
- `email: string` - attributo pubblico di tipo stringa. Indirizzo email dell'utente. Deve rispettare il formato definito da `EMAIL_REGEX` in `RegisterModel`.
- `password: string` - attributo pubblico di tipo stringa. Password scelta dall'utente, trasmessa in chiaro tramite HTTPS e cifrata lato backend^G con bcrypt. Deve rispettare il formato definito da `PASSWORD_REGEX` in `FormModel`.
- `confirmPwd: string` - attributo pubblico di tipo stringa. Conferma della password inserita dall'utente, utilizzata lato frontend^G per verificare che i due campi password coincidano prima di inviare la richiesta al backend.

8.1.1.10.3 Metodi e funzioni

`RegisterDto` non definisce metodi o funzioni.

8.1.1.11 AuthResponse

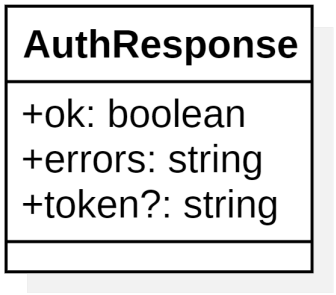

Figure 14: Autenticazione e Registrazione: AuthResponse

8.1.1.11.1 Descrizione

`AuthResponse` è un'interfaccia TypeScript che rappresenta la risposta restituita dal backend^G a seguito di una richiesta di login o registrazione.
Viene deserializzata dal modulo `AuthAPI` e utilizzata dal Model per determinare l'esito dell'operazione e propagare eventuali errori al ViewModel.

8.1.1.11.2 Attributi

- `ok: boolean` - attributo pubblico booleano. Indica se l'operazione di autenticazione o registrazione ha avuto successo. Se `true`, il token^G JWT è presente e l'utente può accedere al sistema.
- `errors: string*` - attributo pubblico di tipo array di stringhe. Contiene i messaggi di errore restituiti dal backend^G in caso di fallimento.
- `token?: string` - attributo pubblico opzionale di tipo stringa. Contiene il token^G JWT restituito dal backend^G in caso di autenticazione o registrazione avvenuta con successo. È assente (`undefined`) in caso di errore.

8.1.1.11.3 Metodi e funzioni

`AuthResponse` non definisce metodi o funzioni.

8.1.1.12 FieldConfig

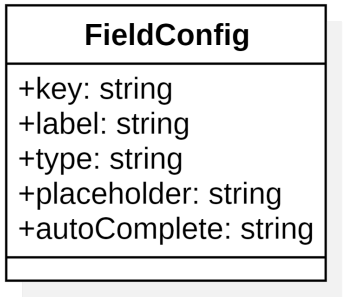

Figure 15: Autenticazione e Registrazione: FieldConfig

8.1.1.12.1 Descrizione

`FieldConfig` è un'interfaccia TypeScript che descrive la configurazione di un singolo campo del form. Viene utilizzata da `FormModel`, `FormProps` e `FormView` per la renderizzazione dinamica dei campi, rendendo il componente View indipendente dalla specifica struttura del form (login o registrazione).

8.1.1.12.2 Attributi

- `key: string` - attributo pubblico di tipo stringa. Identificatore univoco del campo, utilizzato come chiave nelle mappe `values` e `fieldErrors` del ViewModel.
- `label: string` - attributo pubblico di tipo stringa. Etichetta testuale visualizzata sopra al campo di input nel form.
- `type: string` - attributo pubblico di tipo stringa. Tipo HTML dell'input, utilizzato per impostare il corretto attributo `type` del tag `<input>`.
- `placeholder?: string` - attributo pubblico opzionale di tipo stringa. Testo suggerito visualizzato all'interno del campo quando è vuoto, a supporto dell'usabilità del form.
- `autoComplete?: string` - attributo pubblico opzionale di tipo stringa. Valore dell'attributo HTML `autocomplete` del campo, per supportare il riempimento automatico del browser.

8.1.1.12.3 Metodi e funzioni

`FieldConfig` non definisce metodi o funzioni.

8.1.1.13 SubmitResult

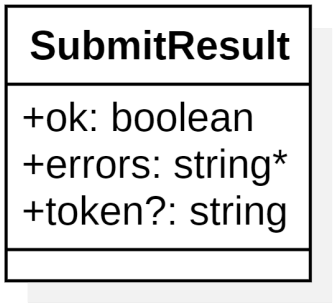

Figure 16: Autenticazione e Registrazione: SubmitResult

8.1.1.13.1 Descrizione

`SubmitResult` è un'interfaccia TypeScript che rappresenta il risultato dell'operazione di submit restituito dal metodo `FormModel.submit()` al ViewModel.
Standardizza il formato della risposta del Model, indipendentemente dal tipo di operazione (login o registrazione), permettendo al ViewModel di aggiornare lo stato della View in modo uniforme.

8.1.1.13.2 Attributi

- `ok: boolean` - attributo pubblico booleano. Indica se l'operazione di submit è terminata con successo.
- `errors: string*` - attributo pubblico di tipo array di stringhe. Contiene i messaggi di errore da mostrare all'utente in caso di fallimento.
- `token?: string` - attributo pubblico opzionale di tipo stringa. Contiene il token^G JWT in caso di submit andato a buon fine.

8.1.1.13.3 Metodi e funzioni

`SubmitResult` non definisce metodi o funzioni.

8.1.1.14 RegisterModel

Figure 17: Autenticazione e Registrazione: RegisterModel

8.1.1.14.1 Descrizione

`RegisterModel` è una classe concreta che estende `FormModel`, specializzandone il comportamento per il caso d'uso^G della registrazione di un nuovo utente. Inizializza i campi del form con la configurazione specifica per la registrazione (username, email, password, conferma password) e sovrascrive `validate` per aggiungere la validazione^G del formato email e la verifica^G della corrispondenza delle due password.

8.1.1.14.2 Attributi

- `EMAIL_REGEX: RegExp` - attributo privato di tipo `RegExp`. Espressione regolare utilizzata per la validazione^G del formato dell'indirizzo email inserito dall'utente durante la registrazione.

8.1.1.14.3 Metodi e funzioni

`RegisterModel` sovrascrive il metodo `validate` ereditato da `FormModel`, estendendolo con la validazione^G del formato username tramite `USERNAME_REGEX`, del formato email tramite `EMAIL_REGEX`, della complessità della password tramite `PASSWORD_REGEX` e della corrispondenza tra i campi `password` e `confirmPwd`.

8.1.1.15 LoginModel

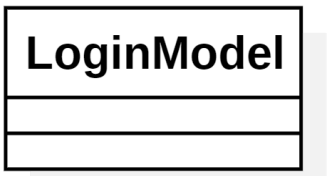

Figure 18: Autenticazione e Registrazione: LoginModel

8.1.1.15.1 Descrizione

`LoginModel` è una classe concreta che estende `FormModel`, specializzandone il comportamento per il caso d'uso^G dell'autenticazione di un utente esistente. Inizializza i campi del form con la configurazione specifica per il login (username e password) e utilizza le regole di validazione^G ereditate da `FormModel` senza aggiungerne di nuove.

8.1.1.15.2 Attributi

`LoginModel` non definisce attributi aggiuntivi rispetto alla classe padre `FormModel`.

8.1.1.15.3 Metodi e funzioni

`LoginModel` eredita senza modifiche i metodi `validate` e `validateField` da `FormModel`. Il metodo `submit` viene specializzato per costruire un oggetto `LoginDto` con i campi username e password.

8.1.2 Backend

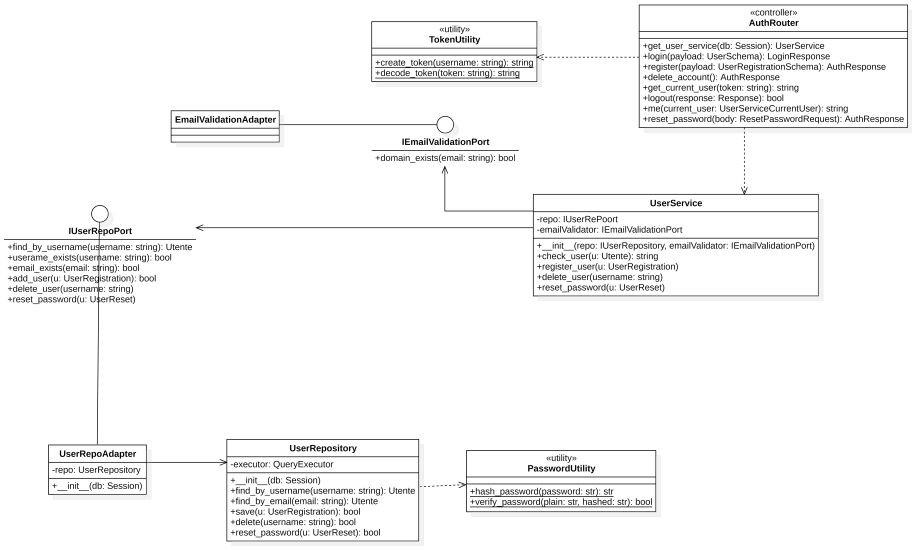

Figure 19: Diagramma delle classi per Autenticazione e Registrazione: Backend

8.1.2.1 UserService

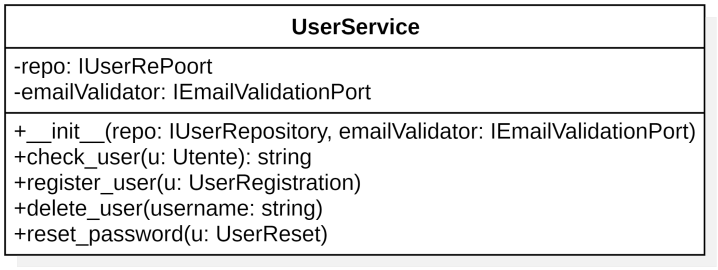

Figure 20: Autenticazione e Registrazione: UserService

8.1.2.1.1 Descrizione

`UserService` è il servizio applicativo che orchestra i casi d'uso di autenticazione e registrazione.
Implementa la logica di business orchestrando i servizi di persistenza (`IUserRepoPort`) e di validazione^G email (`IEmailValidationPort`), lanciando eccezioni mirate quando vengono violate le regole applicative.

8.1.2.1.2 Attributi

- `repo: IUserRepoPort` - porta secondaria verso la persistenza degli utenti; iniettata tramite costruttore.
- `email_validator: IEmailValidationPort` - porta secondaria verso la validazione^G DNS del dominio email; iniettata tramite costruttore.

8.1.2.1.3 Metodi e funzioni

- `__init__(repo: IUserRepository, email_validator: IEmailValidationPort)` - costruttore che riceve le due porte secondarie per dependency injection.
- `check_user(u: Utente) -> string` - verifica^G le credenziali di un utente: recupera l'utente dal repository^G tramite username e controlla la password con `PasswordUtility` restituendo lo username autenticato. Solleva `InvalidCredentialsError` se l'utente non esiste o la password non corrisponde.
- `register_user(u: UserRegistration) -> None` - orchestra la registrazione di un nuovo utente eseguendo in sequenza: la verifica^G dell'univocità dello username (`UsernameAlreadyExistsError`), la validazione^G del DNS del dominio email (`InvalidEmailFormatError`), la verifica^G dell'univocità dell'email (`EmailAlreadyExistsError`) e l'inserimento nel database^G (`UserCreationError`).
- `delete_user(username: str) -> None` - elimina un utente autenticato: verifica^G l'esistenza dell'utente (`UserNotFoundError`) e ne richiede la cancellazione al repository^G (`UserDeletionError`).
- `reset_password(u: UserReset) -> None` - orchestra il reset della password di un utente, verificando l'esistenza dell'utente e aggiornando la password nel sistema di persistenza tramite `PasswordUtility`.

8.1.2.2 AuthRouter

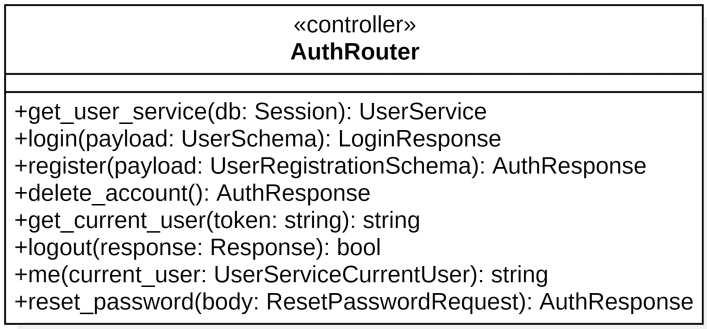

Figure 21: Autenticazione e Registrazione: AuthRouter

8.1.2.2.1 Descrizione

`AuthRouter` è il modulo FastAPI che espone gli endpoint REST^G del dominio autenticazione sotto il prefisso `/auth`.
Non è una classe nel senso stretto, ma un insieme di route handler e funzioni di dipendenza che traducono le richieste HTTP in chiamate al `UserService`, gestendo la serializzazione dei payload e la conversione delle eccezioni dominio in risposte HTTP appropriate.

8.1.2.2.2 Attributi

Il modulo non definisce una classe e pertanto non possiede attributi di istanza. La variabile `router: APIRouter` è definita a livello di modulo e funge da punto di registrazione degli endpoint.

8.1.2.2.3 Metodi e funzioni

- `get_user_service(db`^{G}`: Session) -> UserService` - funzione di dipendenza FastAPI che istanzia `UserService` iniettando `UserRepoAdapter` e `EmailValidationAdapter`; riceve la sessione database^G tramite `get_conn`.
- `login(payload: UserSchema) -> LoginResponse` - handler `POST /auth/login`; riceve un payload `UserSchema` contenente username e password, e delega la verifica^G a `UserService.check_user`. In caso di successo genera e restituisce un `LoginResponse` contenente il JWT prodotto da `TokenUtility.create_token`; in caso di `InvalidCredentialsError` risponde con `HTTP 400`.
- `register(payload: UserRegistrationSchema) -> AuthResponse` - handler `POST /auth/register`; riceve un payload `UserRegistrationSchema` che estende `UserSchema` con email e conferma password, applicando a livello Pydantic la validazione^G del formato di username, password ed email e la corrispondenza tra le due password. Delega la registrazione a `UserService.register_user` e restituisce un `AuthResponse`; mappa le eccezioni del dominio in `HTTP 400` per errori di validazione^G e `HTTP 500` per errori di persistenza.
- `delete_account(current_user: str) -> AuthResponse` - handler `DELETE /auth/account`, richiede un token^G JWT valido tramite `get_current_user`, delega la cancellazione a `UserService.delete_user` e restituisce un `AuthResponse`. Risponde con `HTTP 404` se l'utente non esiste e `HTTP 500` in caso di errore durante la cancellazione.
- `get_current_user(token`^{G}`: str) -> str` - funzione di dipendenza che decodifica il JWT tramite `TokenUtility.decode_token`; solleva `HTTPException 401` se il token^G è assente o non valido. Restituisce lo username estratto dal payload.
- `reset_password(body: ResetPasswordRequest) -> AuthResponse` - handler `POST /auth/reset-password`; riceve i dati necessari al reset della

password e delega l'operazione a `UserService.reset_password`. Restituisce un `AuthResponse` e mappa le eccezioni del dominio in risposte HTTP appropriate.

8.1.2.3 TokenUtility

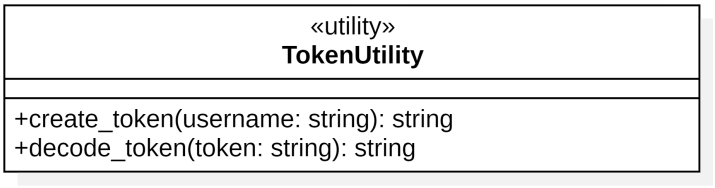

Figure 22: Autenticazione e Registrazione: TokenUtility

8.1.2.3.1 Descrizione

`TokenUtility` è una classe di utilità che fornisce metodi statici per la generazione e la verifica^G dei JWT.
Utilizza la libreria `python-jose` con algoritmo `HS256` e una chiave segreta letta dalla variabile d'ambiente `SECRET_KEY`.

8.1.2.3.2 Attributi

`TokenUtility` non possiede attributi di istanza.
Sono definite le seguenti costanti a livello di modulo:

- `SECRET_KEY: str` - chiave segreta per la firma del token^G, letta dalla variabile d'ambiente `SECRET_KEY`.
- `ALGORITHM: str` - algoritmo di firma, impostato a `HS256`.
- `TOKEN_EXPIRY_HOURS: int` - durata del token^G in ore, impostata a `24`.

8.1.2.3.3 Metodi e funzioni

- `create_token(username: str) -> str` - genera un JWT firmato con payload contenente lo username nel campo `sub` e la scadenza calcolata a partire dal momento corrente. Restituisce il token^G come stringa.
- `decode_token(token`^{G}`: str) -> str | None` - decodifica e verifica^G un JWT; restituisce lo username contenuto nel campo `sub` se il token^G è valido e non scaduto, `None` altrimenti.

8.1.2.4 IEmailValidationPort

Figure 23: Autenticazione e Registrazione: IEmailValidationPort

8.1.2.4.1 Descrizione

`IEmailValidationPort` è una porta secondaria dell'architettura esagonale che definisce il contratto per la validazione[G] del dominio email.
È una classe astratta Python[G] che disaccoppia il dominio applicativo dalla concreta implementazione della validazione[G], permettendo di sostituire o simulare l'adapter in fase di test.

8.1.2.4.2 Attributi

`IEmailValidationPort` non possiede attributi.

8.1.2.4.3 Metodi e funzioni

- `domain_exists(email: str) -> bool` - metodo astratto che verifica[G] l'esistenza del dominio dell'email fornita. L'implementazione concreta è demandata all'adapter `EmailValidationAdapter`.

8.1.2.5 EmailValidationAdapter

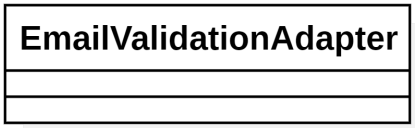

Figure 24: Autenticazione e Registrazione: EmailValidationAdapter

8.1.2.5.1 Descrizione

`EmailValidationAdapter` è l'implementazione concreta di `IEmailValidationPort`. Effettua la validazione[G] del dominio email tramite una query DNS di tipo `MX`, verificando che il dominio sia effettivamente in grado di ricevere posta elettronica.

8.1.2.5.2 Attributi

`EmailValidationAdapter` non possiede attributi.

8.1.2.5.3 Metodi e funzioni

- `domain_exists(email: str) -> bool` - implementazione concreta del metodo astratto definito in `IEmailValidationPort`. Estrae il dominio dall'email e tenta di risolvere il relativo record `MX` tramite la libreria `dnspython`, con un timeout di 3 secondi. Restituisce `True` se il record esiste, `False` in caso di qualsiasi eccezione.

8.1.2.6 IUserRepoPort

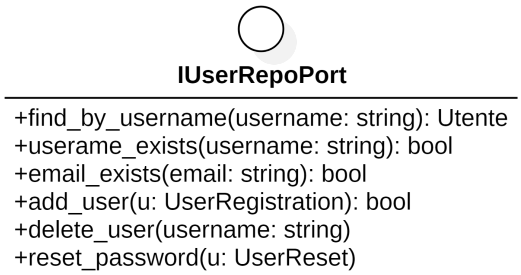

Figure 25: Autenticazione e Registrazione: IUserRepoPort

8.1.2.6.1 Descrizione

`IUserRepoPort` è una porta secondaria dell'architettura esagonale che definisce il contratto per la persistenza degli utenti.
È una classe astratta Python[G] che disaccoppia il dominio applicativo dalla concreta implementazione del repository[G], permettendo di sostituire o simulare l'adapter in fase di test.

8.1.2.6.2 Attributi

`IUserRepoPort` non possiede attributi.

8.1.2.6.3 Metodi e funzioni

- `find_by_username(username: str) -> Utente` - metodo astratto che recupera un utente dal sistema di persistenza tramite username; restituisce `None` se non trovato.
- `username_exists(username: str) -> bool` - metodo astratto che verifica[G] se uno username è già presente nel sistema di persistenza.
- `email_exists(email: str) -> bool` - metodo astratto che verifica[G] se un'email è già presente nel sistema di persistenza.
- `add_user(u: UserRegistration) -> bool` - metodo astratto che inserisce un nuovo utente nel sistema di persistenza; restituisce `True` se l'operazione ha successo.

- `delete_user(username: str) -> bool` - metodo astratto che elimina un utente dal sistema di persistenza tramite username; restituisce `True` se l'operazione ha successo.
- `reset_password(u: UserReset) -> bool` - metodo astratto che aggiorna la password di un utente esistente nel sistema di persistenza; restituisce `True` se l'operazione ha successo.

8.1.2.7 UserRepoAdapter

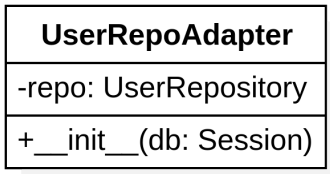

Figure 26: Autenticazione e Registrazione: UserRepoAdapter

8.1.2.7.1 Descrizione

`UserRepoAdapter` è l'implementazione concreta di `IUserRepoPort`.
Funge da adapter verso il livello di persistenza, delegando tutte le operazioni a `UserRepository` che gestisce l'interazione con il database[G] tramite SQLModel.

8.1.2.7.2 Attributi

- `repo: UserRepository` - istanza del repository[G] concreto, inizializzata nel costruttore con la sessione database[G] ricevuta.

8.1.2.7.3 Metodi e funzioni

- `__init__(db`[G]`: Session) -> None` - costruttore che riceve la sessione database[G] e istanzia `UserRepository`.
- `find_by_username(username: str) -> Utente | None` - implementazione concreta che delega la ricerca per username a `UserRepository.find_by_username`.
- `username_exists(username: str) -> bool` - implementazione concreta che verifica[G] l'esistenza dello username delegando a `UserRepository.find_by_username` e controllando che il risultato non sia `None`.
- `email_exists(email: str) -> bool` - implementazione concreta che verifica[G] l'esistenza dell'email delegando a `UserRepository.find_by_email` e controllando che il risultato non sia `None`.
- `add_user(u: UserRegistration) -> bool` - implementazione concreta che delega l'inserimento del nuovo utente a `UserRepository.save`.

- `delete_user(username: str) -> bool` - implementazione concreta che delega la cancellazione dell'utente a `UserRepository.delete`.

8.1.2.8 UserRepository

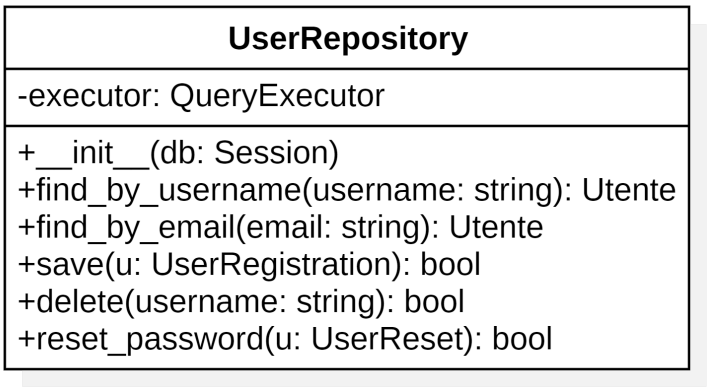

Figure 27: Autenticazione e Registrazione: UserRepository

8.1.2.8.1 Descrizione

`UserRepository` è il repository^G concreto che gestisce le operazioni di persistenza sugli utenti.
Traduce le operazioni di dominio in query SQL^G tramite `QueryExecutor`, utilizzando SQLModel e SQLAlchemy Core per costruire ed eseguire le istruzioni.

8.1.2.8.2 Attributi

- `executor: QueryExecutor` - istanza del query executor, inizializzata nel costruttore con la sessione database^G ricevuta.

8.1.2.8.3 Metodi e funzioni

- `__init__(db`^{G}`: Session) -> None` - costruttore che riceve la sessione database^G e istanzia `QueryExecutor`.
- `find_by_username(username: str) -> Utente` - recupera un utente dal database^G tramite username usando una `SELECT` con filtro su `Utente.username`; restituisce `None` se non trovato.
- `find_by_email(email: str) -> Utente` - recupera un utente dal database^G tramite email usando una `SELECT` con filtro su `Utente.email`; restituisce `None` se non trovato.
- `save(u: UserRegistration) -> bool` - inserisce un nuovo utente nel database^G tramite una `INSERT`, ricevendo un oggetto `UserRegistration` contenente username, email e password in chiaro. La trasmissione della

password in chiaro è considerata sicura in quanto il sistema opera esclusivamente su HTTPS, che garantisce la cifratura a livello di trasporto. La password viene hashata tramite `PasswordUtility.hash_password` prima della persistenza e la descrizione viene impostata a `CLIENTE`; restituisce `True` se l'operazione ha successo.

- `delete(username: str) -> bool` - elimina un utente dal database^G tramite una `DELETE` con filtro su `Utente.username`; restituisce `True` se l'operazione ha successo.
- `reset_password(u: UserReset) -> bool` - aggiorna la password di un utente esistente nel database^G tramite una `UPDATE` con filtro su `Utente.username`, hashando la nuova password tramite `PasswordUtility.hash_password`; restituisce `True` se l'operazione ha successo.

8.1.2.9 PasswordUtility

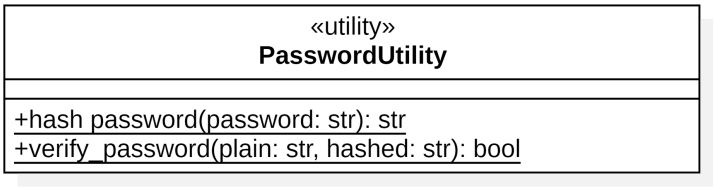

Figure 28: Autenticazione e Registrazione: PasswordUtility

8.1.2.9.1 Descrizione

`PasswordUtility` è una classe di utilità che fornisce metodi statici per l'hashing e la verifica^G delle password tramite la libreria `bcrypt`.
Non mantiene stato interno e viene utilizzata da `UserRepository` in fase di registrazione e da `UserService` in fase di autenticazione.

8.1.2.9.2 Attributi

`PasswordUtility` non possiede attributi di istanza.

8.1.2.9.3 Metodi e funzioni

- `hash_password(password: str) -> str` - genera l'hash bcrypt della password in chiaro tramite `bcrypt.hashpw` con salt casuale; restituisce l'hash come stringa.
- `verify_password(plain: str, hashed: str) -> bool` - verifica^G che la password in chiaro corrisponda all'hash salvato nel database^G tramite `bcrypt.checkpw`; restituisce `False` direttamente se `hashed` è `None`.

9 Stato dei requisiti funzionali

La presente sezione fornisce una visione d'insieme dello stato di avanzamento dei requisiti funzionali identificati durante la fase di analisi. I requisiti funzionali sono stati classificati in base alla loro importanza (obbligatori, desiderabili e opzionali) come definito nel documento Analisi dei Requisiti^{G} v.2.0.0.

9.1 Riepilogo dei requisiti

Durante la fase di analisi sono stati individuati 273 requisiti, fra cui:

- 110 obbligatori, identificati con il codice RF-OB
- 146 desiderabili, identificati con il codice RF-DE
- 17 opzionali, identificati con il codice RF-OP

9.2 Tabella dei requisiti funzionali

Codice	Descrizione	Stato
RF-OB_01	L'Utente deve poter visualizzare il campo di input per inserire lo username	Da implementare
RF-OB_02	L'Utente deve poter inserire lo username	Da implementare
RF-OB_03	Lo username deve essere univoco	Da implementare
RF-OB_04	L'Utente deve poter visualizzare una notifica di errore se lo username è già presente nel sistema	Da implementare
RF-OB_05	Lo username deve avere un massimo di 24 caratteri	Da implementare
RF-OB_06	L'Utente deve poter visualizzare una notifica di errore se lo username eccede il numero massimo di caratteri	Da implementare
RF-OB_07	L'Utente deve poter visualizzare il campo di input per inserire la password	Da implementare
RF-OB_08	L'Utente deve poter inserire la password	Da implementare
RF-OB_09	La password deve avere almeno 1 lettera maiuscola, 1 lettera minuscola, 1 numero, 1 carattere speciale e almeno 8 caratteri	Da implementare

Codice	Descrizione	Stato
RF-OB_10	La password deve avere un massimo di 24 caratteri	Da implementare
RF-OB_11	L'Utente deve poter visualizzare una notifica di errore se la password non è conforme ai criteri	Da implementare
RF-OB_12	L'Utente deve poter visualizzare una notifica di errore se la password supera 24 caratteri	Da implementare
RF-OB_13	L'Utente deve poter tornare alla login	Da implementare
RF-OB_14	L'Utente deve poter visualizzare il campo di input della nuova password	Da implementare
RF-OB_15	L'Utente deve poter confermare la password tramite nuovo campo di input	Da implementare
RF-OB_16	L'Utente deve poter visualizzare una notifica di errore se la password e la sua conferma non coincidono	Da implementare
RF-OB_17	L'Utente deve poter visualizzare il campo di input per l'inserimento dell'email	Da implementare
RF-OB_18	L'Utente deve poter inserire l'email	Da implementare
RF-OB_19	L'email deve essere univoca	Da implementare
RF-OB_20	L'Utente deve poter visualizzare l'errore nel caso in cui l'email non fosse nel formato corretto	Da implementare
RF-OB_21	L'Utente deve poter visualizzare l'errore nel caso in cui l'email sia già presente nel sistema	Da implementare
RF-OB_22	L'Utente deve essere indirizzato alla pagina della chat una volta che la registrazione ha avuto successo	Da implementare
RF-OB_23	L'Utente deve poter visualizzare il campo per inserire lo username	Da implementare
RF-OB_24	L'Utente deve poter inserire lo username	Da implementare
RF-OB_25	L'Utente deve poter visualizzare il campo per inserire la password	Da implementare
RF-OB_26	L'Utente deve poter inserire la password	Da implementare
RF-OB_27	Dev'essere data la possibilità di tornare alla registrazione	Da implementare

Codice	Descrizione	Stato
RF-OB_28	L'Utente deve poter visualizzare il messaggio "Username o password errati" se l'autenticazione fallisce	Da implementare
RF-OB_29	Deve poter avvenire un logout automatico a seguito della cancellazione account	Da implementare
RF-OB_30	L'Utente deve poter visualizzare il pulsante "Logout" nella pagina del profilo	Da implementare
RF-OB_31	L'Utente deve poter effettuare il logout tramite il pulsante "Logout"	Da implementare
RF-OB_32	Il Cliente deve poter inserire input testuali con lunghezza massima di 4096 caratteri	Da implementare
RF-OB_33	Il Cliente deve poter vedere il tasto di invio bloccato al superamento di 4096 caratteri	Da implementare
RF-OB_34	Il Cliente deve poter visualizzare il campo di input	Da implementare
RF-OB_35	Il Cliente deve poter visualizzare il tasto di invio	Da implementare
RF-OB_36	Il Cliente deve poter premere il tasto di invio	Da implementare
RF-OB_37	Il Cliente in assenza di contenuto di input non potrà premere il tasto di invio che sarà disabilitato	Da implementare
RF-OB_38	Il Cliente dopo l'invio dovrà visualizzare il campo di testo vuoto	Da implementare
RF-OB_39	Il Cliente deve poter visualizzare un pulsante con l'icona del microfono	Da implementare
RF-OB_40	Il Cliente deve poter cliccare il pulsante con l'icona del microfono per registrare messaggi audio	Da implementare
RF-OB_41	Il Cliente deve poter visualizzare la durata della registrazione	Da implementare
RF-OB_42	Il Cliente premendo nuovamente l'icona del microfono ferma la registrazione	Da implementare
RF-OB_43	Il Cliente deve poter visualizzare un pulsante con icona "clip"	Da implementare

Codice	Descrizione	Stato
RF-OB_44	Il Cliente deve poter premere il pulsante per selezionare il file da caricare dal dispositivo	Da implementare
RF-OB_45	Il Cliente cliccando sul pulsante deve poter aprire il dialogo di selezione file del sistema operativo	Da implementare
RF-OB_46	Il Cliente deve poter selezionare i file	Da implementare
RF-OB_47	Il Cliente deve poter vedere il tasto “Seleziona” nella selezione file	Da implementare
RF-OB_48	Il Cliente deve poter inviare audio solo se <= 120 secondi	Da implementare
RF-OB_49	Il Cliente deve poter visualizzare un messaggio di errore se il messaggio vocale supera i 120 secondi	Da implementare
RF-OB_50	Il Cliente deve poter inviare file audio con dimensione massima di 10MB in caso contrario ne sarà impedito l’invio	Da implementare
RF-OB_51	Il Cliente deve poter visualizzare un messaggio di errore se le dimensioni dell’audio superano i 10MB	Da implementare
RF-OB_52	Il Cliente deve poter visualizzare un messaggio di errore se il messaggio vocale supera i 10MB	Da implementare
RF-OB_53	Il Cliente deve poter selezionare file audio in formato .mp3	Da implementare
RF-OB_54	Il Cliente deve poter selezionare file audio in formato .m4a	Da implementare
RF-OB_55	Il Cliente deve poter selezionare file audio in formato .m4p	Da implementare
RF-OB_56	Il Cliente deve poter selezionare file audio in formato .wav	Da implementare
RF-OB_57	Il Cliente deve poter visualizzare un messaggio di errore qualora i formati non rientrassero tra quelli accettati	Da implementare
RF-OB_58	Il Cliente deve poter visualizzare un messaggio di errore qualora il chatbot non riconosca gli articoli	Da implementare
RF-OB_59	Il Cliente deve poter aggiungere una massimo di 10 prodotti per volta al carrello tramite input	Da implementare

Codice	Descrizione	Stato
RF-OB_60	Il Cliente deve ricevere conferma dell'aggiunta dei prodotti	Da implementare
RF-OB_61	Il Cliente deve visualizzare un messaggio di errore nel caso in cui tenti di aggiungere più di 10 articoli	Da implementare
RF-OB_62	Il Cliente deve poter rimuovere articoli tramite input	Da implementare
RF-OB_63	Il Cliente deve ricevere una conferma della rimozione degli articoli	Da implementare
RF-OB_64	Il Cliente deve essere avvisato nel caso in cui richiedesse la rimozione di alcuni articoli dal carrello ma quest'ultimo fosse vuoto	Da implementare
RF-OB_65	Il Cliente deve poter inviare il comando testuale "/carrello"	Da implementare
RF-OB_66	Il Cliente deve poter visualizzare il prezzo di ogni articolo	Da implementare
RF-OB_67	Il Cliente deve poter visualizzare il nome di ogni articolo	Da implementare
RF-OB_68	Il Cliente deve poter visualizzare la quantità di ogni articolo	Da implementare
RF-OB_69	Il Cliente deve poter visualizzare i singoli elementi del carrello	Da implementare
RF-OB_70	Il Cliente deve poter visualizzare l'anteprima del carrello	Da implementare
RF-OB_71	Il Cliente deve poter ricevere riceve un avviso nel caso in cui nel carrello non ci fossero elementi	Da implementare
RF-OB_72	Il Cliente deve poter inviare l'ordine	Da implementare
RF-OB_73	Il Cliente deve poter inserire il comando "/invia" nella chat per avviare la conferma dell'ordine	Da implementare
RF-OB_74	Il Cliente deve poter visualizzare il riepilogo ordine nella chat	Da implementare
RF-OB_75	Il Cliente a seguito del messaggio riepilogativo deve decidere se annullare o confermare l'invio dell'ordine	Da implementare
RF-OB_76	Il Cliente deve poter visualizzare i singoli elementi del riepilogo ordine dalla chat	Da implementare

Codice	Descrizione	Stato
RF-OB_77	Il Cliente deve poter confermare l'invio dell'ordine con un input	Da implementare
RF-OB_78	Il Cliente deve poter visualizzare la conferma dell'invio dell'ordine	Da implementare
RF-OB_79	Il Cliente deve poter annullare l'invio dell'ordine con un input	Da implementare
RF-OB_80	Se il carrello è vuoto dopo il comando "/invia" il Cliente visualizza un messaggio di errore che avvisa che il carrello è vuoto	Da implementare
RF-OB_81	Il Cliente deve poter visualizzare un messaggio riguardante l'ambiguità dei prodotti	Da implementare
RF-OB_82	Il Cliente deve poter riformulare l'ordine per prodotti ambigui	Da implementare
RF-OB_83	Il Cliente deve poter annullare la disambiguazione	Da implementare
RF-OB_84	Il Cliente deve poter rispondere "/annulla" per uscire dalla disambiguazione	Da implementare
RF-OB_85	Il Cliente deve poter ricevere conferma dell'annullamento della disambiguazione	Da implementare
RF-OB_86	Il Cliente deve poter vedere il pulsante "Nuova chat"	Da implementare
RF-OB_87	L'Utente deve poter cliccare sul pulsante "Nuova chat"	Da implementare
RF-OB_88	Il Cliente deve poter visualizzare una notifica di errore nel caso in cui il prodotto non venga duplicato correttamente	Da implementare
RF-OB_89	L'Utente deve poter visualizzare il pulsante "visualizza storico ordini"	Da implementare
RF-OB_90	L'Utente deve poter schiacciare il pulsante "visualizza storico ordini"	Da implementare
RF-OB_91	L'Admin deve poter visualizzare una pagina separata contenente lo storico totale degli ordini di tutti i Clienti	Da implementare
RF-OB_92	Il Cliente deve poter visualizzare una pagina separata contente lo storico totale dei suoi ordini	Da implementare

Codice	Descrizione	Stato
RF-OB_93	La lista completa degli ordini deve essere visualizzata in pagine contenenti al massimo 10 ordini ciascuna	Da implementare
RF-OB_94	Il caricamento di una pagina non implica il caricamento delle pagine successive	Da implementare
RF-OB_95	L'Utente deve poter visualizzare il singolo ordine nello storico	Da implementare
RF-OB_96	L'Utente deve poter visualizzare il codice ordine	Da implementare
RF-OB_97	L'Utente deve poter visualizzare la data dell'ordine	Da implementare
RF-OB_98	L'Admin deve poter visualizzare lo username del Cliente	Da implementare
RF-OB_99	L'Utente deve poter visualizzare il dettaglio di un ordine specifico	Da implementare
RF-OB_100	L'Admin deve poter visualizzare lo username del Cliente	Da implementare
RF-OB_101	L'Utente deve poter visualizzare i prodotti dell'ordine	Da implementare
RF-OB_102	L'Utente deve poter visualizzare il nome dei prodotti	Da implementare
RF-OB_103	L'Utente deve poter visualizzare la descrizione dei prodotti	Da implementare
RF-OB_104	L'Utente deve poter visualizzare la quantità dei prodotti	Da implementare
RF-OB_105	L'Utente deve poter visualizzare la data dell'ordine	Da implementare
RF-OB_106	L'Utente deve poter visualizzare il numero dell'ordine	Da implementare
RF-OB_107	Il Cliente deve poter visualizzare il pulsante per duplicare l'ordine	Da implementare
RF-OB_108	Il Cliente deve poter cliccare sul pulsante per duplicare l'ordine	Da implementare
RF-OB_109	Il Cliente deve poter confermare la duplicazione dell'ordine	Da implementare
RF-OB_110	Il Cliente deve poter visualizzare la notifica di errore duplicazione ordine	Da implementare
RF-DE_01	L'Utente deve poter visualizzare il link HTML per la password dimenticata	Da implementare

Codice	Descrizione	Stato
RF-DE_02	L'Utente deve poter cliccare il link HTML per la password dimenticata	Da implementare
RF-DE_03	L'Utente deve poter visualizzare il form per inserire l'indirizzo mail che ha usato quando si è registrato	Da implementare
RF-DE_04	L'Utente deve ricevere via mail la password generata automaticamente	Da implementare
RF-DE_05	L'Utente deve ricevere un messaggio di errore nel caso in cui l'email non fosse valida	Da implementare
RF-DE_06	L'Utente deve ricevere un messaggio di errore nel caso in cui l'email non fosse presente nel sistema	Da implementare
RF-DE_07	L'Utente deve poter visualizzare il pulsante "Elimina Account" nella pagina delle informazioni del profilo	Da implementare
RF-DE_08	L'Utente deve poter premere il pulsante per eliminare il suo account	Da implementare
RF-DE_09	L'Utente deve poter visualizzare la conferma dell'eliminazione dell'account	Da implementare
RF-DE_10	L'Utente deve poter visualizzare nella home un'icona che porta alla pagina delle info del profilo	Da implementare
RF-DE_11	L'Utente deve poter cliccare sul link, rappresentato da un'icona, che porta sulla pagina delle info del proprio profilo	Da implementare
RF-DE_12	L'Utente deve poter visualizzare il link alla pagina dello storico ordini nelle info del proprio profilo	Da implementare
RF-DE_13	L'Utente deve poter visualizzare il proprio username nella pagina delle info del proprio profilo	Da implementare
RF-DE_14	L'Utente deve poter visualizzare la propria ragione sociale nelle info del proprio profilo	Da implementare
RF-DE_15	L'Utente deve poter visualizzare il proprio indirizzo email nelle info del proprio profilo	Da implementare
RF-DE_16	L'Utente deve poter visualizzare il pulsante "Reimposta password"	Da implementare

Codice	Descrizione	Stato
RF-DE_17	L'Utente deve poter cliccare il pulsante "Reimposta password"	Da implementare
RF-DE_18	L'Utente deve poter visualizzare il campo di input "Vecchia password"	Da implementare
RF-DE_19	L'Utente deve poter inserire la vecchia password nel campo di input "Vecchia password"	Da implementare
RF-DE_20	L'Utente deve poter visualizzare un messaggio di errore nel caso in cui l'inserimento della vecchia password non corrispondesse con la password presente nel sistema	Da implementare
RF-DE_21	L'Utente deve poter visualizzare il campo di input "Nuova password"	Da implementare
RF-DE_22	L'Utente deve poter inserire la nuova password nel campo di input "Nuova password"	Da implementare
RF-DE_23	L'Utente deve poter visualizzare un messaggio di errore nel caso in cui la nuova password inserita non fosse conforme ai criteri imposti	Da implementare
RF-DE_24	L'Utente deve poter visualizzare il campo di input "Conferma nuova password"	Da implementare
RF-DE_25	L'Utente deve poter confermare la nuova password nel campo di input "Conferma nuova password"	Da implementare
RF-DE_26	L'Utente deve poter visualizzare un messaggio di errore nel caso in cui la conferma della nuova password non corrisponda alla nuova password inserita nel campo precedente	Da implementare
RF-DE_27	L'Utente deve essere notificato qualora il cambio password vada a buon fine tramite un messaggio di conferma	Da implementare
RF-DE_28	Durante la digitazione il Cliente deve poter visualizzare un contatore x/4096 vicino al campo di testo	Da implementare
RF-DE_29	Il Cliente deve poter visualizzare il pulsante di caricamento file immagine (icona fotocamera)	Da implementare

Codice	Descrizione	Stato
RF-DE_30	Il Cliente deve poter cliccare l'icona della fotocamera	Da implementare
RF-DE_31	Il Cliente a seguito del click sull'icona a forma di clip deve poter vedere un dropdown tra cui: "Seleziona dal dispositivo" oppure "Scatta la foto"	Da implementare
RF-DE_32	Il Cliente deve poter selezionare "Seleziona dal dispositivo"	Da implementare
RF-DE_33	Se Il Cliente seleziona "Seleziona dal dispositivo" deve poter visualizzare i media nel dispositivo	Da implementare
RF-DE_34	Il Cliente deve poter selezionare "Scatta foto"	Da implementare
RF-DE_35	Alla selezione "Scatta foto" il Cliente visualizza l'attivazione della fotocamera	Da implementare
RF-DE_36	Il Cliente deve poter inviare immagini con dimensione massima di 15MB e con risoluzione pari o superiore a 800x600 pixel	Da implementare
RF-DE_37	Nel caso in cui le immagini superino i 15MB abbiano risoluzione inferiore di 800x600 pixel sarà visibile al Cliente un messaggio di errore "L'immagine non rispetta i requisiti richiesti"	Da implementare
RF-DE_38	Il Cliente deve premere il pulsante di invio per confermare l'invio dell'immagine	Da implementare
RF-DE_39	Il Cliente deve poter visualizza un messaggio "Non sono riuscito a rilevare il testo nell'immagine" se il modello non riesce a rilevare testo	Da implementare
RF-DE_40	Il Cliente deve poter selezionare immagini nei formati .jpg	Da implementare
RF-DE_41	Il Cliente deve poter selezionare immagini nei formati .jpeg	Da implementare
RF-DE_42	Il Cliente deve poter selezionare immagini nei formati .png	Da implementare
RF-DE_43	Il Cliente deve poter visionare l'anteprima dell'immagine caricata	Da implementare

Codice	Descrizione	Stato
RF-DE_44	Il Cliente deve poter visualizzare il pulsante a fianco all'ultima risposta "lascia Feedback"	Da implementare
RF-DE_45	Il Cliente deve poter premere il pulsante affianco ad ogni risposta "lascia Feedback" per fornire feedback all'AI	Da implementare
RF-DE_46	Il Cliente premendo su "lascia Feedback" deve visualizzare l'apertura di un form in una nuova pagina	Da implementare
RF-DE_47	Il Cliente deve vedere il dropdown relativo alla tipologia di feedback: "Prodotto sbagliato", "Quantità errata", "Incomprensione", "Altro"	Da implementare
RF-DE_48	Il Cliente deve selezionare obbligatoriamente una tipologia: "Prodotto sbagliato", "Quantità errata", "Incomprensione", "Altro"	Da implementare
RF-DE_49	Il Cliente visualizza l'input testuale della descrizione	Da implementare
RF-DE_50	Il Cliente deve inserire obbligatoriamente una descrizione	Da implementare
RF-DE_51	Il Cliente nella descrizione deve poter inserire un massimo di 300 caratteri	Da implementare
RF-DE_52	Il Cliente deve vedere un contatore x/300 vicino al campo di testo	Da implementare
RF-DE_53	Al superamento dei 300 caratteri il Cliente non potrà più scrivere nell'input	Da implementare
RF-DE_54	Il Cliente visualizza in fondo alla pagina il pulsante "Invia feedback"	Da implementare
RF-DE_55	Il Cliente deve poter premere il pulsante "Invia feedback"	Da implementare
RF-DE_56	Il Cliente deve poter ricevere la notifica di la conferma di invio feedback	Da implementare
RF-DE_57	Il Cliente deve poter vedere nella home il pulsante "Segnala un problema"	Da implementare
RF-DE_58	Il Cliente deve poter cliccare sul pulsante "Segnala un problema"	Da implementare

Codice	Descrizione	Stato
RF-DE_59	Il Cliente deve visualizzare il dropdown: "Bug^G", "Richiesta di supporto", "Suggerimento"	Da implementare
RF-DE_60	Il Cliente deve poter selezionare il tipo da dropdown	Da implementare
RF-DE_61	Il Cliente deve poter visualizzare il tasto "Invia segnalazione"	Da implementare
RF-DE_62	Il Cliente deve poter inviare il form cliccando "Invia segnalazione"	Da implementare
RF-DE_63	Il Cliente deve poter ricevere una notifica di conferma dell'invio con un messaggio "Ticket creato con successo"	Da implementare
RF-DE_64	L'Admin deve poter visualizzare la pagina web delle performance	Da implementare
RF-DE_65	L'Admin deve poter visualizzare il link per la pagina delle performance in qualsiasi pagina	Da implementare
RF-DE_66	L'Admin deve poter schiacciare il link per la pagina delle performance in qualsiasi pagina	Da implementare
RF-DE_67	L'Admin deve poter visualizzare il tempo medio di risposta del chatbot	Da implementare
RF-DE_68	L'Admin deve poter visualizzare il tempo medio di permanenza nella web-app	Da implementare
RF-DE_69	L'Admin deve poter visualizzare un pulsante relativo all'esportazione del log	Da implementare
RF-DE_70	L'Admin deve poter schiacciare il pulsante relativo all'esportazione del log	Da implementare
RF-DE_71	L'Admin deve poter ottenere il file di log in formato JSON^G	Da implementare
RF-DE_72	L'Admin deve poter ottenere il file JSON^G dello storico ordini	Da implementare
RF-DE_73	L'Admin deve poter visualizzare il tasto "Esporta" nella pagina dello storico ordini	Da implementare
RF-DE_74	L'Admin deve poter schiacciare il tasto "Esporta" nella pagina dello storico ordini	Da implementare

Codice	Descrizione	Stato
RF-DE_75	L'Admin deve poter visualizzare la conferma dell'esportazione dello storico ordini	Da implementare
RF-DE_76	L'Admin deve poter visualizzare un errore nel caso in cui non fosse stato possibile scaricare nel proprio dispositivo il file	Da implementare
RF-DE_77	Il file JSON^G esportato deve contenere il campo id [integer]: identificativo univoco dell'ordine	Da implementare
RF-DE_78	Il file JSON^G esportato deve contenere il campo cod_cli [integer]: identificativo univoco del cliente	Da implementare
RF-DE_79	Il file JSON^G esportato deve contenere il campo cod_art [varchar(13)]: identificativo univoco dell'articolo	Da implementare
RF-DE_80	Il file JSON^G esportato deve contenere il campo data_ord [date]: data di inserimento dell'ordine	Da implementare
RF-DE_81	Il file JSON^G esportato deve contenere il campo qta_ordinata [float]: quantità che l'Utente ha ordinato nell'unità di misura dell'articolo	Da implementare
RF-DE_82	Il file JSON^G esportato deve contenere il campo rif [integer/string]: rappresenta un riferimento al testo originario	Da implementare
RF-DE_83	Il Cliente deve poter richiedere la duplicazione indicando il codice dell'ordine dentro il comando comando "/duplica{xx}"	Da implementare
RF-DE_84	Il Cliente deve poter richiedere la duplicazione dell'ultimo ordine effettuato indicando il comando "/duplica"	Da implementare
RF-DE_85	Il Cliente deve poter richiedere la duplicazione di un ordine tramite un input testuale, immagine o di audio	Da implementare
RF-DE_86	Il Cliente deve poter visualizzare una notifica di conferma della duplicazione	Da implementare
RF-DE_87	L'Admin deve poter visualizzare le statistiche	Da implementare

Codice	Descrizione	Stato
RF-DE_88	L'Admin deve poter visualizzare il numero di utenti presenti in quel momento	Da implementare
RF-DE_89	L'Admin deve poter visualizzare il numero di acquisti completati tramite la piattaforma	Da implementare
RF-DE_90	L'Admin deve poter visualizzare, in forma di testo, il modello AI utilizzato in quel momento nella pagina delle performance	Da implementare
RF-DE_91	L'Utente deve poter visualizzare l'elenco dei comandi in una pagina web dedicata	Da implementare
RF-DE_92	L'Utente deve poter visualizzare il pulsante contenente il link per la pagina dell'elenco dei comandi in qualsiasi pagina	Da implementare
RF-DE_93	L'Utente deve poter schiacciare il pulsante contenente il link per la pagina dell'elenco dei comandi in qualsiasi pagina	Da implementare
RF-DE_94	Il Cliente deve poter visualizzare l'elenco dei comandi con la relativa spiegazione	Da implementare
RF-DE_95	Il Cliente deve poter visualizzare l'elenco dei comandi con la relativa spiegazione	Da implementare
RF-DE_96	Il Cliente deve poter visualizzare il comando "/duplica" con la relativa spiegazione	Da implementare
RF-DE_97	Il Cliente deve poter visualizzare il comando "/carrello" con la relativa spiegazione	Da implementare
RF-DE_98	Il Cliente deve poter visualizzare il comando "/duplica{xx}" con la relativa spiegazione	Da implementare
RF-DE_99	Il Cliente deve poter visualizzare il comando "/invia" con la relativa spiegazione	Da implementare
RF-DE_100	Il Cliente deve poter visualizzare il comando "/annulla" con la relativa spiegazione	Da implementare

Codice	Descrizione	Stato
RF-DE_101	Il Cliente deve poter visualizzare il comando "/comandi" con la relativa spiegazione	Da implementare
RF-DE_102	Il Cliente deve poter visualizzare i comandi inline	Da implementare
RF-DE_103	Il Cliente deve poter visualizzare i comandi disponibili	Da implementare
RF-DE_104	Il Cliente deve poter visualizzare il comando "/duplica"	Da implementare
RF-DE_105	Il Cliente deve poter visualizzare il comando "/carrello"	Da implementare
RF-DE_106	Il Cliente deve poter visualizzare il comando "/duplica{xx}"	Da implementare
RF-DE_107	Il Cliente deve poter visualizzare il comando "/invia"	Da implementare
RF-DE_108	Il Cliente deve poter visualizzare il comando "/annulla"	Da implementare
RF-DE_109	Il Cliente deve poter visualizzare il comando "/comandi"	Da implementare
RF-DE_110	Il Cliente deve poter inviare un comando nella chat	Da implementare
RF-DE_111	Il Cliente deve poter inviare il comando "/duplica" nella chat	Da implementare
RF-DE_112	Il Cliente deve poter inviare il comando "/duplica{xx}" nella chat	Da implementare
RF-DE_113	Il Cliente deve poter inviare il comando "/invia" nella chat	Da implementare
RF-DE_114	Il Cliente deve poter inviare il comando "/annulla" nella chat	Da implementare
RF-DE_115	Il Cliente deve poter inviare il comando "/comandi" nella chat	Da implementare
RF-DE_116	L'Utente deve poter impostare dei filtri	Da implementare
RF-DE_117	L'Utente deve poter filtrare per data gli ordini presenti nello storico ordini	Da implementare
RF-DE_118	L'Utente deve poter visualizzare l'icona del filtro data a forma di calendario	Da implementare
RF-DE_119	L'Utente deve poter cliccare sull'icona del filtro data	Da implementare

Codice	Descrizione	Stato
RF-DE_120	Il filtro per data deve poter permettere di visualizzare tutti gli ordini successivi o uguali ad una data scelta	Da implementare
RF-DE_121	Il filtro per data deve poter permettere di visualizzare tutti gli ordini compresi tra due date scelte	Da implementare
RF-DE_122	L'Utente deve poter vedere il campo testuale del filtro prodotti	Da implementare
RF-DE_123	L'Utente deve poter scrivere nel campo testuale del filtro prodotti	Da implementare
RF-DE_124	L'Utente deve poter vedere un riquadro di prodotti presenti che iniziano o sono uguali alla sequenza di caratteri inserita	Da implementare
RF-DE_125	L'Utente, digitando il nome del prodotto, deve poter vedere un riquadro dei prodotti presenti che contengono quella sequenza di caratteri	Da implementare
RF-DE_126	L'Admin deve poter vedere il campo testuale del filtro Cliente	Da implementare
RF-DE_127	L'Admin deve poter scrivere nel campo testuale del filtro Cliente	Da implementare
RF-DE_128	L'Admin deve poter scegliere dal riquadro dei Clienti presenti il Cliente desiderato	Da implementare
RF-DE_129	L'Admin deve poter vedere un riquadro di Clienti presenti che iniziano con la sequenza di caratteri inserita	Da implementare
RF-DE_130	L'Admin deve poter vedere un riquadro di Clienti presenti che contengono la sequenza di caratteri inserita	Da implementare
RF-DE_131	L'Admin deve poter scegliere dal riquadro dei Clienti presenti lo username del Cliente desiderato	Da implementare
RF-DE_132	Il Cliente deve poter ricevere un avviso nel caso in cui invii più comandi nello stesso input	Da implementare
RF-DE_133	Il Cliente deve poter ricevere un avviso nel caso in cui invii il comando con altro testo	Da implementare

Codice	Descrizione	Stato
RF-DE_134	Il Cliente deve poter visualizzare le chat	Da implementare
RF-DE_135	Il Cliente deve poter visualizzare una singola chat	Da implementare
RF-DE_136	Il Cliente deve poter visualizzare la lista dei messaggi di una singola chat	Da implementare
RF-DE_137	Il Cliente deve poter visualizzare un messaggio singolo all'interno della chat	Da implementare
RF-DE_138	Il Cliente deve poter visualizzare un messaggio singolo dell'Utente all'interno della chat	Da implementare
RF-DE_139	Il Cliente deve poter visualizzare un messaggio singolo dell'AI all'interno della chat	Da implementare
RF-DE_140	Il Cliente deve poter selezionare la chat	Da implementare
RF-DE_141	Il Cliente deve poter cancellare una chat	Da implementare
RF-DE_142	Il Cliente deve poter vedere il pulsante per cancellare la chat	Da implementare
RF-DE_143	Il Cliente deve poter premere il pulsante per cancellare la chat	Da implementare
RF-DE_144	Il Cliente deve poter confermare la cancellazione della chat	Da implementare
RF-DE_145	Il Cliente deve poter visualizzare il pulsante di invio ordine dal carrello	Da implementare
RF-DE_146	Il Cliente deve poter premere il pulsante abilitato di invio ordine	Da implementare
RF-OP_01	Tutte le informazioni riguardanti l'Utente utili ai fini del logging devono essere anonimizzate	Da implementare
RF-OP_02	L'Admin deve poter creare nuovi utenti	Da implementare
RF-OP_03	L'Admin deve poter visualizzare il pulsante relativo alla creazione di un nuovo Utente	Da implementare
RF-OP_04	L'Admin deve poter schiacciare il pulsante relativo alla creazione di un nuovo Utente	Da implementare
RF-OP_05	L'Admin deve poter selezionare il ruolo di quest'ultimo	Da implementare
RF-OP_06	L'Admin deve poter selezionare il ruolo Cliente	Da implementare

Codice	Descrizione	Stato
RF-OP_07	L'Admin deve poter selezionare il ruolo Admin	Da implementare
RF-OP_08	L'Admin deve poter visualizzare il campo di input per inserire lo username	Da implementare
RF-OP_09	L'Admin deve poter inserire lo username	Da implementare
RF-OP_10	Lo username deve essere univoco	Da implementare
RF-OP_11	L'Admin deve poter visualizzare una notifica di errore se lo username è già presente nel sistema	Da implementare
RF-OP_12	L'Admin deve poter visualizzare il campo di input per l'inserimento dell'email	Da implementare
RF-OP_13	L'Admin deve poter inserire l'email	Da implementare
RF-OP_14	L'Admin deve poter visualizzare l'errore nel caso in cui l'email non fosse nel formato corretto	Da implementare
RF-OP_15	L'Admin deve poter visualizzare l'errore nel caso in cui l'email sia già presente nel sistema	Da implementare
RF-OP_16	Deve essere visualizzata una notifica a schermo che informa della corretta creazione dell'Utente	Da implementare
RF-OP_17	L'Admin deve poter visualizzare la notifica di errore creazione utente	Da implementare

Table 2: Stato dei Requisiti Funzionali