



UNIVERSITÀ DEGLI STUDI DI TRENTO  
FACOLTÀ DI INFORMATICA  
Corso di Ingegneria del Software

# Fix Mi Application Implementation and Documentation

Gruppo G43:  
Giovanni Santini  
Riginel Ungureanu  
Valerio Asaro

Anno Accademico 2023/2024  
Trento

## CONTENTS

0.1	Scopo del documento . . . . .	4
0.2	Implementazione . . . . .	4
0.3	Informazioni del Documento . . . . .	4
<b>1</b>	<b>Microservizi</b>	<b>5</b>
1.1	Architettura dei Microservizi . . . . .	5
1.2	Visione Generale . . . . .	6
1.2.1	Tecnologie . . . . .	6
1.2.2	Specifiche dell'Infrastruttura . . . . .	7
1.3	Codice . . . . .	9
1.3.1	Codice: Struttura . . . . .	10
1.3.2	Codice: docker-compose.yaml . . . . .	11
1.3.3	Codice: nginx.conf . . . . .	14
1.4	Parti comuni ad ogni microservizio . . . . .	18
1.4.1	Scelta del Linguaggio . . . . .	18
1.4.2	Frameworks e Dipendenze . . . . .	18
1.4.3	Struttura . . . . .	19
1.4.4	Comandi npm . . . . .	22
1.4.5	Elementi Front-End . . . . .	22
1.5	Modellazione dati nel database . . . . .	26
<b>2</b>	<b>Microservizio Autenticazione</b>	<b>30</b>
2.1	Back-End . . . . .	30
2.1.1	Struttura della Back-End . . . . .	30
2.1.2	Specifiche delle risorse . . . . .	31
2.1.3	Login . . . . .	33

2.1.4	Logout . . . . .	40
2.1.5	Registrazione . . . . .	46
2.1.6	Cambio password . . . . .	53
2.1.7	Eliminazione Profilo . . . . .	59
2.1.8	Autentifica Token . . . . .	66
2.1.9	Two Factor Authentication . . . . .	72
2.2	Front-End . . . . .	76
2.2.1	Struttura del Front-End . . . . .	77
2.2.2	UserFlow . . . . .	78
2.2.3	Login . . . . .	80
2.2.4	Registrazione . . . . .	83
2.2.5	Two Factor Authentication . . . . .	86
2.2.6	Profilo . . . . .	90
<b>3</b>	<b>Microservizio Task</b>	<b>94</b>
3.1	Back-End . . . . .	94
3.1.1	Lista delle Risorse . . . . .	94
3.1.2	Specifiche delle Risorse . . . . .	95
3.1.3	Task . . . . .	95
3.1.4	Scegli Task . . . . .	105
3.1.5	Modifica Stato Task . . . . .	115
3.1.6	Get Lista Task In Lavorazione . . . . .	126
3.1.7	Get Lista Task Da Eseguire . . . . .	134
3.1.8	Get Lista Task In Pausa . . . . .	142
3.1.9	Get Storico Task . . . . .	150
3.1.10	Crea Task . . . . .	158
3.2	Front-End . . . . .	170
3.2.1	Struttura del Front-End . . . . .	170
3.2.2	UserFlow . . . . .	171
3.2.3	Tasks . . . . .	173
3.2.4	Task . . . . .	178
<b>4</b>	<b>Microservizio Gestione Dipendenti</b>	<b>182</b>
4.1	Back-End . . . . .	182
4.1.1	Crea profilo Dipendente . . . . .	184
4.1.2	Elimina profilo . . . . .	195

<b>5 Microservizio Home</b>	<b>217</b>
5.1 Front-End . . . . .	217
5.1.1 Struttura del Front-End . . . . .	217
5.1.2 UserFlow . . . . .	219
5.1.3 Homepage . . . . .	221
5.1.4 Riparazione . . . . .	223
5.1.5 Assistenza . . . . .	225
5.1.6 Feedback . . . . .	227
<b>6 Deploy</b>	<b>229</b>
6.1 Dipendenze . . . . .	229
6.2 Ottenere il codice . . . . .	229
6.3 Istruzioni . . . . .	230
6.4 Eseguire Singoli Microservizi . . . . .	230

## 0.1 Scopo del documento

Questo documento denominato "Application Implementation and Documentation" descrive in dettaglio l'implementazione dell'applicazione nelle sue molteplici parti, seguendo quanto visto nei precedenti documenti. Il documento è strutturato in diversi capitoli: il primo dedicato alla specifica dell'infrastruttura in generale, con grafici e codice, seguito da quattro capitoli dedicati alle varie funzionalità implementate, con una visione della Back-End e della Front-End. Il documento si chiude con le istruzioni per il deploy dell'applicazione.

## 0.2 Implementazione

Il team di sviluppo, valutando i requisiti del progetto ed il tempo a disposizione, ha deciso di implementare i seguenti microservizi:

- Autenticazione
- Task
- Home
- Gestione Dipendenti

Inoltre vengono implementati i seguenti servizi:

- Reverse Proxy
- Database

## 0.3 Informazioni del Documento

Campo	Valore
Titolo del Documento	Application Implementation and Documentation
Titolo del Progetto	Fix Mi
Autori del Documento	Giovanni Santini Riginel Ungureanu Valerio Asaro
Amministratore Progetto	Riginel Ungureanu
Versione del documento	1.0

---

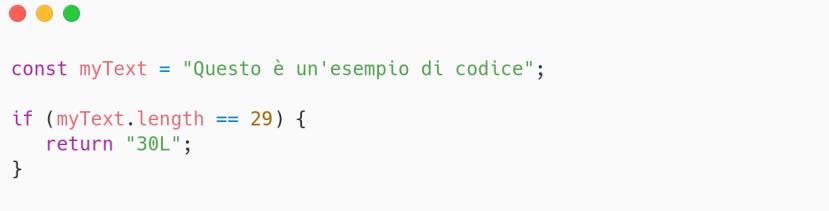
CHAPTER  
**ONE**

---

## MICROSERVIZI

Il seguente capitolo descrive l'implementazione, con il supporto di diagrammi "UML" e codice, dell'architettura dell'applicazione, in riferimento a quanto detto nei precedenti documenti quali "Analisi dei Requisiti", "Specifiche dei Requisiti" e del "Documento di Architettura". Il capitolo è strutturato partendo da una visione generale dell'infrastruttura, andando ad analizzare più nello specifico i suoi componenti. I successivi capitoli andranno a descrivere nello specifico ogni microservizio.

Gli estratti di codice verranno mostrati nel seguente formato:



```
● ● ●  
const myText = "Questo è un'esempio di codice";  
  
if (myText.length == 29) {  
    return "30L";  
}
```

Esempio di estratto di codice

### 1.1 Architettura dei Microservizi

L'architettura proposta si basa sulla divisione logica (e fisica) delle funzionalità dell'applicazione tramite la distinzione di *Micsoservizi*. Ciascun microservizio è indipendente dagli altri ed è composto a sua volta da una Front-End e una

Back-End distinte. Per tale ragione, è più accurato parlare di micro-Front-End e micro-Back-End.

### Vantaggi di un'architettura basata su microservizi

Da un punto di vista di sviluppo, un'architettura non monolitica permette lo sviluppo asincrono dei singoli microservizi, oltre a facilitare la divisione del lavoro nelle varie parti. Tale architettura ha anche dei vantaggi a livello di performance in quanto il carico di lavoro che l'applicazione processa viene distribuito su più processi diminuendo il carico sul singolo. Altri vantaggi sono la scalabilità dia una componente dell'applicazione in base alle esigenze e la ridondanza in quanto possono esistere più istanze di un microservizio contemporaneamente.

### Svantaggi

Un'architettura a microservizi è intrinsecamente più complessa nella progettazione e nella manutenzione rispetto ad una architettura tradizionale. La necessità di orchestrare e mettere in comunicazione i diversi microservizi richiede particolari accortezze nella parte di design e di deploy. Tali problematiche sono state valutate con cura dal team di sviluppo.

## 1.2 Visione Generale

Prima di analizzare il singolo microservizio, questa sezione illustra una visione generale dell'applicazione e degli strumenti utilizzati per la realizzazione dell'infrastruttura, per poi concentrarsi sulle parti comuni di ogni microservizio e infine sul singolo microservizio.

### 1.2.1 Tecnologie

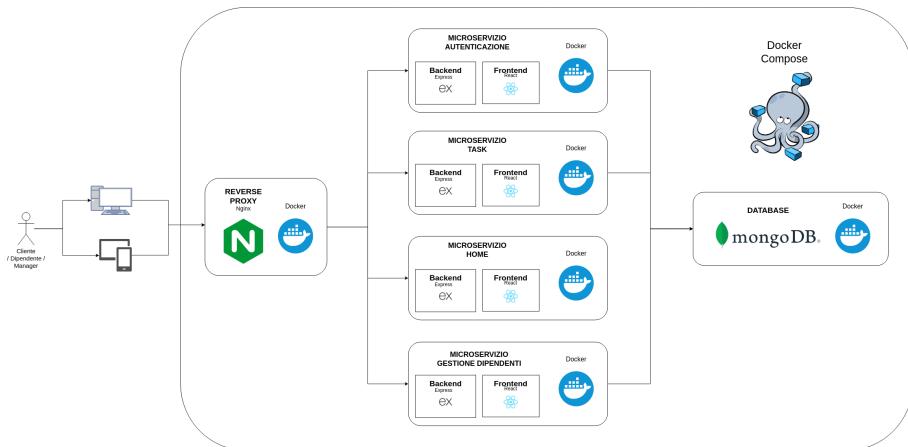
In questo capitolo verranno menzionate più volte alcune tecnologie, dunque ne viene riportata una breve descrizione sotto:

- "Docker": tecnologia che raccoglie il software in unità standardizzate chiamate container, offrendo tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Il logo ricorda una balena di colore azzurro con dei rettangoli al di sopra rappresentanti dei containers, come se la balena stessa fosse una nave.

- "Docker Compose": software per la gestione di molteplici Docker containers contemporaneamente. L'icona rappresenta un polpo con dei parallelepipedi azzurri tra i tentacoli.
- "Nginx": intermediario tra le richieste da parte dei client e il server. Può essere usato come load balancer, cache o proxy. Il logo è raffigurato da un N bianca con sfondo verde.
- "MongoDB": DBMS non relazionale. Il logo rappresenta una foglia verde.
- "Express": framework Back-End per applicazioni web in javascript. Il logo è formato dalle due lettere "e" e "x" in nero.
- "React": framework Front-End lo sviluppo di applicazioni web. L'icona ricorda un atomo.

### 1.2.2 Specifica dell'Infrastruttura

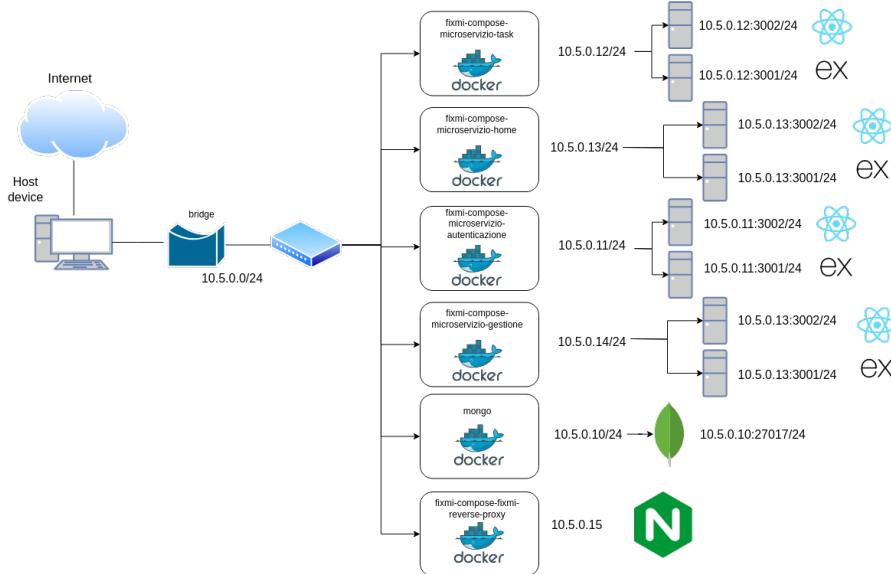
Si presta attenzione alla seguente infografica dell'infrastruttura implementata:



Schema dei microservizi

L'intera infrastruttura viene inizializzata con *Docker Compose*. In particolare, *Docker Compose* si occupa di impostare i containers sullo stesso network con un IP statico e le variabili di ambiente come le porte e gli IP dei rispettivi microservizi, oltre ad inizializzare i containers, le porte e i volumi condivisi.

Se eseguito su un singolo host, il network di default segue la seguente struttura:



Schema del network dei microservizi

Docker imposta i microservizi sul network 10.5.0.0/24 con i seguenti IP:

Nome Microservizio	Network IP
MongoDB	10.5.0.10/24
Microservizio Autenticazione	10.5.0.11/24
Microservizio Task	10.5.0.12/24
Microservizio Home	10.5.0.13/24
Microservizio Gestione Dipendenti	10.5.0.14/24
Reverse Proxy	127.0.0.1

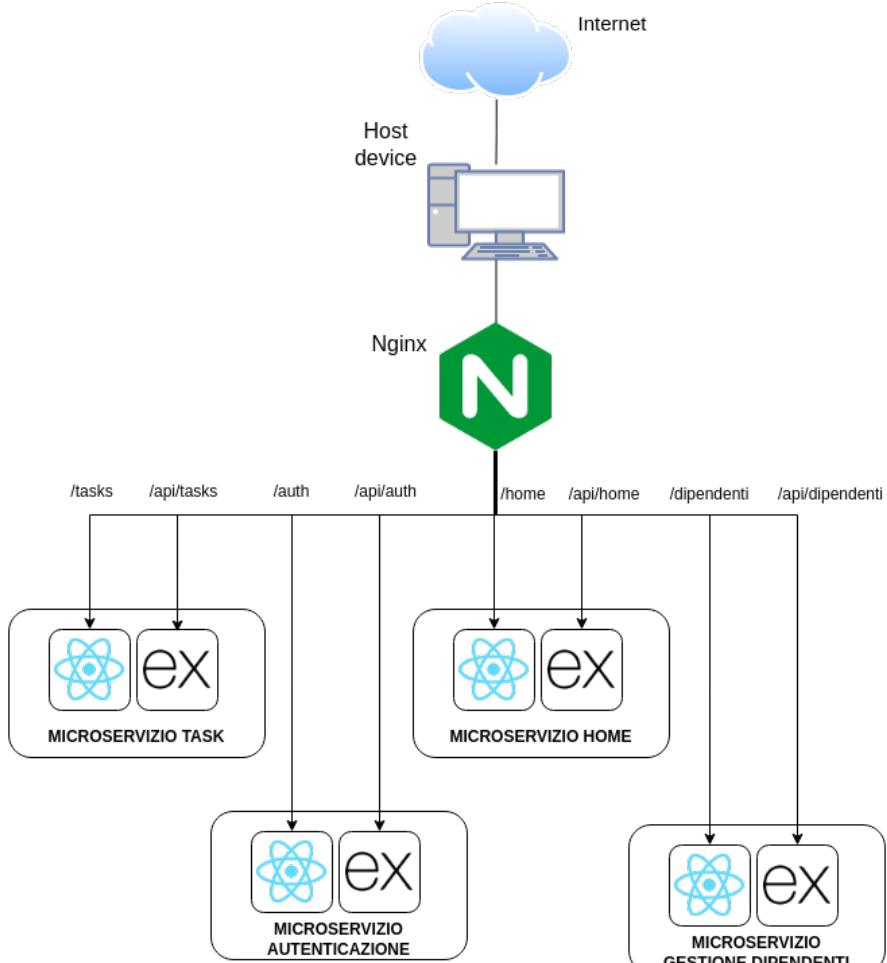
Ad ogni servizio sono state assegnate le seguenti porte:

Nome servizio	Porta
MongoDB	27017
Back-End	3001
Front-End	3002
Reverse Proxy	7777

Due microservizi particolari sono il database e il reverse proxy. Il database fornisce la possibilità di salvare in modo permanente i dati dell'applicazione,

mentre il reverse proxy permette di accedere facilmente all'ip di un microservizio attraverso una mappatura di ip. I software scelti sono rispettivamente *MongoDB* e *Nginx*.

In particolare, *Nginx* associa le routes nel seguente modo:



Schema delle routes di Nginx

### 1.3 Codice

In questa sezione verrà mostrato il codice che implementa quanto sopra, con opportuna spiegazione del funzionamento quando necessaria.

### 1.3.1 Codice: Struttura

La directory radice del progetto contiene i seguenti files:

```
● ● ●
$> tree -L 1
.
├── db
├── docker-compose.yaml
├── fixmi-microservice-gestione-dipendenti
├── fixmi-microservizio-autenticazione
├── fixmi-microservizio-home
├── fixmi-microservizio-task
├── fixmi-reverse-proxy
└── init_scripts
    └── README.md
```

Output del comando "tree", mostra i contenuti della directory root del progetto.

Segue una descrizione degli stessi:

- "db": volume condiviso tra il Docker container del database e il filesystem dell'host per mantenere persistenza dei dati quando il container viene riavviato o rimosso.
- "docker-compose.yaml": file di configurazione utilizzato da Docker Compose per inizializzare l'infrastruttura. Contiene le informazioni per avviare gli altri microservizi.
- "fixmi-microservice-gestione-dipendenti": cartella contenente il microservizio "Gestione Dipendenti".
- "fixmi-microservizio-autenticazione": cartella contenente il microservizio "Autenticazione".
- "fixmi-microservizio-home": cartella contenente il microservizio "Home".
- "fixmi-microservizio-task": cartella contenente il microservizio "task".
- "fixmi-reverse-proxy": cartella contenente il reverse proxy.
- "init\_scripts": script per inizializzare il database con dei dati di esempio.
- "README.md": contiene informazioni e documentazione sul deploy dell'infrastruttura.

### 1.3.2 Codice: docker-compose.yaml

Si prenda come esempio questo frammento di codice responsabile del setup del microservizio task:

```
version: "3"

services:
  microservizio-task:
    build: ./fixmi-microservizio-task
    ports:
      - "3004:3002"
      - "3003:3001"

    volumes:
      - ./fixmi-microservizio-task:/app

    restart: always

    environment:
      MICROSERVIZIO_DB_IP: ${MICROSERVIZIO_DB_IP}
      MICROSERVIZIO_DB_USERNAME: ${DB_USERNAME}
      MICROSERVIZIO_DB_PASSWORD: ${DB_PASSWORD}
      MICROSERVIZIO_AUTENTICAZIONE_IP: ${MICROSERVIZIO_AUTENTICAZIONE_IP}

    networks:
      fixmi-network:
        ipv4_address: ${MICROSERVIZIO_TASK_IP}
```

Estratto dal file "docker-compose.yaml"

In questo codice, notiamo che le porte vengono impostate sotto la sezione "ports", così come i volumi e il network. Le variabili d'ambiente contenenti i vari ip e le credenziali del database sono contenute nel file ".env" presente nella radice della cartella. La configurazione per gli altri microservizi è simile, con qualche piccola differenza. Qua sotto vengono riportati gli altri microservizi:

```
● ● ●

microservizio-gestione-dipendenti:
  build: ./fixmi-microservizio-gestione-dipendenti
  ports:
    - "3008:3002"
    - "3007:3001"
  volumes:
    - ./fixmi-microservizio-gestione-dipendenti:/app
  restart: always
  environment:
    MICROSERVIZIO_DB_IP: ${MICROSERVIZIO_DB_IP}
    MICROSERVIZIO_DB_USERNAME: ${DB_USERNAME}
    MICROSERVIZIO_DB_PASSWORD: ${DB_PASSWORD}
    MICROSERVIZIO_AUTENTICAZIONE_IP: ${MICROSERVIZIO_AUTENTICAZIONE_IP}
  networks:
    fixmi-network:
      ipv4_address: ${MICROSERVIZIO_GESTIONE_DIPENDENTI_IP}
```

Codice responsabile per l'avvio del microservizio "Gestione Dipendenti"

```
● ● ●

microservizio-autenticazione:
  build: ./fixmi-microservizio-autenticazione
  ports:
    - "3002:3002"
    - "3001:3001"
  volumes:
    - ./fixmi-microservizio-autenticazione:/app
  restart: always
  environment:
    MICROSERVIZIO_DB_IP: ${MICROSERVIZIO_DB_IP}
    MICROSERVIZIO_DB_USERNAME: ${DB_USERNAME}
    MICROSERVIZIO_DB_PASSWORD: ${DB_PASSWORD}
    MICROSERVIZIO_AUTENTICAZIONE_IP: ${MICROSERVIZIO_AUTENTICAZIONE_IP}
  networks:
    fixmi-network:
      ipv4_address: ${MICROSERVIZIO_AUTENTICAZIONE_IP}
```

Codice responsabile per l'avvio del microservizio "Autenticazione"

```
microservizio-home:
  build: ./fixmi-microservizio-home
  ports:
    - "3006:3002"
    - "3005:3001"
  volumes:
    - ./fixmi-microservizio-home:/app
  restart: always
  environment:
    MICROSERVIZIO_DB_IP: ${MICROSERVIZIO_DB_IP}
    MICROSERVIZIO_DB_USERNAME: ${DB_USERNAME}
    MICROSERVIZIO_DB_PASSWORD: ${DB_PASSWORD}
    MICROSERVIZIO_AUTENTICAZIONE_IP: ${MICROSERVIZIO_AUTENTICAZIONE_IP}
  networks:
    fixmi-network:
      ipv4_address: ${MICROSERVIZIO_HOME_IP}
```

Codice responsabile per l'avvio del microservizio "Home"

```
microservizio-database:
  container_name: "fixmidb"
  image: mongo:5.0.8-focal
  restart: always
  ports:
    - 27017:27017
  networks:
    fixmi-network:
      ipv4_address: ${MICROSERVIZIO_DB_IP}
  volumes:
    - ./init_scripts/init-mongo.js:/docker-entrypoint-initdb.d/
      init-mongo.js # Mount init script
    - ./db:/data/db # Data persistence outside docker
  environment:
    MONGO_INITDB_ROOT_USERNAME: ${DB_USERNAME}
    MONGO_INITDB_ROOT_PASSWORD: ${DB_PASSWORD}
  # listen to all ips, enable authentication
  command: ["mongod", "--bind_ip_all", "--auth"]
```

Codice responsabile per l'avvio del database

```
fixmi-reverse-proxy:
  build: ./fixmi-reverse-proxy
  ports:
    - "7777:7777"
  volumes:
    - ./fixmi-reverse-proxy/nginx.conf:/etc/nginx/nginx.conf
  tty: true
  networks:
    fixmi-network:
      ipv4_address: ${REVERSE_PROXY_IP}
```

Codice responsabile per l'avvio del reverse proxy

### 1.3.3 Codice: nginx.conf

Il file *nginx.conf* all'interno della cartella *fixmi-reverse-proxy* imposta la configurazione per il reverse proxy. Questo microservizio si occupa di instradare le richieste ai vari microservizi. Segue un'estratto della configurazione:



```
events {
    worker_connections 1024;
}

http {
    server {
        listen 7777;
        server_name localhost;

        location /home/ {
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
            proxy_pass http://10.5.0.13:3002/;
        }

        location /api/home/ {
            proxy_pass http://10.5.0.13:3001/api/home/;
        }
    }
}
```

Estratto da "fixmi-reverse-proxy/nginx.conf", definizione delle routes "/home" e "/api/home"

La variabile "worker\_connections" definisce il numero di connessioni simultanee che il servizio può gestire. Il blocco "http" e il blocco "server" contengono le configurazioni delle routes, in particolare in questo esempio notiamo che la route "/home/" viene passata a "http://10.5.0.13:3002/" ossia la Front-End del microservizio "home", mentre "/api/home/" rimanda a "http://10.5.0.13:3001/api/home/" ossia la Back-End del microservizio "home". Vengono inoltre impostati alcuni headers dopo la dicitura "proxy\_set\_header".

Di seguito vengono riportate le altre routes:

```
● ● ●

location /tasks/ {
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
    proxy_pass http://10.5.0.12:3002/;
}

location /api/tasks/ {
    proxy_pass http://10.5.0.12:3001/api/tasks/;
}
```

Routes "/tasks" e "/api/tasks"

```
● ● ●

location /dipendenti/ {
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
    proxy_pass http://10.5.0.14:3002/;
}

location /api/dipendenti/ {
    proxy_pass http://10.5.0.14:3001/api/dipendenti/;
}
```

Routes "/dipendenti" e "/api/dipendenti"

```

● ○ ●

location /auth/ {
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
    proxy_pass http://10.5.0.11:3002/;
}

location /api/auth/ {
    proxy_pass http://10.5.0.11:3001/api/auth/;
}

```

Routes "/auth" e "/api/auth"

### Codice: init-mongo.js

Il file contiene l'inizializzazione delle entrate del database in formato json. Sono stati individuati due database: "Tasks" e "Users" e le rispettive collezioni "tasks" e "users" (da notare la differenza della prima lettera da maiuscola a minuscola). I contenuti sono i seguenti:

```

● ○ ●

db_tasks = db.getSiblingDB('Tasks');
db_tasks.createCollection("tasks");

db_tasks.tasks.insertMany([
{
    taskid: ObjectId(),
    name: 'Task1',
    description: 'Description1',
    taskTag: 'Riparazione',
    taskStatus: 'Da Eseguire',
    nomeRichiedente: 'Mario',
    cognomeRichiedente: 'Rossi',
    emailRichiedente: 'email@email.com',
    phoneNumber: '3334445566',
    assignedTo: '663159d0ec263e41dc9d9f96'
},
...

```

Inizializzazione database "Task"



```

db_users = db.getSiblingDB('Users');
db_users.createCollection("users");

db_users.users.insertMany([
{
  id: ObjectId(),
  email: "test@test.com",
  password_hash:
"9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08",
//spoiler: it's 'test'
  permissionLevel: "Manager",
  nome: "Pietro",
  cognome: "Smusi",
  dataDiNascita: new Date("1980-01-01"),
  giornoDiAssunzione: Date(),
  workTags: ["Negozio", "Riparazione", "Magazzino", "Assistenza",
"Feedback"],
},
...

```

Inizializzazione database "Users"

## 1.4 Parti comuni ad ogni microservizio

In questa sezione verrà descritta la struttura che accomuna i vari microservizi.

### 1.4.1 Scelta del Linguaggio

Il linguaggio scelto per la Back-End e Front-End è *TypeScript*: un linguaggio tipizzato che viene tradotto da un compilatore ("transpiler") in Javascript e dunque può essere eseguito in un web browser. Qualsiasi script JavaScript è anche codice TypeScript valido, questo permette all'applicazione di utilizzare l'ampia collezione di pacchetti JavaScript.

### 1.4.2 Frameworks e Dipendenze

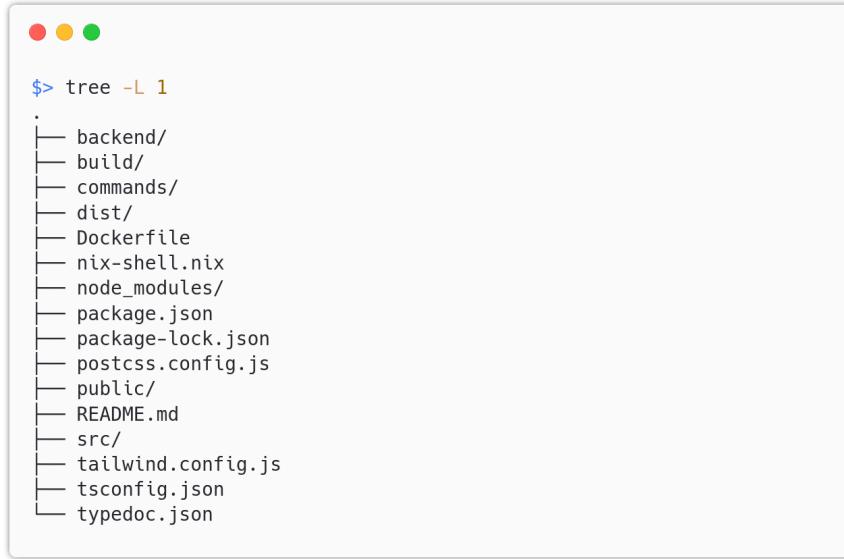
Ogni microservizio utilizza:

- *ts-node* v10.9.2: ambiente runtime Typescript open source e multipiattaforma per la creazione del server di Back-End.
- *eslint* v8.56.0: analizzatore di codice statico.

- *nodemon* v3.0.3: tool di assistenza per Node.js. permette il riavvio automatico del server quando viene rilevata una modifica nel codice.
- *express* v4.18.2: framework web per Node.js, utilizzato per la realizzazione di api.
- *React* v18.3.0: libreria UI per javascript, utilizzata per la Front-End.
- *tailwindcss* v3.4.1: framework CSS.
- *Jest* v29.7.0 : framework per il testing
- *supertest* v7.0.0: framework per il testing di api, utilizzato insieme a Jest.
- *concurrently* v8.2.2: esegue più comandi contemporaneamente, utilizzato per eseguire e Back-End insieme.
- *serve* v14.2.3: hosta un sito statico, utilizzato per hostare la Front-End in produzione.
- *dotenv* v16.4.5: permette l'accesso alle variabili di ambiente, utilizzate per la configurazione dei microservizi.
- *body-parser* v1.20.2: permette di leggere il body di una richiesta http.
- *mongodb* v6.5.0: interfaccia per la connessione al microservizio database.

### 1.4.3 Struttura

La cartella di un microservizio presenta i seguenti files:



```
$> tree -L 1
.
├── backend/
├── build/
├── commands/
├── dist/
├── Dockerfile
├── nix-shell.nix
├── node_modules/
├── package.json
├── package-lock.json
├── postcss.config.js
├── public/
├── README.md
├── src/
└── tailwind.config.js
    └── tsconfig.json
        └── typedoc.json
```

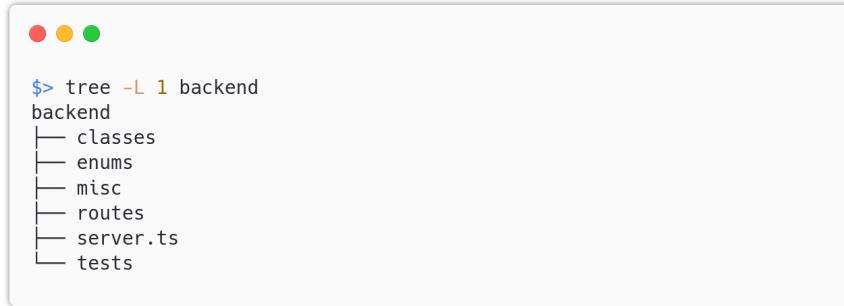
Output del comando "tree" all'interno di un microservizio.

Segue una descrizione degli stessi, dall'alto verso il basso:

- "backend": cartella contenente tutto il codice di Back-End in TypeScript.
- "build": cartella contenente l'applicazione web statica generata da React.
- "commands": cartella contenente degli appunti degli sviluppatori durante la creazione del progetto.
- "dist": cartella contenente il codice Back-End per la produzione, generato dal compilatore di TypeScript.
- "Dockerfile": file contenente le istruzioni per la generazione del contenitore di Docker.
- "nix-shell.nix": file contenente l'ambiente di sviluppo per chi sviluppa su NixOS.
- "node\_modules/": cartella contenente i moduli utilizzati dall'applicazione
- "package.json": file che definisce le dipendenze e le impostazioni di vari moduli.
- "package-lock.json": contiene le versioni delle dipendenze dei moduli.

- "postcss.config.js": script di configurazione necessario per tailwindcss
- "public": contiene file di accesso pubblico per la Front-End, come favicon.ico, robots.txt, index.html.
- "README.md": contiene le informazioni del progetto e le istruzioni per il deploy.
- "src/": cartella contenente tutto il codice per la frontend in tsx.
- "tailwind.config.js": file di configurazione per il funzionamento di tailwind CSS
- "tsconfig.json": configurazione per TypeScript.

#### Directory backend/



```
$> tree -L 1 backend
backend
├── classes
├── enums
├── misc
├── routes
└── server.ts
└── tests
```

Output del comando "tree" all'interno della cartella "backend/".

La struttura della directory "backend/" segue il seguente formato:

- server.ts: è il file principale. Si occupa di istanziare l'App di express, connettersi al database e definire le route delle API.
- classes/ : directory contenente le classi relative ai profili, che vengono manipolate ed immagazzinate nel database.
- enums/ : directory contenente gli enum necessari.
- routes/ : directory contenente gli endpoint delle API del sistema.
- tests/ : directory contenente i test delle API

- `utils/` : directory contenente strutture dati e funzioni utili al funzionamento del sistema.

#### 1.4.4 Comandi npm

Vengono definiti i seguenti scripts per l'esecuzione dell'applicazione tramite npm. Ogni comando può essere eseguito digitando "npm run" ed il nome del comando.

- "startfront": esegue solo la Front-End, si aggiorna automaticamente ad ogni modifica del codice
- "buildfront": genera la Front-End statica per la production
- "startback": esegue soltanto il server di Back-End, si aggiorna automaticamente ad ogni modifica del codice
- "start": esegue sia la Front-End sia la Back-End in developement mode
- "production": esegue sia la Front-End sia la Back-End in production mode
- "lint": esegue un'analisi statica del codice
- "test": esegue gli script di testing del codice.

#### 1.4.5 Elementi Front-End

Ogni microservizio condivide due componenti grafici, "navbar.tsx" e "footer.tsx" posizionati sempre rispettivamente in cima e in fondo ad ogni pagina dell'applicazione. Il loro scopo è rendere la navigazione per l'utente più semplice e chiara

##### Navbar



Figure 1.1: Immagine del componente "navbar.tsx"

Contiene bottoni per accedere ai diversi microservizi del sito, ha uno stile ed un design semplice ed intuitivo in linea con il resto dell'applicazione.

```

● ● ●
import { useState } from 'react';
import './style.css';

export default function Navbar() {
  const [isMenuOpen, setIsMenuOpen] = useState(false);

  const toggleMenu = () => {
    setIsMenuOpen(!isMenuOpen);
  };

  return (
    <nav className="bg-gradient-to-r from-blue-500 to-purple-600 shadow-lg sticky top-0 z-50">
      <div className="max-w-screen-xl flex flex-wrap items-center justify-between mx-auto p-4">
        <a href="/" className="flex items-center space-x-3 rtl:space-x-reverse">
          <img src={require('../assets/fixml-logo.png')} className="h-8" alt="FixML" />
        </a>
        <button
          onClick={toggleMenu}
          type="button"
          className="inline-flex items-center p-2 w-10 h-10 justify-center text-sm text-gray-500 rounded-lg md:hidden hover:bg-gray-100
focus:outline-none focus:ring-2 focus:ring-gray-200 dark:text-gray-400 dark:hover:bg-gray-700 dark:focus:ring-gray-600"
          aria-controls="navbar-default"
          aria-expanded={isMenuOpen ? "true" : "false"}>
          <span className="sr-only">Open main menu</span>
          <svg className="w-5 h-5" aria-hidden="true" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 17 14">
            <path stroke="currentColor" strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M1 1h15M1 7h15M1 13h15" />
          </svg>
        </button>
        <div className="w-full md:flex md:w-auto ${isMenuOpen ? '' : 'hidden'}" id="navbar-default">
          <ul className="font-large flex flex-col items-center md:items-start p-4 md:p-0 mt-4 border border-gray-100 rounded-lg md:flex-row
md:space-x-8 md:mt-0 md:border-0">
            <li>
              <a href="/riparazione" className="block py-2 px-3 rounded dark:hover:bg-gray-700 dark:hover:text-white bg-blue-200 text-blue-
800">
                Riparazione
              </a>
            </li>
            <li>
              <a href="/assistenza" className="block py-2 px-3 rounded dark:hover:bg-gray-700 dark:hover:text-white bg-green-200 text-green-
800">
                Assistenza
              </a>
            </li>
            <li>
              <a href="#" className="block py-2 px-3 rounded dark:hover:bg-gray-700 dark:hover:text-white bg-red-200 text-red-800">
                Negozio
              </a>
            </li>
            <li>
              <a href="/tasks" className="block py-2 px-3 rounded dark:hover:bg-gray-700 dark:hover:text-white bg-purple-200 text-purple-
800">
                Tasks
              </a>
            </li>
            <li>
              <a href="/auth" className="block py-2 px-3 rounded dark:hover:bg-gray-700 dark:hover:text-white bg-yellow-200 text-yellow-
800">
                Profilo
              </a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  );
}

```

Figure 1.2: Codice del componente "navbar.tsx"

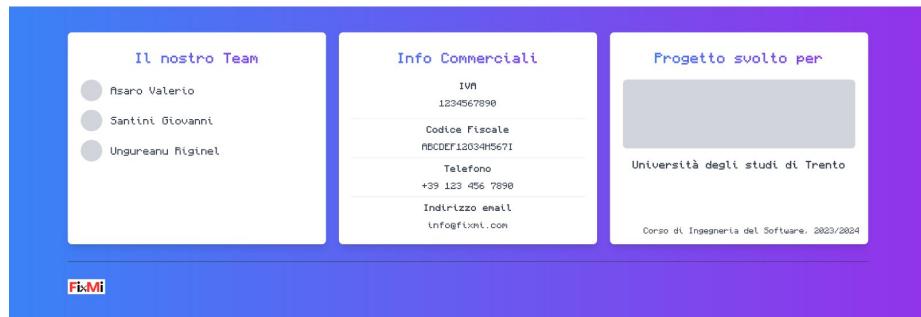
**Footer**

Figure 1.3: Immagine del componente "footer.tsx"

Contiene le principali informazioni sugli autori dell'applicazione "FixMi", dell'azienda e contatti di cui l'utente potrebbe eventualmente avere bisogno

```

● ● ●

import '../style.css'

export default function Footer() {
  return (
    <footer className="bg-gradient-to-r from-blue-500 to-purple-600 dark:bg-gray-900">
      <div className="container px-6 py-12 mx-auto">
        <div className="grid grid-cols-1 gap-6 sm:grid-cols-3">
          <div className="bg-white p-6 rounded-lg shadow-lg">
            <h5 className="text-center text-2xl font-semibold bg-white text-transparent bg-clip-text bg-gradient-to-r from-blue-500 to-purple-600 p-2 rounded-t-lg">
              Il nostro Team
            </h5>
            <div className="pt-4">
              <ul className="text-gray-700">
                <li className="flex items-center mb-4">
                  <div className="w-10 h-10 rounded-full bg-gray-300 mr-4"></div>
                  <span className="text-lg font-medium">Asaro Valerio</span>
                </li>
                <li className="flex items-center mb-4">
                  <div className="w-10 h-10 rounded-full bg-gray-300 mr-4"></div>
                  <span className="text-lg font-medium">Santini Giovanni</span>
                </li>
                <li className="flex items-center mb-4">
                  <div className="w-10 h-10 rounded-full bg-gray-300 mr-4"></div>
                  <span className="text-lg font-medium">Ungureanu Reginel</span>
                </li>
              </ul>
            </div>
          </div>
          <div className="bg-white p-6 rounded-lg shadow-lg">
            <h5 className="text-center text-2xl font-semibold bg-white text-transparent bg-clip-text bg-gradient-to-r from-blue-500 to-purple-600 p-2 rounded-t-lg">
              Info Commerciali
            </h5>
            <div className="pt-4">
              <div className="text-gray-700">
                <h5 className="text-center font-semibold mb-2">IVA</h5>
                <p className="text-center mb-4">1234567890</p>
                <hr className="border-gray-300 my-2" />
                <h5 className="text-center font-semibold mb-2">Codice Fiscale</h5>
                <p className="text-center">ABCDEF1234H567I</p>
                <hr className="border-gray-300 my-2" />
                <h5 className="text-center font-semibold mb-2">Telefono</h5>
                <p className="text-center">+39 123 456 7890</p>
                <hr className="border-gray-300 my-2" />
                <h5 className="text-center font-semibold mb-2">Indirizzo email</h5>
                <p className="text-center">info@fixmi.com</p>
              </div>
            </div>
          </div>
          <div className="bg-white p-6 rounded-lg shadow-lg relative">
            <h5 className="text-center text-2xl font-semibold bg-white text-transparent bg-clip-text bg-gradient-to-r from-blue-500 to-purple-600 p-2 rounded-t-lg">
              Progetto svolto per
            </h5>
            <div className="pt-4">
              <div className="w-full h-32 rounded-lg bg-gray-300 mb-4"></div>
              <div className="text-center text-gray-700 font-semibold text-lg mb-2">
                Università degli studi di Trento
              </div>
              <div className="text-gray-700 text-sm absolute bottom-4 right-4">
                Corso di Ingegneria del Software, 2023/2024
              </div>
            </div>
          </div>
          <hr className="my-6 border-gray-200 md:my-8 dark:border-gray-700" />
          <div className="flex items-center justify-between">
            <a href="#">
              <img className="w-auto h-7" src={require("../assets/fixmi-logo.png")} alt="FixMi Logo" />
            </a>
          </div>
        </div>
      </div>
    </footer>
  );
}

```

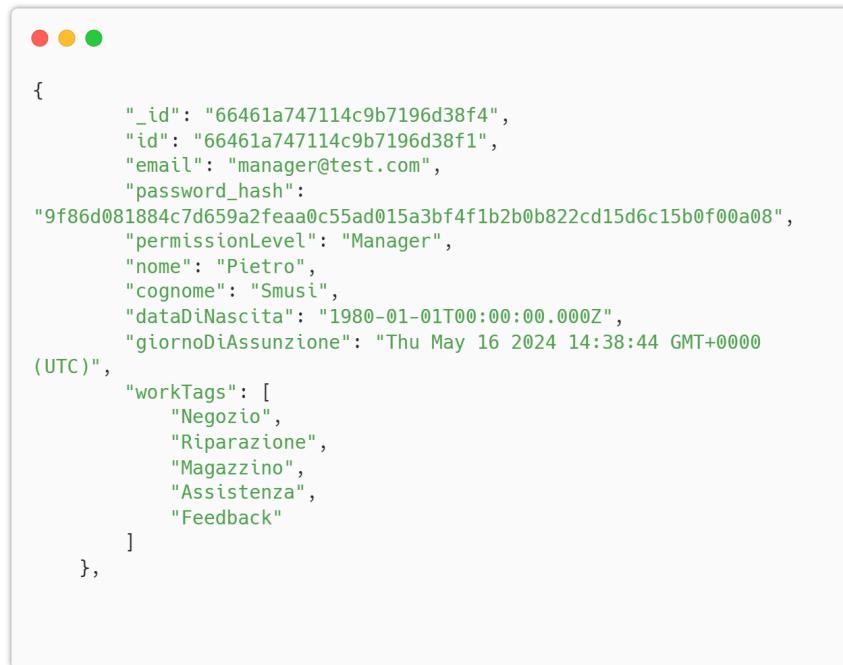
Figure 1.4: Codice del componente "footer.tsx"

## 1.5 Modellazione dati nel database

Sono presenti due database MongoDB con i quali i microservizi interagiscono:

### Users

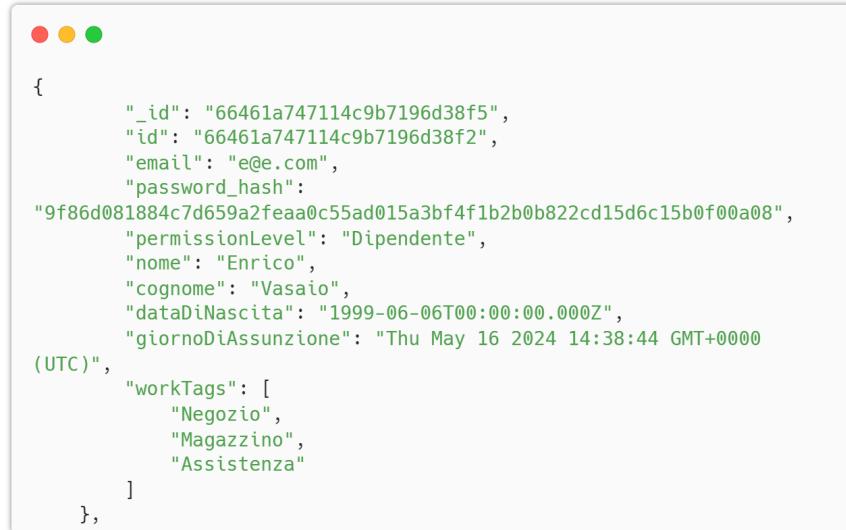
Il database Users contiene i profili dei clienti, dipendenti e manager



The screenshot shows a MongoDB document representing a manager profile. The document is displayed in a light gray box with three colored dots (red, yellow, green) at the top left. The JSON code is as follows:

```
{  
    "_id": "66461a747114c9b7196d38f4",  
    "id": "66461a747114c9b7196d38f1",  
    "email": "manager@test.com",  
    "password_hash":  
        "9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08",  
    "permissionLevel": "Manager",  
    "nome": "Pietro",  
    "cognome": "Smusi",  
    "dataDiNascita": "1980-01-01T00:00:00.000Z",  
    "giornoDiAssunzione": "Thu May 16 2024 14:38:44 GMT+0000  
        (UTC)",  
    "workTags": [  
        "Negozio",  
        "Riparazione",  
        "Magazzino",  
        "Assistenza",  
        "Feedback"  
    ]  
},
```

Figure 1.5: Profilo manager nel database



```
{  
    "_id": "66461a747114c9b7196d38f5",  
    "id": "66461a747114c9b7196d38f2",  
    "email": "e@e.com",  
    "password_hash":  
        "9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08",  
    "permissionLevel": "Dipendente",  
    "nome": "Enrico",  
    "cognome": "Vasaio",  
    "dataDiNascita": "1999-06-06T00:00:00.000Z",  
    "giornoDiAssunzione": "Thu May 16 2024 14:38:44 GMT+0000  
    (UTC)",  
    "workTags": [  
        "Negozio",  
        "Magazzino",  
        "Assistenza"  
    ]  
},
```

Figure 1.6: Profilo dipendente nel database

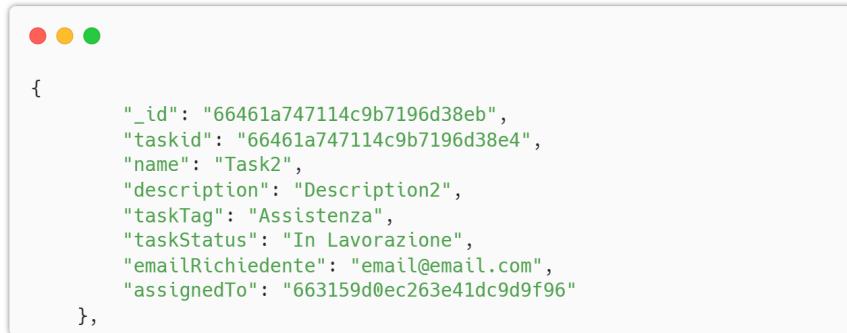


```
{  
    "_id": "66461a747114c9b7196d38f6",  
    "id": "66461a747114c9b7196d38f3",  
    "email": "cliente@hotmail.com",  
    "password_hash":  
        "9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08",  
    "permissionLevel": "Cliente"  
},
```

Figure 1.7: Profilo cliente nel database

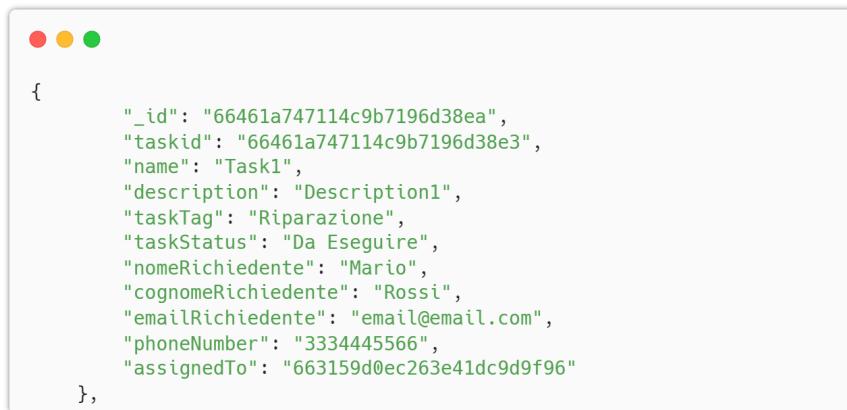
## Tasks

Il database Tasks contiene le task



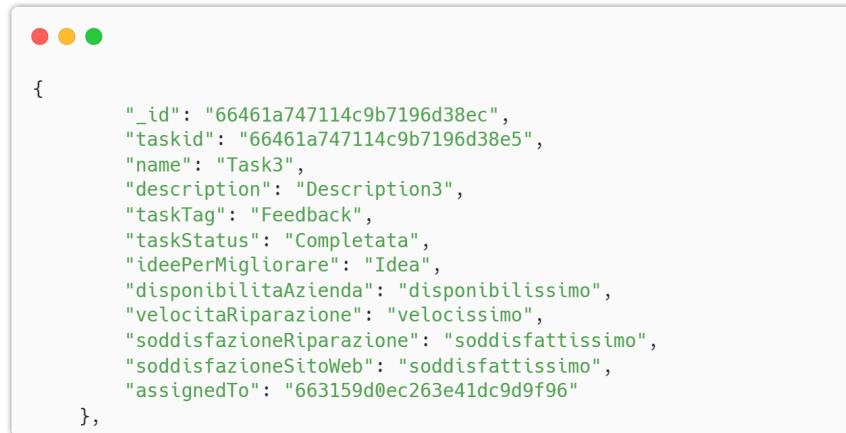
```
{  
    "_id": "66461a747114c9b7196d38eb",  
    "taskid": "66461a747114c9b7196d38e4",  
    "name": "Task2",  
    "description": "Description2",  
    "taskTag": "Assistenza",  
    "taskStatus": "In Lavorazione",  
    "emailRichiedente": "email@email.com",  
    "assignedTo": "663159d0ec263e41dc9d9f96"  
},
```

Figure 1.8: Task Assistenza nel database



```
{  
    "_id": "66461a747114c9b7196d38ea",  
    "taskid": "66461a747114c9b7196d38e3",  
    "name": "Task1",  
    "description": "Description1",  
    "taskTag": "Riparazione",  
    "taskStatus": "Da Eseguire",  
    "nomeRichiedente": "Mario",  
    "cognomeRichiedente": "Rossi",  
    "emailRichiedente": "email@email.com",  
    "phoneNumber": "3334445566",  
    "assignedTo": "663159d0ec263e41dc9d9f96"  
},
```

Figure 1.9: Task Riparazione nel database



A screenshot of a MongoDB database interface showing a single document. The document is represented by a JSON object with the following fields:

```
{  
    "_id": "66461a747114c9b7196d38ec",  
    "taskid": "66461a747114c9b7196d38e5",  
    "name": "Task3",  
    "description": "Description3",  
    "taskTag": "Feedback",  
    "taskStatus": "Completata",  
    "ideePerMigliorare": "Idea",  
    "disponibilitaAzienda": "disponibilissimo",  
    "velocitaRiparazione": "velocissimo",  
    "soddisfazioneRiparazione": "soddisfattissimo",  
    "soddisfazioneSitoWeb": "soddisfattissimo",  
    "assignedTo": "663159d0ec263e41dc9d9f96"  
},
```

Figure 1.10: Task Feedback nel database



A screenshot of a MongoDB database interface showing a single document. The document is represented by a JSON object with the following fields:

```
{  
    "_id": "66461a747114c9b7196d38ee",  
    "taskid": "66461a747114c9b7196d38e7",  
    "name": "Task5",  
    "description": "Description5",  
    "taskTag": "Magazzino",  
    "taskStatus": "Completata",  
    "assignedTo": "663159d0ec263e41dc9d9f96"  
},
```

Figure 1.11: Task Magazzino nel database

---

CHAPTER

**TWO**

---

MICROSERVIZIO AUTENTICAZIONE

## 2.1 Back-End

Questa sezione comprende lo sviluppo e la documentazione della Back-End del Microservizio "Autenticazione"

### 2.1.1 Struttura della Back-End

La struttura della Back-End del microservizio autenticazione è riportata nella figura sotto.

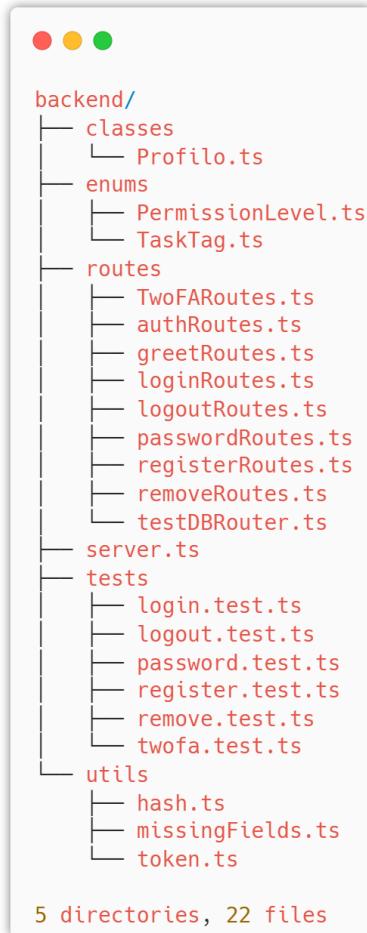


Figure 2.1: Output del comando "tree" all'interno di un microservizio.

### 2.1.2 Specifica delle risorse

Di seguito le risorse estratte dal diagramma delle classi e utilizzate da questo microservizio.

**«resource» Cliente** Rappresenta il profilo di un Cliente. Ha i seguenti attributi

- id
- email

- password\_hash
- permissionLevel
- token

**«resource» Dipendente** Rappresenta il profilo di un Dipendente. Ha tutti gli attributi di cliente, in più:

- nome
- cognome
- data di assunzione
- data di nascita
- worktags

**Manager** Rappresenta il profilo del Manager. Ha tutti gli attributi di Dipendente, ma ha come permissionLevel "Manager"

Metodi

**«resource» login** Permette a un utente già esistente di accedere all'applicazione. I parametri in ingresso sono email, password e codice 2FA. E' un'API del Back-End.

**«resource» logout** Permette a un utente che ha già eseguito l'accesso di svolgere il logout dall'applicazione. Richiede in ingresso il token di sessione ricevuto al login.

**«resource» twofa** Permette a un utente che vuole eseguire l'accesso, registrarsi per la prima volta, cambiare la password o rimuovere il proprio account di ricevere attraverso email un codice 2FA. Richiede in ingresso la propria email.

**«resource» registrazione** Permette a un nuovo cliente di registrarsi all'applicazione. Richiede in ingresso l'email, la password e il codice 2FA

**«resource» cambioPassword** Permette a un utente già esistente di cambiare la propria password. Richiede in ingresso l'email, la nuova password e il codice 2FA.

**«resource» `eliminaProfilo`** Permette a un utente già esistente di eliminare il proprio profilo. Richiede in ingresso l'email, la password e il codice 2FA.

**«resource» `autenticaUtente`** Permette a un altro microservizio di autenticare un utente che desidera utilizzare una funzionalità che richiede autorizzazione. Richiede in ingresso il token dell'utente.

### 2.1.3 Login

#### Specifiche

Questa API, all'indirizzo `/api/auth/login`, ha un metodo POST e viene utilizzata per far accedere utenti all'applicazione. La request body è in formato x-www-form-urlencoded e deve contenere i seguenti campi:

- email
- password
- twofa

La response body è in formato application/json. Di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{text: "successfully logged in", token} cookie: token	Login avvenuto con successo
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
404 NOT FOUND	{error: "user not found"}	non esiste utente con la mail fornita
404 NOT FOUND	{error: "wrong password" }	la password inserita non è corretta
400 BAD REQUEST	{error: "2fa not correct" }	Il codice twofa inserito non è corretto.

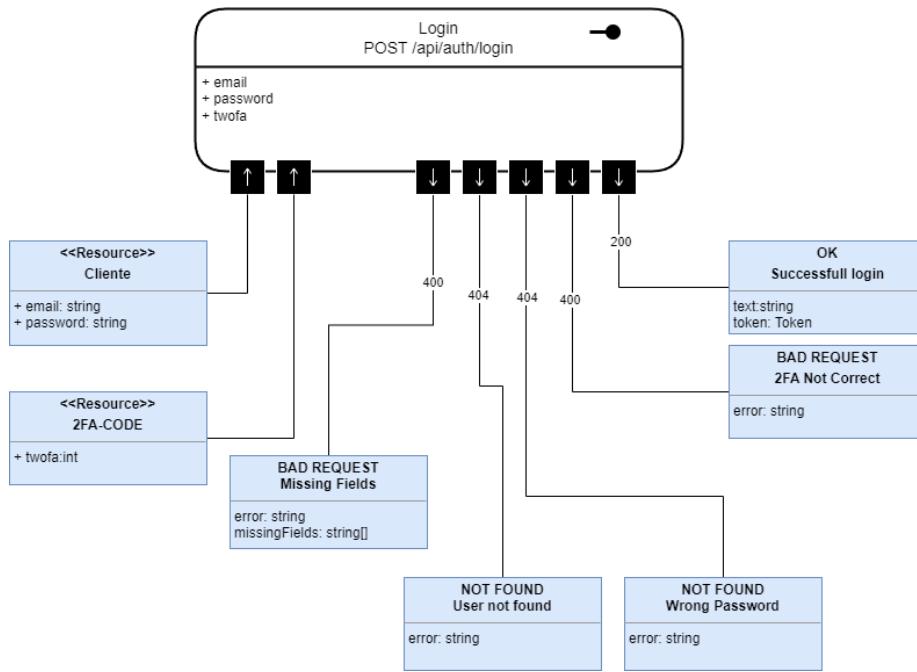


Figure 2.2: Diagramma dell'endpoint "login"

### Specifiche OpenAPI

Di seguito la specifica openapi dell'endpoint login contenuta nel file 'openapi.yaml' della directory del microservizio

```

/login:
  post:
    summary: Login method for user authentication
    description: This endpoint allows users to log in by providing email, password, and tw
    requestBody:
      required: true
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:

```

```
email:
  type: string
  format: email

password:
  type: string
  format: password

twofa:
  type: string
  format: twofa-code

required:
  - email
  - password
  - twofa

responses:
  '200':
    description: Successfully logged in
    content:
      application/json:
        schema:
          type: object
          properties:
            text:
              type: string
              example: "successfully logged in!"
            token:
              type: string
              example: "12415343463452"

headers:
  Set-Cookie:
    schema:
      type: string
    description: Session token
```

```
'400':  
    description: Bad Request - Missing fields or 2fa not correct  
    content:  
        application/json:  
            schema:  
                oneOf:  
                    - $ref: '#/components/schemas/missingFieldsSchema'  
                    - $ref: '#/components/schemas/wrongTwoFaSchema'  
  
'404':  
    description: User not found or wrong password  
    content:  
        application/json:  
            schema:  
                oneOf:  
                    - $ref: '#/components/schemas/notFoundSchema'  
                    - $ref: '#/components/schemas/wrongPassSchema'
```

### Codice

Questa API viene utilizzata da un utente per eseguire l'accesso. Una volta ricevuta la richiesta, viene controllata la presenza di tutti i campi richiesti. Successivamente si controlla se l'utente esiste o meno nel database, e successivamente la validità della password. Dopo aver controllato che il codice 2FA sia corretto, viene generato un token, aggiunto alla sessione e inviato sia nel body sia come cookie nella risposta.

```
● ● ●

loginRouter.post('/', async (req, res) => {
  let [email, password, twofa] = [req.body.email, req.body.password, req.body.twofa];
  let missingFields= getMissingFields([["email",email],["password",password],["twofa",twofa]]);

  //missing fields: returns an error
  if(missingFields.length!=0){
    res.status(400);
    res.json({error: "missing fields", missingFields: missingFields})
    return;
  }
  let user = await db.collection("users").findOne({email: email}) ;
  if (user == null){
    res.status(404);
    res.json({error: "user not found"});
    return;
  }
  let hash = hash_pass(password);
  if (hash != user.password_hash ){
    res.status(404);
    res.json({error: "wrong password"})
    return;
  }
  if (twofa != "12345"){
    res.status(400);
    res.json({error:"2fa not correct"});
  }
  let token = generate_token();
  let b = tokenSession.insert(token,new ObjectId(user._id));
  res.cookie('token', token);

  // Return a json
  res.json({text: "Successfully logged in!",token: token});
});
```

Codice dell'API api/auth/login

## Testing

Per questo endpoint sono stati creati i seguenti test:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Login con successo	{email, password, twofa}	email e password già esistenti nel sistema, twofa corretto	DB	200	200
2	Mancanza campi	{ }			400	400
3	Utente non esiste	{ email, password, twofa }	L'account non esiste	DB	404	404
4	Password non valida	{ email, password, twofa }	email già esistente nel sistema	DB	404	404

```

● ● ●

it('should successfully login', async() => {
  const res = await request("localhost:3001")
    .post('/api/auth/login')
    .type('form')

  .send({email:"manager@test.com",password:"test",twofa:"12345"});
  expect(res.statusCode).toEqual(200);
})

```

Figure 2.3: Test Login 1

```
● ● ● ● ●  
it("should return an error, because we didn't include fields",  
async() => {  
    const res = await request("localhost:3001")  
        .post('/api/auth/login')  
        .type('form')  
        .send({});  
    expect(res.statusCode).toEqual(400);  
})
```

Figure 2.4: Test Login 2

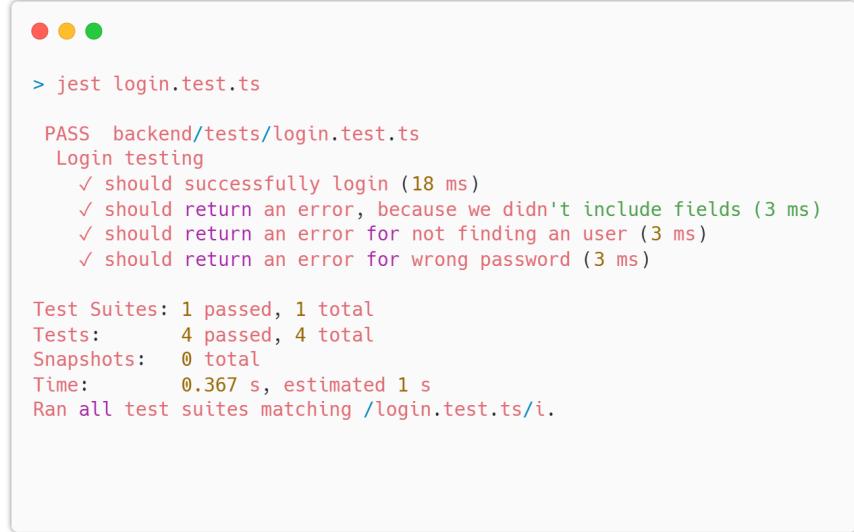
```
● ● ● ● ●  
it('should return an error for not finding an user', async() => {  
    const res = await request("localhost:3001")  
        .post('/api/auth/login')  
        .type('form')  
        .send({email:"pippo",password:"test",twofa:"12345"});  
    expect(res.statusCode).toEqual(404);  
})
```

Figure 2.5: Test Login 3

```
● ● ● ● ●  
it('should return an error for wrong password', async() => {  
    const res = await request("localhost:3001")  
        .post('/api/auth/login')  
        .type('form')  
  
        .send({email:"manager@test.com",password:"test1",twofa:"12345"});  
    expect(res.statusCode).toEqual(404);  
})
```

Figure 2.6: Test Login 4

I risultati dei test sono i seguenti:



```

> jest login.test.ts

PASS  backend/tests/login.test.ts
  Login testing
    ✓ should successfully login (18 ms)
    ✓ should return an error, because we didn't include fields (3 ms)
    ✓ should return an error for not finding an user (3 ms)
    ✓ should return an error for wrong password (3 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.367 s, estimated 1 s
Ran all test suites matching /login.test.ts/i.

```

Figure 2.7: Risultati test

### 2.1.4 Logout

#### Specifica

Questa API, all'indirizzo/api/auth/logout, ha un metodo DELETE e viene utilizzata da un utente autenticato per svolgere il logout. La request body è in formato x-www-form-urlencoded e deve contenere il token dell'utente. La response body è in formato application/json. di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{text: "successfully logged out"}	Logout avvenuto con successo
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
404 NOT FOUND	{error: "user not found with the given token"}	non esiste utente nella sessione con il token fornito.
404 NOT FOUND	{error: "wrong password" }	la password inserita non è corretta

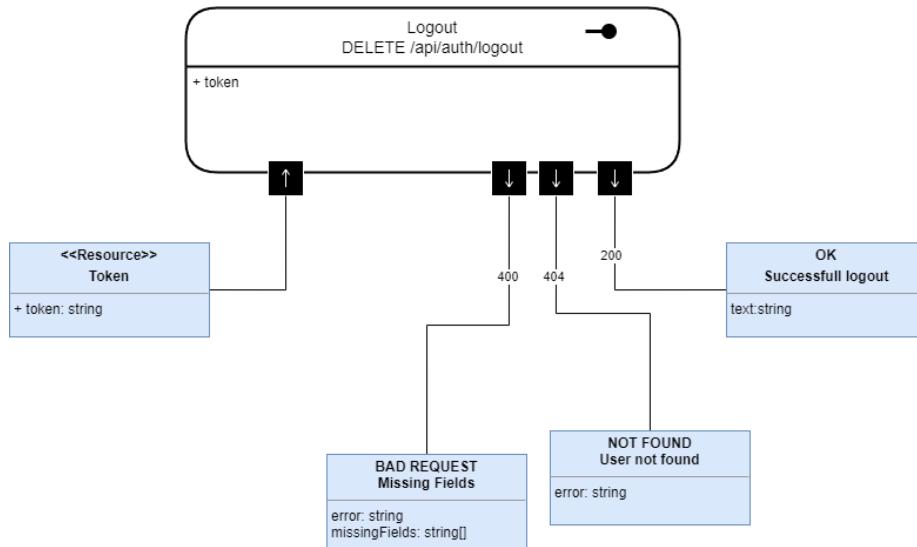


Figure 2.8: Diagramma dell'endpoint "logout"

### Specifica OpenAPI

Di seguito la specifica openapi dell'endpoint logout contenuta nel file 'openapi.yaml' della directory del microservizio

```

/logout:
  delete:
    summary: Method for logging out
    description: This endpoint allows users who have already logged in to log out by providing their token.
    requestBody:
      required: true
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:
              token:
                type: string
                required:

```

```
        - token

responses:
  '200':
    description: Succesfully logged out
    content:
      application/json:
        schema:
          type: object
          properties:
            text:
              type: string
              example: "Succesfully logged out!"
  '400':
    description: Bad Request - Missing fields
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/missingFieldsSchema'
  '404':
    description: User not found with the given token
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/notFoundSchema'
```

### Codice

Questa API viene utilizzata da un utente già autenticato per eseguire il logout. Una volta ricevuta la richiesta, viene controllata la presenza di tutti i campi richiesti. Successivamente si controlla se il token fornito è presente nella sessione. A operazione completata, viene rimosso il token dalla sessione.

```
logoutRouter.delete('/', async (req, res) => {
  let token = req.body.token;
  /*
  let missingFields: any = {}
  if (token == null){
    missingFields.token = "UNSPECIFIED"
  }

  */
  let missingFields= getMissingFields([["token",token]]);

  //missing fields: returns an error
  if(missingFields.length!=0) {
    res.status(400);
    res.json({error: "missing fields", missingFields: missingFields});
    return;
  }
  let id = tokenSession.get(token);
  if (id == undefined){
    res.status(404);
    res.json({error: "user not found with the given token"});
    return;
  }
  tokenSession.remove(token);

  res.json({text: "Successfully logged out"});
})
```

Codice dell'API api/auth/logout

## Testing

Per questo endpoint sono stati creati i seguenti test:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Logout con successo	{token}	Token ottenuto dal login		200	200
2	Mancanza campi	{ }			400	400
3	Token non presente nella session	{ token}	Token non presente nella sessione		404	404



```

● ● ●

it('should successfully logout', async() => {
  const resLogin = await request
    .post('/api/auth/login')
    .type('form')

  .send({email:"manager@test.com",password:"test",twofa:"12345"});
  let token = resLogin.body.token;

  const res = await request
    .delete('/api/auth/logout')
    .type('form')
    .send({token: token})
  expect(res.statusCode).toEqual(200);
});

```

Figure 2.9: Test Logout 1

```
it("should return an error, because we didn't include fields",  
  async() => {  
    const res = await request  
      .delete('/api/auth/logout')  
      .type('form')  
      .send({});  
  
    expect(res.statusCode).toEqual(400);  
  });
```

Figure 2.10: Test Logout 2

```
it('should return an error for getting the token wrong', async() =>  
{  
  
  const resLogin = await request  
    .post('/api/auth/login')  
    .type('form')  
  
  .send({email:"manager@test.com",password:"test",twofa:"12345"});  
  let token = resLogin.body.token;  
  
  const res = await request  
    .delete('/api/auth/logout')  
    .type('form')  
    .send({token: token+1})  
  expect(res.statusCode).toEqual(404);  
});
```

Figure 2.11: Test Logout 3

I risultati dei test sono i seguenti:



```
jest logout.test.ts

PASS  backend/tests/logout.test.ts
Logout testing
  ✓ should successfully logout (22 ms)
  ✓ should return an error, because we didn't include fields (2 ms)
  ✓ should return an error for getting the token wrong (6 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.369 s, estimated 1 s
Ran all test suites matching /logout.test.ts/i.
```

Figure 2.12: Risultati test

### 2.1.5 Registrazione

#### Specifiche

Questa API, all'indirizzo /api/auth/register, ha un metodo POST e viene utilizzata per la creazione di un account cliente. La request body è in formato x-www-form-urlencoded e deve contenere i seguenti campi:

- email
- password
- twofa

La response body è in formato application/json. Di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{text: "successfully registered", token} cookie: token	Login avvenuto con successo
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
404 NOT FOUND	{error: "user already exists with the given email"}	Esiste già un utente con la mail fornita
400 BAD REQUEST	{error: "2fa not correct" }	Il codice twofa inserito non è corretto.

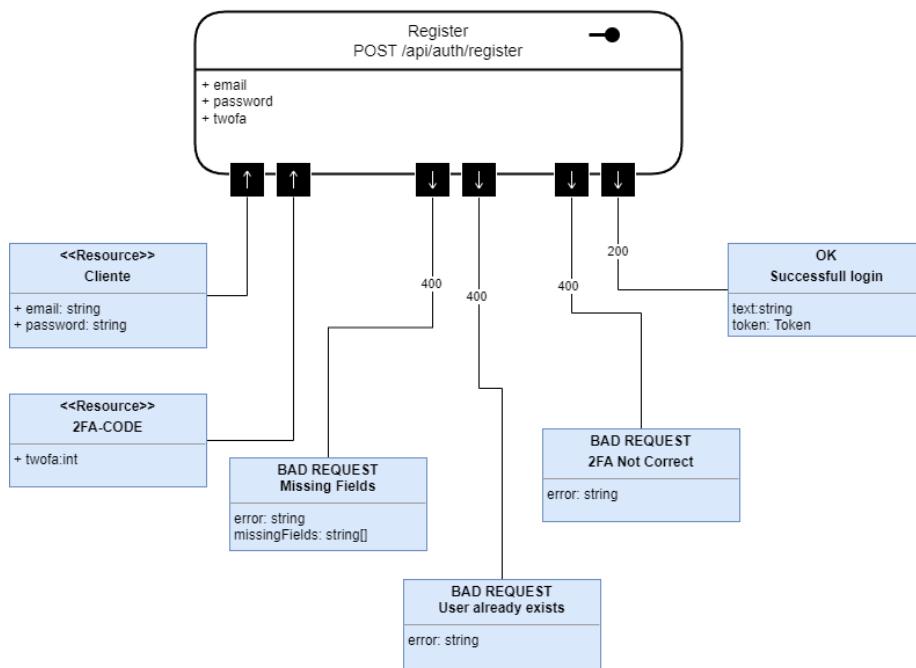


Figure 2.13: Diagramma dell'endpoint "register"

### Specifiche OpenAPI

Di seguito la specifica openapi dell'endpoint register contenuta nel file 'openapi.yaml' della directory del microservizio

```
/register:  
  post:  
    summary: Registering method for new users  
    description: This endpoint allows new users to register by providing email, password,  
    requestBody:  
      required: true  
      content:  
        application/x-www-form-urlencoded:  
          schema:  
            type: object  
            properties:  
              email:  
                type: string  
                format: email  
  
              password:  
                type: string  
                format: password  
              twofa:  
                type: string  
                format: twofa-code  
            required:  
              - email  
              - password  
              - twofa  
  responses:  
    '200':  
      description: Succesfully registered  
      content:  
        application/json:  
          schema:  
            type: object  
            properties:  
              text:  
                type: string  
                example: "Successfully registered"  
              token:
```

```
        type: string
        example: "1231345321532151"

    headers:
        Set-Cookie:
            schema:
                type: string
                description: Session token

    '400':
        description: Bad Request - Missing fields or 2fa not correct or User already exists
        content:
            application/json:
                schema:
                    oneOf:
                        - $ref: '#/components/schemas/missingFieldsSchema'
                        - $ref: '#/components/schemas/wrongTwoFaSchema'
                        - $ref: '#/components/schemas/alreadyExistsSchema'
```

### Codice

Questa API viene utilizzata da un utente per creare un proprio account cliente. Una volta ricevuta la richiesta, viene controllata la presenza di tutti i campi richiesti. Successivamente si controlla se un utente con la stessa email sia già esistente nel database. Dopo aver controllato che il codice 2FA sia corretto, viene inserito il nuovo profilo cliente nel database, viene generato un token, aggiunto alla sessione e inviato sia nel body sia come cookie nella risposta.

```
● ● ●

registerRouter.post('/', async (req, res) => {
  let [email, password, twofa] = [req.body.email, req.body.password, req.body.twofa];
  /*
  let missingFields: any={};
  if (email == null){
    missingFields.email = "UNSPECIFIED";
  }
  if (password == null){
    missingFields.password = "UNSPECIFIED";
  }
  if (twofa == null){
    missingFields.twofa = "UNSPECIFIED";
  }
  */
  let missingFields= getMissingFields([["email",email],["password",password],["twofa",twofa]]);

  //missing fields: returns an error
  if(missingFields.length!=0){
    res.status(400);
    res.json({error: "missing fields", missingFields: missingFields});
    return;
  }
  let user = await db.collection("users").findOne({email: email}) ;
  if (user != null){
    res.status(400);
    res.json({error: "user already exists with given email"});
    return;
  }
  if (twofa != "12345"){
    res.status(400);
    res.json({error:"2fa not correct"});
  }
  let cliente = new Cliente(email,password);
  await db.collection("users").insertOne(cliente);
  let token = generate_token();
  let b = tokenSession.insert(token,new ObjectId(cliente.id));
  res.cookie('token', token);
  // Return a json
  res.json({text: "Successfully registered!",token: token});
});
```

Codice dell'API api/auth/register

## Testing

Per questo endpoint sono stati creati i seguenti test:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Registrato con successo	{email, password, twofa}	email non presente nel sistema, twofa corretto	DB	200	200
2	Mancanza campi	{ }			400	400
3	Utente esistente	{ email, password, twofa}	L'account esiste	DB	400	400

```

● ● ●

it('should successfully register', async() => {

    const resRegister = await request
        .post('/api/auth/register')
        .type('form')

    .send({email:"registertest1@test.com",password:"test",twofa:"12345"});
    expect(resRegister.statusCode).toEqual(200);
    const resLogin = await request
        .post('/api/auth/login')
        .type('form')

    .send({email:"registertest1@test.com",password:"test",twofa:"12345"});
    expect(resLogin.statusCode).toEqual(200);
    //cleanup
    await request.delete('/api/auth/remove')
        .type('form')

    .send({email:"registertest1@test.com",password:"test",twofa:"12345"});
});

});

```

Figure 2.14: Test Register 1

```
it("should return an error, because we didn't include fields", async() => {
  const res = await request
    .post('/api/auth/register')
    .type('form')
    .send({});

  expect(res.statusCode).toEqual(400);
});
```

Figure 2.15: Test Register 2

```
it('should return an error, because user already exists', async() => {
  const resRegister = await request
    .post('/api/auth/register')
    .type('form')

  .send({email:"manager@test.com",password:"test",twofa:"12345"});
  expect(resRegister.statusCode).toEqual(400);

});
```

Figure 2.16: Test Register 3

I risultati dei test sono i seguenti:



```
/app # npm test register.test.ts

> fixmi-microservice-template@1.0.0 test
> jest register.test.ts

PASS  backend/tests/register.test.ts
  Register testing
    ✓ should successfully register (30 ms)
    ✓ should return an error, because we didn't include fields (3 ms)
    ✓ should return an error, because user already exists (5 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.369 s, estimated 1 s
Ran all test suites matching /register.test.ts/i.
/app #
```

Figure 2.17: Risultati test

### 2.1.6 Cambio password

#### Specifica

Questa API, all’indirizzo /api/auth/changepass, ha un metodo PATCH e viene utilizzata per cambiare la password di un utente. La request body è in formato x-www-form-urlencoded e deve contenere i seguenti campi:

- email
- new\_password
- twofa

La response body è in formato application/json. Di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{text: "successfully changed your password", token} cookie: token	Cambio password avvenuto con successo
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
404 NOT FOUND	{error: "user not found"}	non esiste utente con la mail fornita
400 BAD REQUEST	{error: "2fa not correct" }	Il codice twofa inserito non è corretto.

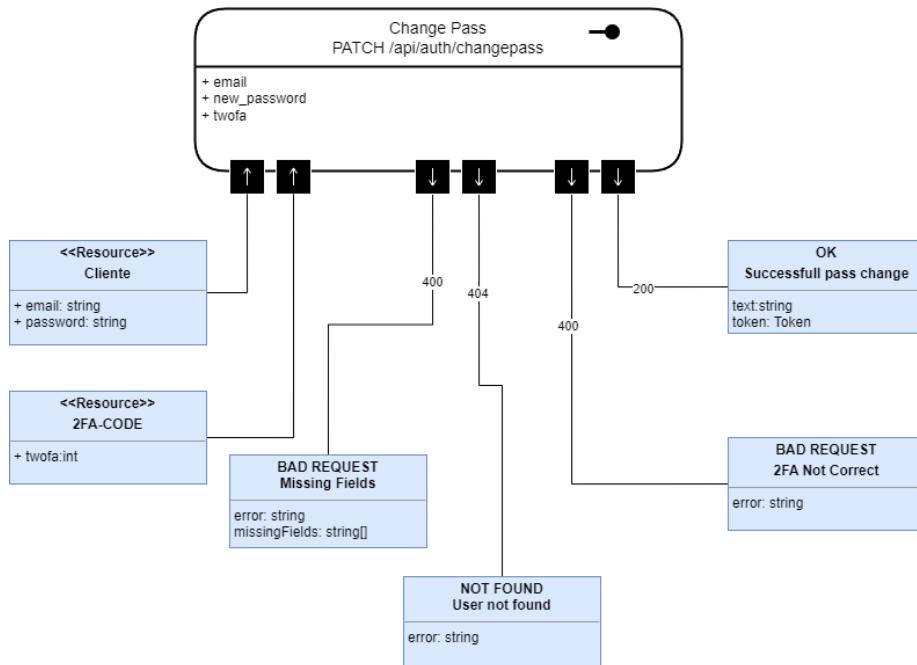


Figure 2.18: Diagramma dell'endpoint "changepass"

### Specifica OpenAPI

Di seguito la specifica openapi dell'endpoint changepass contenuta nel file 'openapi.yaml'

```
/changepass:
  patch:
    summary: Password Changing Method
    description: This endpoint allows existing users to change their passwords by providing their email and a new password. It also supports two-factor authentication (2FA) via a code sent to the user's email.
    requestBody:
      required: true
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:
              email:
                type: string
                format: email
              new_password:
                type: string
                format: password
              twofa:
                type: string
                format: twofa-code
            required:
              - email
              - new_password
              - twofa
    responses:
      '200':
        description: Successfully changed your password
        content:
          application/json:
            schema:
              type: object
              properties:
                text:
```

```
        type: string
        example: "successfully changed your password"
      token:
        type: string
        example: "12312434325321234"
      headers:
        Set-Cookie:
          schema:
            type: string
            description: Session token
    '400':
      description: Bad Request - Missing fields or 2fa not correct
      content:
        application/json:
          schema:
            oneOf:
              - $ref: '#/components/schemas/missingFieldsSchema'
              - $ref: '#/components/schemas/wrongTwoFaSchema'

    '404':
      description: User not found
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/notFoundSchema'
```

### Codice

Questa API viene utilizzata da un utente per cambiare la password del proprio account. Una volta ricevuta la richiesta, viene controllata la presenza di tutti i campi richiesti. Successivamente si controlla se l'utente esista o meno nel database. Dopo aver controllato che il codice 2FA sia corretto, viene modificata la password nel database, viene generato un token, aggiunto alla sessione e inviato sia nel body sia come cookie nella risposta.

```

● ● ●

pwdChangeRouter.patch('/', async (req, res) => {
    let [email, new_password, twofa] = [req.body.email, req.body.new_password, req.body.twofa];

    let missingFields= getMissingFields([["email",email],["new_password",new_password],["twofa",twofa]]);

    //missing fields: returns an error
    if(missingFields.length!=0) {
        res.status(400);
        res.json({error: "missing fields", missingFields: missingFields, your_body: req.body});
        return;
    }
    let user = await db.collection("users").findOne({email: email}) ;
    if(user == null){
        res.status(404);
        res.json({error:"user not found"});
        return;
    }

    if (twofa != "12345"){
        res.status(400);
        res.json({error:"2fa not correct"});
        return;
    }
    await db.collection("users").updateOne({email:email},{$set : {password_hash: hash_pass(new_password)})})
    let token = generate_token();
    let b = tokenSession.insert(token,new ObjectId(user.id));
    res.cookie('token', token);
    // Return a json
    res.json({text: "Successfully changed your password!",token: token});
});
```

Codice dell'API api/auth/changepass

## Testing

Per questo endpoint sono stati creati i seguenti test:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Cambio password con successo	{email, new_password, twofa}	creazione account con richiesta register, twofa corretto	DB	200	200
2	Mancanza campi	{ }			400	400



```
it('should successfully change password', async() => {
    const resRegister = await request
        .post('/api/auth/register')
        .type('form')

    .send({email:"pwdtest@test.com",password:"test",twofa:"12345"});

    const res = await request
        .patch('/api/auth/changepass')
        .type('form')

    .send({email:"pwdtest@test.com",new_password:"test1",twofa:"12345"})
    ;
    expect(res.statusCode).toEqual(200);

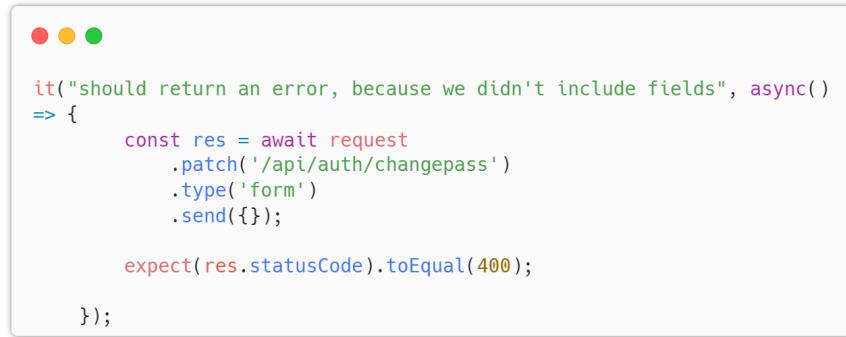
    const resLogin = await request
        .post('/api/auth/login')
        .type('form')

    .send({email:"pwdtest@test.com",password:"test1",twofa:"12345"});
    expect(resLogin.statusCode).toEqual(200);
    //cleanup
    await request.delete('/api/auth/remove')
        .type('form')

    .send({email:"pwdtest@test.com",password:"test1",twofa:"12345"});
    });

});
```

Figure 2.19: Test Pass 1



```

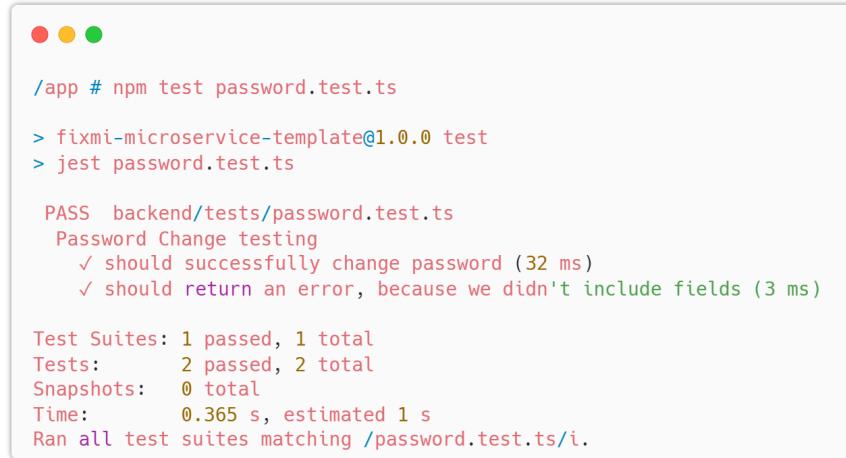
it("should return an error, because we didn't include fields", async() => {
  const res = await request
    .patch('/api/auth/changepass')
    .type('form')
    .send({});

  expect(res.statusCode).toEqual(400);
});

```

Figure 2.20: Test Pass 2

I risultati dei test sono i seguenti:



```

/app # npm test password.test.ts
> fixmi-microservice-template@1.0.0 test
> jest password.test.ts

PASS  backend/tests/password.test.ts
  Password Change testing
    ✓ should successfully change password (32 ms)
    ✓ should return an error, because we didn't include fields (3 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:  0 total
Time:        0.365 s, estimated 1 s
Ran all test suites matching /password.test.ts/i.

```

Figure 2.21: Risultati test

### 2.1.7 Eliminazione Profilo

#### Specifiche

Questa API, all'indirizzo /api/auth/remove, ha un metodo POST e viene utilizzata per rimuovere il profilo di un utente. La request body è in formato x-www-form-urlencoded e deve contenere i seguenti campi:

- email
- password
- twofa

La response body è in formato application/json. Di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{text: "successfully removed your account", token} cookie: token	L'account è stato rimosso con successo
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
404 NOT FOUND	{error: "user not found"}	non esiste utente con la mail fornita
404 NOT FOUND	{error: "wrong password" }	la password inserita non è corretta
400 BAD REQUEST	{error: "2fa not correct" }	Il codice twofa inserito non è corretto.

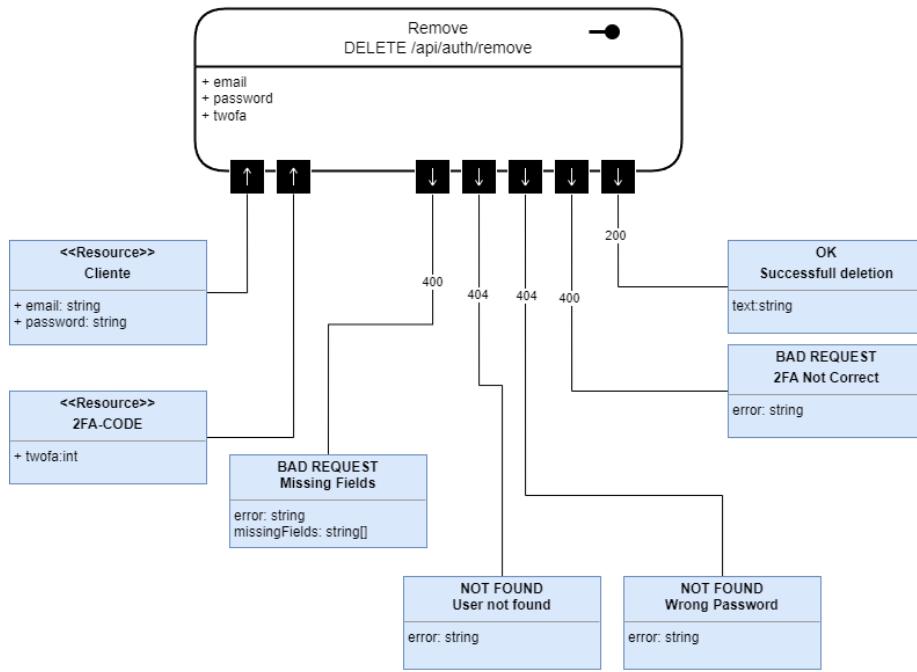


Figure 2.22: Diagramma dell'endpoint "remove"

### Specifiche OpenAPI

Di seguito la specifica openapi dell'endpoint remove contenuta nel file 'openapi.yaml' della directory del microservizio

```

/remove:
  delete:
    summary: Account removal method
    description: This endpoint allows existing users to remove their accounts by providing
    requestBody:
      required: true
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:

```

```
email:
  type: string
  format: email

password:
  type: string
  format: password

twofa:
  type: string
  format: twofa-code

required:
- email
- password
- twofa

responses:
'200':
  description: Succesfully removed the account
  content:
    application/json:
      schema:
        type: object
        properties:
          text:
            type: string
            example: "Successfully removed your account."
'400':
  description: Bad Request - Missing fields or 2fa not correct
  content:
    application/json:
      schema:
        oneOf:
- $ref: '#/components/schemas/missingFieldsSchema'
- $ref: '#/components/schemas/wrongTwoFaSchema'

'404':
  description: User not found or wrong password
  content:
    application/json:
```

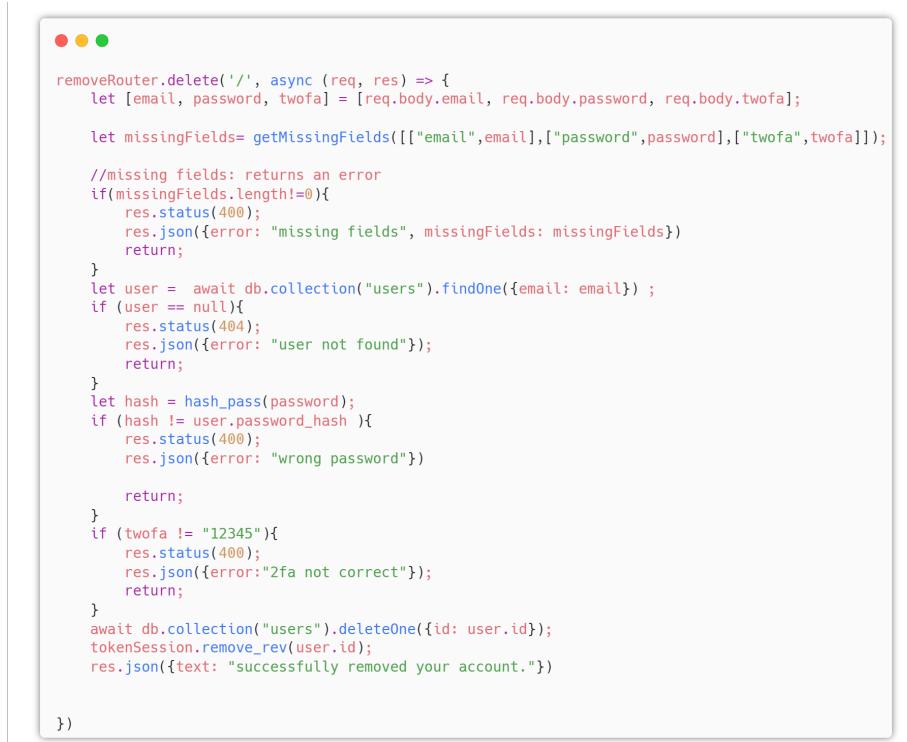
```

schema:
oneOf:
  - $ref: '#/components/schemas/notFoundSchema'
  - $ref: '#/components/schemas/wrongPassSchema'

```

### Codice

Questa API viene utilizzata da un utente per eliminare il proprio profilo. Una volta ricevuta la richiesta, viene controllata la presenza di tutti i campi richiesti. Successivamente si controlla se l'utente esiste o meno nel database, e successivamente la validità della password. Dopo aver controllato che il codice 2FA sia corretto, il profilo viene eliminato dal database e il suo token rimosso dalla sessione.



```

removeRouter.delete('/', async (req, res) => {
  let [email, password, twofa] = [req.body.email, req.body.password, req.body.twofa];

  //missing fields: returns an error
  if(missingFields.length!=0){
    res.status(400);
    res.json({error: "missing fields", missingFields: missingFields})
    return;
  }
  let user = await db.collection("users").findOne({email: email});
  if (user == null){
    res.status(404);
    res.json({error: "user not found"});
    return;
  }
  let hash = hash_pass(password);
  if (hash != user.password_hash ){
    res.status(400);
    res.json({error: "wrong password"})
    return;
  }
  if (twofa != "12345"){
    res.status(400);
    res.json({error:"2fa not correct"});
    return;
  }
  await db.collection("users").deleteOne({id: user.id});
  tokenSession.remove_rev(user.id);
  res.json({text: "successfully removed your account."})
})

```

Codice dell'API api/auth/remove

## Testing

Per questo endpoint sono stati creati i seguenti test:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Eliminazione con successo	{email, password, twofa}	utente già creato con la richiesta register, twofa corretto	DB	200	200
2	Mancanza campi	{ }			400	400
3	Utente non esiste	{ email, password, twofa}	L'account non esiste	DB	404	404

```

● ● ●

it('should successfully remove an account', async() => {
  const resRegister = await request
    .post('/api/auth/register')
    .type('form')

  .send({email:"removetest@test.com",password:"test",twofa:"12345"});

  //cleanup
  const resRemove =await request
    .delete('/api/auth/remove')
    .type('form')

  .send({email:"removetest@test.com",password:"test",twofa:"12345"});
  expect(resRemove.statusCode).toEqual(200);
});

```

Figure 2.23: Test Remove 1

```
it("should return an error, because we didn't include fields", async() => {
  const res = await request
    .delete('/api/auth/remove')
    .type('form')
    .send({});

  expect(res.statusCode).toEqual(400);
});
```

Figure 2.24: Test Remove 2

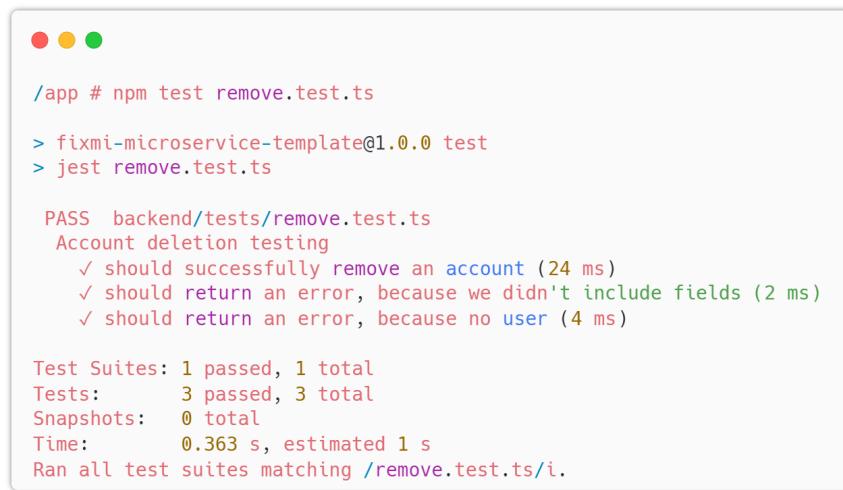
```
it('should return an error, because no user', async() => {
  const resRemove = await request
    .delete('/api/auth/remove')
    .type('form')

  .send({email:"removetest@test.com",password:"test",twofa:"12345"});
  expect(resRemove.statusCode).toEqual(404);

});
```

Figure 2.25: Test Remove 3

I risultati dei test sono i seguenti:



```
/app # npm test remove.test.ts
> fixmi-microservice-template@1.0.0 test
> jest remove.test.ts

PASS  backend/tests/remove.test.ts
  Account deletion testing
    ✓ should successfully remove an account (24 ms)
    ✓ should return an error, because we didn't include fields (2 ms)
    ✓ should return an error, because no user (4 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.363 s, estimated 1 s
Ran all test suites matching /remove.test.ts/i.
```

Figure 2.26: Risultati test

### 2.1.8 Autentifica Token

#### Specifiche

Questa API, all'indirizzo/api/auth/authenticate, ha un metodo POST e viene utilizzata da un altro microservizio per autenticare un utente utilizzando il suo token. La request body è in formato x-www-form-urlencoded e deve contenere il token dell'utente. La response body è in formato application/json. di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{text: "success", user_info: {permission, id} }	L'utente esiste, e ha il seguente id e permission-Level
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
404 NOT FOUND	{error: "user not found with the given token"}	non esiste utente nella sessione con il token fornito.
404 NOT FOUND	{error: "user doesn't exist or is deleted" }	Caso impossibile a meno di bug: il token esiste nella sessione, ma l'id utente non corrisponde ad alcun utente del database.

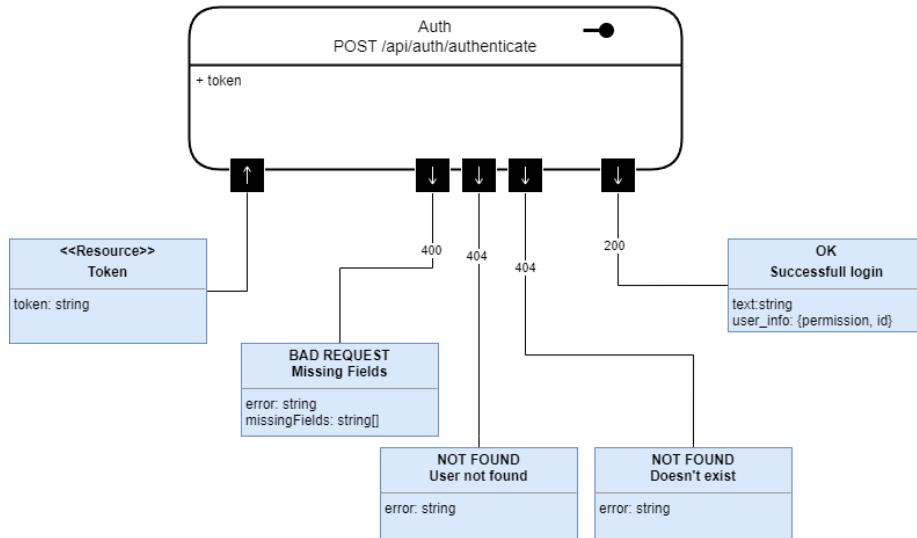


Figure 2.27: Diagramma dell'endpoint "authenticate"

### Specifica OpenAPI

Di seguito la specifica openapi dell'endpoint authenticate contenuta nel file 'openapi.yaml' della directory del microservizio

```
/authenticate:
  post:
    summary: Method for authentication
    description: This endpoint allows other microservices to authenticate a user by providing a token
    requestBody:
      required: true
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:
              token:
                type: string

      required:
        - token
    responses:
      '200':
        description: Success
        content:
          application/json:
            schema:
              type: object
              properties:
                text:
                  type: string
                  example: "success"
                user_info:
                  type: object
                  properties:
                    permission:
                      type: string
                      example: "Manager"
```

```
        id:  
          type: string  
          example: "aa231e3421bd1223"  
  
'400':  
  description: Bad Request - Missing fields  
  content:  
    application/json:  
      schema:  
        $ref: '#/components/schemas/missingFieldsSchema'  
'404':  
  description: User not found with the given token or user doesn't exist  
  content:  
    application/json:  
      schema:  
        oneOf:  
        - $ref: '#/components/schemas/notFoundSchema'  
        - $ref: '#/components/schemas/maybeDeletedSchema'
```

### Codice

Questa API viene utilizzata da un microservizio per autenticare un utente che desidera farne uso. Una volta ricevuta la richiesta, viene controllata la presenza di tutti i campi richiesti. Successivamente si controlla se il token fornito abbia un utente corrispondente nella sessione. Infine il livello di permesso associato all'utente e il suo id vengono inviati nella risposta.

```

● ● ●

authRouter.post('/', async (req,res) => {
    let token = req.body.token;
    let missingFields= getMissingFields([["token",token]]);

    //missing fields: returns an error
    if(missingFields.length!=0) {
        res.status(400);
        res.json({error: "missing fields", missingFields: missingFields})
        return;
    }
    let id = tokenSession.get(token);
    if (id == undefined){
        res.status(404);
        res.json({error: "user not found with the given token"});
        return;
    }
    let user = await db.collection("users").findOne({_id: id});
    if(user == null){
        res.status(404);
        res.json({error: "user doesn't exist or is deleted"});
        return;
    }

    res.json({text: "Success", user_info: {permission: user.permissionLevel,id: user.id}});
}
)

```

Codice dell'API api/auth/authenticate

## Testing

Per questo endpoint sono stati creati i seguenti test

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Autentificazione con successo	{token}	token presente nella sessione	DB	200	200
2	Mancanza campi	{ }			400	400
3	Utente non esiste	{ token}	token non presente nella sessione	DB	404	404

```
it('should successfully authenticate the token', async() => {
    //first login
    const loginRes = await request("localhost:3001")
        .post('/api/auth/login')
        .type('form')

    .send({email:"manager@test.com",password:"test",twofa:"12345"});
    expect(loginRes.statusCode).toEqual(200);
    let token = loginRes.body.token;
    const res = await request("localhost:3001")
        .post('/api/auth/authenticate')
        .type('form')
        .send({token: token})
    expect(res.status).toEqual(200);
})
```

Figure 2.28: Test Auth 1

```
it("should return an error, because we didn't include fields", async() => {
    const res = await request("localhost:3001")
        .post('/api/auth/authenticate')
        .type('form')
        .send({});
    expect(res.statusCode).toEqual(400);
})
```

Figure 2.29: Test Auth 2



```

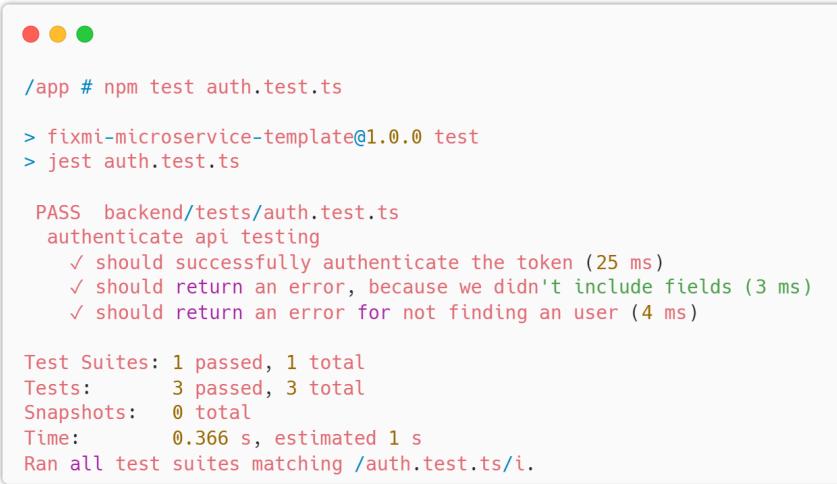
● ● ●

it('should return an error for not finding an user', async() => {
  const res = await request("localhost:3001")
    .post('/api/auth/authenticate')
    .type('form')
    .send({token:'a1234124s221'});
  expect(res.statusCode).toEqual(404);
})

```

Figure 2.30: Test Auth 3

I risultati dei test sono i seguenti:



```

● ● ●

/app # npm test auth.test.ts

> fixmi-microservice-template@1.0.0 test
> jest auth.test.ts

PASS  backend/tests/auth.test.ts
  authenticate api testing
    ✓ should successfully authenticate the token (25 ms)
    ✓ should return an error, because we didn't include fields (3 ms)
    ✓ should return an error for not finding an user (4 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        0.366 s, estimated 1 s
Ran all test suites matching /auth.test.ts/i.

```

Figure 2.31: Risultati test

### 2.1.9 Two Factor Authentication

#### Specifica

Questa API, all'indirizzo/api/auth/twofa, ha un metodo POST e viene utilizzata da un utente per richiedere un codice 2FA attraverso mail. La request body è in formato x-www-form-urlencoded e deve contenere l'indirizzo email dell'utente. La response body è in formato application/json. di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{text: "Successfully sent the 2FA code to your email"}	Il codice è stato inviato alla mail correttamente
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request

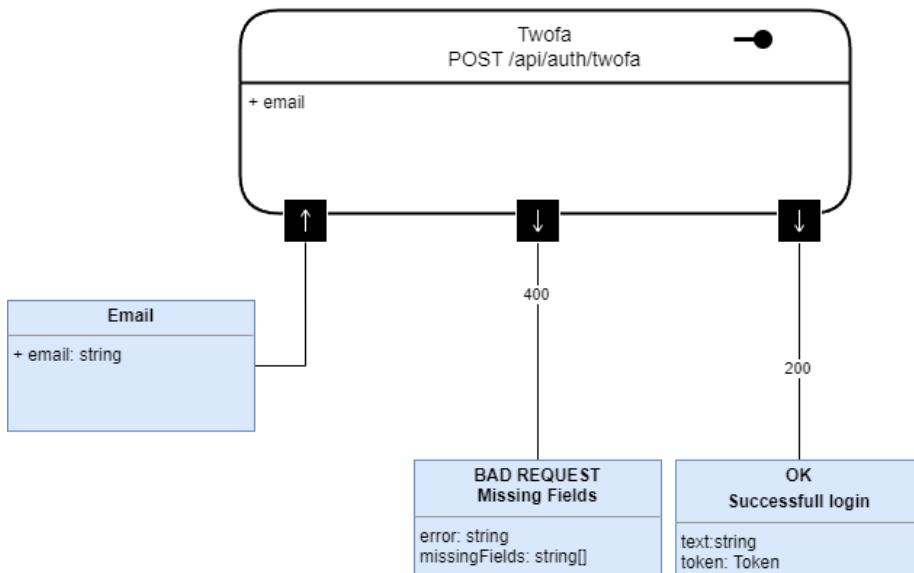


Figure 2.32: Diagramma dell'endpoint "changepass"

### Specifica OpenAPI

Di seguito la specifica openapi dell'endpoint twofa contenuta nel file 'openapi.yaml' della directory del microservizio

```

/twofa:
  post:
    summary: Method for receiving a 2FA code
    description: This endpoint allows users to request a Two-Factor Authentication code by
    requestBody:

```

```
required: true
content:
  application/x-www-form-urlencoded:
    schema:
      type: object
      properties:
        token:
          type: string
          required:
            - token
responses:
  '200':
    description: Succesfully sent the 2FA
    content:
      application/json:
        schema:
          type: object
          properties:
            text:
              type: string
            example: "Successfully sent the 2FA code to your email(SPOILER: IT'S 123"
  '400':
    description: Bad Request - Missing fields
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/missingFieldsSchema'
```

### Codice

Questa API viene utilizzata da un utente per ricevere un codice 2FA attraverso email. Una volta ricevuta la richiesta, viene controllata la presenza di tutti i campi richiesti. Successivamente viene inviata l'email con il codice 2FA.

```

● ● ●

twoFArouter.post('/', (req, res) => {
  let email = req.body.email;
  let missingFields = getMissingFields([["email",email]]);
  if(missingFields.length != 0){
    res.status(400);
    res.json({error: "missing fields", missingFields: missingFields})
    return;
  }
  // Return a json
  res.json({text: "Successfully sent the 2FA code to your email(SPOILER: IT'S 12345)"});
});

```

Codice dell'API api/auth/twofa

## Testing

Per questo endpoint sono stati creati i seguenti test

Numerico Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Twofa inviato con successo	{email}			200	200
2	Mancanza campi	{ }			400	400

```

● ● ●

it('should successfully send twofa', async() => {
  const res = await request("localhost:3001")
    .post('/api/auth/twofa')
    .type('form')
    .send({email:"test@test.com"});
  expect(res.statusCode).toEqual(200);
})

```

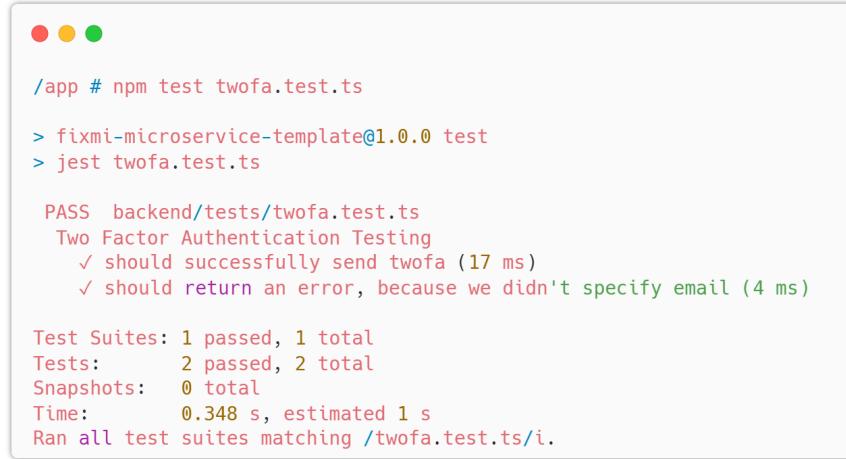
Figure 2.33: TwoFa Testing 1



```
it("should return an error, because we didn't specify email", async() => {
  const res = await request("localhost:3001")
    .post('/api/auth/twofa')
    .type('form')
    .send({});
  expect(res.statusCode).toEqual(400);
})
```

Figure 2.34: TwoFa Testing 2

I risultati dei test sono i seguenti:



```
/app # npm test twofa.test.ts
> fixmi-microservice-template@1.0.0 test
> jest twofa.test.ts

PASS  backend/tests/twofa.test.ts
  Two Factor Authentication Testing
    ✓ should successfully send twofa (17 ms)
    ✓ should return an error, because we didn't specify email (4 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:  0 total
Time:        0.348 s, estimated 1 s
Ran all test suites matching /twofa.test.ts/i.
```

Figure 2.35: Risultati test

## 2.2 Front-End

Questa sezione comprende lo sviluppo e la documentazione del Front-End del Microservizio Autenticazione

### 2.2.1 Struttura del Front-End

La struttura del Front-End del microservizio autenticazione è riportata nella figura sotto.

```
src/
├── assets
│   └── fixmi-logo.png
├── components
│   ├── changepass
│   │   └── changepass.tsx
│   ├── login
│   │   ├── formLogin.tsx
│   │   └── login.tsx
│   ├── profile
│   │   ├── formRemove.tsx
│   │   ├── logout.tsx
│   │   ├── notLoggedIn.tsx
│   │   ├── profile.tsx
│   │   └── remove.tsx
│   ├── register
│   │   ├── formRegister.tsx
│   │   └── register.tsx
│   ├── twofa
│   │   ├── twoFa.tsx
│   │   └── twoFaForm.tsx
│   ├── footer.tsx
│   └── navbar.tsx
└── utils
    ├── connection.ts
    ├── cookie.tsx
    └── statusEnum.ts
App.tsx
index.tsx
style.css

7 directories, 21 files
```

Figure 2.36: Output del comando "tree" all'interno del microservizio autenticazione.

```

● ● ●

import { BrowserRouter, Route, Routes } from 'react-router-dom';
import React from 'react';
import LoginPage from './components/login/login';
import RegisterPage from './components/register/register';
import ChangePassPage from './components/changepass/changepass';
import ProfilePage from './components/profile/profile';
import RemovePage from './components/profile/remove';
import LogoutPage from './components/profile/logout';

function App() {

  return (
    <BrowserRouter basename='auth/'>
      <main>
        <Routes>
          {/* Routing */}
          <Route path="/" element={<ProfilePage />} />
          <Route path="profile" element={<ProfilePage/>} /></Route>
          <Route path="login" element={<LoginPage />} />
          <Route path="register" element={<RegisterPage/>} />
          <Route path="changepass" element={<ChangePassPage/>} />
          <Route path="remove" element={<RemovePage/>} />
          <Route path="logout" element={<LogoutPage/>} />
          <Route path="*" element={<ProfilePage />} />
        </Routes>
      </main>
    </BrowserRouter>
  );
}

export default App;

```

Codice del componente "App.tsx"

### Specifiche

Tutte le componenti facenti parte del microservizio "Autenticazione" vengono gestite dalla componente "App.tsx", quest'ultimo stabilisce gli indirizzi di ciascuna pagina del microservizio e quale componente deve gestire l'indirizzo.

#### 2.2.2 UserFlow

Lo userflow del microservizio autenticazione viene mostrato sotto nel suo intero. Verrà successivamente suddiviso ed analizzato in ciascun suo componente.

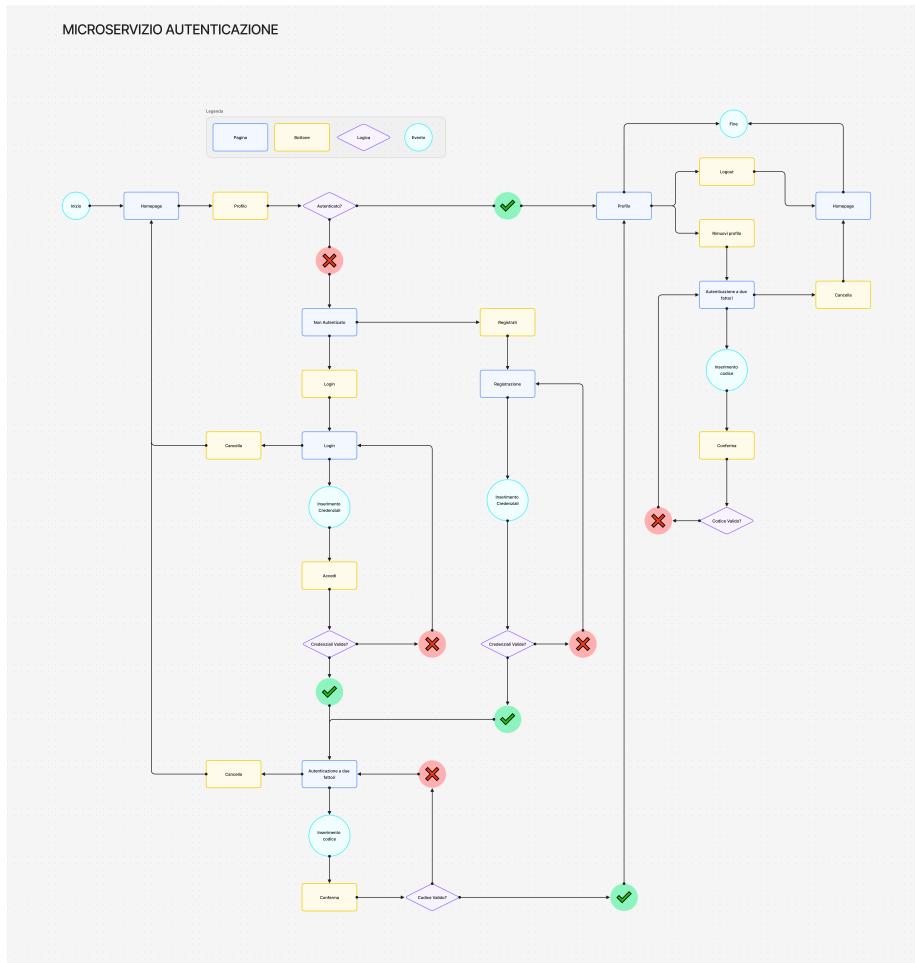


Figure 2.37: UserFlow del microservizio "Autenticazione"

### 2.2.3 Login

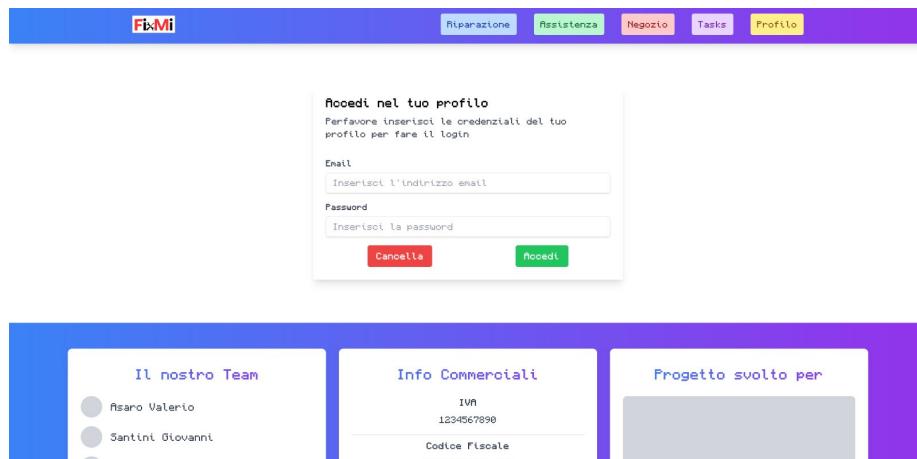


Figure 2.38: Schermata per il login

#### Specifiche

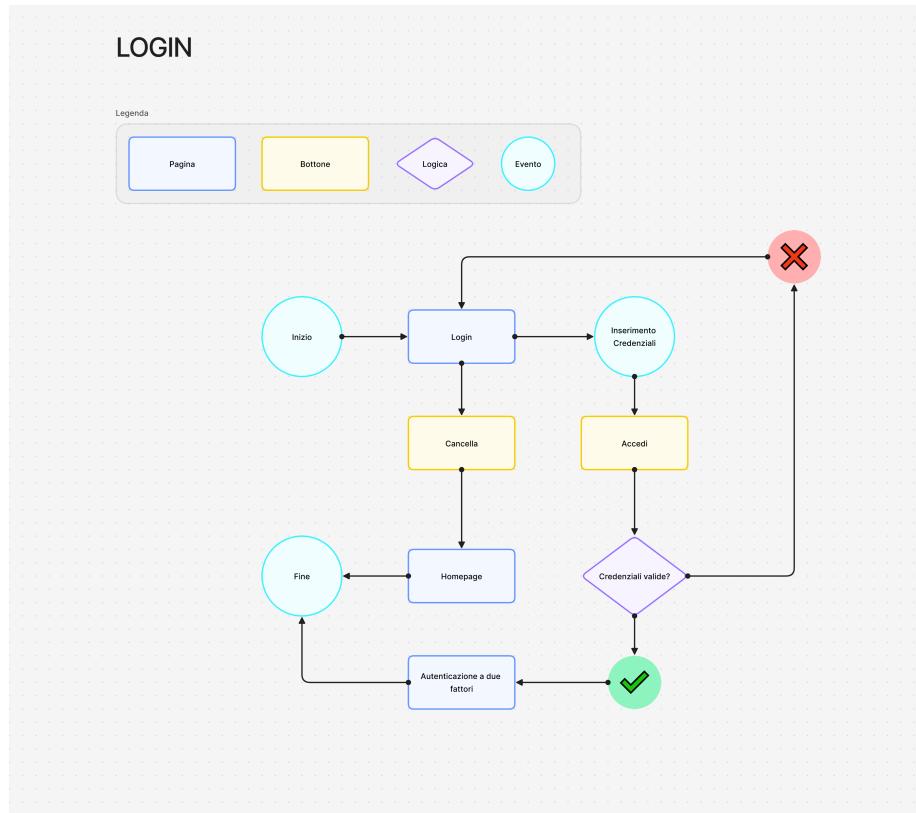
La pagina si occupa di chiedere le credenziali necessarie per effettuare l'accesso nel proprio profilo.

E' possibile tornare indietro cliccando il bottone rosso "Cancella"

Dopo avere inserito le credenziali è possibile continuare cliccando il bottone verde "Accedi"

#### Userflow

Partendo dalla pagina "Homepage" l'utente non autenticato deve prima cliccare il tasto "Profilo" presente nella "navbar", successivamente può inserire le proprie credenziali e passare alla pagina "Two Factor Authentication"



Userflow della pagina "Login"

### Codice

La pagina "Login" è composta da due componenti chiamati "login.tsx" e "formLogin.tsx" con le seguenti caratteristiche:

- login.tsx
  - Contiene al suo interno "formLogin.tsx"
  - Contiene controlli e "Fallbacks" per eventuali errori
- formLogin.tsx
  - Contiene al suo interno un modulo che deve essere compilato dall'utente inserendo l'email e la password.
  - Contiene controlli sulle credenziali inserite

```

● ● ●
import React from 'react';

function FormLogin({ onSubmit }) {
  return (
    <div className="flex justify-center items-center h-full pt-20 pb-20">
      <div className="w-3/4 lg:w-3/4 xl:w-2/4">
        <div className="max-w-xl mx-auto rounded overflow-hidden shadow-lg">
          <div className="px-6 py-4">
            <p className="font-bold text-xl mb-2">Accedi nel tuo profilo</p>
            <p className="text-gray-700 text-base">
              Per favore inserisci le credenziali del tuo profilo per fare il login
            </p>
          </div>
          <div className="px-6 pt-4 pb-2">
            <form onSubmit={onSubmit}>
              <div className="mb-4">
                <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="email">
                  Email
                </label>
                <input
                  className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none
focus:shadow-outline"
                  type="email"
                  name="email"
                  placeholder="Inserisci l'indirizzo email"
                />
              </div>
              <div className="mb-4">
                <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="password">
                  Password
                </label>
                <input
                  className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none
focus:shadow-outline"
                  name="password"
                  type="password"
                  placeholder="Inserisci la password"
                />
              </div>
              <div className="mb-4 justify-around flex">
                <button
                  className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
                  type="button"
                  onClick={() => {
                    document.getElementsByName("email")[0].value = "";
                    document.getElementsByName("password")[0].value = "";
                  }}
                >
                  Cancella
                </button>
                <button
                  className="my_button bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded focus:outline-none
focus:shadow-outline"
                  type="submit"
                >
                  Accedi
                </button>
              </div>
            </form>
          </div>
        </div>
      </div>
    );
}

export default FormLogin;

```

Codice del componente "formLogin.tsx"

- Spazi vuoti
- Numero elevato di caratteri

non causa problemi di alcun tipo e vengono gestiti correttamente.

### 2.2.4 Registrazione



Schermata per la registrazione

#### Specifiche

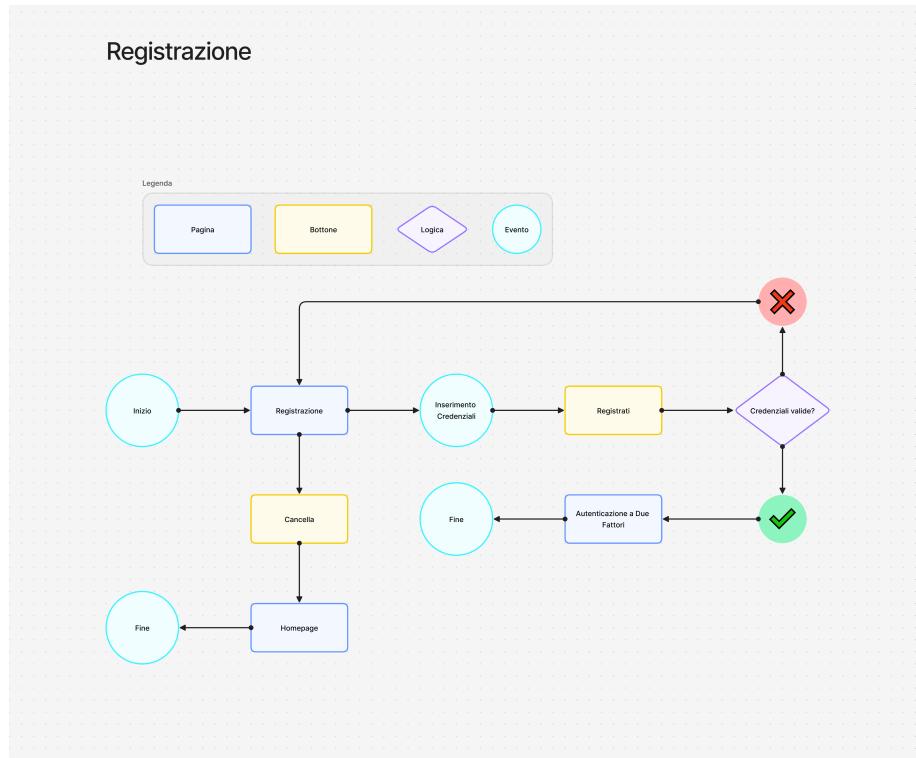
La pagina si occupa di chiedere le credenziali necessarie per la creazione di un nuovo profilo per l'utente.

E' possibile tornare indietro cliccando il bottone rosso "Cancella"

Dopo avere inserito le credenziali è possibile continuare cliccando il bottone verde "Registrati"

#### Userflow

Partendo dalla pagina "Homepage" l'utente non autenticato deve prima cliccare il tasto "Profilo" presente nella "navbar", successivamente può inserire le proprie credenziali e passare alla pagina "Two Factor Authentication"



Userflow della pagina "Registrazione"

### Codice

La pagina "Registrazione" è composta da due componenti chiamati "register.tsx" e "formRegister.tsx" con le seguenti caratteristiche:

- register.tsx
  - Contiene al suo interno "formRegister.tsx"
  - Contiene controlli e "Fallbacks" per eventuali errori
- formRegister.tsx
  - Contiene al suo interno un modulo che deve essere compilato dall'utente inserendo l'email, la password ed una ripetizione della password.
  - Contiene controlli sulle credenziali inserite

```

import React from 'react';

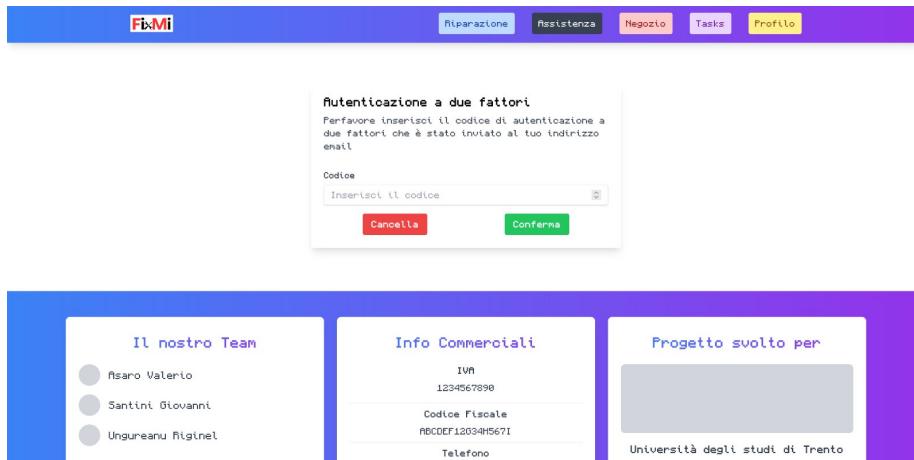
function FormRegister({ onSubmit, samePass }) {
  return (
    <div className="flex justify-center items-center h-full pt-20 pb-20">
      <div className="w-3/4 lg:w-3/4 xl:w-2/4">
        <div className="max-w-xl mx-auto rounded overflow-hidden shadow-lg">
          <div className="px-6 py-4">
            <p><b>Crea un nuovo profilo</b></p>
            <p>Perfavore inserisci le credenziali del tuo nuovo profilo per registrarti</p>
          </div>
          <div className="px-6 pt-4 pb-2">
            <form onSubmit={onSubmit}>
              <div className="mb-4">
                <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="email">
                  Email
                </label>
                <input
                  className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
                  type="email"
                  name="email"
                  placeholder="Inserisci l'indirizzo email"
                  required
                />
              </div>
              <div className="mb-4">
                <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="password">
                  Password
                </label>
                <input
                  className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
                  name="password"
                  type="password"
                  placeholder="Inserisci la password"
                  required
                />
              </div>
              <div className="mb-4">
                <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="repeat_password">
                  Ripeti Password
                </label>
                <input
                  className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline"
                  name="repeat_password"
                  type="password"
                  placeholder="Ripeti la password"
                  required
                />
              </div>
              <div className="mb-4 justify-around flex">
                <button
                  className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
                  type="button"
                  onClick={() => {
                    document.getElementsByName("email")[0].value = "";
                    document.getElementsByName("password")[0].value = "";
                    document.getElementsByName("repeat_password")[0].value = "";
                  }}
                >
                  Cancella
                </button>
                <button
                  className="my_button bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
                  type="submit"
                >
                  Registrati
                </button>
              </div>
              <p className="text-red-500 text-xs italic">
                {samePass ? "" : "Le password non coincidono"}
              </p>
            </form>
          </div>
        </div>
      </div>
    );
}

export default FormRegister;

```

Codice del componente "formRegister.tsx"

### 2.2.5 Two Factor Authentication



Schermata per il "Two Factor Authentication"

#### Specifica

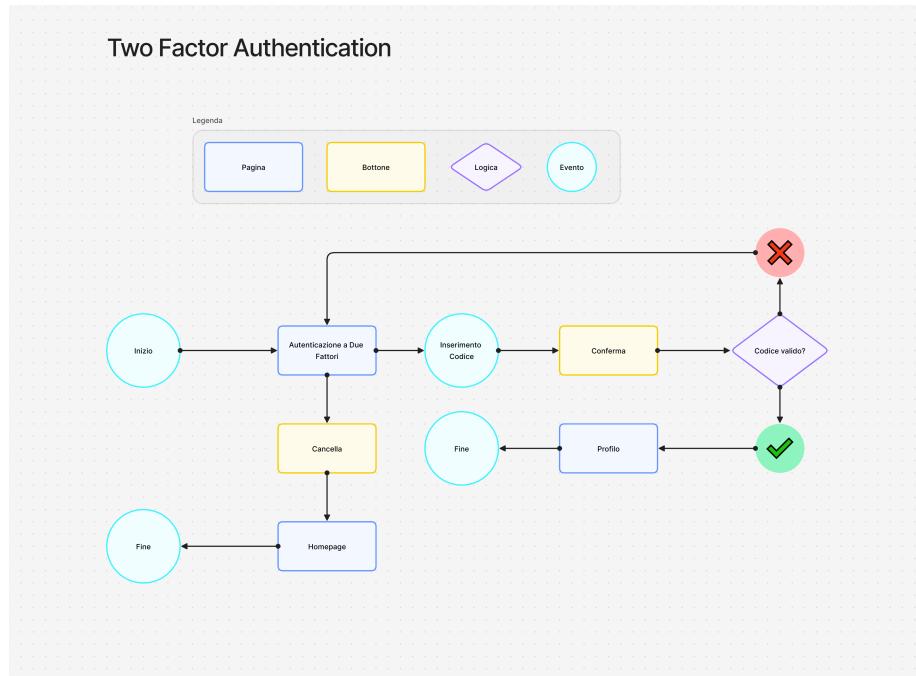
La pagina si occupa di chiedere il codice di autenticazione a due fattori che è stato mandato all'utente.

E' possibile tornare indietro cliccando il bottone rosso "Cancella"

Dopo avere inserito il codice è possibile continuare cliccando il bottone verde "Conferma"

#### Userflow

Partendo dalle pagine "Login", "Registrazione" o "Elimina Profilo" l'utente ha la possibilità di tornare indietro cliccando il tasto "Cancella", oppure completare il modulo inserendo il codice ricevuto e successivamente cliccando "Conferma". Se il codice è valido l'utente viene poi riportato nella pagina "Homepage".



Userflow della pagina "Two Factor Authentication"

### Codice

La pagina "Two Factor Authentication" è composta da due componenti chiamati "twoFa.tsx" e "formTwoFa.tsx" con le seguenti caratteristiche:

- twoFa.tsx
  - Contiene al suo interno "formTwoFa.tsx"
  - Contiene controlli e "Fallbacks" per eventuali errori

```
import React from "react";
import TwoFaForm from "./twoFaForm";
import { ErrorBoundary } from 'react-error-boundary'
function TwoFa({email, setTwofa, setStatus}){
  return (
    <ErrorBoundary FallbackComponent={Fallback}>
      <TwoFaForm email= {email} setTwofa={setTwofa} setStatus={setStatus}/>
    </ErrorBoundary>
  )
}

function Fallback({error}){
  return (
    <>
      <h2>Something went Wrong! {error.data}</h2>
      <a href="home">return to Home</a>
    </>
  )
}

export default TwoFa;
```

Codice del componente "twoFa.tsx"

- formTwoFa.tsx
    - Contiene al suo interno un modulo che deve essere compilato dall'utente inserendo il codice di autenticazione a due fattori
    - Contiene controlli sul codice inserito dall'utente

```

● ● ●

import React, { useEffect, useState } from "react";
import { twoFaRequest } from "./utils/connection";
import Status from ".././utils/statusEnum";
import { useErrorBoundary } from "react-error-boundary";

function TwoFaForm({ email, setTwofa, setStatus }) {
  const [res, setRes] = useState({});
  const { showBoundary } = useErrorBoundary();

  useEffect(() => {
    const fetchData = async () => {
      await fetch(twoFaRequest(email))
        .then(
          response => {
            setRes(response.body);
          },
          error => {
            showBoundary(error);
          }
        )
    };
    fetchData();
  }, [email, showBoundary]);

  async function handleSubmit(e) {
    e.preventDefault();
    const twofa = e.target.elements.code.value;
    setTwofa(twofa);
    setStatus(Status.TwoFaSubmitted);
  }

  return (
    <div className="flex justify-center items-center h-full pt-20 pb-20">
      <div className="w-3/4 lg:w-2/4">
        <div className="px-6 py-4 w-full rounded shadow-lg">
          <div className="px-4 mb-4">
            <div className="font-bold text-xl mb-2">Autenticazione a due fattori</div>
            <p className="text-gray-700 text-base">
              Per favore inserisci il codice di autenticazione a due fattori che è stato inviato al tuo indirizzo email
            </p>
          </div>
          <div className="px-6 pt-4 pb-2">
            <form onSubmit={handleSubmit}>
              <div className="mb-4">
                <label className="block text-gray-700 text-sm font-bold mb-2" htmlFor="code">
                  Codice
                </label>
                <input
                  className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none"
                  type="number"
                  name="code"
                  placeholder="Inserisci il codice"
                  required
                />
              </div>
              <div className="mb-4 justify-around flex">
                <button
                  className="bg-red-500 hover:bg-red-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
                  type="button"
                >
                  Cancella
                </button>
                <button
                  className="my_button bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline"
                  type="submit"
                >
                  Conferma
                </button>
              </div>
            </form>
            <p className="text-gray-700 text-xs italic">{res.text}</p>
          </div>
        </div>
      </div>
    );
}

export default TwoFaForm;

```

Codice del componente "formTwoFa.tsx"

### 2.2.6 Profilo

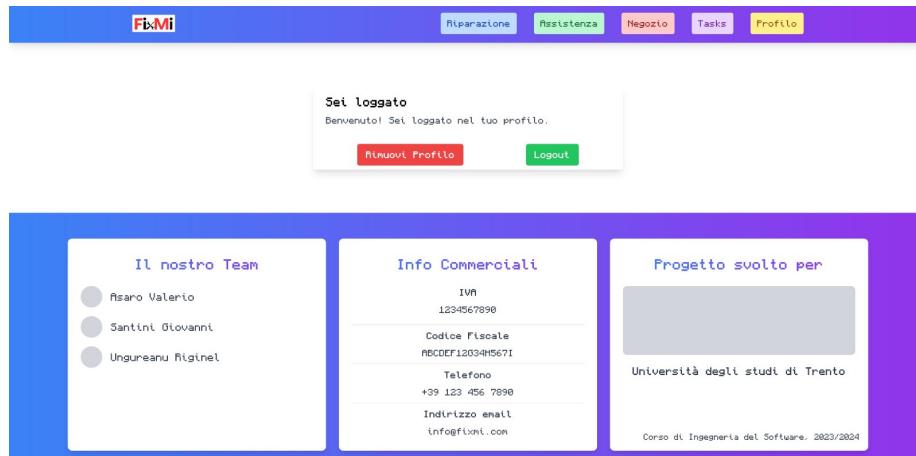


Figure 2.39: Schermata per il profilo quando si è autenticati



Figure 2.40: Schermata per il profilo quando non si è autenticati

### Specifiche

La pagina può assumere due aspetti diversi se chi sta cercando di visualizzare la pagina è autenticato oppure no. Se non si ha ancora fatto il login la pagina si occupa di informare l'utente che non ha ancora fatto l'accesso, ma che tuttavia sta cercando di visualizzare il proprio profilo.

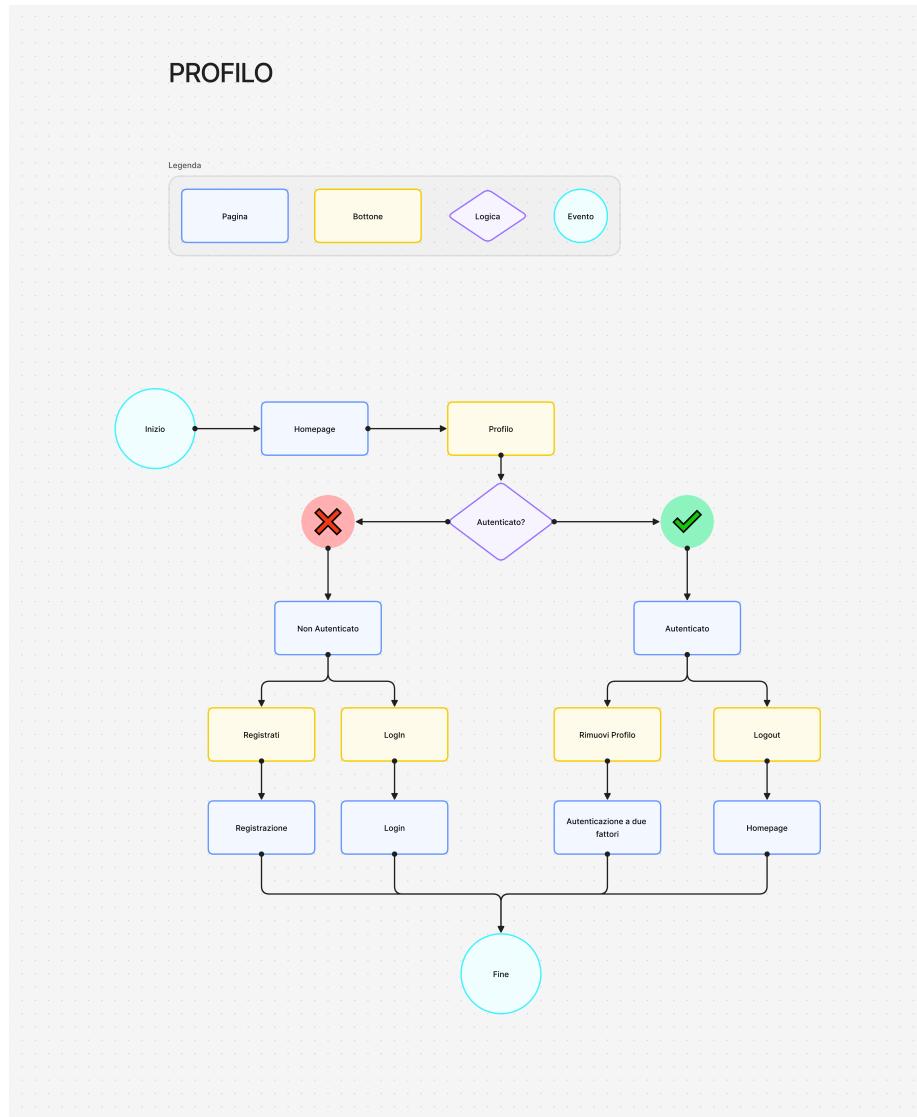
E' possibile fare il login cliccando il bottone blu "Login"

E' possibile registrarsi cliccando il bottone verde "Registrati" Se il login è già stato effettuato la pagina si occupa di informare l'utente che ha fatto l'accesso. E' possibile eliminare il proprio profilo cliccando il bottone rosso "Rimuovi Profilo"

E' possibile uscire dal proprio profilo cliccando il bottone verde "Logout"

### **Userflow**

Partendo dalla pagina "Homepage" l'utente deve cliccare il tasto "Profilo" presente nella "navbar".



Userflow della pagina "Profilo"

### Codice

La pagina "Profilo" è composta da due componenti chiamati "profile.tsx" e "notLoggedIn.tsx" con le seguenti caratteristiche:

- profile.tsx
  - Contiene al suo interno "notLoggedIn.tsx", "logout.tsx", "remove.tsx"

- Se l'utente è autenticato mostra due buttoni "Rimuovi Profilo" e "Logout"
- Contiene controlli e "Fallbacks" per eventuali errori

- notLoggedIn.tsx

- Mostra all'utente due buttoni "Login" e "Registrati"
- Contiene controlli sulle credenziali inserite



```

export default function NotLoggedIn() {
  return (
    <div className="flex justify-center items-center h-screen bg-gray-50">
      <div className="w-full max-w-4xl min-h-80">
        <div className="rounded-lg overflow-hidden shadow-lg bg-white">
          <div className="px-8 py-6 bg-red-100">
            <div className="font-bold text-4xl text-center mb-4 text-red-600">Non sei ancora autenticato!</div>
          </div>
          <div className="px-8 pt-4 pb-6">
            <p className="text-gray-700 text-lg text-center mb-20 mt-20">
              È necessario fare il login per poter visualizzare il proprio profilo.
            </p>
            <div className="flex justify-around">
              <a href="#" className="bg-blue-500 text-white text-4xl py-2 px-6 rounded hover:bg-blue-600 transition duration-300 ease-in-out" >
                Login
              </a>
              <a href="#" className="bg-green-500 text-white text-4xl py-2 px-6 rounded hover:bg-green-600 transition duration-300 ease-in-out" >
                Registrati
              </a>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}

```

Codice del componente "notLoggedIn.tsx"

---

CHAPTER  
**THREE**

---

## MICROSERVIZIO TASK

### 3.1 Back-End

Questa sezione comprende lo sviluppo e la documentazione della Back-End del Microservizio "Task".

#### 3.1.1 Lista delle Risorse

Segue una lista delle risorse implementate dal microservizio:

- "Task": contiene tutte le informazioni e i metodi che appartengono alla classe Task (vedi sotto).
- "Crea Task": permette all'utente di creare una nuova task.
- "Modifica Stato Task": permette al dipendente o al manager di modificare lo stato di una task.
- "Scegli Task": permette al dipendente o al manager di assegnare una task al proprio profilo.
- "Get Lista Task In Lavorazione": permette al dipendente o al manager di ottenere la lista delle task con lo status di "In Lavorazione" in formato json.
- "Get Lista Task Da Eseguire": permette al dipendente o al manager di ottenere la lista delle task con lo status di "Da Eseguire" in formato json.

- "Get Lista Task In Pausa": permette al dipendente o al manager di ottenere la lista delle task con lo status di "In Pausa" in formato json.
- "Get Storico Task" permette al dipendente o al manager di ottenere la lista delle task con lo status di "Completata" in formato json. Il dipendente riceverà le tasks che gli sono state assegnate e che sono completate, mentre il dipendente riceverà in risposta tutte le task completate da ogni dipendente.

### 3.1.2 Specifica delle Risorse

Per ogni risorsa elencata nella sezione precedente, si descrive la risorsa attraverso l'utilizzo di illustrazioni e testo.

### 3.1.3 Task

#### Specifiche

La risorsa "Task" implementa la classe "Task" vista nel "Documento di Architettura".

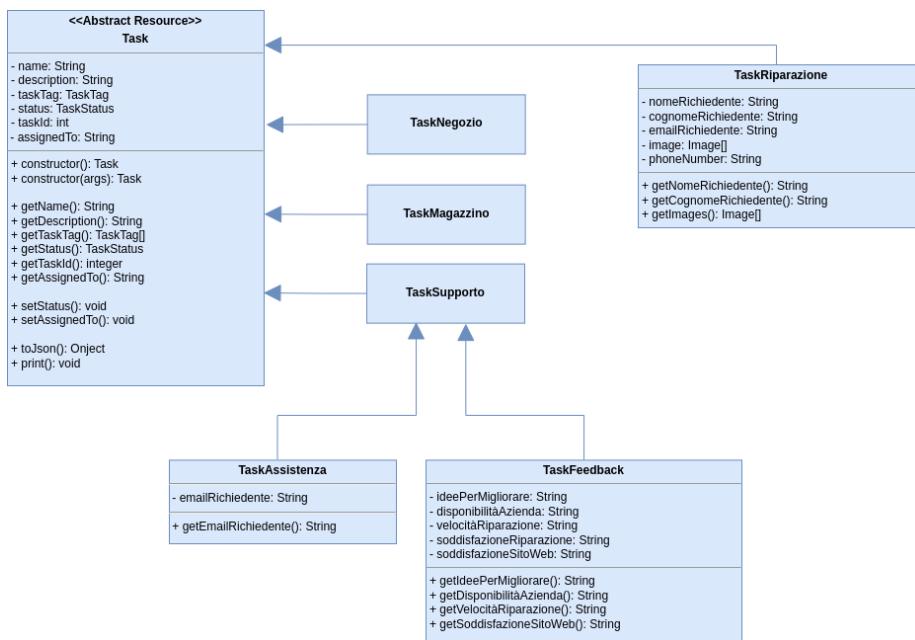


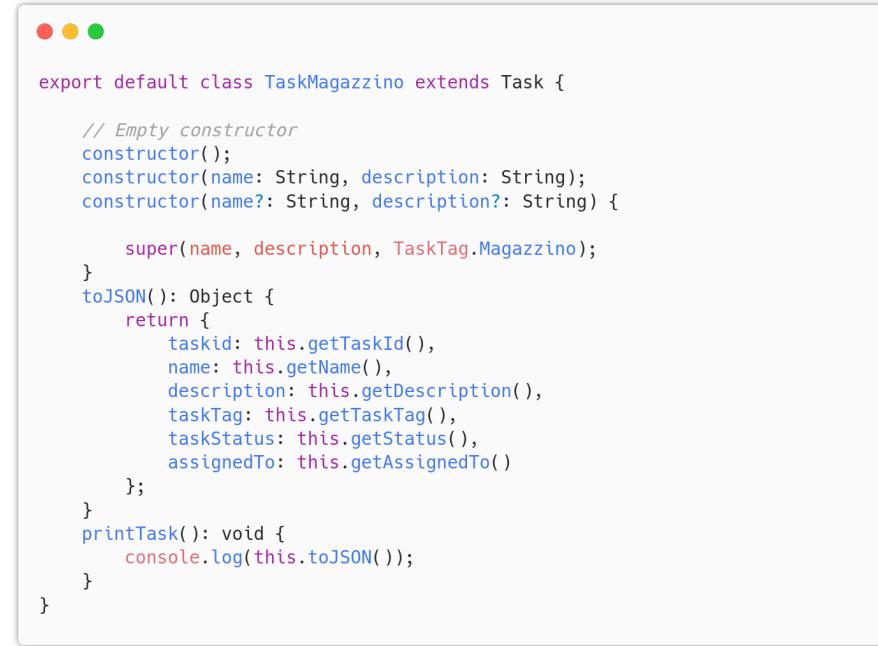
Diagramma della risorsa "Task".

**Codice**

Segeue l'implementazione nel linguaggio "TypeScript":

```
export default abstract class Task {  
  
    #name: String;  
    #description: String;  
    #taskTag: TaskTag;  
    #taskStatus: TaskStatus;  
    #taskId: ObjectId;  
    #assignedTo: String;  
  
    // Empty constructor  
    constructor();  
    constructor(name: String, description: String,  
               taskTag: TaskTag);  
    constructor(name?: String, description?: String,  
               taskTag?: TaskTag) {  
  
        this.#name = name;  
        this.#description = description;  
        this.#taskTag = taskTag;  
        this.#taskStatus = TaskStatus.DaEseguire;  
        this.#taskId = new ObjectId();  
        this.#assignedTo = null; // When a task is created,  
                               // it is not assigned to anyone  
    }  
    getName(): String {  
        return this.#name;  
    }  
    getDescription(): String {  
        return this.#description;  
    }  
    getTaskTag(): TaskTag {  
        return this.#taskTag;  
    }  
    getStatus(): TaskStatus {  
        return this.#taskStatus;  
    }  
    setStatus(taskStatus: TaskStatus) {  
        this.#taskStatus = taskStatus;  
    }  
    getTaskId(): String {  
        return this.#taskId.toString();  
    }  
    getAssignedTo(): String {  
        return this.#assignedTo;  
    }  
    setAssignedTo(assignedTo: String) {  
        this.#assignedTo = assignedTo;  
    }  
    toJSON(): Object { // This is an abstract class  
        return {};  
        // should not return anything  
    }  
}
```

File situato in "backend/classes/Task.ts".

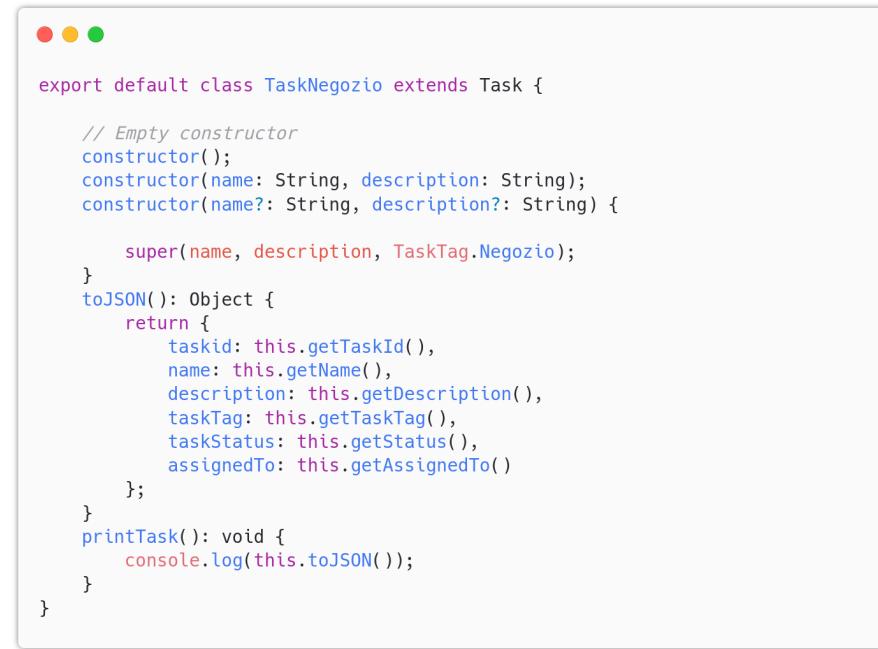


```
export default class TaskMagazzino extends Task {

    // Empty constructor
    constructor();
    constructor(name: String, description: String);
    constructor(name?: String, description?: String) {

        super(name, description, TaskTag.Magazzino);
    }
    toJSON(): Object {
        return {
            taskid: this.getTaskId(),
            name: this.getName(),
            description: this.getDescription(),
            taskTag: this.getTaskTag(),
            taskStatus: this.getStatus(),
            assignedTo: this.getAssignedTo()
        };
    }
    printTask(): void {
        console.log(this.toJSON());
    }
}
```

File situato in "backend/classes/TaskMagazzino.ts".



```
export default class TaskNegozio extends Task {

    // Empty constructor
    constructor();
    constructor(name: String, description: String);
    constructor(name?: String, description?: String) {

        super(name, description, TaskTag.Negozio);
    }
    toJSON(): Object {
        return {
            taskid: this.getTaskId(),
            name: this.getName(),
            description: this.getDescription(),
            taskTag: this.getTaskTag(),
            taskStatus: this.getStatus(),
            assignedTo: this.getAssignedTo()
        };
    }
    printTask(): void {
        console.log(this.toJSON());
    }
}
```

File situato in "backend/classes/TaskNegozio.ts".

```
● ● ●

export default class TaskRiparazione extends Task {

    #nomeRichiedente: String;
    #cognomeRichiedente: String;
    #emailRichiedente: String;
    // #images: Image[];
    #phoneNumber: String;

    // Empty constructor
    constructor();
    constructor(name: String, description: String,
                nomeRichiedente: String, cognomeRichiedente: String,
                emailRichiedente: String, phoneNumber: String);
    constructor(name?: String, description?: String,
                nomeRichiedente?: String, cognomeRichiedente?: String,
                emailRichiedente?: String, phoneNumber?: String) {

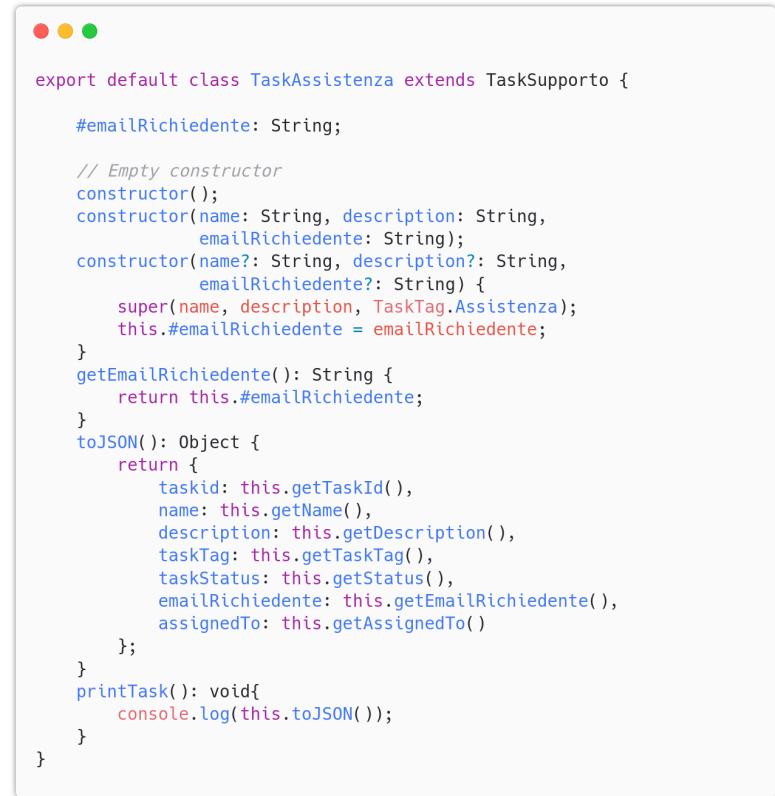
        super(name, description, TaskTag.Riparazione);
        this.#nomeRichiedente = nomeRichiedente;
        this.#cognomeRichiedente = cognomeRichiedente;
        this.#emailRichiedente = emailRichiedente;
        this.#phoneNumber = phoneNumber;
    }
    getNomeRichiedente(): String {
        return this.#nomeRichiedente;
    }
    getCognomeRichiedente(): String {
        return this.#cognomeRichiedente;
    }
    getEmailRichiedente(): String {
        return this.#emailRichiedente;
    }
    getPhoneNumber(): String {
        return this.#phoneNumber;
    }
    toJSON(): Object {
        return {
            taskid: this.getTaskId(),
            name: this.getName(),
            description: this.getDescription(),
            taskTag: this.getTaskTag(),
            taskStatus: this.getStatus(),
            nomeRichiedente: this.getNomeRichiedente(),
            cognomeRichiedente: this.getCognomeRichiedente(),
            emailRichiedente: this.getEmailRichiedente(),
            phoneNumber: this.getPhoneNumber(),
            assignedTo: this.getAssignedTo()
        };
    }
    printTask(): void{
        console.log(this.toJSON());
    }
}
```

File situato in "backend/classes/TaskRiparazione.ts".



```
export default abstract class TaskSupporto extends Task {  
    // Empty constructor  
    constructor();  
    constructor(name: String, description: String,  
               taskTag: TaskTag);  
    constructor(name?: String, description?: String,  
               taskTag?: TaskTag) {  
        super(name, description, taskTag);  
    }  
}
```

File situato in "backend/classes/TaskSupporto.ts".



```
export default class TaskAssistenza extends TaskSupporto {  
  
    #emailRichiedente: String;  
  
    // Empty constructor  
    constructor();  
    constructor(name: String, description: String,  
               emailRichiedente: String);  
    constructor(name?: String, description?: String,  
               emailRichiedente?: String) {  
        super(name, description, TaskTag.Assistenza);  
        this.#emailRichiedente = emailRichiedente;  
    }  
    getEmailRichiedente(): String {  
        return this.#emailRichiedente;  
    }  
    toJSON(): Object {  
        return {  
            taskId: this.getTaskId(),  
            name: this.getName(),  
            description: this.getDescription(),  
            taskTag: this.getTaskTag(),  
            taskStatus: this.getStatus(),  
            emailRichiedente: this.getEmailRichiedente(),  
            assignedTo: this.getAssignedTo()  
        };  
    }  
    printTask(): void{  
        console.log(this.toJSON());  
    }  
}
```

File situato in "backend/classes/TaskAssistenza.ts".

```
● ○ ●

export default class TaskFeedback extends TaskSupporto {

    #ideePerMigliorare: String;
    #disponibilitaAzienda: String;
    #velocitaRiparazione: String;
    #soddisfazioneRiparazione: String;
    #soddisfazioneSitoWeb: String;

    // Empty constructor
    constructor();
    constructor(name: String, description: String,
                ideePerMigliorare: String, disponibilitaAzienda: String,
                velocitaRiparazione: String,
                soddisfazioneRiparazione: String,
                soddisfazioneSitoWeb: String);

    constructor(name?: String, description?: String,
                ideePerMigliorare?: String, disponibilitaAzienda?: String,
                velocitaRiparazione?: String,
                soddisfazioneRiparazione?: String,
                soddisfazioneSitoWeb?: String) {

        super(name, description, TaskTag.Feedback);
        this.#ideePerMigliorare = ideePerMigliorare;
        this.#disponibilitaAzienda = disponibilitaAzienda;
        this.#velocitaRiparazione = velocitaRiparazione;
        this.#soddisfazioneRiparazione = soddisfazioneRiparazione;
        this.#soddisfazioneSitoWeb = soddisfazioneSitoWeb;
    }
    getIdeePerMigliorare(): String {
        return this.#ideePerMigliorare;
    }
    getDisponibilitaAzienda(): String {
        return this.#disponibilitaAzienda;
    }
    getVelocitaRiparazione(): String {
        return this.#velocitaRiparazione;
    }
    getSoddisfazioneRiparazione(): String {
        return this.#soddisfazioneRiparazione;
    }
    getSoddisfazioneSitoWeb(): String {
        return this.#soddisfazioneSitoWeb;
    }
    toJSON(): Object {
        return {
            taskid: this.getTaskId(),
            name: this.getName(),
            description: this.getDescription(),
            taskTag: this.getTaskTag(),
            taskStatus: this.getStatus(),
            ideePerMigliorare: this.getIdeePerMigliorare(),
            disponibilitaAzienda: this.getDisponibilitaAzienda(),
            velocitaRiparazione: this.getVelocitaRiparazione(),
            soddisfazioneRiparazione: this.getSoddisfazioneRiparazione(),
            soddisfazioneSitoWeb: this.getSoddisfazioneSitoWeb(),
            assignedTo: this.getAssignedTo()
        };
    }
    printTask(): void {
        console.log(this.toJSON()); 102
    }
}
```

File situato in "backend/classes/TaskFeedback.ts".

**Specifica OpenAPI**

Segue la definizione in OpenAPI specification:

```
components:  
schemas:  
    Task:  
        type: object  
        properties:  
            nome:  
                type: string  
                example: "Task di esempio"  
            descrizione:  
                type: string  
                example: "Descrizione del task di esempio"  
            taskTag:  
                type: string  
                enum:  
                    - Negozio  
                    - Riparazione  
                    - Magazzino  
                    - Assistenza  
                    - Feedback  
                example: "Riparazione"  
            taskStatus:  
                type: string  
                enum:  
                    - Da Eseguire  
                    - In Lavorazione  
                    - In Pausa  
                    - Completata  
                example: "In Lavorazione"  
        required:  
            - nome  
            - descrizione  
            - taskTag  
            - taskStatus  
    TaskAssistenza:
```

```
allOf:  
  - $ref: '#/components/schemas/Task'  
  - type: object  
    properties:  
      emailRichiedente:  
        type: string  
        format: email  
        example: "richiedente@example.com"  
    required:  
      - emailRichiedente  
TaskFeedback:  
  allOf:  
    - $ref: '#/components/schemas/Task'  
    - type: object  
      properties:  
        ideePerMigliorare:  
          type: string  
          example: "Suggerimenti per migliorare il servizio"  
        disponibilitaAzienda:  
          type: string  
          example: "Disponibilità dell'azienda a migliorare"  
        velocitaRiparazione:  
          type: string  
          example: "8"  
        soddisfazioneRiparazione:  
          type: sting  
          example: 9  
        soddisfazioneSitoWeb:  
          type: string  
          example: 7  
    required:  
      - ideePerMigliorare  
      - disponibilitaAzienda  
      - velocitaRiparazione  
      - soddisfazioneRiparazione  
      - soddisfazioneSitoWeb
```

```

TaskRiparazione:
  allOf:
    - $ref: '#/components/schemas/Task'
    - type: object
      properties:
        nomeRichiedente:
          type: string
          example: "Mario"
        cognomeRichiedente:
          type: string
          example: "Rossi"
        emailRichiedente:
          type: string
          format: email
          example: "mario.rossi@example.com"
        numeroTelefono:
          type: string
          example: "391234567890"
      required:
        - nomeRichiedente
        - cognomeRichiedente
        - emailRichiedente
        - numeroTelefono

TaskMagazzino:
  allOf:
    - $ref: '#/components/schemas/Task'
    - type: object

TaskNegozio:
  allOf:
    - $ref: '#/components/schemas/Task'
    - type: object

```

### 3.1.4 Scegli Task

#### Specifica

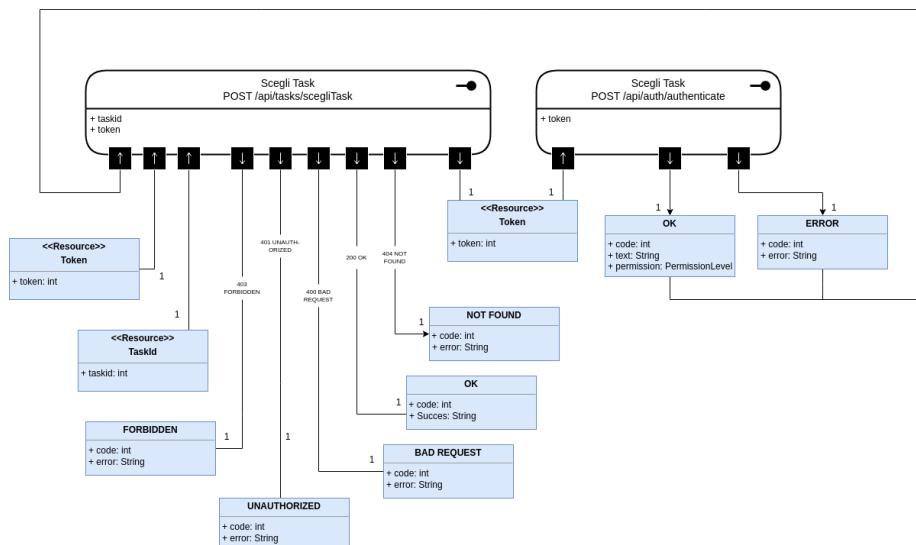
La risorsa "Scegli Task", accessibile tramite la route "/api/tasks/scegliTask" permette di assegnare una task ad un dipendente o ad un manager. L'API

richiede tutti i seguenti campi:

- "token": il codice numerico che identifica una sessione autenticata, ottenuto attraverso il microservizio autenticazione.
  - "taskId": il codice univoco che identifica una task nel database.

La response body è in formato application/json. Di seguito le possibili risposte:

Status Code	Body	Spiegazione
200 OK	{Success: "task assigned to the user" }	La procedura è terminata con successo
400 BAD REQUEST	{error: "missing fields", missingFields }	I campi richiesti sono mancanti.
401 UNAUTHORIZED	{ error: "User not found with the given token" }	Il token non è valido
403 FORBIDDEN	{ error: "User not authorized" }	L'utente non ha permessi elevati per accedere all'API
404 NOT FOUND	{ error: "Task not found" }	Non è stata trovata alcuna task con il taskid inviato.



## Diagramma delle risorse per l'API "Scegli Task".

**Specifica OpenAPI**

Di seguito viene derscritta la risorsa secondo OpenAPI specification:

```
openapi: 3.0.3
  info:
    title: Scegli Task
    version: 1.0.0
    description: Permette di assegnare una
                  task ad un dipendente o ad un manager
  paths:
    /api/tasks/scegliTask:
      post:
        requestBody:
          required: true
          content:
            application/json:
              schema:
                type: object
                properties:
                  token:
                    type: string
                    description: Il token ottenuto attraverso
                                  il microservizio autenticazione.
                    example: 171595638452613
                  taskid:
                    type: string
                    description: L'identificativo univoco
                                  della task
                    example: 66476aa0726f8460bce4a26d
          required:
            - token
            - taskid
        responses:
          '200':
            description: Task assigned to the user
            content:
              application/json:
```

```
schema:
  type: object
  properties:
    Success:
      type: string
      example: "Task assigned to the user"
    '400':
      description: Bad Request
      content:
        application/json:
          schema:
            type: object
            properties:
              error:
                type: string
                example: "missing fields"
              missingFields:
                type: array
                items:
                  type: string
                  example: token
            required:
              - error
              - missingFields
    '401':
      description: Unauthorized
      content:
        application/json:
          schema:
            type: object
            properties:
              error:
                type: string
                example: "user not found with
                          the given token"
            required:
              - error
```

```
'403':  
    description: Forbidden  
    content:  
        application/json:  
            schema:  
                type: object  
                properties:  
                    error:  
                        type: string  
                        example: "User not authorized"  
                required:  
                    - error  
'404':  
    description: Not Found  
    content:  
        application/json:  
            schema:  
                type: object  
                properties:  
                    error:  
                        type: string  
                        example: "taskidask not found"  
                required:  
                    - error
```

### Codice

Il codice che implementa quanto detto sopra è il seguente:

```
scegliTaskRouter.post('/', async (req, res) => {
  try {
    let token = getToken(req);
    if (token === undefined) {
      let e = {'value': 'Missing fields', missingfields: ['token']};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(400);
      res.json(e);
      return;
    }
    // Check authentication
    let profile = await getProfileInfo(token);
    if (profile == null) {
      let e = {'value': 'User not found with the given token'};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(401);
      res.json(e);
      return;
    }
    // Check authorization
    if (!isAuthorized(profile)) {
      let e = {'value': 'User not authorized'};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(403);
      res.json(e);
      return;
    }
    // Get the task id
    let missingFields = getMissingFields([["taskid", req.body.taskid]]);
    if (missingFields.length != 0) {
      let e = {'value': 'Missing fields', missingfields: missingFields};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(400);
      res.json(e);
      return;
    }
    // Check if the task exists
    if (!await taskExists(req)) {
      let e = {'value': 'Task not found'};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(404);
      res.json(e);
      return;
    }
    // Query the DB
    let profileId = getProfileId(profile);
    let taskId = getTaskId(req);
    await executeQuery(profileId, taskId);
    // OK
    res.status(200);
    res.json({'Success': 'Task assigned to the user'});
  }
  catch(e) {
    dbg("(ERROR)", e);
    res.status(400);
    res.json(JSON.parse(e.message));
  }
});
```

```
// Get the task id
function getTaskId(req) {
    return req.body.taskid;
}
// Check if a task exists with the given taskId
async function taskExists(req) {

    let taskId = getTaskId(req);

    // Used for testing purposes
    if (taskId == 'test') return true;

    return db.collection("tasks")
        .findOne({ taskid: taskId })
        .then(task => {
            if (task != null) {
                return true;
            }
            return false;
        });
}
```

Funzione "taskExists()".

```
// Query the DB
async function executeQuery(profileId, taskId) {

    dbg("Executing query", "");

    // Used for testing purposes
    if (taskId == 'test') return;

    return db.collection("tasks")
        .updateOne({ taskid: taskId }, { $set: { assignedTo: profileId } });
}
```

Funzione "executeQuery()".

## Testing

Sono stati realizzati i seguenti tests per coprire ogni possibile risultato della chiamata:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Nessun field	-	-	-	400	400
2	Invio Token non valido	{token: "fake" }	-	-	401	401
3	Invio Token da un cliente	{ token: tokenCliente }	Esistenza di un cliente nel DB	Microservizio Autenticazione, DB	403	403
4	Invio di un token valido ma senza taskid	{ token: tokenManager }	Esistenza di un manager nel DB	Microservizio Autenticazione, DB	400	400
5	Invio token valido, taskid non valido	{ token: tokenManager, taskid: "fake" }	Esistenza di un Manager nel DB	Microservizio Autenticazione, DB	404	404
6	Invio token valido con taskid valido	{ token: tokenManager, taskid: validTaskid }	Esistenza di un Manager e Task nel DB	Microservizio Autenticazione, DB	200	200

Di seguito il codice dei tests e l'output:

```
● ● ●

it('Sending request with missing fields, should return 400', async () => {
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/scegliTask')
    expect(response.status).toBe(400);
});
```

Testcase 1

```
● ● ●

it('Sending a wrong token, should return 401 user not found with given
token', async () => {
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/scegliTask')
    .type('form')
    .send({token: "fake"});
  expect(response.status).toBe(401);
});
```

Testcase 2

```
● ● ●

it('Sending a token from a Cliente, should return 403 not allowed', async
() => {

  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({email:"cliente@hotmail.com",
           password:"test",twofa:"12345"});
  let token = resLogin.body.token;

  // sending the request
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/scegliTask')
    .type('form')
    .send({token: token});
  expect(response.status).toBe(403);
});
```

Testcase 3

```
● ● ●

it('Sending the right token with no taskid, should return 400 not enough arguments', async () => {

    // getting the token
    const resLogin = await request('10.5.0.11:3001')
        .post('/api/auth/login')
        .type('form')
        .send({email:"manager@test.com",password:"test",twofa:"12345"});
    let token = resLogin.body.token;

    // sending the request
    const response = await request('10.5.0.12:3001')
        .post('/api/tasks/scegliTask')
        .type('form')
        .send({token: token});
    expect(response.status).toBe(400);
});
```

Testcase 4

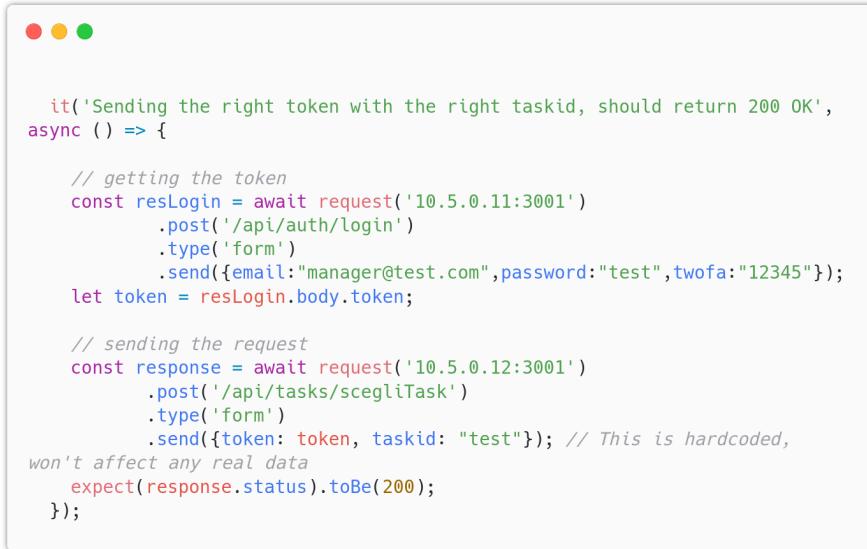
```
● ● ●

it('Sending the right token with wrong taskid, should return 404 not found', async () => {

    // getting the token
    const resLogin = await request('10.5.0.11:3001')
        .post('/api/auth/login')
        .type('form')
        .send({email:"manager@test.com",password:"test",twofa:"12345"});
    let token = resLogin.body.token;

    // sending the request
    const response = await request('10.5.0.12:3001')
        .post('/api/tasks/scegliTask')
        .type('form')
        .send({token: token, taskid: "fake"});
    expect(response.status).toBe(404);
});
```

Testcase 5



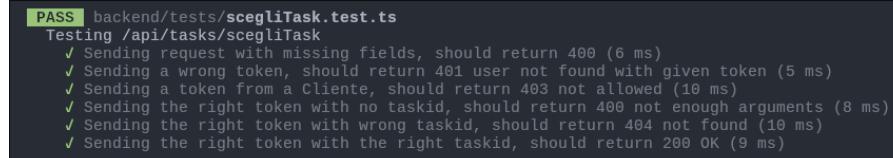
```

it('Sending the right token with the right taskid, should return 200 OK',
  async () => {
    // getting the token
    const resLogin = await request('10.5.0.11:3001')
      .post('/api/auth/login')
      .type('form')
      .send({email:"manager@test.com",password:"test",twofa:"12345"});
    let token = resLogin.body.token;

    // sending the request
    const response = await request('10.5.0.12:3001')
      .post('/api/tasks/scegliTask')
      .type('form')
      .send({token: token, taskid: "test"}); // This is hardcoded,
    // won't affect any real data
    expect(response.status).toBe(200);
  });
}

```

Testcase 6



```

PASS backend/tests/scegliTask.test.ts
Testing /api/tasks/scegliTask
  ✓ Sending request with missing fields, should return 400 (6 ms)
  ✓ Sending a wrong token, should return 401 user not found with given token (5 ms)
  ✓ Sending a token from a Cliente, should return 403 not allowed (10 ms)
  ✓ Sending the right token with no taskid, should return 400 not enough arguments (8 ms)
  ✓ Sending the right token with wrong taskid, should return 404 not found (10 ms)
  ✓ Sending the right token with the right taskid, should return 200 OK (9 ms)

```

Risultato dei tests

### 3.1.5 Modifica Stato Task

#### Specifiche

La risorsa "Modifica Stato Task" è accessibile tramite la route "/api/tasks/modificaStatoTask". Permette al dipendente o al manager di impostare un nuovo stato ad una task presente nel database. Gli stati possibili sono definiti nell'enum TaskStatus e sono i seguenti: "DaEseguire", "InLavorazione", "InPausa", "Completata".



```
export enum TaskStatus {
  DaEseguire = 'Da Eseguire',
  Inlavorazione = 'In Lavorazione',
  InPausa = 'In Pausa',
  Completata = 'Completata',
  Fallita = 'Fallita'
}
```

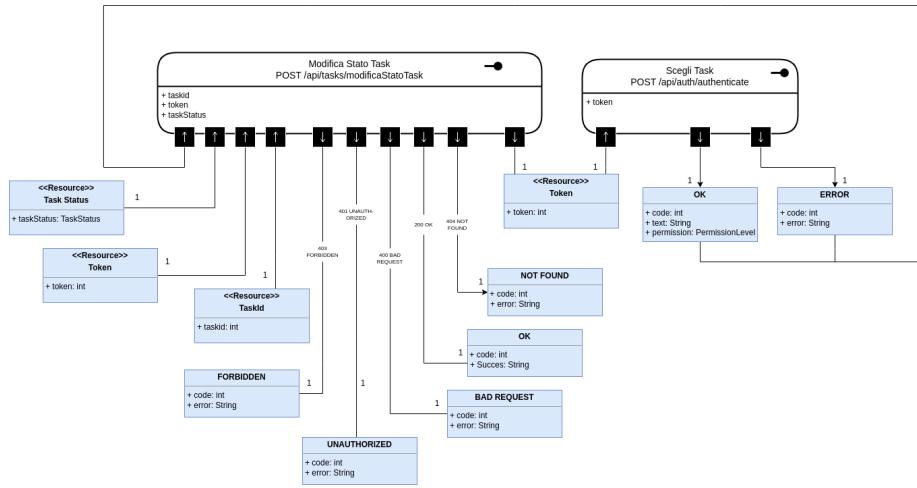
Definizione dell'enum "TaskStatus" in "backend/enums/TaskStatus.ts".

La risorsa richiede tutti i seguenti campi nel body:

- "taskid": identificativo univoco di una task
- "token": token ottenuto dal microservizio autenticazione
- "taskStatus": il nuovo stato che verrà impostato nella task

Il corpo della risposta è in formato application/json e riporta le seguenti risposte:

Status Code	Body	Spiegazione
200 OK	{Success: "Task status changed successfully" }	La procedura è terminata con successo
400 BAD REQUEST	{error: "missing fields", missingFields }	I campi richiesti sono mancanti.
400 BAD REQUEST	{error: "Wrong status", missingFields }	Lo status inviato non è valido (non corrisponde ad uno dei valori dell'enum).
401 UNAUTHORIZED	{ error: "User not found with the given token" }	Il token non è valido
403 FORBIDDEN	{ error: "User not authorized" }	L'utente non ha permessi elevati per accedere all'API
404 NOT FOUND	{ error: "Task not found" }	Non è stata trovata alcuna task con il taskid inviato.



Modello delle risorse per "/api/tasks/modificaStatoTask".

### Specifica OpenAPI

Segue la definizione secondo OpenAPI:

```

openapi: 3.0.3
  info:
    title: Modifica Stato Tas
    version: 1.0.0
  paths:
    /api/tasks/modificaStatoTask:
      post:
        summary: Modifica lo stato di una task
        description: Questa route permette al dipendente o al manager di cambiare lo stato di una task.
        requestBody:
          required: true
          content:
            application/json:
              schema:
                type: object
                properties:
                  token:
                    type: string
  
```

```
        description: Il token dell'utente
        example: 171595638452613

    taskid:
        type: string
        description: Identificativo della task
        example: 66476aa0726f8460bce4a26d

    taskStatus:
        type: TaskStatus
        description: Il nuovo stato della task.
        example: Completata

    required:
        - token
        - taskid
        - taskStatus

responses:
  '200':
    description: Task status changed successfully
    content:
      application/json:
        schema:
          type: object
          properties:
            Success:
              type: string
              example: "Task status
changed successfully"
  '400':
    description: Bad Request
    content:
      application/json:
        schema:
          type: object
          properties:
            error:
              type: string
              example: "missing fields"
            missingFields:
```

```
        type: array
        items:
            type: string
        required:
            - error
            - missingFields
'401':
    description: Unauthorized
    content:
        application/json:
            schema:
                type: object
                properties:
                    error:
                        type: string
                        example: "user not found
                                with the given token"
                required:
                    - error
'403':
    description: Forbidden
    content:
        application/json:
            schema:
                type: object
                properties:
                    error:
                        type: string
                        example: "User not authorized"
                required:
                    - error
'404':
    description: Not Found
    content:
        application/json:
            schema:
                type: object
```

```
properties:  
  error:  
    type: string  
    example: "Task not found"  
required:  
  - error
```

**Codice**

Il codice è il seguente:



```

modificaStatoTaskRouter.post('/', async (req, res) => {
  try {
    let token = getToken(req);
    dbg("Token", token); // qui il token a UNDEFINED DIO LUPO
    if (token === undefined) {
      let e = {'value': 'Missing fields', missingfields: ['token']};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(400);
      res.json(e);
      return;
    }
    // Check authentication
    let profile = await getProfileInfo(token);
    if (profile === null) {
      let e = {'value': 'User not found with the given token'};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(401);
      res.json(e);
      return;
    }
    // Check authorization
    if (!isAuthorized(profile)) {
      let e = {'value': 'User not authorized'};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(403);
      res.json(e);
      return;
    }

    let taskId = getTaskId(req);
    let statusVal = getStatus(req);
    let missingFields2 = getMissingFields([["taskid", taskId], ["taskStatus", statusVal]]);
    if (missingFields2.length != 0) {
      let e = {'value': 'Missing fields', missingfields: missingFields2};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(400);
      res.json(e);
      return;
    }
    // Check if the status is correct
    if (!checkStatus(statusVal)) {
      let e = {'value': 'Wrong status'};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(400);
      res.json(e);
      return;
    }
    // Check if the task exists
    if (! await taskExists(req)) {
      let e = {'value': 'Task not found'};
      dbg("(ERROR)", JSON.stringify(e));
      res.status(404);
      res.json(e);
      return;
    }
    // Query the DB
    let profileId = getProfileId(profile);
    await executeQuery(taskId, statusVal);

    // OK
    res.json({'Success': 'Task status changed successfully'});
  }
  catch(e) {
    dbg("(ERROR)", e);
    res.status(400);
    res.json(JSON.parse(e.message));
  }
});

```

Estratto da "backend/routes/modificaStatoTask".

```
// Check if the status is correct
function checkStatus(statusVal) {
    // Check if the status is a value in the enum
    if (!Object.values(TaskStatus).includes(statusVal)) {
        return false;
    }
    return true;
}

// Query the DB
async function executeQuery(taskId, statusVal) {
    dbg("Executing query", "");
    // Used for testing purposes
    if (taskId === 'test') return;

    return db.collection("tasks")
        .updateOne({ taskId: taskId }, { $set: { taskStatus: statusVal } });
}
```

Funzione "checkStatus()" e "executeQuery()".

## Testing

Sono stati creati i seguenti tests:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Nessun field	-	-	-	400	400
2	Invio Token non valido	{token: "fake" }	-	-	401	401
3	Invio Token da un cliente	{ token: tokenCliente }	Esistenza di un cliente nel DB	Microservizio Autenticazione, DB	403	403
4	Invio di un token valido ma senza taskid	{ token: tokenManager }	Esistenza di un manager nel DB	Microservizio Autenticazione, DB	400	400
5	Invio token valido con taskid valido ma con uno stato sbagliato	{ token: tokenManager, taskid: validTaskid, taskStatus: "fake" }	Esistenza di un Manager e Task nel DB	Microservizio Autenticazione, DB	400	400
6	Invio token valido con taskid valido e uno stato valido	{ token: tokenManager, taskid: validTaskid, taskStatus: validStatus }	Esistenza di un Manager e Task nel DB	Microservizio Autenticazione, DB	200	200

Di seguito il codice dei tests:

```
● ● ●
it('Sending request with missing fields, should return 400', async () => {
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/modificaStatoTask')
    .expect(response.status).toBe(400);
});
```

Testcase 1

```
it('Sending a wrong token, should return 401 user not found with given token', async () => {
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/modificaStatoTask')
    .type('form')
    .send({token: "fake"});
  expect(response.status).toBe(401);
});
```

Testcase 2

```
it('Sending a token from a Cliente, should return 403 not allowed', async () => {
  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({email:"cliente@hotmail.com",
           password:"test",twofa:"12345"});
  let token = resLogin.body.token;

  // sending the request
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/modificaStatoTask')
    .type('form')
    .send({token: token});
  expect(response.status).toBe(403);
});
```

Testcase 3

```
it('Sending the right token with no taskid, should return 400 not enough arguments', async () => {
  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({email:"manager@test.com",password:"test",twofa:"12345"});
  let token = resLogin.body.token;

  // sending the request
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/modificaStatoTask')
    .type('form')
    .send({token: token, taskid: "fake"});
  expect(response.status).toBe(400);
});
```

Testcase 4

```

it('Sending the right token with the right taskid but with a wrong status, should return 400 wrong status', async () => {
    // getting the token
    const resLogin = await request('10.5.0.11:3001')
        .post('/api/auth/login')
        .type('form')
        .send({email:"manager@test.com",password:"test",twofa:"12345"});
    let token = resLogin.body.token;

    // sending the request
    const response = await request('10.5.0.12:3001')
        .post('/api/tasks/modificaStatoTask')
        .type('form')
        .send({token: token, taskid: "test", taskStatus: "fake"});
    // This is hardcoded, won't affect any real data
    expect(response.status).toBe(400);
});

```

Testcase 5

```

it('Sending the right token with the right taskid with right status, should return 200 OK', async () => {
    // getting the token
    const resLogin = await request('10.5.0.11:3001')
        .post('/api/auth/login')
        .type('form')
        .send({email:"manager@test.com",password:"test",twofa:"12345"});
    let token = resLogin.body.token;

    // sending the request
    const response = await request('10.5.0.12:3001')
        .post('/api/tasks/modificaStatoTask')
        .type('form')
        .send({token: token, taskid: "test", taskStatus: "Completata"});
    // This is hardcoded, won't affect any real data
    expect(response.status).toBe(200);
});

```

Testcase 6

```

PASS | backend/tests/modificaStatoTask.test.ts
Testing /api/tasks/modificaStatoTask
  ✓ Sending request with missing fields, should return 400 (14 ms)
  ✓ Sending a wrong token, should return 401 user not found with given token (47 ms)
  ✓ Sending a token from a Cliente, should return 403 not allowed (14 ms)
  ✓ Sending the right token with no taskid, should return 400 not enough arguments (12 ms)
  ✓ Sending the right token with the right taskid but with a wrong status, should return 400 wrong status (13 ms)
  ✓ Sending the right token with the right taskid with right status, should return 200 OK (14 ms)

```

Risultato dei tests

### 3.1.6 Get Lista Task In Lavorazione

#### Specifiche

La risorsa "Get Lista Task In Lavorazione" è accessibile tramite la route

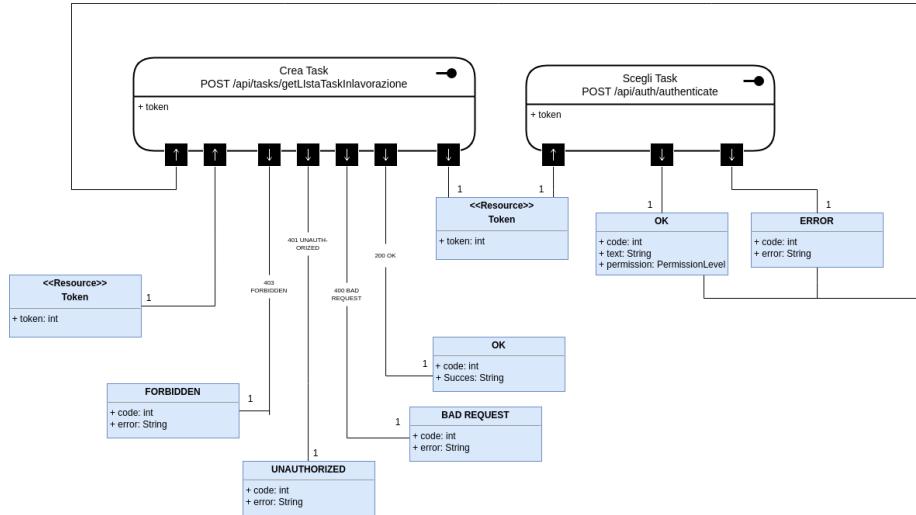
"`/api/tasks/getListaTaskInLavorazione`". Permette al dipendente o al manager di ottenere una lista in formato json delle task nello stato di "In Lavorazione". Gli elementi della rispecchiano gli attributi della risorsa "Task" vista precedentemente.

La risorsa richiede tutti i seguenti campi nel body:

- "token": token ottenuto dal microservizio autenticazione che identifica la sessione

Il corpo della risposta è in formato application/json e riporta le seguenti risposte:

Status Code	Body	Spiegazione
200 OK	{ [ task1, task2, ... ] }	La procedura è terminata con successo
400 BAD REQUEST	{ error: "missing fields", missingFields }	I campi richiesti sono mancanti o malformati.
401 UNAUTHORIZED	{ error: "User not found with the given token" }	Il token non è valido
403 FORBIDDEN	{ error: "User not authorized" }	L'utente non ha permessi elevati per accedere all'API



Modello delle risorse per `"/api/tasks/getListaTaskInLavorazione"`.

### Specifiche OpenAPI

```

openapi: 3.0.3
info:
  title: Get Lista Task In lavorazione
  description: API getListaTaskInLavorazione
  version: 1.0.0
paths:
  /api/tasks/getListaTaskInLavorazione:
    post:
      summary: Ritorna tutte le tasks on lo stato di "In Lavorazione"
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                token:
                  type: string
                  description: User's authentication token
                  example: 171595638452613

```

```
        required:
          - token

      responses:
        '200':
          description: Successfully retrieved tasks in progress
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/TaskInLavorazione'

        '400':
          description: Missing fields in the request
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    type: string
                    example: missing fields
                  missingFields:
                    type: array
                    items:
                      type: string
                      example: token

        '401':
          description: User not found with the given token
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    type: string
                    example: user not found with the given token
```

```
'403':  
    description: User not authorized  
    content:  
        application/json:  
            schema:  
                type: object  
                properties:  
                    error:  
                        type: string  
                example: User not authorized
```

### Codice

Il codice che implementa quanto sopra è il seguente:

```
getListaTaskInLavorazioneRouter.post('/', async (req, res) => {
  try {
    let token = getToken(req)
    if (token === undefined) {
      let e = { value: 'Missing fields', missingfields: ['token'] }
      dbg('ERROR', JSON.stringify(e))
      res.status(400)
      res.json(e)
      return
    }
    // Check authentication
    let profile = await getProfileInfo(token)
    if (profile == null) {
      let e = { value: 'User not found with the given token' }
      dbg('ERROR', JSON.stringify(e))
      res.status(401)
      res.json(e)
      return
    }
    // Get the profile id
    let profileId = getProfileId(profile)
    let permission = getPermission(profile)
    // Check authorization
    if (!isAuthorized(profile)) {
      let e = { value: 'User not authorized' }
      dbg('ERROR', JSON.stringify(e))
      res.status(403)
      res.json(e)
      return
    }
    // Query the DB
    let allTasks = await executeQuery(profileId, permission)
    if (allTasks == null) {
      let e = { value: 'Query error' }
      dbg('ERROR', e)
      res.status(400)
      res.json(e)
      return
    }
    dbg('AllTasks', JSON.stringify(allTasks))
    // OK
    // Return the tasks
    res.json(allTasks)
  } catch (e) {
    dbg('ERROR', e)
    res.status(400)
    res.json(JSON.parse(e.message))
  }
})
```

Codice estratto da "backend/routes/getListaTaskInLavorazione".

```

● ● ●

async function executeQuery(profileId, permission) {
  dbg('Executing query', '')

  // The manager can see all the tasks in lavorazione
  if (permission == 'Manager') {
    return db
      .collection('tasks')
      .find({ taskStatus: 'In Lavorazione' })
      .toArray()
  }

  return db
    .collection('tasks')
    .find({ taskStatus: 'In Lavorazione', assignedTo: profileId })
    .toArray()
}

```

## Testing

Sono stati individuati i seguenti test cases:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Nessun field	-	-	-	400	400
2	Invio Token non valido	{token: "fake" }	-	-	401	401
3	Invio Token da un cliente	{ token: tokenCliente }	Esistenza di un cliente nel DB	Microservizio Autenticazione, DB	403	403
4	Invio token valido	{ token: tokenManager }	Esistenza di un Manager e Task nel DB	Microservizio Autenticazione, DB	200	200

Di seguito il codice dei suddetti tests:

```
● ● ●

it('Sending request with missing fields, should return 400', async () => {
  const response = await request('10.5.0.12:3001').post(
    '/api/tasks/getListaTaskInLavorazione'
  )
  expect(response.status).toBe(400)
});
```

Testcase 1

```
● ● ●

it('Sending a wrong token, should return 401 user not found with given token', async () => {
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/getListaTaskInLavorazione')
    .type('form')
    .send({ token: 'fake' })
  expect(response.status).toBe(401)
})
```

Testcase 2

```
it('Sending a token from a Cliente, should return 403 not allowed',  
async () => {  
    // getting the token  
    const resLogin = await request('10.5.0.11:3001')  
        .post('/api/auth/login')  
        .type('form')  
        .send({ email: 'cliente@hotmail.com', password: 'test', twofa:  
'12345' })  
    let token = resLogin.body.token  
  
    // sending the request  
    const response = await request('10.5.0.12:3001')  
        .post('/api/tasks/getListaTaskInLavorazione')  
        .type('form')  
        .send({ token: token })  
    expect(response.status).toBe(403)  
})
```

### Testcase 3

```
it('Sending the right token, should return 200 OK', async () => {
  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({ email: 'manager@test.com', password: 'test', twofa: '12345' })
  let token = resLogin.body.token

  // sending the request
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/getListaTaskInLavorazione')
    .type('form')
    .send({ token: token })
  expect(response.status).toBe(200)
})
```

## Testcase 4

```
PASS backend/tests/getListaTaskInLavorazione.test.ts
Testing /api/tasks/getListaTaskInLavorazione
  ✓ Sending request with missing fields, should return 400 (2 ms)
  ✓ Sending a wrong token, should return 401 user not found with given token (4 ms)
  ✓ Sending a token from a Cliente, should return 403 not allowed (11 ms)
  ✓ Sending the right token, should return 200 OK (14 ms)
```

Risultato dei tests

### 3.1.7 Get Lista Task Da Eseguire

#### Specifiche

La risorsa "Get Lista Task Da Eseguire" è accessibile tramite la route

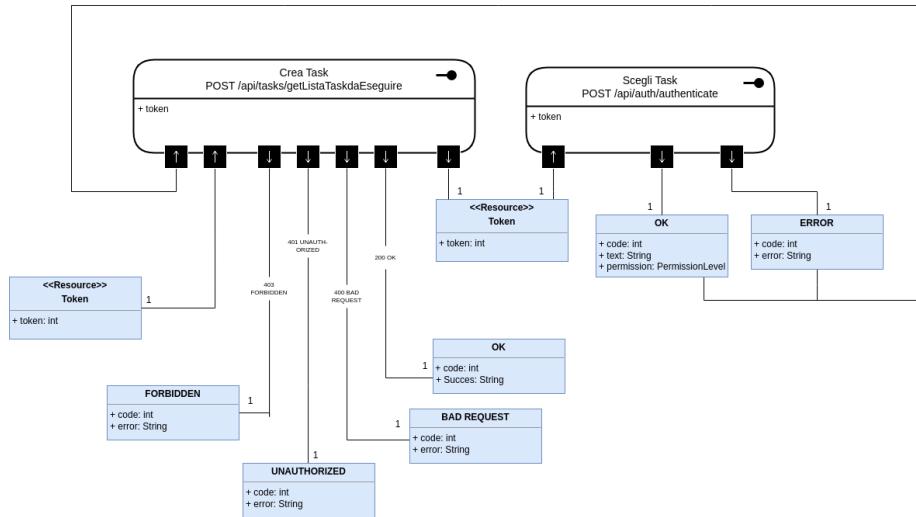
"`/api/tasks/getListaTaskDaEdeguire`". Permette al dipendente o al manager di ottenere una lista in formato json delle task nello stato di "Da Eseguire". Gli elementi della rispecchiano gli attributi della risorsa "Task" vista precedentemente.

La risorsa richiede tutti i seguenti campi nel body:

- "token": token ottenuto dal microservizio autenticazione che identifica la sessione

Il corpo della risposta è in formato application/json e riporta le seguenti risposte:

Status Code	Body	Spiegazione
200 OK	{ [ task1, task2, ... ] }	La procedura è terminata con successo
400 BAD REQUEST	{error: "missing fields", missingFields }	I campi richiesti sono mancanti o malformati.
401 UNAUTHORIZED	{ error: "User not found with the given token" }	Il token non è valido
403 FORBIDDEN	{ error: "User not authorized" }	L'utente non ha permessi elevati per accedere all'API



Modello delle risorse per "/api/tasks/getListaTaskDaEseguire".

### Specifiche OpenAPI

```

openapi: 3.0.3
info:
  title: Get Lista Task Da Eseguire
  description: API getListaTaskDaEseguire
  version: 1.0.0
paths:
  /api/tasks/getListaTaskDaEseguire:
    post:
      summary: Ritorna tutte le tasks on lo stato di "Da Eseguire"
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                token:
                  type: string
                  description: User's authentication token
                  example: 171595638452613

```

```
        required:  
          - token  
  
responses:  
'200':  
  description: Successfully retrieved tasks in progress  
  content:  
    application/json:  
      schema:  
        type: array  
        items:  
          $ref: '#/components/schemas/TaskInLavorazione'  
'400':  
  description: Missing fields in the request  
  content:  
    application/json:  
      schema:  
        type: object  
        properties:  
          error:  
            type: string  
            example: missing fields  
          missingFields:  
            type: array  
            items:  
              type: string  
              example: token  
'401':  
  description: User not found with the given token  
  content:  
    application/json:  
      schema:  
        type: object  
        properties:  
          error:  
            type: string  
            example: user not found with the given token  
'403':
```

```
description: User not authorized
content:
  application/json:
    schema:
      type: object
      properties:
        error:
          type: string
      example: User not authorized
```

### Codice

Il codice è il seguente:

```
getListaTaskDaEseguireRouter.post('/', async (req, res) => {
  try {
    let token = getToken(req)
    if (token === undefined) {
      let e = { value: 'Missing fields', missingfields: ['token'] }
      dbg('ERROR', JSON.stringify(e))
      res.status(400)
      res.json(e)
      return
    }
    // Check authentication
    let profile = await getProfileInfo(token)
    if (profile === null) {
      let e = { value: 'User not found with the given token' }
      dbg('ERROR', JSON.stringify(e))
      res.status(401)
      res.json(e)
      return
    }
    // Check authorization
    if (!isAuthorized(profile)) {
      let e = { value: 'User not authorized' }
      dbg('ERROR', JSON.stringify(e))
      res.status(403)
      res.json(e)
      return
    }
    // Query the DB
    let allTasks = await executeQuery()
    if (allTasks === null) {
      let e = { value: 'Query error' }
      dbg('ERROR', JSON.stringify(e))
      res.status(400)
      res.json(e)
      return
    }
    dbg('AllTasks', JSON.stringify(allTasks))
    // OK
    // Return the tasks
    res.status(200)
    res.json(allTasks)
  } catch (e) {
    dbg('ERROR', e)
    res.status(400)
    res.json(e.message)
  }
})
```

Codice estratto da "backend/routes/getListaTaskDaEseguire".

```

● ● ●

async function executeQuery() {
  dbg('Executing query', '')

  return db.collection('tasks').find({ taskStatus: 'Da Eseguire'
}).toArray()
}

```

## Testing

Sono stati creati i seguenti test cases:

Numero Test- case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscon- trato
1	Nessun field	-	-	-	400	400
2	Invio Token non valido	{token: "fake" }	-	-	401	401
3	Invio Token da un cliente	{ token: to- kenCliente }	Esistenza di un cliente nel DB	Microservizio Autenti- cazione, DB	403	403
4	Invio token valido	{ token: to- kenManager }	Esistenza di un Manager e Task nel DB	Microservizio Autenti- cazione, DB	200	200

Di seguito il loro codice:

```
it('Sending request with missing fields, should return 400', async () => {
  const response = await request('10.5.0.12:3001').post(
    '/api/tasks/getListaTaskDaEseguire'
  )
  expect(response.status).toBe(400)
})
```

Testcase 1

```
it('Sending a wrong token, should return 401 user not found with given token', async () => {
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/getListaTaskDaEseguire')
    .type('form')
    .send({ token: 'fake' })
  expect(response.status).toBe(401)
})
```

Testcase 2

```
it('Sending a token from a Cliente, should return 403 not allowed',  
async () => {  
    // getting the token  
    const resLogin = await request('10.5.0.11:3001')  
        .post('/api/auth/login')  
        .type('form')  
        .send({ email: 'cliente@hotmail.com', password: 'test', twofa:  
'12345' })  
    let token = resLogin.body.token  
  
    // sending the request  
    const response = await request('10.5.0.12:3001')  
        .post('/api/tasks/getListaTaskDaEseguire')  
        .type('form')  
        .send({ token: token })  
    expect(response.status).toBe(403)  
})
```

### Testcase 3

```
it('Sending the right token, should return 200 OK', async () => {
  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({ email: 'manager@test.com', password: 'test', twofa: '12345' })
  let token = resLogin.body.token

  // sending the request
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/getListaTaskDaEseguire')
    .type('form')
    .send({ token: token })
  expect(response.status).toBe(200)
})
```

## Testcase 4

```
PASS backend/tests/getListaTaskDaEseguire.test.ts
Testing /api/tasks/getListaTaskDaEseguire
✓ Sending request with missing fields, should return 400 (4 ms)
✓ Sending a wrong token, should return 401 user not found with given token (6 ms)
✓ Sending a token from a Cliente, should return 403 not allowed (12 ms)
✓ Sending the right token, should return 200 OK (15 ms)
```

Risultato tests

### 3.1.8 Get Lista Task In Pausa

#### Specifiche

La risorsa "Get Lista Task In Pausa" è accessibile tramite la route

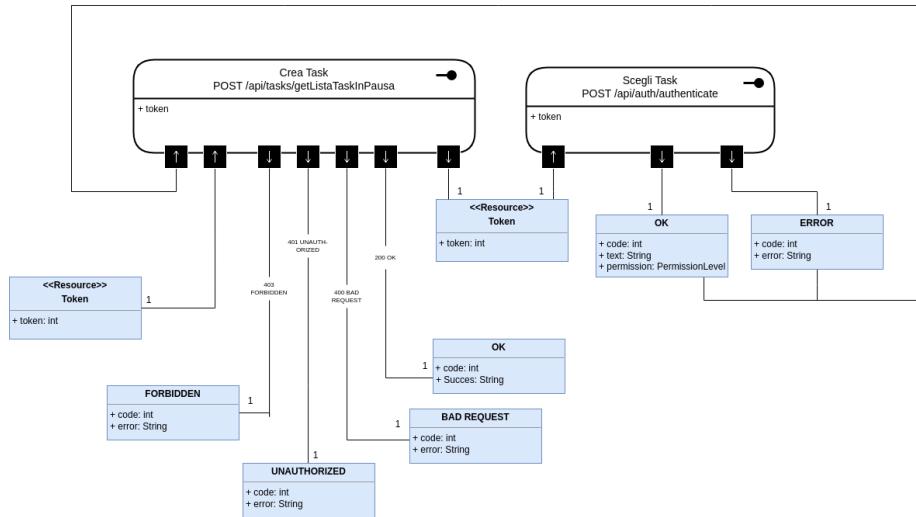
"/api/tasks/getListaTaskInPausa". Permette al dipendente o al manager di ottenere una lista in formato json delle task nello stato di "In Pausa". Gli elementi della rispecchiano gli attributi della risorsa "Task" vista precedentemente.

La risorsa richiede tutti i seguenti campi nel body:

- "token": token ottenuto dal microservizio autenticazione che identifica la sessione

Il corpo della risposta è in formato application/json e riporta le seguenti risposte:

Status Code	Body	Spiegazione
200 OK	{ [ task1, task2, ... ] }	La procedura è terminata con successo
400 BAD REQUEST	{error: "missing fields", missingFields }	I campi richiesti sono mancanti o malformati.
401 UNAUTHORIZED	{ error: "User not found with the given token" }	Il token non è valido
403 FORBIDDEN	{ error: "User not authorized" }	L'utente non ha permessi elevati per accedere all'API



Modello delle risorse per "/api/tasks/getListaTaskInPausa".

### Specifiche OpenAPI

```

openapi: 3.0.3
  info:
    title: Get Lista Task In Pausa
    edescription: API getListaTaskInPausa
  version: 1.0.0
  paths:
    /api/tasks/getListaTaskInPausa:
      post:
        summary: Ritorna tutte le tasks on lo stato di "In Pausa"
        requestBody:
          required: true
          content:
            application/json:
              schema:
                type: object
                properties:
                  token:
                    type: string
                    description: User's authentication token
                    example: 171595638452613
  
```

```
        required:  
          - token  
  
responses:  
'200':  
  description: Successfully retrieved tasks in progress  
  content:  
    application/json:  
      schema:  
        type: array  
        items:  
          $ref: '#/components/schemas/TaskInLavorazione'  
'400':  
  description: Missing fields in the request  
  content:  
    application/json:  
      schema:  
        type: object  
        properties:  
          error:  
            type: string  
            example: missing fields  
          missingFields:  
            type: array  
            items:  
              type: string  
              example: token  
'401':  
  description: User not found with the given token  
  content:  
    application/json:  
      schema:  
        type: object  
        properties:  
          error:  
            type: string  
            example: user not found with the given token  
'403':
```

```
description: User not authorized
content:
  application/json:
    schema:
      type: object
      properties:
        error:
          type: string
example: User not authorized
```

**Codice**

Il codice è quanto segue:

```
● ○ ●

getListaTaskInPausaRouter.post('/', async (req, res) => {
  try {
    let token = getToken(req)
    if (token === undefined) {
      let e = { value: 'Missing fields', missingfields: ['token'] }
      dbg('ERROR', JSON.stringify(e))
      res.status(400)
      res.json(e)
      return
    }
    // Check authentication
    let profile = await getProfileInfo(token)
    if (profile == null) {
      let e = { value: 'User not found with the given token' }
      dbg('ERROR', JSON.stringify(e))
      res.status(401)
      res.json(e)
      return
    }
    // Get the profile id
    let profileId = getProfileId(profile)
    let permission = getPermission(profile)
    // Check authorization
    if (!isAuthorized(profile)) {
      let e = { value: 'User not authorized' }
      dbg('ERROR', JSON.stringify(e))
      res.status(403)
      res.json(e)
      return
    }
    // Query the DB
    let allTasks = await executeQuery(profileId, permission)
    if (allTasks == null) {
      let e = { value: 'Query error' }
      dbg('ERROR', JSON.stringify(e))
      res.status(400)
      res.json(e)
      return
    }
    dbg('AllTasks', JSON.stringify(allTasks))
    // OK
    // Return the tasks
    res.json(allTasks)
  } catch (e) {
    dbg('ERROR', e)
    res.status(400)
    res.json(JSON.parse(e.message))
  }
})
```

Codice estratto da "backend/routes/getlistaTaskInPausa".

```

● ○ ●

async function executeQuery(profileId, permission) {
    dbg('Executing query', '')

    // The manager can see all the tasks in pause
    if (permission == 'Manager') {
        return db.collection('tasks').find({ taskStatus: 'In Pausa'
    }).toArray()
    }

    return db
        .collection('tasks')
        .find({ taskStatus: 'In Pausa', assignedTo: profileId })
        .toArray()
}

```

## Testing

Sono stati creati i seguenti test cases:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Nessun field	-	-	-	400	400
2	Invio Token non valido	{token: "fake" }	-	-	401	401
3	Invio Token da un cliente	{ token: tokenCliente }	Esistenza di un cliente nel DB	Microservizio Autenticazione, DB	403	403
4	Invio token valido	{ token: tokenManager }	Esistenza di un Manager e Task nel DB	Microservizio Autenticazione, DB	200	200

```
it('Sending request with missing fields, should return 400', async () => {
  const response = await request('10.5.0.12:3001').post('/api/tasks/getListaTaskInPausa')
  expect(response.status).toBe(400)
})
```

Testcase 1

```
it('Sending a wrong token, should return 401 user not found with given token', async () => {
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/getListaTaskInPausa')
    .type('form')
    .send({ token: 'fake' })
  expect(response.status).toBe(401)
})
```

Testcase 2

```
it('Sending a token from a Cliente, should return 403 not allowed',  
async () => {  
    // getting the token  
    const resLogin = await request('10.5.0.11:3001')  
        .post('/api/auth/login')  
        .type('form')  
        .send({ email: 'cliente@hotmail.com', password: 'test', twofa: '12345' })  
    let token = resLogin.body.token  
  
    // sending the request  
    const response = await request('10.5.0.12:3001')  
        .post('/api/tasks/getListaTaskInPausa')  
        .type('form')  
        .send({ token: token })  
    expect(response.status).toBe(403)  
})
```

### Testcase 3

```
it('Sending the right token, should return 200 OK', async () => {
  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({ email: 'manager@test.com', password: 'test', twofa: '12345' })
  let token = resLogin.body.token

  // sending the request
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/getListaTaskInPausa')
    .type('form')
    .send({ token: token })
  expect(response.status).toBe(200)
})
```

## Testcase 4

```
PASS backend/tests/getListaTaskInPausa.test.ts
Testing /api/tasks/getListaTaskInPausa
  ✓ Sending request with missing fields, should return 400 (2 ms)
  ✓ Sending a wrong token, should return 401 user not found with given token (6 ms)
  ✓ Sending a token from a Cliente, should return 403 not allowed (9 ms)
  ✓ Sending the right token, should return 200 OK (9 ms)
```

Risultato dei tests.

### 3.1.9 Get Storico Task

#### Specifiche

La risorsa "Get Storico Task" è accessibile tramite la route

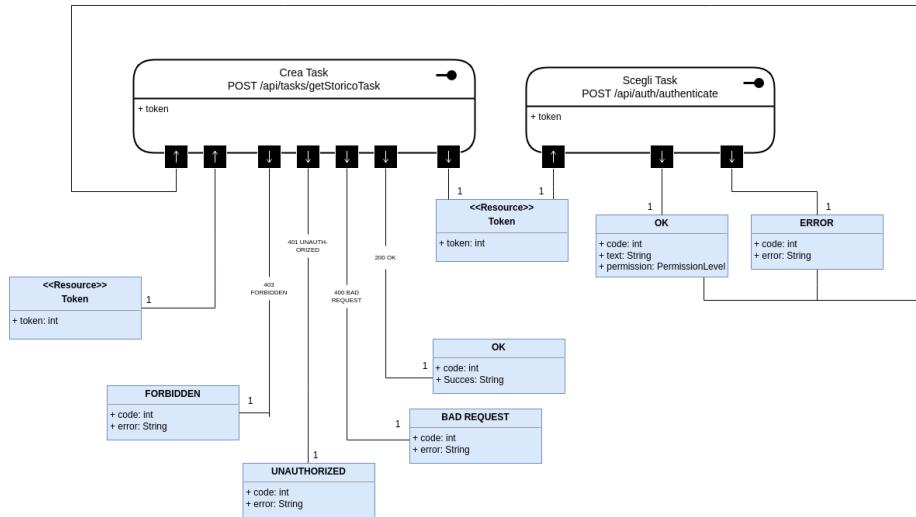
"/api/tasks/getStoricoTask". Permette al dipendente o al manager di ottenere una lista in formato json delle task nello stato di "Completata". Gli elementi della rispecchiano gli attributi della risorsa "Task" vista precedentemente. Il dipendente otterrà solo le task alle quali è stato assegnato, mentre il manager otterrà tutte le task completate presenti nel database.

La risorsa richiede tutti i seguenti campi nel body:

- "token": token ottenuto dal microservizio autenticazione che identifica la sessione

Il corpo della risposta è in formato application/json e riporta le seguenti risposte:

Status Code	Body	Spiegazione
200 OK	{ [ task1, task2, ... ] }	La procedura è terminata con successo
400 BAD REQUEST	{error: "missing fields", missingFields }	I campi richiesti sono mancanti o malformati.
401 UNAUTHORIZED	{ error: "User not found with the given token" }	Il token non è valido
403 FORBIDDEN	{ error: "User not authorized" }	L'utente non ha permessi elevati per accedere all'API



Modello delle risorse per `"/api/tasks/getStoricoTask"`.

### Specifiche OpenAPI

```

openapi: 3.0.3
  info:
    title: Get Storico Task
    description: API getStoricoTask
  version: 1.0.0
  paths:
    /api/tasks/getStoricoTask:
      post:
        summary: Ritorna tutte le tasks on lo stato di "Completate"
        requestBody:
          required: true
          content:
            application/json:
              schema:
                type: object
                properties:
                  token:
                    type: string
                    description: User's authentication token
                    example: 171595638452613
  
```

```
        required:  
          - token  
  
responses:  
'200':  
  description: Successfully retrieved tasks in progress  
  content:  
    application/json:  
      schema:  
        type: array  
        items:  
          $ref: '#/components/schemas/TaskInLavorazione'  
'400':  
  description: Missing fields in the request  
  content:  
    application/json:  
      schema:  
        type: object  
        properties:  
          error:  
            type: string  
            example: missing fields  
          missingFields:  
            type: array  
            items:  
              type: string  
              example: token  
'401':  
  description: User not found with the given token  
  content:  
    application/json:  
      schema:  
        type: object  
        properties:  
          error:  
            type: string  
            example: user not found with the given token  
'403':
```

```
description: User not authorized
content:
  application/json:
    schema:
      type: object
      properties:
        error:
          type: string
example: User not authorized
```

**Codice**

Il codice è il seguente:

```
● ○ ●

getStoricoTaskRouter.post('/', async (req, res) => {
  try {
    let token = getToken(req)
    if (token === undefined) {
      let e = { value: 'Missing fields', missingfields: ['token'] }
      dbg('ERROR', JSON.stringify(e))
      res.status(400)
      res.json(e)
      return
    }
    // Check authentication
    let profile = await getProfileInfo(token)
    if (profile == null) {
      let e = { value: 'User not found with the given token' }
      dbg('ERROR', JSON.stringify(e))
      res.status(401)
      res.json(e)
      return
    }
    // Get the profile id
    let profileId = getProfileId(profile)
    let permission = getPermission(profile)
    // Check authorization
    if (!isAuthorized(profile)) {
      let e = { value: 'User not authorized' }
      dbg('ERROR', JSON.stringify(e))
      res.status(403)
      res.json(e)
      return
    }
    // Query the DB
    let allTasks = await executeQuery(profileId, permission)
    if (allTasks === null) {
      let e = { value: 'Query error' }
      dbg('ERROR', JSON.stringify(e))
      res.status(400)
      res.json(e)
      return
    }
    dbg('AllTasks', JSON.stringify(allTasks))
    // OK
    // Return the tasks
    res.json(allTasks)
  } catch (e) {
    dbg('ERROR', e)
    res.status(400)
    res.json(JSON.parse(e.message))
  }
});
```

Estratto di codice dal file "backend/routes/getStoricoTask".

```

● ○ ●

async function executeQuery(profileId, permission) {
  dbg('Executing query', '')

  // The manager can see all the tasks in lavorazione
  if (permission == 'Manager') {
    return db.collection('tasks').find({ taskStatus: 'Completata'
  }).toArray()
  }

  return db
    .collection('tasks')
    .find({ taskStatus: 'Completata', assignedTo: profileId })
    .toArray()
}

```

## Testing

Sono stati individuati i seguenti test cases:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Nessun field	-	-	-	400	400
2	Invio Token non valido	{token: "fake" }	-	-	401	401
3	Invio Token da un cliente	{ token: tokenCliente }	Esistenza di un cliente nel DB	Microservizio Autenticazione, DB	403	403
4	Invio token valido	{ token: tokenManager }	Esistenza di un Manager e Task nel DB	Microservizio Autenticazione, DB	200	200

```
it('Sending request with missing fields, should return 400', async () => {
  const response = await request('10.5.0.12:3001').post('/api/tasks/getStoricoTask')
  expect(response.status).toBe(400)
})
```

Testcase 1

```
it('Sending a wrong token, should return 401 user not found with given token', async () => {
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/getStoricoTask')
    .type('form')
    .send({ token: 'fake' })
  expect(response.status).toBe(401)
})
```

Testcase 2

```
● ○ ● ●  
it('Sending a token from a Cliente, should return 403 not allowed',  
async () => {  
    // getting the token  
    const resLogin = await request('10.5.0.11:3001')  
        .post('/api/auth/login')  
        .type('form')  
        .send({ email: 'cliente@hotmail.com', password: 'test', twofa:  
'12345' })  
    let token = resLogin.body.token  
  
    // sending the request  
    const response = await request('10.5.0.12:3001')  
        .post('/api/tasks/getStoricoTask')  
        .type('form')  
        .send({ token: token })  
    expect(response.status).toBe(403)  
})
```

Testcase 3

```
● ○ ● ●  
it('Sending the right token, should return 200 OK', async () => {  
    // getting the token  
    const resLogin = await request('10.5.0.11:3001')  
        .post('/api/auth/login')  
        .type('form')  
        .send({ email: 'manager@test.com', password: 'test', twofa:  
'12345' })  
    let token = resLogin.body.token  
  
    // sending the request  
    const response = await request('10.5.0.12:3001')  
        .post('/api/tasks/getStoricoTask')  
        .type('form')  
        .send({ token: token })  
    expect(response.status).toBe(200)  
})
```

Testcase 4

```
PASS backend/tests/getStoricoTask.test.ts
Testing /api/tasks/getStoricoTask
✓ Sending request with missing fields, should return 400 (3 ms)
✓ Sending a wrong token, should return 401 user not found with given token (5 ms)
✓ Sending a token from a Cliente, should return 403 not allowed (8 ms)
✓ Sending the right token, should return 200 OK (10 ms)
```

Testcase 4

### 3.1.10 Crea Task

#### Specifica

La risorsa "Crea Task" è accessibile tramite la route

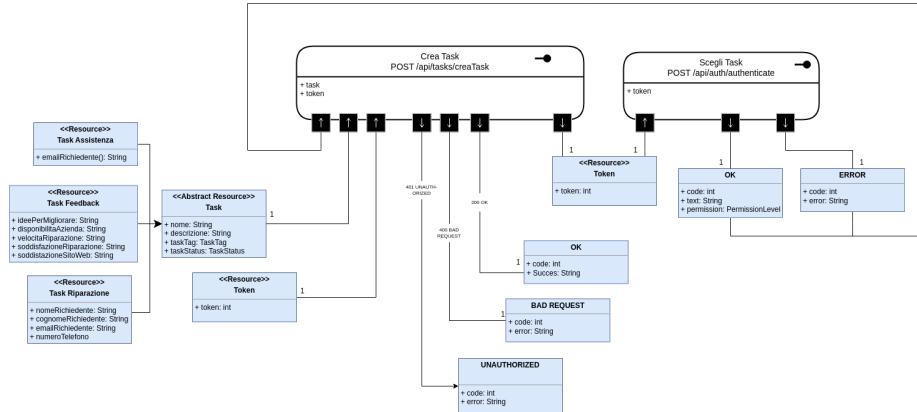
"/api/tasks/creaTask". Permette a chiunque sia autenticato di creare una task. L'oggetto task da creare deve essere compatibile con il costruttore di una delle classi che ereditano la risorsa "Task" definita precedentemente.

La risorsa richiede tutti i seguenti campi nel body:

- "token": token ottenuto dal microservizio autenticazione che identifica la sessione

Il corpo della risposta è in formato application/json e riporta le seguenti risposte:

Status Code	Body	Spiegazione
200 OK	{ Secces: "task created" }	La procedura è terminata con successo
400 BAD REQUEST	{error: "missing fields", missingFields }	I campi richiesti sono mancanti o malformati.
401 UNAUTHORIZED	{ error: "User not found with the given token" }	Il token non è valido
403 FORBIDDEN	{ error: "User not authorized" }	L'utente non ha permessi elevati per accedere all'API



Modello delle risorse per "/api/tasks/creaTask".

### Specifica OpenAPI

```

openapi: 3.0.3
  info:
    title: API Crea Task
    description: API pre /api/tasks/creaTask
    version: 1.0.0
  paths:
    /api/tasks/creaTask:
      post:
        summary: Create a task
        description: Questa route permette agli utenti autenticati di creare una nuova task
        requestBody:
          required: true
          content:
            application/json:
              schema:
                oneOf:
                  - $ref: '#/components/schemas/TaskAssistenza'
                  - $ref: '#/components/schemas/TaskFeedback'
                  - $ref: '#/components/schemas/TaskRiparazione'
                  - $ref: '#/components/schemas/TaskMagazzino'
                  - $ref: '#/components/schemas/TaskNegozio'
  
```

```
responses:
  '200':
    description: Successfully created the task
    content:
      application/json:
        schema:
          type: object
          properties:
            Success:
              type: string
              example: "class created"
  '400':
    description: Missing fields in the request
    content:
      application/json:
        schema:
          type: object
          properties:
            error:
              type: string
              example: missing fields
            missingFields:
              type: array
              items:
                type: string
  '401':
    description: User not found with the given token
    content:
      application/json:
        schema:
          type: object
          properties:
            error:
              type: string
              example: User not found
              with the given token
```

**Codice**

Il codice è il seguente:

```
creaTaskRouter.post('/', async (req, res) => {
  try {
    let token = getToken(req)
    if (token === undefined) {
      let e = { value: 'Missing fields', missingfields: ['token'] }
      dbg('ERROR', JSON.stringify(e))
      res.status(400)
      res.json(e)
      return
    }
    // Check authentication
    let profile = await getProfileInfo(token)
    if (profile === null) {
      let e = { value: 'User not found with the given token' }
      dbg('ERROR', JSON.stringify(e))
      res.status(401)
      res.json(e)
      return
    }
    let taskTag = getTaskTag(req)
    // Get the task
    let task = getTask(req, taskTag)
    dbg('Task', JSON.stringify(task.toJSON()))
    // Query the DB
    executeQuery(task)
      .then((ret) => {
        // OK
        dbg('DB response', JSON.stringify(ret))
        res.status(200)
        res.json({ Success: 'Task created' })

        // Errors
      })
      .catch((e) => {
        // DB error
        dbg('ERROR', e)
        res.status(400)
        res.json(JSON.parse(e.message))
      })
  } catch (e) {
    // Missing fields
    dbg('ERROR', e)
    res.status(400)
    res.json(JSON.parse(e.message))
  }
})
```

Estratto di codice dal file "backend/routes/creaTask".

```
● ○ ●

function getTaskTag(req) : TaskTag {
    let taskTagString = req.body.taskTag;
    dbg("TaskTag", taskTagString);

    // Check if the tasktag is missing
    let missingFields = getMissingFields([["taskTag", taskTagString]]);
    if (missingFields.length != 0) {
        let e = {'value': 'Missing fields', missingfields: missingFields};
        throw new JSONError(e);
    }

    if (!Object.values(TaskTag).includes(taskTagString as TaskTag)) {
        let e = {'value': 'Invalid taskTag', 'taskTag': taskTagString};
        throw new JSONError(e);
    }

    let taskTag = taskTagString as TaskTag;
    return taskTag;
}
```

```
function getTask(req, taskTag: TaskTag): Task {
    let missingFields = []
    switch (taskTag) {
        case TaskTag.Assistenza:
            missingFields = getMissingFields([
                ['nome', req.body.nome],
                ['descrizione', req.body.descrizione],
                ['emailRichiedente', req.body.emailRichiedente],
            ])

            if (missingFields.length != 0) {
                let e = { value: 'Missing fields', missingfields:
                    missingFields }
                throw new JSONError(e)
            }

            return new TaskAssistenza(
                req.body.nome,
                req.body.descrizione,
                req.body.emailRichiedente
            )
            break

        case TaskTag.Feedback:
            missingFields = getMissingFields([
                ['nome', req.body.nome],
                ['descrizione', req.body.descrizione],
                ['ideePerMigliorare', req.body.ideePerMigliorare],
                ['disponibilitaAzienda', req.body.disponibilitaAzienda],
                ['velocitaRiparazione', req.body.velocitaRiparazione],
                ['soddisfazioneRiparazione',
                    req.body.soddisfazioneRiparazione],
                ['soddisfazioneSitoWeb', req.body.soddisfazioneSitoWeb],
            ])

            if (missingFields.length != 0) {
                let e = { value: 'Missing fields', missingfields:
                    missingFields }
                throw new JSONError(e)
            }

            return new TaskFeedback(
                req.body.nome,
                req.body.descrizione,
                req.body.ideePerMigliorare,
                req.body.disponibilitaAzienda,
                req.body.velocitaRiparazione,
                req.body.soddisfazioneRiparazione,
                req.body.soddisfazioneSitoWeb
            )
            break
    ...
}
```

```
● ● ●

async function executeQuery(task) {
  dbg('Executing query', '')

  // Used for testing purposes
  if (task.getName() == 'justatest') return

  return db.collection('tasks').insertOne(task.toJSON())
}
```

## Testing

Sono stati individuati i seguenti test cases:

Numerico Test- case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscon- trato
1	Nessun field	-	-	-	400	400
2	Invio Token non valido	{token: "fake" }	-	-	401	401
3	Invio token senza task tag	{ token: validToken }	Esistenza Manager nel DB	Microservizio Autenti- cazione, DB	400	400
4	Invio token con task tag sbagliato	{ token: validToken, taskTag: "fake" }	Esistenza Manager nel DB	Microservizio Autenti- cazione, DB	400	400
5	Invio Token con task tag ma senza al- tri attributi	{ token: to- kenCliente, taskTag: "Magazzino" }	Esistenza di un cliente nel DB	Microservizio Autenti- cazione, DB	400	400
6	Tutto valido	{ token: to- kenManager, taskTak: "Magazz- ino", nome: "test", de- scrizione: "test" }	Esistenza di un Manager nel DB	Microservizio Autenti- cazione, DB	200	200

Codice dei testcases:



```
it('Sending request with missing fields, should return 400', async () => {
  const response = await request('10.5.0.12:3001').post('/api/tasks/creaTask')
  expect(response.status).toBe(400)
})
```

Testcase 1



```
it('Sending request with wrong token, should return 401', async () => {
  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/creaTask')
    .type('form')
    .send({ token: 'fake' })
  expect(response.status).toBe(401)
})
```

Testcase 2

```
● ○ ●

it('Sending right token but no task tag, should return 400 missing
fileds', async () => {
  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({ email: 'manager@test.com', password: 'test', twofa:
'12345' })
  let token = resLogin.body.token

  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/creaTask')
    .type('form')
    .send({ tasktag: token })
  expect(response.status).toBe(400)
})
```

Testcase 3

```
● ○ ●

it('Sending right token but a wrong task tag, should return 400
tasktag not found', async () => {
  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({ email: 'manager@test.com', password: 'test', twofa:
'12345' })
  let token = resLogin.body.token

  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/creaTask')
    .type('form')
    .send({ tasktag: token, taskTag: 'fake' })
  expect(response.status).toBe(400)
})
```

Testcase 4

```
● ○ ●

it('Sending a good task tag but without enouth attributes, should return 400 not enough attributes', async () => {
  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({ email: 'manager@test.com', password: 'test', twofa: '12345' })
  let token = resLogin.body.token

  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/creaTask')
    .type('form')
    .send({ token: token, taskTag: 'Magazzino', nome: 'test' })
  expect(response.status).toBe(400)
})
```

Testcase 5

```
● ○ ●

it('Sending everything right, should return 200 OK', async () => {
  // getting the token
  const resLogin = await request('10.5.0.11:3001')
    .post('/api/auth/login')
    .type('form')
    .send({ email: 'manager@test.com', password: 'test', twofa: '12345' })
  let token = resLogin.body.token

  const response = await request('10.5.0.12:3001')
    .post('/api/tasks/creaTask')
    .type('form')
    .send({
      token: token,
      taskTag: 'Magazzino',
      nome: 'justatest',
      descrizione: 'test',
    })
  expect(response.status).toBe(200)
})
```

Testcase 6

```
PASS backend/tests/creaTask.test.ts
Testing /api/tasks/creaTask
  ✓ Sending request with missing fields, should return 400 (3 ms)
  ✓ Sending request with wrong token, should return 401 (6 ms)
  ✓ Sending right token but no task tag, should return 400 missing fileds (5 ms)
  ✓ Sending right token but a wrong task tag, should return 400 tasktag not found (6 ms)
  ✓ Sending a good task tag but without enouth attributes, should return 400 not enough attributes (8 ms)
  ✓ Sending everything right, should return 200 OK (11 ms)
```

Risultato testcases

## 3.2 Front-End

Questa sezione comprende lo sviluppo e la documentazione del Front-End del Microservizio Task

### 3.2.1 Struttura del Front-End

La struttura del Front-End del microservizio task è riportata nella figura sotto.

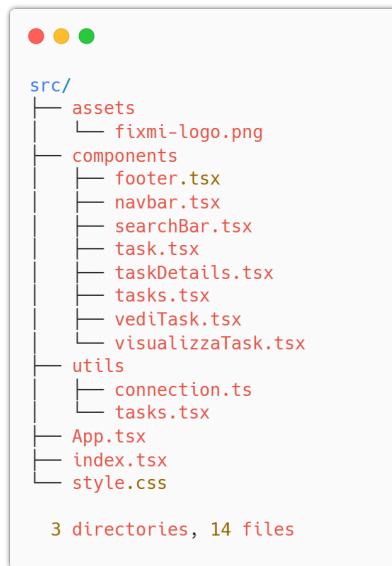


Figure 3.1: Output del comando "tree" all'interno del microservizio task.

```
import { BrowserRouter, Route, Routes } from 'react-router-dom';
import React from 'react';
import VisualizzaTask from './components/visualizzaTask';
import TaskDetails from './components/TaskDetails.tsx';

function App() {
  return (
    <BrowserRouter basename='tasks/'>
      <main>
        <Routes>
          {/* Routing */}
          <Route path="/" element={<VisualizzaTask />} />
          <Route path="task-details" element={<TaskDetails />} />
          <Route path="*" element={<VisualizzaTask />} />
        </Routes>
      </main>
    </BrowserRouter>
  );
}

export default App;
```

Codice del componente "App.tsx"

### Specifiche

Tutte le componenti facenti parte del microservizio "Task" vengono gestite dalla componente "App.tsx", quest'ultimo stabilisce gli indirizzi di ciascuna pagina del microservizio e quale componente deve gestire l'indirizzo.

#### 3.2.2 UserFlow

Lo userflow del microservizio task viene mostrato sotto nel suo intero. Verrà successivamente suddiviso ed analizzato in ciascun suo componente.

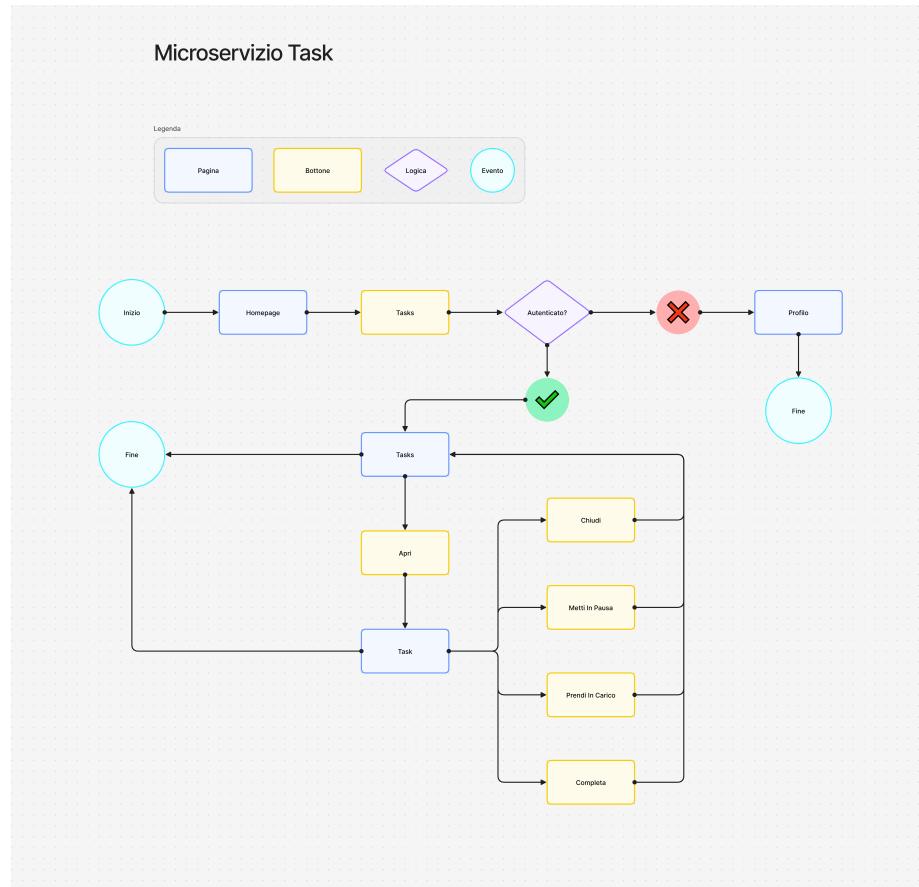


Figure 3.2: UserFlow del microservizio "Task"

### 3.2.3 Tasks

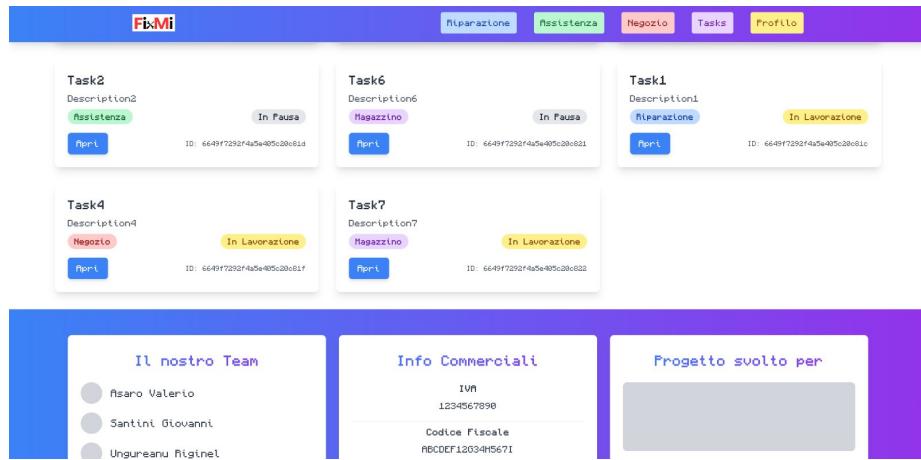


Figure 3.3: Schermata per le tasks

#### Specifica

La pagina si occupa di mostrare l'intera lista di tasks presenti all'interno dell'applicazione. E' possibile cercare determinate tasks in base alla loro categoria o al loro stato selezionandoli tramite i due menu "dropdown" mostrati sotto:

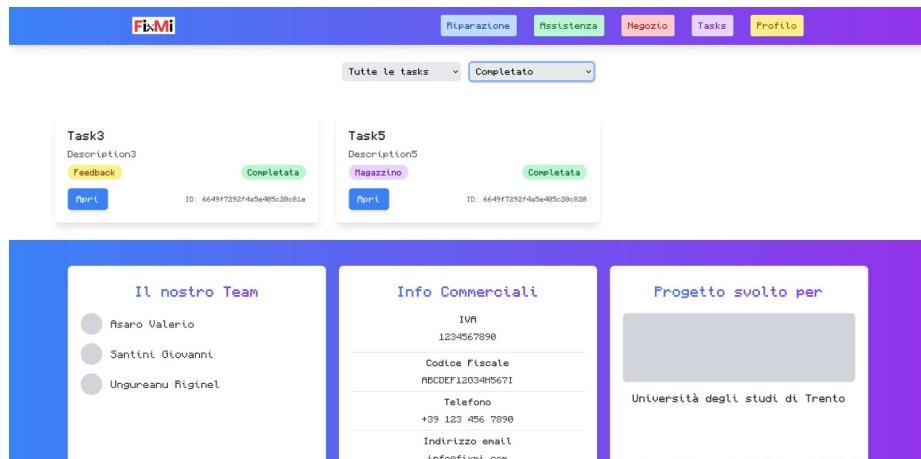


Figure 3.4: I due menu "DropDown", che mostrano solo la lista di task con lo stato "Completato"

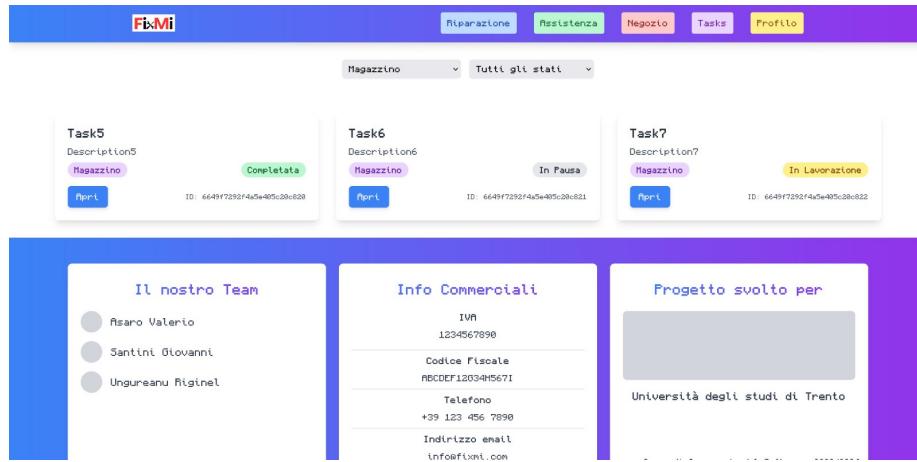
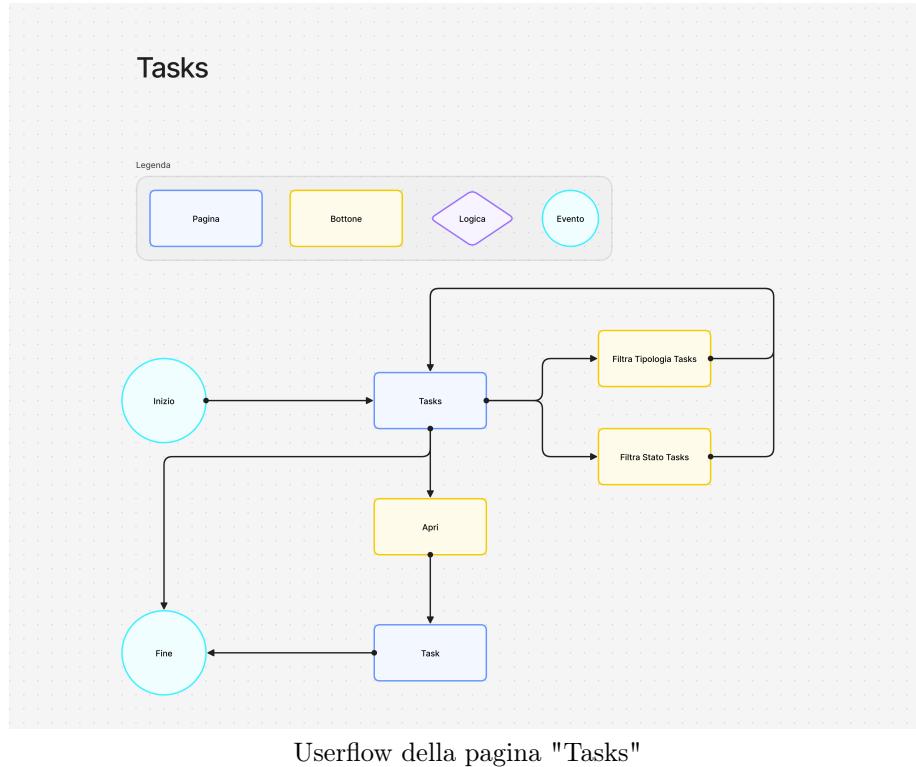


Figure 3.5: I due menu "DropDown", che mostrano solo la lista di task di tipologia "Magazzino"

E' possibile visualizzare i dettagli di ciascuna task cliccando il bottone blu "Apri" della task di cui si è interessati

### Userflow

Partendo dalla pagina "Homepage" il dipendente o manager autenticato devono cliccare il tasto "Tasks" presente nella "navbar".



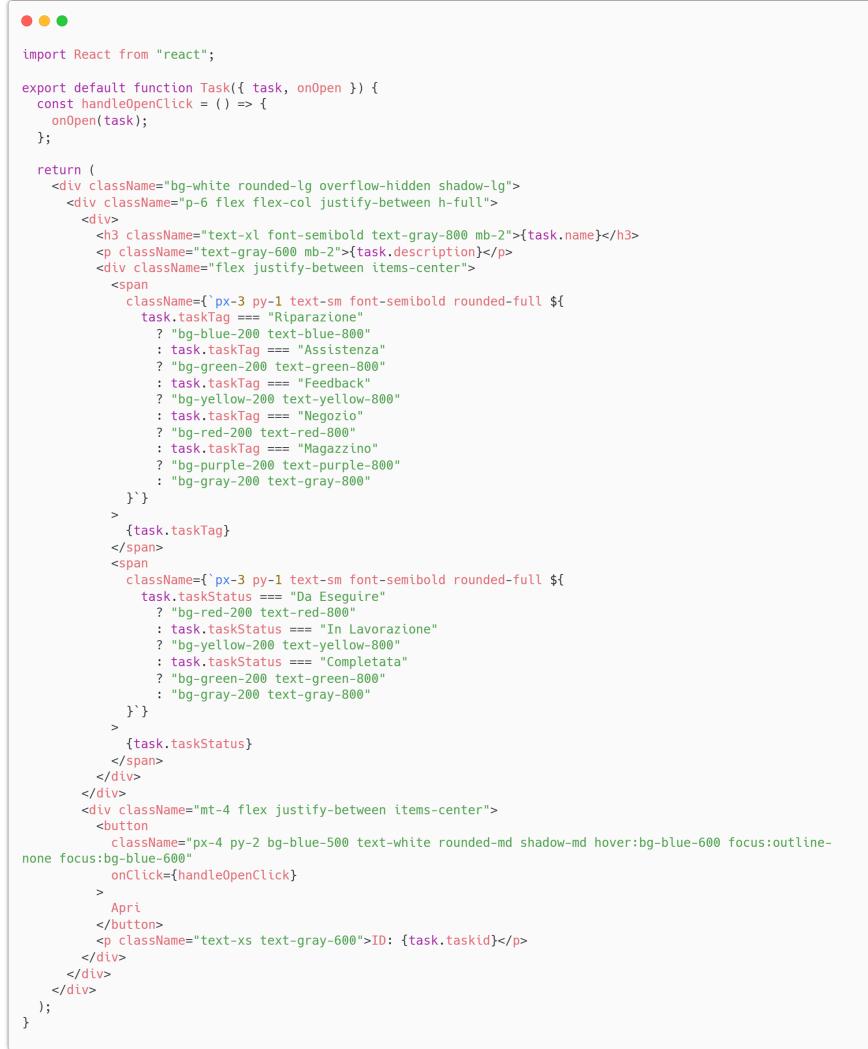
Userflow della pagina "Tasks"

### Codice

La pagina "Tasks" è composta da quattro componenti chiamati "visualizzaTask.tsx", "tasks.tsx", "task.tsx" e "searchBar.tsx" con le seguenti caratteristiche:

- visualizzaTask.tsx
  - Contiene al suo interno "tasks.tsx" e search
  - Contiene controlli e "Fallbacks" per eventuali errori
- tasks.tsx
  - Contiene al suo interno "task.tsx" e "searchBar.tsx"
  - Funge da contenitore ed interlocutore tra i vari componenti del microservizio
- task.tsx
  - Si occupa di mostrare una carta contenente i principali dettagli della task per ciascuna task da visualizzare.

- Contiene un bottone blu "Apri" che visualizza i dettagli della task selezionata



```

import React from "react";

export default function Task({ task, onOpen }) {
  const handleOpenClick = () => {
    onOpen(task);
  };

  return (
    <div className="bg-white rounded-lg overflow-hidden shadow-lg">
      <div className="p-6 flex flex-col justify-between h-full">
        <div>
          <h3 className="text-xl font-semibold text-gray-800 mb-2">{task.name}</h3>
          <p className="text-gray-600 mb-2">{task.description}</p>
          <div className="flex justify-between items-center">
            <span>
              className={`px-3 py-1 text-sm font-semibold rounded-full ${task.taskTag === "Riparazione"
                ? "bg-blue-200 text-blue-800"
                : task.taskTag === "Assistenza"
                ? "bg-green-200 text-green-800"
                : task.taskTag === "Feedback"
                ? "bg-yellow-200 text-yellow-800"
                : task.taskTag === "Negozio"
                ? "bg-red-200 text-red-800"
                : task.taskTag === "Magazzino"
                ? "bg-purple-200 text-purple-800"
                : "bg-gray-200 text-gray-800"
              }`}
            >
              {task.taskTag}
            </span>
            <span>
              className={`px-3 py-1 text-sm font-semibold rounded-full ${task.taskStatus === "Da Eseguire"
                ? "bg-red-200 text-red-800"
                : task.taskStatus === "In Lavorazione"
                ? "bg-yellow-200 text-yellow-800"
                : task.taskStatus === "Completata"
                ? "bg-green-200 text-green-800"
                : "bg-gray-200 text-gray-800"
              }`}
            >
              {task.taskStatus}
            </span>
          </div>
        </div>
        <div className="mt-4 flex justify-between items-center">
          <button
            className="px-4 py-2 bg-blue-500 text-white rounded-md shadow-md hover:bg-blue-600 focus:outline-none focus:bg-blue-600"
            onClick={handleOpenClick}>
            Apri
          </button>
          <p className="text-xs text-gray-600">ID: {task.taskid}</p>
        </div>
      </div>
    </div>
  );
}

```

Codice del componente "task.tsx"

- searchBar.tsx

- Si occupa di filtrare la lista di task mostrate.
- Contiene due menu "Dropdown" in cui è possibile scegliere la tipologia e lo stato che le task devono possedere per essere mostrate

```
● ● ●

import React, { useState } from 'react';

export default function SearchBar({ onTypeFilter, onStatusFilter }) {
  const [selectedTaskType, setSelectedTaskType] = useState('Tutti');
  const [selectedStatus, setSelectedStatus] = useState('Tutti');

  const handleTypeChange = (event) => {
    const selectedValue = event.target.value;
    setSelectedTaskType(selectedValue);
    onTypeFilter(selectedValue);
  };

  const handleStatusChange = (event) => {
    const selectedValue = event.target.value;
    setSelectedStatus(selectedValue);
    onStatusFilter(selectedValue);
  };

  return (
    <div className="flex justify-center py-8">
      <div className="w-full max-w-lg md:w-auto flex items-center space-x-4">
        <select
          value={selectedTaskType}
          onChange={handleTypeChange}
          className="block w-auto py-2 pl-3 pr-10 text-base border rounded-md shadow-sm focus:outline-none focus:ring
          focus:border-blue-500"
        >
          <option value="Tutti">Tutte le tasks</option>
          <option value="Assistenza">Assistenza</option>
          <option value="Feedback">Feedback</option>
          <option value="Riparazione">Riparazione</option>
          <option value="Negozio">Negozio</option>
          <option value="Magazzino">Magazzino</option>
        </select>
        <select
          value={selectedStatus}
          onChange={handleStatusChange}
          className="block w-auto py-2 pl-3 pr-10 text-base border rounded-md shadow-sm focus:outline-none focus:ring
          focus:border-blue-500"
        >
          <option value="Tutti">Tutti gli stati</option>
          <option value="Completata">Completato</option>
          <option value="In Lavorazione">In Lavorazione</option>
          <option value="Da Eseguire">Da Eseguire</option>
          <option value="In Pausa">In Pausa</option>
        </select>
      </div>
    </div>
  );
}
```

Codice del componente "searchBar.tsx"

### 3.2.4 Task

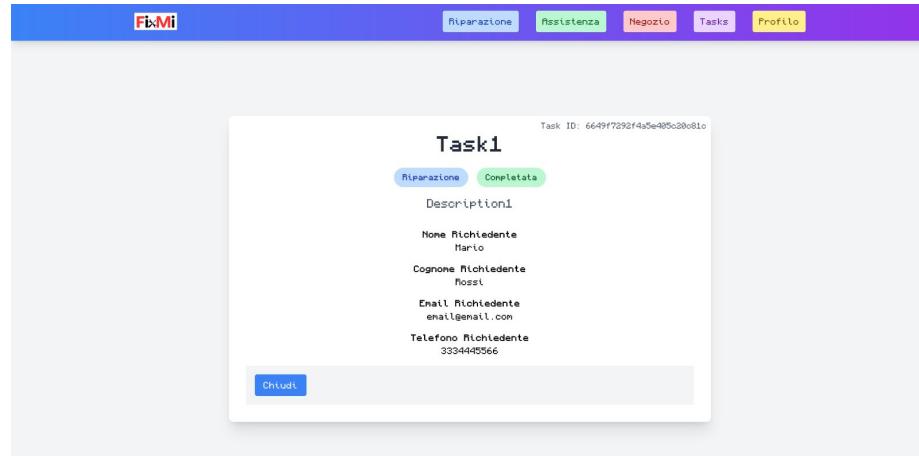


Figure 3.6: Schermata per la task in stato "Completato"

#### Specifiche

La pagina si occupa di mostrare tutti i dettagli legati ad una singola task.

La pagina assume diversi aspetti in base allo stato della task che si sta visualizzando.

E' sempre possibile tornare alla pagina "Tasks" cliccando il bottone blu "Chiudi"

E' possibile, se lo stato della Task lo permette, cliccare i seguenti bottoni:

- Bottone arancione "Metti in Pausa" per mettere in pausa la task
- Bottone verde "Completa" per completare la task

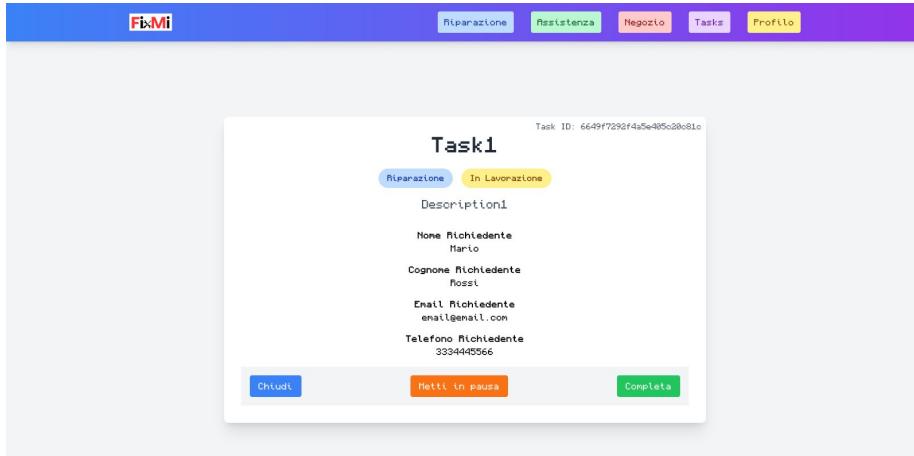


Figure 3.7: Schermata per la task in stato "In Lavorazione"

- Bottone arancione "Prendi in Carico" per prendere in carico la task

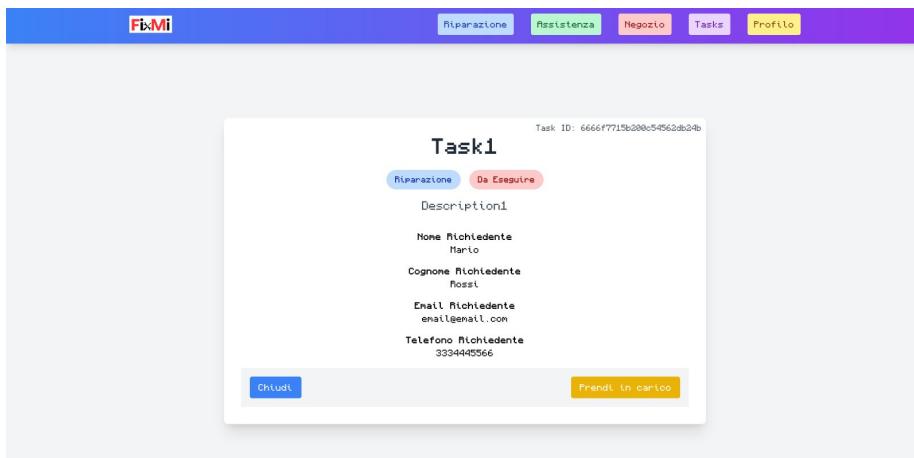


Figure 3.8: Schermata per la task in stato "Da Eseguire"

- Bottone verde "Riprendi" per riprendere la task

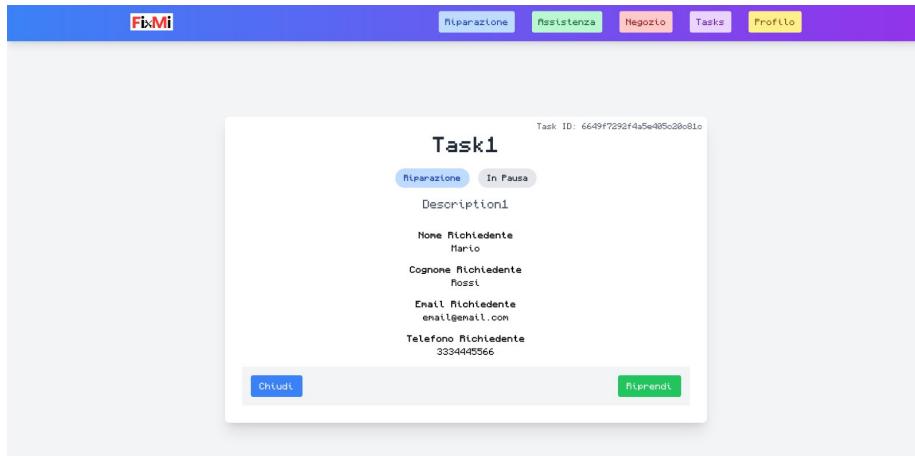
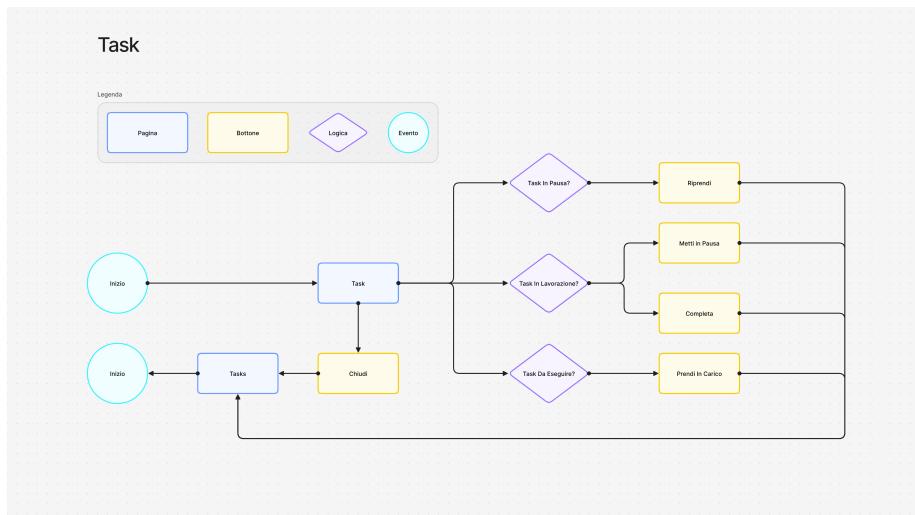


Figure 3.9: Schermata per la task in stato "In Pausa"

### Userflow

Partendo dalla pagina "Homepage" il dipendente o manager autenticato devono cliccare il tasto "Tasks" presente nella "navbar", dopodichè devono scegliere una Task e cliccare il corrispettivo bottone blu "Apri".



Userflow della pagina "Task"

**Codice**

La pagina "Task" è composta da due componenti chiamati "TaskDetails.tsx" e "vediTask.tsx" con le seguenti caratteristiche:

- TaskDetails.tsx
  - Contiene al suo interno "veditask.tsx"
  - Contiene controlli e "Fallbacks" per eventuali errori
- vediTask.tsx
  - Si occupa di mostrare una carta contenente i principali dettagli della task.
  - Contiene sempre un bottone blu "Chiudi" che ritorna alla pagina "Tasks".

---

CHAPTER

**FOUR**

---

## MICROSERVIZIO GESTIONE DIPENDENTI

### 4.1 Back-End

Questa sezione comprende lo sviluppo e la documentazione del Back-End del Microservizio "Gestione Dipendenti"

#### **Struttura del Back-End**

La struttura della Back-End del microservizio "Gestione Dipendenti" è riportata nella figura sotto.

```
backend/
├── classes
│   └── Profilo.ts
├── enums
│   ├── PermissionLevel.ts
│   └── TaskTag.ts
├── routes
│   ├── TwoFARoutes.ts
│   ├── authRoutes.ts
│   ├── greetRoutes.ts
│   ├── loginRoutes.ts
│   ├── logoutRoutes.ts
│   ├── passwordRoutes.ts
│   ├── registerRoutes.ts
│   ├── removeRoutes.ts
│   └── testDBRouter.ts
├── server.ts
└── tests
    ├── login.test.ts
    ├── logout.test.ts
    ├── password.test.ts
    ├── register.test.ts
    ├── remove.test.ts
    └── twofa.test.ts
└── utils
    ├── hash.ts
    ├── missingFields.ts
    └── token.ts

5 directories, 22 files
```

Figure 4.1: Output del comando "tree" all'interno di un microservizio.

## Modellazione dati nel database

### Specifiche delle Risorse

Di seguito le risorse estratte dal diagramma delle classi e utilizzate da questo microservizio.

**«resource» Dipendente** Rappresenta il profilo di un Dipendente. Ha tutti gli attributi di cliente, in più:

- nome

- cognome
- data di assunzione
- data di nascita
- worktags

**Manager** Rappresenta il profilo del Manager. Ha tutti gli attributi di Dipendente, ma ha come permissionLevel "Manager"

Metodi

**«resource» crea** Permette al manager di creare un profilo dipendente. E' un'API della Back-End

**«resource» rimuovi** Permette al manager di rimuovere un profilo dipendente E' un'API della Back-End

**«resource» cerca** Permette al manager di cercare profili dipendente E' un'API della Back-End

**«resource» storico** Permette al manager di visualizzare lo storico task di un determinato dipendente E' un'API della Back-End

#### 4.1.1 Crea profilo Dipendente

##### Specifiche

Questa API, all'indirizzo /api/dipendenti/create, ha un metodo POST e viene utilizzata per la creazione di un profilo dipendente da parte del Manager. La request body è in formato x-www-form-urlencoded e deve contenere i seguenti campi:

- email
- password
- data di nascita
- data di assunzione
- lista worktag

Inoltre, per verificare che il manager sia autenticato, il suo token va inserito o all'interno del body o attraverso un cookie. La response body è in formato application/json. Di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{text: "successfully created the profile"}	Profilo dipendente creato con successo
401 UNAUTHORIZED	{error: "missing token"}	Il token non è presente nella richiesta
403 FORBIDDEN	{error: "permission denied"}	Il token è presente nella richiesta, ma l'utente associato non è un manager
404 NOT FOUND	{error: "No user found with the given token"}	Il token è presente nella richiesta, ma non corrisponde a nessun profilo autenticato
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
400 BAD REQUEST	{error: "user already exists with the given email"}	Esiste già un utente con la mail fornita
400 BAD REQUEST	{error: "invalid birth date" }	La data di nascita fornita non è in un formato riconosciuto
400 BAD REQUEST	{error: "invalid assunzione date" }	La data di assunzione fornita non è in un formato riconosciuto
400 BAD REQUEST	{error: "no worktag array" }	il campo della richiesta 'worktag' non è un array
400 BAD REQUEST	{error: "elements provided are not worktags" }	Gli elementi forniti all'interno di 'worktag' non sono worktags

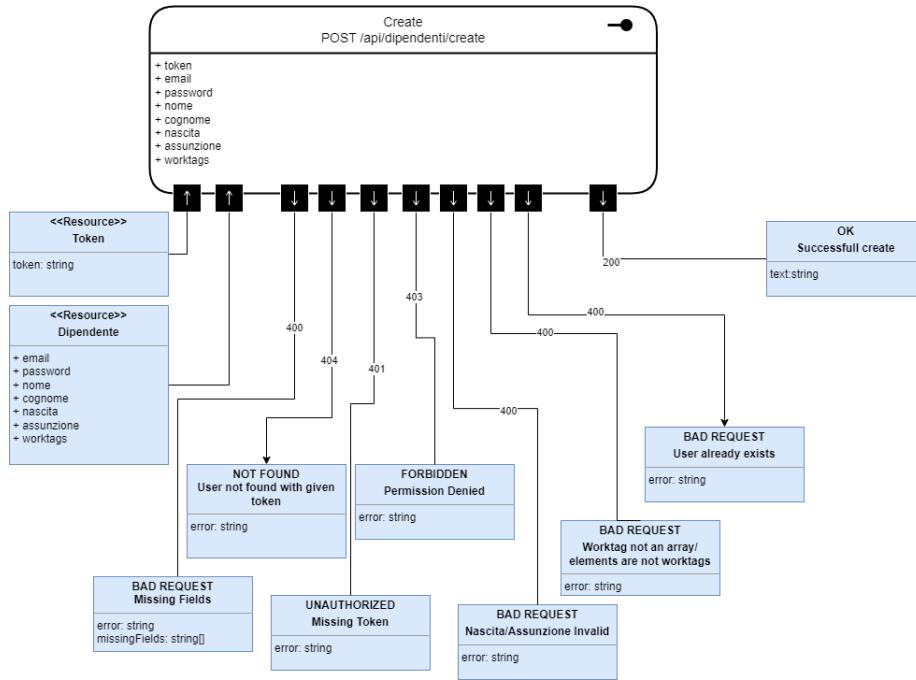


Figure 4.2: Diagramma dell'endpoint create

### Specifica OpenAPI

Di seguito la specifica openapi dell'endpoint remove contenuta nel file 'openapi.yaml' della directory del microservizio

```

/create:
  post:
    summary: Metodo per la creazione di profili dipendente da parte del manager
    description: Questo endpoint permette al manager di creare un profilo dipendente fornendo i campi richiesti.
    requestBody:
      required: true
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:
              email:
                type: string

```

```
        format: email

    password:
        type: string
        format: password

    nome:
        type: string
    cognome:
        type: string
    nascita:
        type: string
        format: date
    assunzione:
        type: string
        format: date
    worktag:
        type: array
    token:
        type: string

    required:
        - email
        - password
        - nome
        - cognome
        - nascita
        - assunzione
        - worktag
        - token

responses:
'200':
    description: "Profilo creato con successo"
    content:
        application/json:
            schema:
                type: object
```

```
        properties:
          text:
            type: string
            example: "successfully created"

      '401':
        description: "Token mancante nella richiesta"
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/missingTokenSchema"
      '403':
        description: "Il token non corrisponde al token del manager"
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/notTheManagerSchema"

      '404':
        description: "Il token non corrisponde a nessun utente"
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/notFoundSchema"
      '400':
        description: Bad Request - 'Missing fields' or 'invalid birth date' or 'invalid as
        content:
          application/json:
            schema:
              oneOf:
                - $ref: '#/components/schemas/missingFieldsSchema'
                - type: object
                  properties:
                    text:
                      type: string
```

```

example:
oneOf:
  - "invalid birth date"
  - "invalid assunzione date"
- type: object
  properties:
    text:
      type: string
example:
oneOf:
  - "no worktag array"
  - "elements provided are not worktags"
- $ref: '#/components/schemas/alreadyExistsSchema'

```

### Codice

Questa api viene utilizzata dal manager per la creazione di un profilo dipendente. Una volta ricevuta la richiesta, viene controllata inizialmente la presenza del token del manager nel body o nel token attraverso il middleware 'permissionMiddleware'. Successivamente viene controllata la presenza di tutti i campi richiesti e dell'integrità dei dati forniti, ossia:

- Se la data di nascita è in un formato valido
- Se la data di assunzione è in un formato valido
- se 'worktag' è un array
- se gli elementi di 'worktag' sono esclusivamente worktags

Dopodichè viene controllata la presenza di un utente con la mail fornita nel database. Infine viene creato il dipendente e memorizzato all'interno del database, ritornando una risposta positiva.

```

createRouter.use(permissionCheck(PermissionLevel.Manager));

createRouter.post("/", async (req, res) => {
  let fields = new Map([
    ["email", req.body.email],
    ["password", req.body.password],
    ["nome", req.body.nome],
    ["cognome", req.body.cognome],
    ["nascita", req.body.nascita],
    ["assunzione", req.body.assunzione],
    ["worktag", req.body.worktag],
  ]);

  let missingFields = getMissingFields(Array.from(fields.entries()));
  //missing fields: returns an error
  if(missingFields.length!=0) {
    res.status(400);
    res.json({error: "missing fields", missingFields});
    return;
  }

  const nascita = new Date(fields.get("nascita"));
  const assunzione = new Date(fields.get("assunzione"));
  if(isNaN(nascita.getDate())){
    res.status(400);
    res.json({error: "invalid birth date"});
    return;
  }
  if(isNaN(assunzione.getDate())){
    res.status(400);
    res.json({error: "invalid assunzione date"});
    return;
  }
  if(!Array.isArray(fields.get("worktag"))){
    res.status(400);
    res.json({error: "you haven't provided an array to worktag"});
    return;
  }
  console.log(fields.get("worktag"));
  let worktag= toTaskTagArray(fields.get("worktag"));
  if(worktag==null){
    res.status(400);
    res.json({error: "elements provided are not worktags"});
    return;
  }
  const lookupEmail = await
db_users.collection("users").findOne({email:fields.get("email")});
  if(lookupEmail != null){
    res.status(400);
    res.json({error: "user with given email already exists!"});
    return;
  }
  const dipendente = new
Dipendente(fields.get("email"),fields.get("password"),fields.get("nome")
),fields.get("cognome"),nascita,assunzione,worktag);
  await db_users.collection("users").insertOne(dipendente);

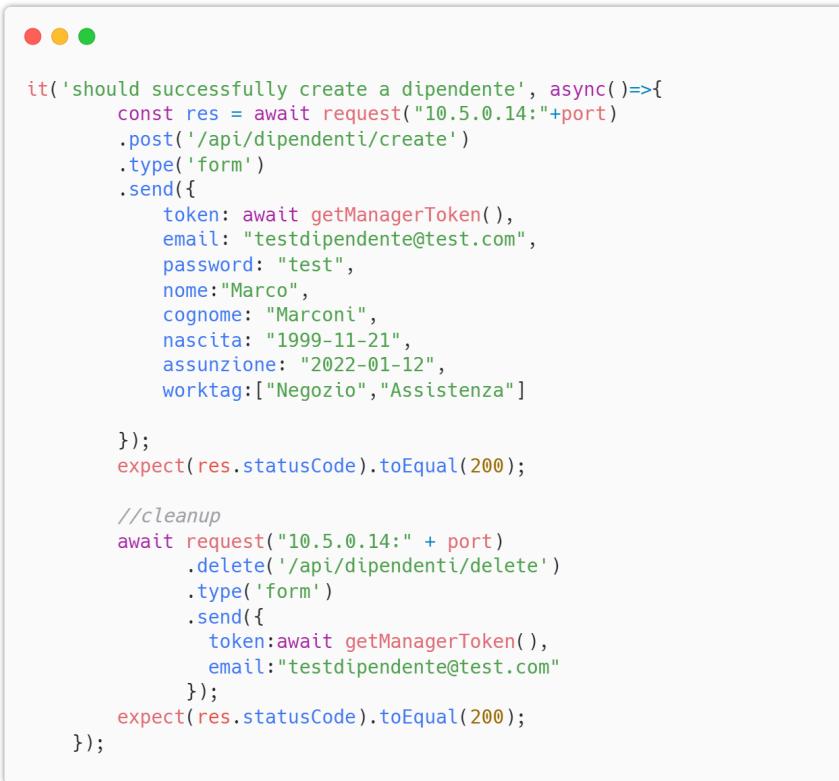
  res.json({text: "successfully created a Dipendente profile!",
dipendente: dipendente})
})

```

### Testing

Per questo endpoint sono stati creati i seguenti test:

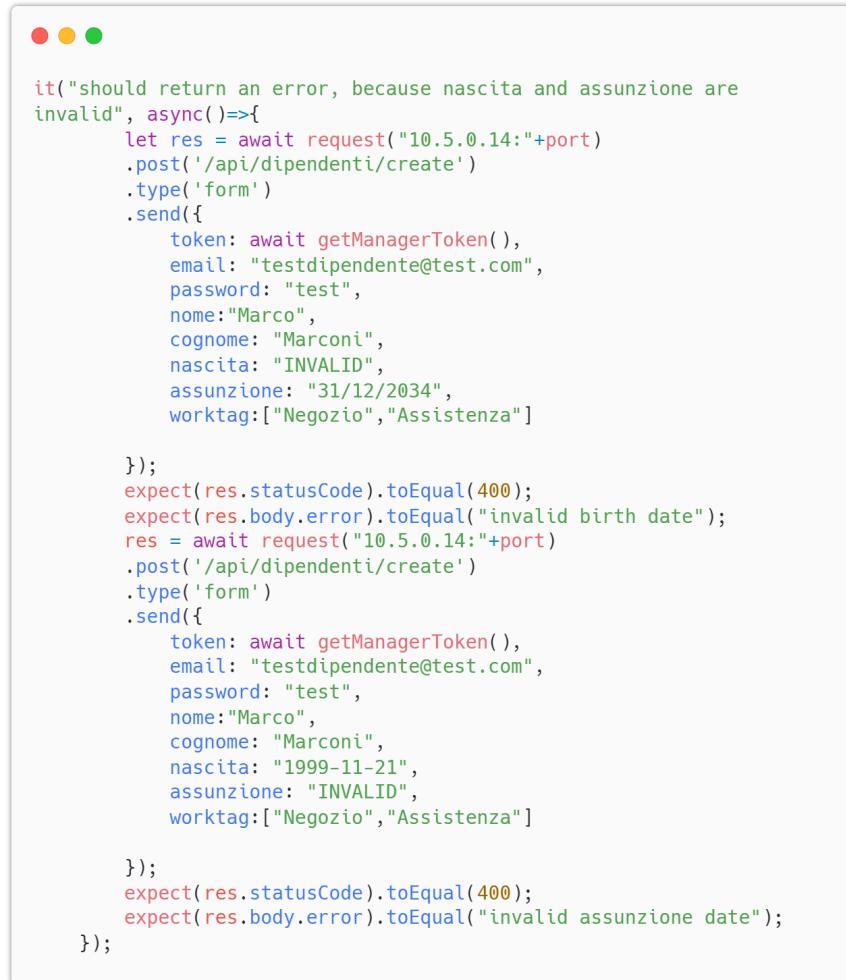
Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Dipendente creato con successo	{email, password, nome, cognome, nascita, assunzione, worktag}	utente non esistente	Microservizio Autenticazione,DB	200	200
2	Nascita e Assunzione non validi	{email, password, nome, cognome, nascita, assunzione, worktag}		Microservizio Autenticazione,DB	400	400
3	Elementi forniti non sono worktags	{email, password, nome, cognome, nascita, assunzione, worktag}		Microservizio Autenticazione,DB	400	400



```
it('should successfully create a dipendente', async()=>{
    const res = await request("10.5.0.14:"+port)
    .post('/api/dipendenti/create')
    .type('form')
    .send({
        token: await getManagerToken(),
        email: "testdipendente@test.com",
        password: "test",
        nome:"Marco",
        cognome: "Marconi",
        nascita: "1999-11-21",
        assunzione: "2022-01-12",
        worktag:["Negozio","Assistenza"]
    });
    expect(res.statusCode).toEqual(200);

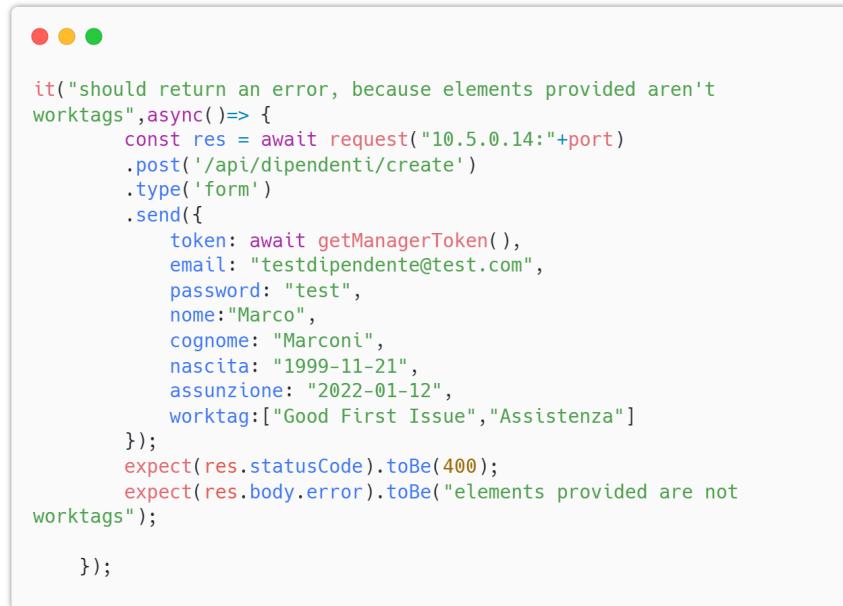
    //cleanup
    await request("10.5.0.14:" + port)
        .delete('/api/dipendenti/delete')
        .type('form')
        .send({
            token:await getManagerToken(),
            email:"testdipendente@test.com"
        });
    expect(res.statusCode).toEqual(200);
});
```

Figure 4.4: Test Case 1



```
it("should return an error, because nascita and assunzione are invalid", async()=>{
    let res = await request("10.5.0.14:"+port)
    .post('/api/dipendenti/create')
    .type('form')
    .send({
        token: await getManagerToken(),
        email: "testdipendente@test.com",
        password: "test",
        nome:"Marco",
        cognome: "Marconi",
        nascita: "INVALID",
        assunzione: "31/12/2034",
        worktag:["Negozio","Assistenza"]
    });
    expect(res.statusCode).toEqual(400);
    expect(res.body.error).toEqual("invalid birth date");
    res = await request("10.5.0.14:"+port)
    .post('/api/dipendenti/create')
    .type('form')
    .send({
        token: await getManagerToken(),
        email: "testdipendente@test.com",
        password: "test",
        nome:"Marco",
        cognome: "Marconi",
        nascita: "1999-11-21",
        assunzione: "INVALID",
        worktag:["Negozio","Assistenza"]
    });
    expect(res.statusCode).toEqual(400);
    expect(res.body.error).toEqual("invalid assunzione date");
});
```

Figure 4.5: Test Case 2



The screenshot shows a terminal window with three colored window control buttons (red, yellow, green) at the top. The main area contains a block of Node.js code. The code is a test case using the Mocha framework and the Chai assertion library. It checks if an error is returned when required fields are missing. The test uses the `request` module to send a POST request to the '/api/dipendenti/create' endpoint with a 'form' type. The request body contains user data with various fields. The test expects a `statusCode` of 400 and the `body.error` message to be "elements provided are not worktags". The code is color-coded with red for strings, green for variables, blue for functions, and black for comments and keywords.

```
it("should return an error, because elements provided aren't worktags",async()=> {
    const res = await request("10.5.0.14:"+port)
        .post('/api/dipendenti/create')
        .type('form')
        .send({
            token: await getManagerToken(),
            email: "testdipendente@test.com",
            password: "test",
            nome:"Marco",
            cognome: "Marconi",
            nascita: "1999-11-21",
            assunzione: "2022-01-12",
            worktag:["Good First Issue","Assistenza"]
        });
    expect(res.statusCode).toBe(400);
    expect(res.body.error).toBe("elements provided are not worktags");
});
```

Figure 4.6: Test Case 3



```
/app # npm test create.test.ts
> fixmi-microservice-template@1.0.0 test
> jest create.test.ts

PASS  backend/tests/create.test.ts
  create testing
    ✓ should successfully create a dipendente (104 ms)
    ✓ should return an error, because nascita and assunzione are
      invalid (18 ms)
      ✓ should return an error, because elements provided aren't worktags
        (9 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        0.441 s, estimated 1 s
Ran all test suites matching /create.test.ts/i.);
```

Figure 4.7: Risultati dei test

#### 4.1.2 Elimina profilo

##### Specifica

Questa API, all'indirizzo /api/dipendenti/delete, ha un metodo delete e viene utilizzata per l'eliminazione di un profilo da parte del Manager. La request body è in formato x-www-form-urlencoded e deve contenere i seguenti campi:

- email

Inoltre, per verificare che il manager sia autenticato, il suo token va inserito o all'interno del body o attraverso un cookie. La response body è in formato application/json. Di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{text: "successfully deleted the profile"}	Profilo eliminato con successo
401 UNAUTHORIZED	{error: "missing token"}	Il token non è presente nella richiesta
403 FORBIDDEN	{error: "permission denied"}	Il token è presente nella richiesta, ma l'utente associato non è un manager
404 NOT FOUND	{error: "No user found with the given token"}	Il token è presente nella richiesta, ma non corrisponde a nessun profilo autenticato
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
404 NOT FOUND	{error: "no user with the given email" }	Non esiste alcun utente con la mail fornita

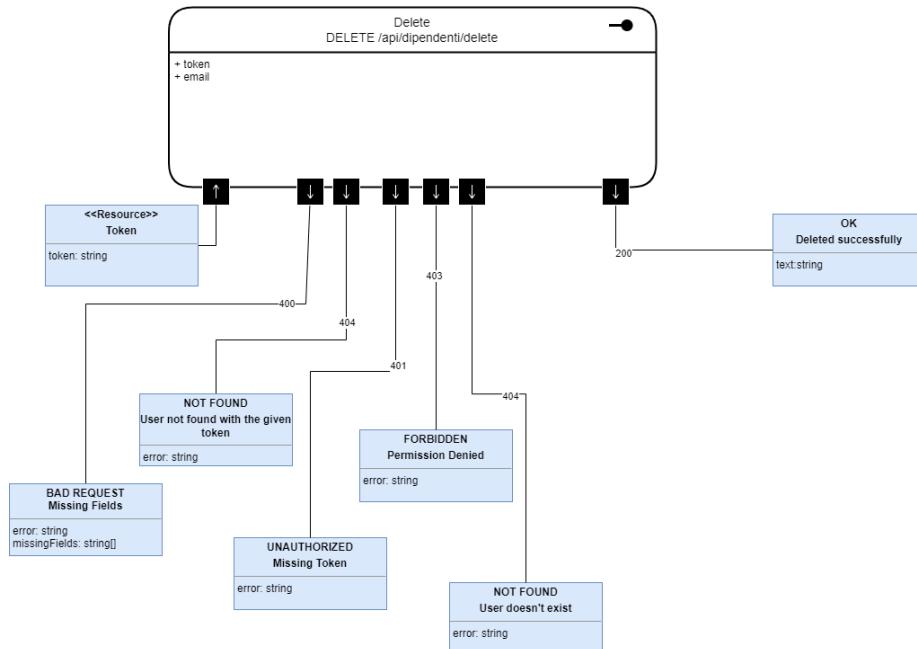


Figure 4.8: Diagramma dell'endpoint delete

**Specifica OpenAPI**

Di seguito la specifica openapi dell'endpoint remove contenuta nel file 'openapi.yaml' della directory del microservizio

```
/delete:  
  delete:  
    summary: Metodo per l'eliminazione di un profilo  
    description: Questo endpoint permette al manager di eliminare un profilo fornendo la s  
    requestBody:  
      required: true  
      content:  
        application/x-www-form-urlencoded:  
          schema:  
            type: object  
            properties:  
              email:  
                type: string  
                format: email  
              token:  
                type: string  
  
            required:  
              - email  
              - token  
    responses:  
      '200':  
        description: "Profilo eliminato con successo con successo"  
        content:  
          application/json:  
            schema:  
              type: object  
              properties:  
                text:  
                  type: string  
                example: "successfully deleted the profile"
```

```
'401':
  description: "Token mancante nella richiesta"
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/missingTokenSchema"
'403':
  description: "Il token non corrisponde al token del manager"
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/notTheManagerSchema"

'404':
  description: "Il token non corrisponde a nessun utente" OR 'Nessun utente trovato'
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/notFoundSchema"
'400':
  description: Bad Request - 'Missing fields'
  content:
    application/json:
      schema:
        oneOf:
          - $ref: '#/components/schemas/missingFieldsSchema'
```

### Codice

Questa api viene utilizzata dal manager per eliminare un profilo già esistente nel sistema. Una volta ricevuta la richiesta, viene controllata inizialmente la presenza del token del manager nel body o nel token attraverso il middleware 'permissionMiddleware'. Successivamente viene controllata la presenza di tutti i campi richiesti. Dopodichè viene controllato se l'email fornita corrisponde al profilo di un utente. Infine viene eliminato il profilo all'interno del database,

ritornando un esito positivo

```

● ● ●

deleteRouter.use(permissionCheck(PermissionLevel.Manager));

deleteRouter.delete("/", async (req, res) => {
  const fields = new Map([
    ["email", req.body.email],
  ]);

  const missingFields =
    getMissingFields(Array.from(fields.entries()));
    //missing fields: returns an error
  if(missingFields.length != 0) {
    res.status(400);
    res.json({error: "missing fields", missingFields});
  }
  const user = await
    db_users.collection("users").findOne({email: fields.get("email")});
  if(user == null){
    res.status(404);
    res.json({error: "user not found"});
    return;
  }
  await db_users.collection("users").deleteOne({id: user.id});
  res.json({text: "successfully deleted the profile!"});

}

}

const dipendente = new
Dipendente(fields.get("email"), fields.get("password"), fields.get("nome"),
fields.get("cognome"), nascita, assunzione, worktag);
await db_users.collection("users").insertOne(dipendente);

res.json({text: "successfully created a Dipendente profile!",
dipendente: dipendente})

})

```

Figure 4.9: Codice dell'endpoint delete

## Testing

Per questo endpoint sono stati creati i seguenti test:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Dipendente eliminato con successo	{email}	utente creato con la richiesta create	Microservizio Autenticazione,DB	200	200
2	utente non trovato	{email}		Microservizio Autenticazione,DB	404	404

```

● ● ●

it('should successfully delete a dipendente', async()=>{
    let rand = Math.floor(Math.random() * 30);
    const res = await request("10.5.0.14:"+port)
    .post('/api/dipendenti/create')
    .type('form')
    .send({
        token: await getManagerToken(),
        email: "testdipendente"+rand+"@test.com",
        password: "test",
        nome:"Marco",
        cognome: "Marconi",
        nascita: "1999-11-21",
        assunzione: "2022-01-12",
        worktag:["Negozio","Assistenza"]
    });
    //cleanup
    let cleanup_res = await request("10.5.0.14:" + port)
    .delete('/api/dipendenti/delete')
    .type('form')
    .send({
        token:await getManagerToken(),
        email:"testdipendente"+rand+"@test.com"
    });
    expect(res.statusCode).toEqual(200);

    expect(cleanup_res.statusCode).toEqual(200);
});

```

Figure 4.10: Test Case 1

```
it("should return an error, because user not found", async()=>{
    const res = await request(`10.5.0.14:${port}`)
    .delete('/api/dipendenti/delete')
    .type('form')
    .send({
        token: await getManagerToken(),
        email: "cthulhu@test.com",
        password: "test",
        nome: "Marco",
        cognome: "Marconi",
        nascita: "29/11/2002",
        assunzione: "31/12/2034",
        worktag: ["Negozio", "Assistenza"]
    });
    expect(res.statusCode).toEqual(404);
    expect(res.body.error).toEqual("user not found");
});
```

Figure 4.11: Test Case 1

```
/app # npm test delete.test.ts
> fixmi-microservice-template@1.0.0 test
> jest delete.test.ts

PASS  backend/tests/delete.test.ts
  delete testing
    ✓ should successfully delete a dipendente (34 ms)
    ✓ should return an error, because user not found (7 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:  0 total
Time:        0.348 s, estimated 1 s
Ran all test suites matching /delete.test.ts/i.
```

Figure 4.12: Risultati dei test

## Cerca profili

### Specifica

Questa API, all'indirizzo /api/dipendenti/find, ha un metodo post e viene utilizzata per la ricerca di uno o più profili da parte del manager. La request body è in formato x-www-form-urlencoded e deve contenere i seguenti campi:

- mode

il campo mode può assumere i valori 'one' o 'many', e indica se la ricerca debba ritornare uno o più profili. Il body può contenere, ma non è obbligatorio, i seguenti campi per una ricerca più specifica:

- email
- nome
- cognome
- data di nascita
- data di assunzione
- worktags

Inoltre, per verificare che il manager sia autenticato, il suo token va inserito o all'interno del body o attraverso un cookie. La response body è in formato application/json. Di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{user: ...}	La ricerca con mode = "one" ha riportato un risultato
200 OK	{users: [user1,user2,user3...]}	La ricerca con mode = 'many' ha riportato uno o più risultati
401 UNAUTHORIZED	{error: "missing token"}	Il token non è presente nella richiesta
403 FORBIDDEN	{error: "permission denied"}	Il token è presente nella richiesta, ma l'utente associato non è un manager
404 NOT FOUND	{error: "No user found with the given token"}	Il token è presente nella richiesta, ma non corrisponde a nessun profilo autenticato
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
404 NOT FOUND	{error: "no user found" }	Non esiste alcun utente avente i parametri di ricerca forniti. viene ritornato solo con mode = one

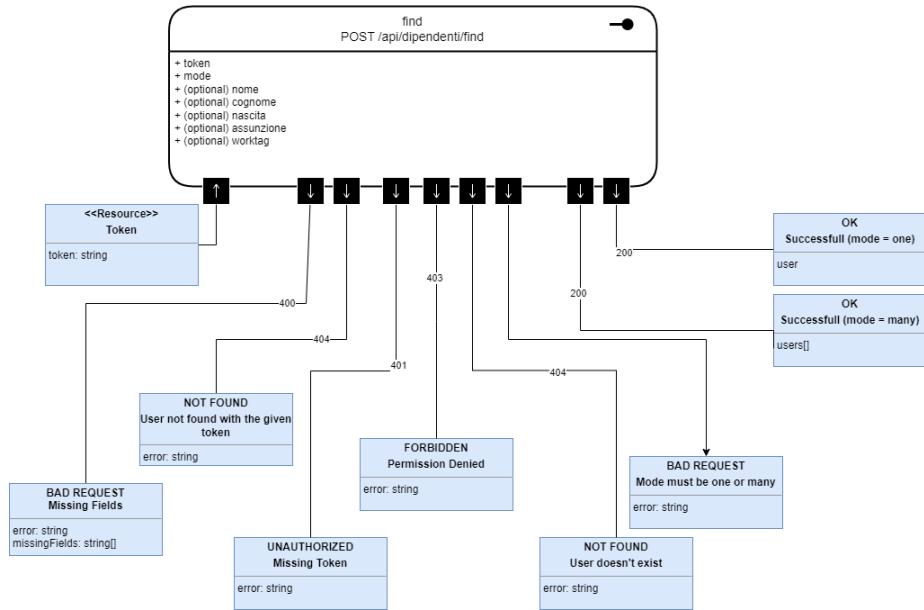


Figure 4.13: Diagramma dell'endpoint find

### Specifica OpenAPI

Di seguito la specifica openapi dell'endpoint remove contenuta nel file 'openapi.yaml' della directory del microservizio

```
/find:
  post:
    summary: Metodo per la ricerca di profili dipendente da parte del manager
    description: Questo endpoint permette al manager di cercare profili dipendente fornendo i criteri di ricerca.
    requestBody:
      required: true
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:
              email:
                type: string
                format: email
```

```
        password:  
          type: string  
          format: password  
  
        nome:  
          type: string  
        cognome:  
          type: string  
        nascita:  
          type: string  
          format: date  
        assunzione:  
          type: string  
          format: date  
        worktag:  
          type: array  
        token:  
          type: string  
        mode:  
          type: string  
          enum: [one,many]  
        required:  
          - token  
          - mode  
  
      responses:  
        '200':  
          description: "(mode=one) Profilo trovato OR (mode=many) Profili trovati "  
          content:  
            application/json:  
              schema:  
                oneOf:  
                  - $ref: '#components/schemas/user'  
                  - $ref: '#components/schemas/users'  
        '401':  
          description: "Token mancante nella richiesta"  
          content:  
            application/json:
```

```
        schema:
          $ref: "#/components/schemas/missingTokenSchema"
'403':
  description: "Il token non corrisponde al token del manager"
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/notTheManagerSchema"

'404':
  description: "Il token non corrisponde a nessun utente OR (mode = one) Utente non
content:
  application/json:
    schema:
      $ref: "#/components/schemas/notFoundSchema"
'400':
  description: Bad Request - 'Missing fields'
  content:
    application/json:
      schema:
        oneOf:
        - $ref: '#/components/schemas/missingFieldsSchema'
```

### Codice

Questa api viene utilizzata dal manager per cercare uno o più profili presenti nel sistema. Una volta ricevuta la richiesta, viene controllata inizialmente la presenza del token del manager nel body o nel token attraverso il middleware 'permissionMiddleware'. Successivamente viene controllata la presenza di tutti i campi richiesti. Dopodichè vengono presi tutti i campi opzionali forniti e inseriti nella query di ricerca nel database. Infine il server risponde con il primo profilo trovato che corrisponde ai criteri di ricerca forniti.

```
● ○ ●

findRouter.use(permissionCheck(PermissionLevel.Manager));
findRouter.post("/", async (req, res) => {
  const fields = new Map([
    ["mode", req.body.mode],
  ]);
  const missingFields =
    getMissingFields(Array.from(fields.entries()));
  //missing fields: returns an error
  if(missingFields.length != 0) {
    res.status(400);
    res.json({error: "missing fields", missingFields});
    return;
  }

  const schema = removeBlankAttributes({
    email: req.body.email,
    nome: req.body.nome,
    cognome: req.body.cognome,
    dataDiNascita: req.body.nascita,
    dataDiAssunzione: req.body.assunzione,
    workTags: req.body.worktag,
    permissionLevel: req.body.permissionLevel
  });
  if( fields.get("mode") == "one"){
    const user = await
    db_users.collection("users").findOne(schema);
    if(user == null){
      res.status(404);
      res.json({error:"user not found", search_criteria:
schema});
      return;
    }
    res.json({user: user});
    return;
  }
  if(fields.get("mode") == "many"){
    const user = await
    db_users.collection("users").find(schema).toArray();
    res.json({users: user});
    return;
  }
  //mode is not one or many
  if(fields["mode"] != "one" || fields["mode"] != "many"){
    res.status(400);
    res.json({error: "mode field must be 'one' or 'many' "});
  }
}
```

Figure 4.14: Codice dell'endpoint find

## Testing

Per questo endpoint sono stati creati i seguenti test:

Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Dipendente trovato con successo	{email, mode=one}	utente esistente	Microservizio Autenticazione,DB	200	200
2	Mode non è 'one' o 'many'	{email, mode}		Microservizio Autenticazione,DB	400	400
3	Utente non trovato	{email, mode=one}	utente non esistente	Microservizio Autenticazione,DB	404	404

```
● ● ●

it('should successfully find an user', async()=>{
  const res = await request("10.5.0.14:"+port)
    .post('/api/dipendenti/find')
    .type('form')
    .send({
      token: await getManagerToken(),
      email: "manager@test.com",
      mode: "one"
    });
  expect(res.statusCode).toEqual(200);
});
```

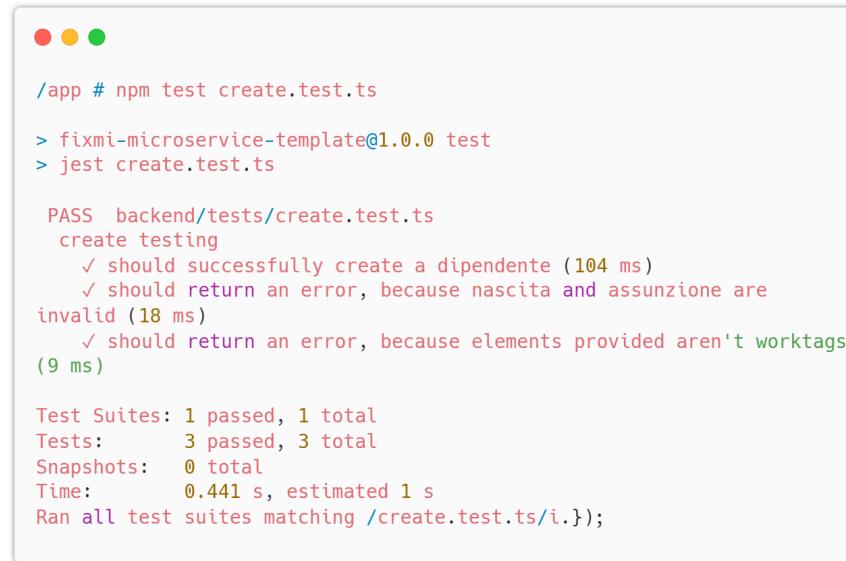
Figure 4.15: Test Case 1

```
it('should return an error, because mode is not one or many',  
  async()=>{  
    const res = await request("10.5.0.14:"+port)  
      .post('/api/dipendenti/find')  
      .type('form')  
      .send({  
        token: await getManagerToken(),  
        email: "manager@test.com",  
        mode: "boh"  
      });  
    expect(res.statusCode).toEqual(400);  
  });
```

Figure 4.16: Test Case 2

```
it("should return an error, because user not found", async()=>{  
  const res = await request("10.5.0.14:"+port)  
    .post('/api/dipendenti/find')  
    .type('form')  
    .send({  
      token: await getManagerToken(),  
      email: "cthulhu@test.com",  
      mode: "one"  
    });  
  expect(res.statusCode).toEqual(404);  
  expect(res.body.error).toEqual("user not found");  
});
```

Figure 4.17: Test Case 3



```
/app # npm test create.test.ts
> fixmi-microservice-template@1.0.0 test
> jest create.test.ts

PASS  backend/tests/create.test.ts
  create testing
    ✓ should successfully create a dipendente (104 ms)
    ✓ should return an error, because nascita and assunzione are
      invalid (18 ms)
      ✓ should return an error, because elements provided aren't worktags
        (9 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.441 s, estimated 1 s
Ran all test suites matching /create.test.ts/i.);
```

Figure 4.18: Risultati dei test

## Storico profili

### Specifiche

Questa API, all'indirizzo /api/dipendenti/history, ha un metodo post e viene utilizzata per la visione dello storico task di un determinato dipendente. La request body è in formato x-www-form-urlencoded e deve contenere i seguenti campi:

- email

Inoltre, per verificare che il manager sia autenticato, il suo token va inserito o all'interno del body o attraverso un cookie. La response body è in formato application/json. Di seguito le possibili risposte:

Status Code	Body e Cookie	Spiegazione
200 OK	{[task1,task2,task3...]}	La ricerca ha riportato lo storico task con successo
401 UNAUTHORIZED	{error: "missing token"}	Il token non è presente nella richiesta
403 FORBIDDEN	{error: "permission denied"}	Il token è presente nella richiesta, ma l'utente associato non è un manager
404 NOT FOUND	{error: "No user found with the given token"}	Il token è presente nella richiesta, ma non corrisponde a nessun profilo autenticato
400 BAD REQUEST	{error: "missing fields", missingFields}	Alcuni campi non sono stati specificati nella request
404 NOT FOUND	{error: "dipendente not found" }	Non esiste alcun dipendente avente la mail fornita

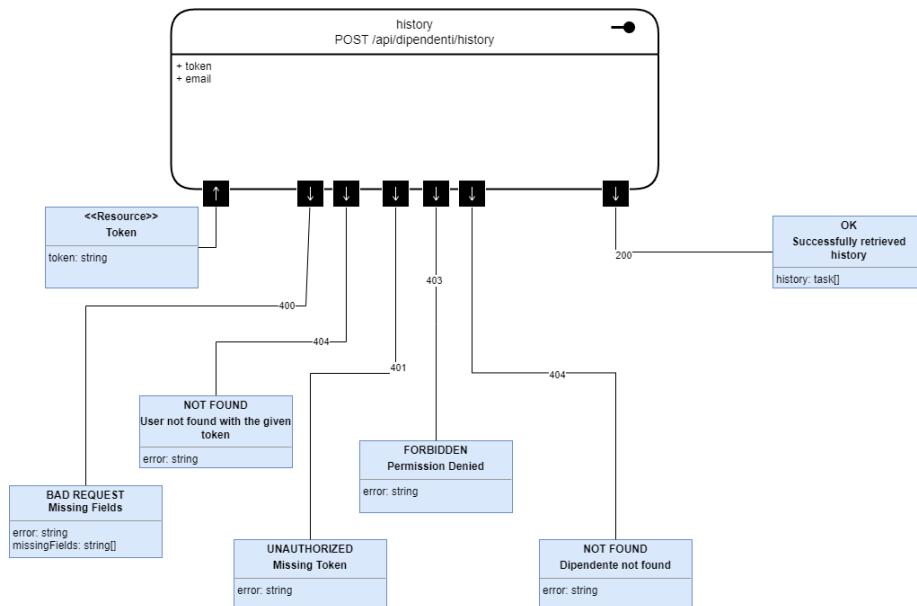


Figure 4.19: Diagramma dell'endpoint history

**Specifica OpenAPI**

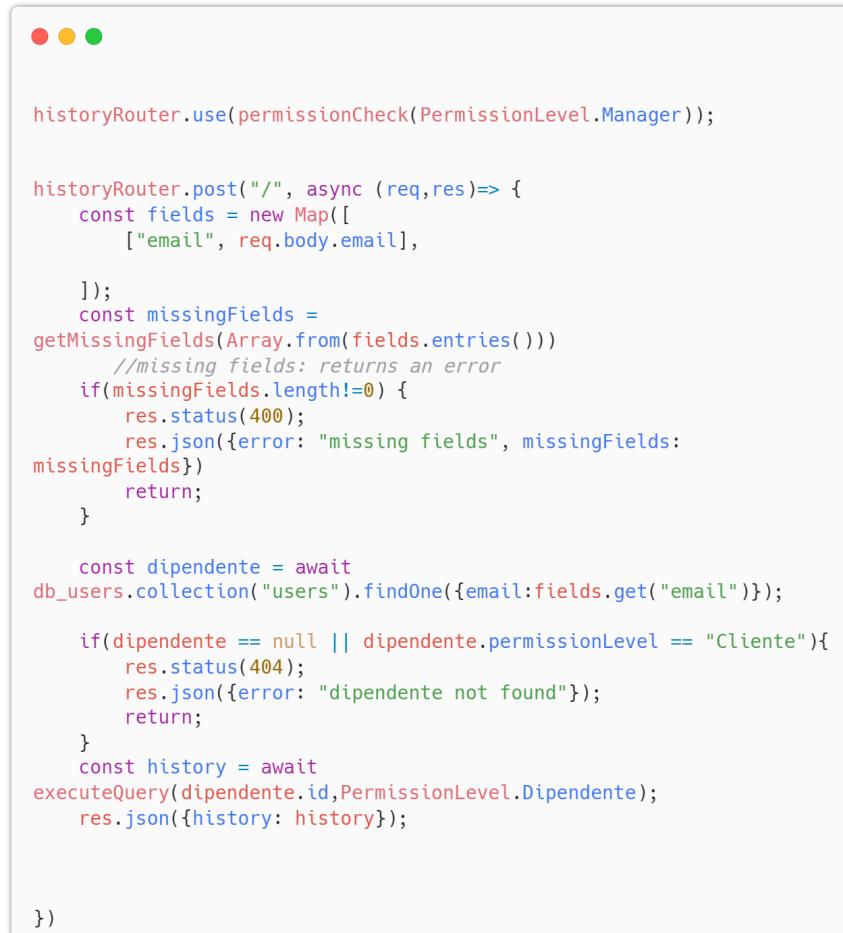
Di seguito la specifica openapi dell'endpoint remove contenuta nel file 'openapi.yaml' della directory del microservizio

```
/history:  
  post:  
    summary: Metodo per la visione dello storico task di un dipendente  
    description: Questo endpoint permette al manager di vedere lo storico task di un determinato dipendente  
    requestBody:  
      required: true  
      content:  
        application/x-www-form-urlencoded:  
          schema:  
            type: object  
            properties:  
              email:  
                type: string  
                format: email  
              token:  
                type: string  
  
            required:  
              - token  
              - email  
    responses:  
      '200':  
        description: "Il dipendente esiste e questo è il suo storico task "  
        content:  
          application/json:  
            schema:  
              type: array  
              items:  
                type: object  
  
      '401':  
        description: "Token mancante nella richiesta"  
        content:
```

```
application/json:  
  schema:  
    $ref: "#/components/schemas/missingTokenSchema"  
'403':  
  description: "Il token non corrisponde al token del manager"  
  content:  
    application/json:  
      schema:  
        $ref: "#/components/schemas/notTheManagerSchema"  
  
'404':  
  description: "Il token non corrisponde a nessun utente OR Utente non trovato nella  
  content:  
    application/json:  
      schema:  
        $ref: "#/components/schemas/notFoundSchema"  
'400':  
  description: Bad Request - 'Missing fields'  
  content:  
    application/json:  
      schema:  
        oneOf:  
        - $ref: '#/components/schemas/missingFieldsSchema'
```

### Codice

Questa api viene utilizzata dal manager per ottenere lo storico task di un determinato dipendente. Una volta ricevuta la richiesta, viene controllata inizialmente la presenza del token del manager nel body o nel token attraverso il middleware 'permissionMiddleware'. Successivamente viene controllata la presenza di tutti i campi richiesti. Dopodichè viene controllata la presenza di un dipendente nel database avente la mail fornita. Infine il server risponde con lo storico task del dipendente.



```
historyRouter.use(permissionCheck(PermissionLevel.Manager));

historyRouter.post("/", async (req, res) => {
  const fields = new Map([
    ["email", req.body.email],
  ]);

  const missingFields =
    getMissingFields(Array.from(fields.entries()))
      //missing fields: returns an error
    if(missingFields.length!=0) {
      res.status(400);
      res.json({error: "missing fields", missingFields: missingFields})
      return;
    }

  const dipendente = await
    db_users.collection("users").findOne({email:fields.get("email")});

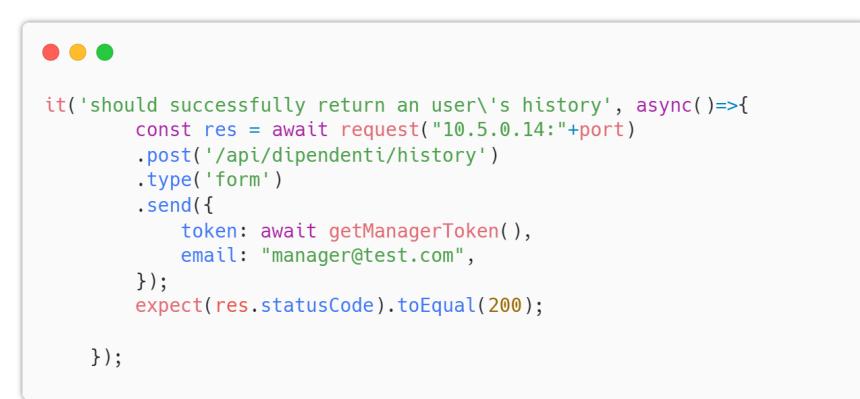
  if(dipendente == null || dipendente.permissionLevel == "Cliente"){
    res.status(404);
    res.json({error: "dipendente not found"});
    return;
  }
  const history = await
    executeQuery(dipendente.id, PermissionLevel.Dipendente);
  res.json({history: history});
})
```

Figure 4.20: Codice dell'endpoint history

## Testing

Per questo endpoint sono stati creati i seguenti test:

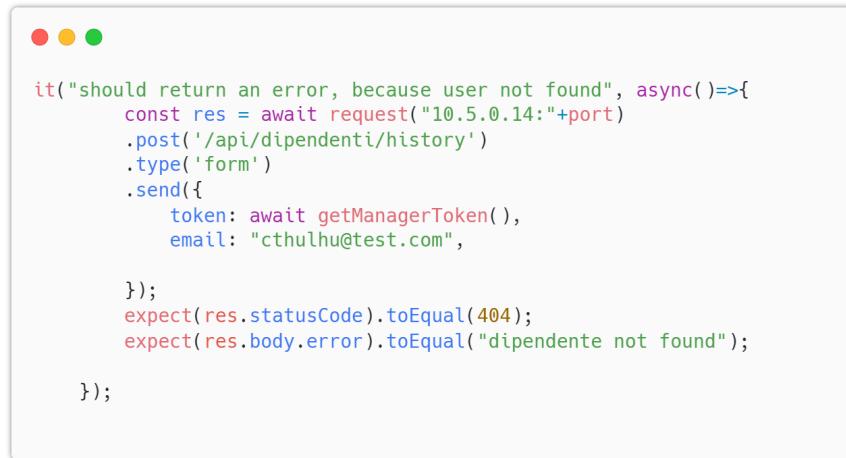
Numero Test-case	Descrizione Test Case	Test Data	Precondizioni	Dipendenze	Res Atteso	Res Riscontrato
1	Storico trovato con successo	{email}	dipendente esistente	Microservizio Autenticazione,DB	200	200
2	Utente non trovato	{email}		Microservizio Autenticazione,DB	404	404



```
it('should successfully return an user\'s history', async()=>{
  const res = await request("10.5.0.14:"+port)
    .post('/api/dipendenti/history')
    .type('form')
    .send({
      token: await getManagerToken(),
      email: "manager@test.com",
    });
  expect(res.statusCode).toEqual(200);

});
```

Figure 4.21: Test Case 1



```
it("should return an error, because user not found", async()=>{
    const res = await request("10.5.0.14:"+port)
        .post('/api/dipendenti/history')
        .type('form')
        .send({
            token: await getManagerToken(),
            email: "cthulhu@test.com",
        });
    expect(res.statusCode).toEqual(404);
    expect(res.body.error).toEqual("dipendente not found");
});
```

Figure 4.22: Test Case 2



```
/app # npm test history.test.ts
> fixmi-microservice-template@1.0.0 test
> jest history.test.ts

PASS  backend/tests/history.test.ts
  history testing
    ✓ should successfully return an user's history (27 ms)
    ✓ should return an error, because user not found (8 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:  0 total
Time:        0.411 s, estimated 1 s
Ran all test suites matching /history.test.ts/i.
```

Figure 4.23: Risultati dei test

---

CHAPTER

**FIVE**

---

MICROSERVIZIO HOME

## 5.1 Front-End

Questa sezione comprende lo sviluppo e la documentazione della Front-End del Microservizio Home

### 5.1.1 Struttura del Front-End

La struttura della Front-End del microservizio Home è riportata nella figura sotto.

```
● ● ●
src/
├── assets
│   └── fixmi-logo.png
├── components
│   ├── Assistenza
│   │   ├── assistenza.tsx
│   │   └── formAssistenza.tsx
│   ├── Feedback
│   │   ├── feedback.tsx
│   │   └── formFeedback.tsx
│   ├── Riparazione
│   │   └── formRiparazione.tsx
│   └── riparazione.tsx
├── error.tsx
├── footer.tsx
├── home.tsx
├── homeContent.tsx
├── navbar.tsx
└── success.tsx
├── utils
│   └── connection.ts
└── App.tsx
└── index.tsx
└── style.css
6 directories, 17 files
```

Figure 5.1: Output del comando "tree" all'interno del microservizio home.

```
● ● ●

import { BrowserRouter, Route, Routes } from 'react-router-dom';
import React from 'react'
import Home from './components/home'
import Riparazione from './components/Riparazione/riparazione.tsx';
import Success from './components/success.tsx';
import Error from './components/error.tsx';
import Assistenza from './components/Assistenza/assistenza.tsx';
import Feedback from './components/Feedback/feedback.tsx';

function App() {

  return (
    <BrowserRouter basename='/'>
      <main>
        <Routes>
          {/* Routing */}
          <Route path="/" element={<Home />} />
          <Route path="/riparazione" element={<Riparazione />} />
          <Route path="/assistenza" element={<Assistenza />} />
          <Route path="/success" element={<Success />} />
          <Route path="/error" element={<Error />} />
          <Route path="/feedback" element={<Feedback />} />
          <Route path="*" element={<Home />} />
        </Routes>
      </main>
    </BrowserRouter>
  );
}

}
```

Codice del componente "App.tsx"

### Specifiche

Tutte le componenti facenti parte del microservizio "Home" vengono gestite dalla componente "App.tsx", quest'ultimo stabilisce gli indirizzi di ciascuna pagina del microservizio e quale componente deve gestire l'indirizzo.

#### 5.1.2 UserFlow

Lo userflow del microservizio home viene mostrato sotto nel suo intero. Verrà successivamente suddiviso ed analizzato in ciascun suo componente.

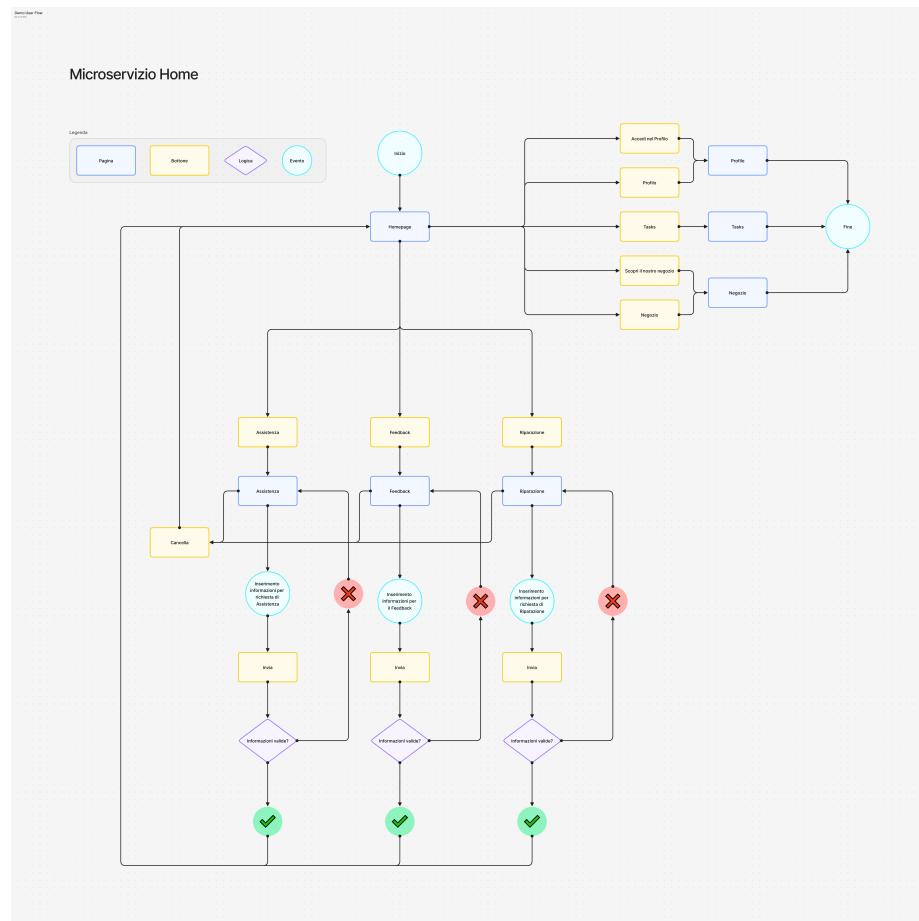


Figure 5.2: UserFlow del microservizio "Home"

### 5.1.3 Homepage



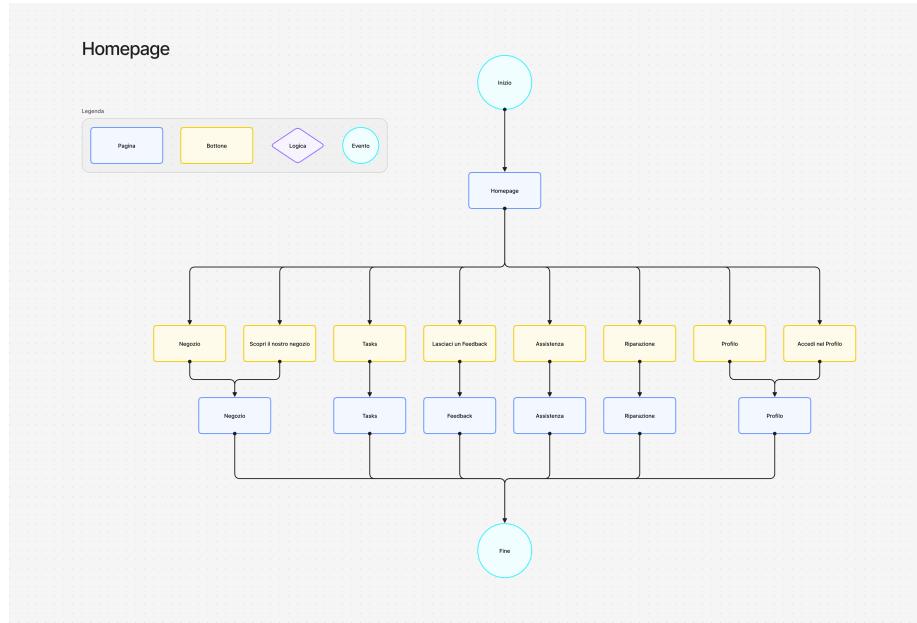
Figure 5.3: Schermata per la homepage

#### Specifiche

La pagina si occupa di mostrare un messaggio di benvenuto all'utente. E' possibile raggiungere tutti gli altri microservizi tramite i bottoni presenti al suo interno.

#### Userflow

E' la prima pagina che viene mostrata quando l'applicazione viene aperta

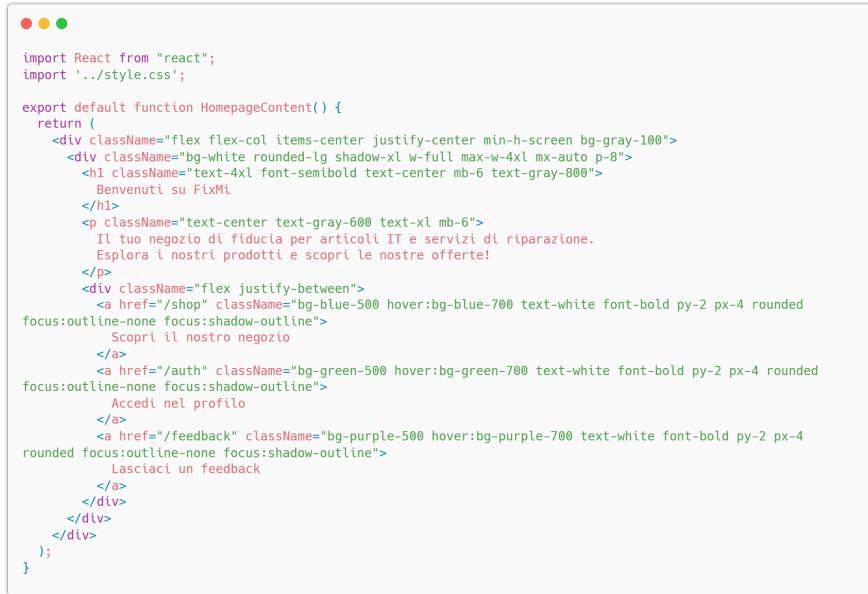


Userflow della pagina "homepage"

### Codice

La pagina "Homepage" è composta da due componenti chiamati "home.tsx" e "homeContent.tsx" con le seguenti caratteristiche:

- home.tsx
  - Contiene al suo interno "homeContent.tsx"
  - Contiene controlli e "Fallbacks" per eventuali errori
- homeContent.tsx
  - Si occupa di mostrare una carta contenente un messaggio di benvenuto e tre button per accedere ai principali servizi dell'applicazione



```

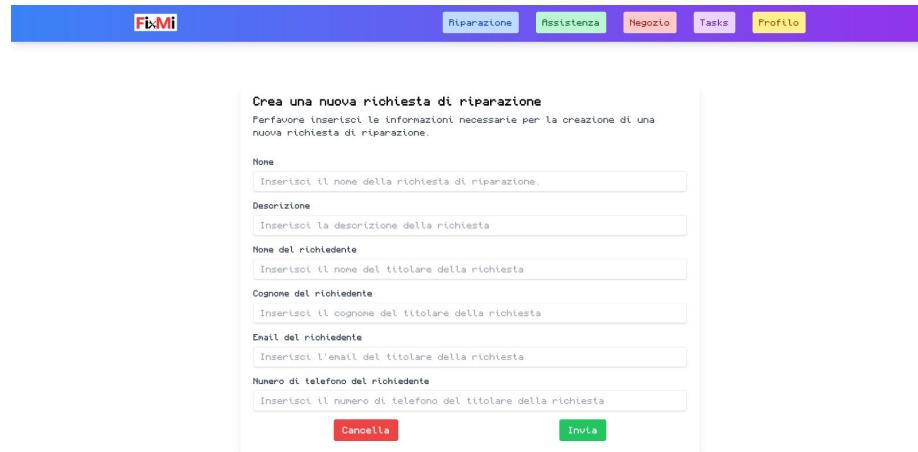
import React from "react";
import './style.css';

export default function HomepageContent() {
  return (
    <div className="flex flex-col items-center justify-center min-h-screen bg-gray-100">
      <div className="bg-white rounded-lg shadow-xl w-full max-w-4xl mx-auto p-8">
        <h1 className="text-4xl font-semibold text-center mb-6 text-gray-800">
          Benvenuti su FixMi
        </h1>
        <p className="text-center text-gray-600 text-xl mb-6">
          Il tuo negozio di fiducia per articoli IT e servizi di riparazione.
          Esplora i nostri prodotti e scopri le nostre offerte!
        </p>
        <div className="flex justify-between">
          <a href="/shop" className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline">
            Scopri il nostro negozio
          </a>
          <a href="/auth" className="bg-green-500 hover:bg-green-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline">
            Accedi nel profilo
          </a>
          <a href="/feedback" className="bg-purple-500 hover:bg-purple-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline">
            Lasciati un feedback
          </a>
        </div>
      </div>
    </div>
  );
}

```

Codice del componente "homeContent.tsx"

### 5.1.4 Riparazione



The screenshot shows a web application interface for creating a repair request. At the top, there is a blue header bar with the 'FixMi' logo on the left and five navigation tabs on the right: 'Riparazione' (highlighted in blue), 'Assistenza' (green), 'Negozio' (red), 'Tasks' (yellow), and 'Profilo' (orange). Below the header, the main content area has a white background. It features a title 'Crea una nuova richiesta di riparazione' followed by a subtitle 'Perfavore inserisci le informazioni necessarie per la creazione di una nuova richiesta di riparazione.' There are six input fields with placeholder text: 'Nome' (Inserisci il nome della richiesta di riparazione.), 'Descrizione' (Inserisci la descrizione della richiesta), 'Nome del richiedente' (Inserisci il nome del titolare della richiesta), 'Cognome del richiedente' (Inserisci il cognome del titolare della richiesta), 'Email del richiedente' (Inserisci l'email del titolare della richiesta), and 'Numero di telefono del richiedente' (Inserisci il numero di telefono del titolare della richiesta). At the bottom of the form are two buttons: a red 'Cancella' button on the left and a green 'Invia' button on the right.

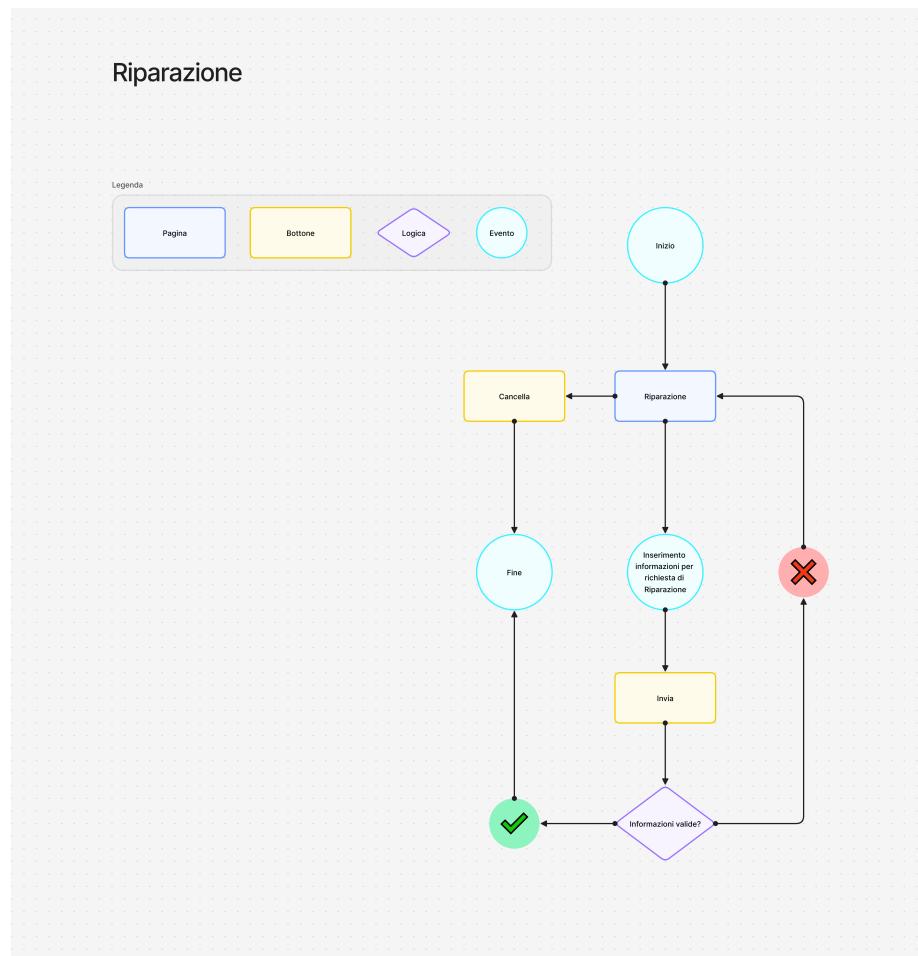
Figure 5.4: Schermata per la richiesta di riparazione

### Specifica

La pagina si occupa di mostrare un modulo per una richiesta di riparazione che può essere compilato ed inviato dall'utente.

### Userflow

Partendo dalla pagina "Homepage" bisogna cliccare il tasto all'interno della "navbar" chiamato "Riparazione"



Userflow della pagina "riparazione"

### Codice

La pagina "Riparazione" è composta da due componenti chiamati "riparazione.tsx" e "formRiparazione.tsx" con le seguenti caratteristiche:

- riparazione.tsx
  - Contiene al suo interno "formRiparazione.tsx"
  - Contiene controlli e "Fallbacks" per eventuali errori
- formRiparazione.tsx
  - Si occupa di mostrare una carta contenente un modulo per una richiesta di Riparazione che può essere compilata dall'utente

#### 5.1.5 Assistenza

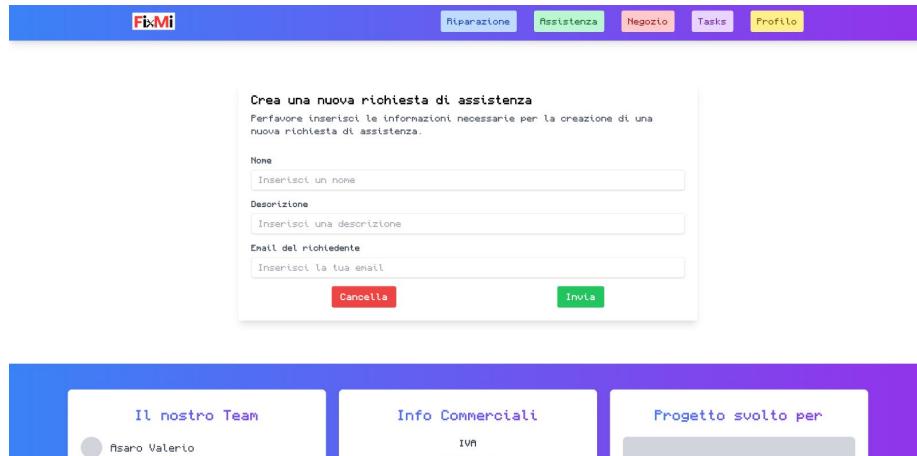


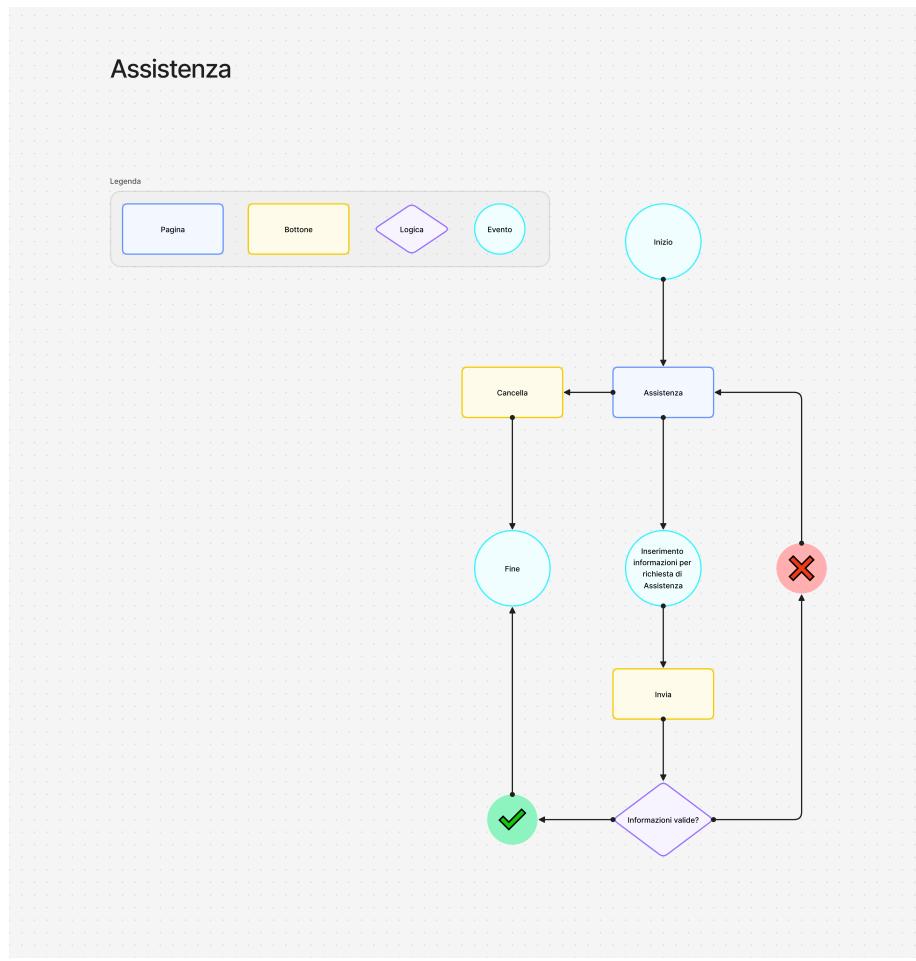
Figure 5.5: Schermata per la richiesta di assistenza

### Specifica

La pagina si occupa di mostrare un modulo per una richiesta di assistenza che può essere compilato ed inviato dall'utente.

### Userflow

Partendo dalla pagina "Homepage" bisogna cliccare il tasto all'interno della "navbar" chiamato "Assistenza"



Userflow della pagina "assistenza"

### Codice

La pagina "Assistenza" è composta da due componenti chiamati "assistenza.tsx" e "formAssistenza.tsx" con le seguenti caratteristiche:

- assistenza.tsx
  - Contiene al suo interno "formAssistenza.tsx"
  - Contiene controlli e "Fallbacks" per eventuali errori
- formAssistenza.tsx

- Si occupa di mostrare una carta contenente un modulo per una richiesta di Assistenza che può essere compilata dall'utente

### 5.1.6 Feedback

The screenshot shows a feedback form titled "Lasciaci un feedback!" (Leave us a feedback!). It includes fields for entering a name, providing a description, suggesting improvements, and rating various aspects of the service. The form is part of a larger application interface with tabs for Reparazione, Assistenza, Negozio, Tasks, and Profilo.

**Lasciaci un feedback!**  
Come ti sei trovato su FixMi? Diccelo qui sotto!

**Nome**  
Inserisci un nome

**Descrizione**  
Inserisci una descrizione

**Idee Per Migliorare**  
Inserisci qui qualche suggerimento per rendere il servizio migliore

**Disponibilità dell'azienda**  
FixMi è stata veloce nel svolgere il suo lavoro?

**Velocità della riparazione**  
I nostri tecnici sono stati rapidi nella riparazione?

**Soddisfazione della riparazione**  
E' soddisfatto del nostro servizio di riparazione?

**Soddisfazione del sito web**  
E' soddisfatto del sito di FixMi?

**Cancella** **Invia**

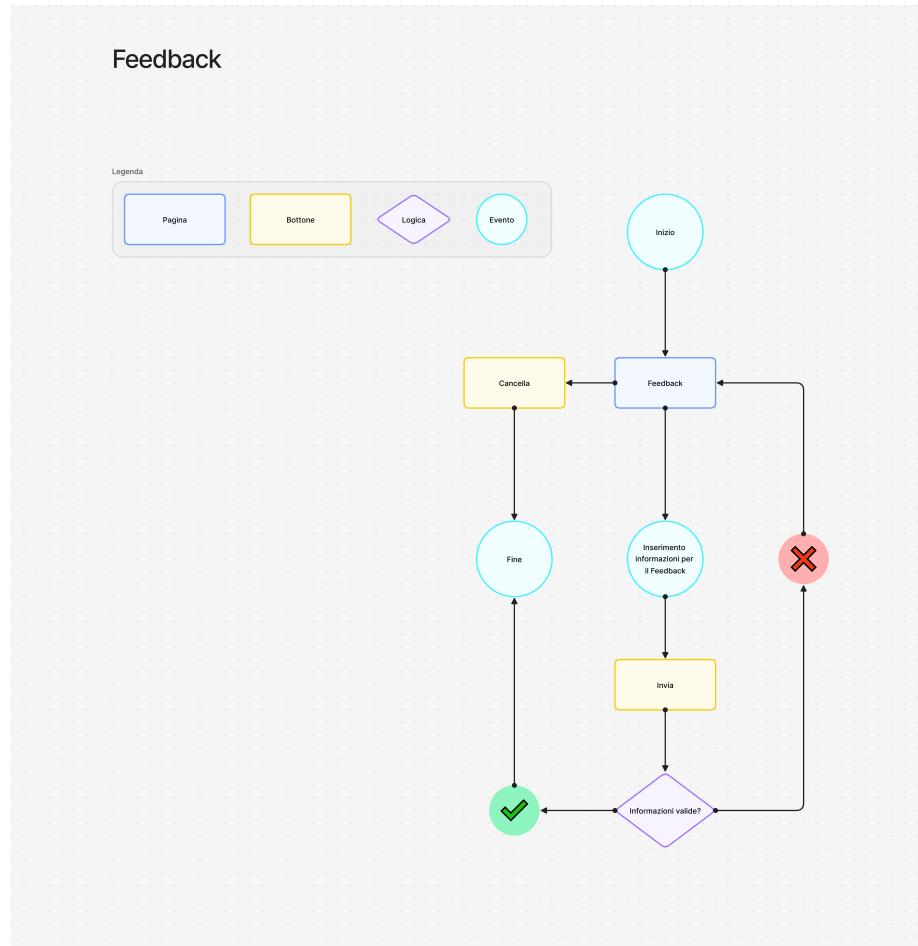
Figure 5.6: Schermata per il feedback

#### Specifiche

La pagina si occupa di mostrare un modulo per l'invio di un feedback che può essere compilato dall'utente.

#### Userflow

Partendo dalla pagina "Homepage" bisogna cliccare il tasto all'interno della carta di benvenuto chiamato "Lasciaci un feedback"



Userflow della pagina "feedback"

## Codice

La pagina "Feedback" è composta da due componenti chiamati "feedback.tsx" e "formFeedback.tsx" con le seguenti caratteristiche:

- feedback.tsx
  - Contiene al suo interno "formFeedback.tsx"
  - Contiene controlli e "Fallbacks" per eventuali errori
- formFeedback.tsx
  - Si occupa di mostrare una carta contenente un modulo per l'invio di feedback che può essere compilata dall'utente

---

## CHAPTER

## SIX

---

## DEPLOY

In questo capitolo si descrivono i procedimenti per effettuare un deploy dell'applicazione con successo in locale.

### 6.1 Dipendenze

E' necessario aver installato Docker nella macchina host. Se l'host si trova su windows, Docker può essere installato tramite "WSL" ("Windows Subsystem for Linux").

### 6.2 Ottenere il codice

Tutto il codice necessario è presente su github, accessibile tramite questo link (le repo sono private quindi se non si è loggati con il profilo autorizzato si otterrà una 404). In particolare, il codice per eseguire l'infrastruttura si trova in "fixmi-compose". Al suo interno sono contenute le altre repo dei microservizi in submoduli. Per clonare la repo si usa il comando:

```
git clone --recurse-submodules https://github.com/IS-FixMi/fixmi-compose.git
```

#### **Nota:**

Dato che i moduli da installare sono molti, il team di sviluppo consiglia di scaricare la directory con già i moduli tramite il seguente link TODO. Successivamente, è possibile estrarre la cartella tramite:

```
tar -xf archive.tar.gz
```

### 6.3 Istruzioni

L'intera infrastruttura può essere eseguita tramite il seguente comando all'interno della cartella "fixmi-compose":

```
sudo docker compose up --build
```

si consiglia di utilizzare quest'ultimo in quanto vengono di impostare tutte le porte e volumi necessari per il corretto funzionamento dell'applicazione.

#### Problemi Noti

- L'applicazione risulta parecchio pesante, dunque il tempo di download potrebbe essere prolioso.
- Sono noti dei problemi nell'installazione dei moduli "concurrently" e "react-scripts". Qualora vi siano dei problemi del tipo "concurrently non è stato trovato", si proceda in questo modo:
  1. Si entri nella cartella del microservizio che riporta il problema
  2. Si esegua il comando "npm i <nome-del-modulo-non-trovato> ."
- Qualora le porte da 3001 a 3008 risultassero già occupate da altri servizi nella macchina host, queste possono essere cambiate nel file "docker-compose.yaml", così come la porta del reverse proxy (7777) e gli ip dei microservizi.

### 6.4 Eseguire Singoli Microservizi

Nonostante eseguire parti dell'infrastruttura indipendenti senza Docker compose non sia consigliato, questa sezione riporta le istruzioni per eseguire in microservizio singolarmente e informazioni per il developement.

#### Microservizio con Docker

Per eseguire un singolo microservizio dobbiamo buildare il Docker container, questo può essere fatto con il seguente comando:

```
cd fixmi-microservizio-<nome-del-microservizio>
docker build -t example-microservice .
```

Successivamente, il Docker container può essere eseguito con il comando:

```
docker run -p <backend_port>:3001 -p <frontend_port>:3002 \\
-v .:/app example-microservice
```

- Al comando può essere aggiunta la flag "-d" per eseguire il container in detached mode, nascondendo l'output nel background. Si ricordi che per fermare il container sarà necessario usare il comando "docker stop <container-id>" e per leggere i logs si può usare il comando "docker logs <container-id>"

## Microservizio senza Docker

Per eseguire un microservizio senza l'utilizzo di Docker è necessario aver installato "npm". I seguenti comandi sono utili per il deploy e lo sviluppo. Prima di tutto è necessario installare i moduli da cui dipende il programma:

```
npm install .
```

Il microservizio può essere eseguito in production con:

```
npm run buildfront
npm run production
```

Per eseguire solo la Back-End:

```
npm run startback
```

Per eseguire solo la Front-End:

```
npm run startfront
```

Per eseguire la Back-End e Front-End in dev mode:

```
npm run start
```

Nota: i microservizi utilizzano il database per il salvataggio e la lettura dei dati, e il reverse proxy per ottenere il token di sessione e are redirect agli altri microservizi.

Per qualsiasi problema e assistenza, si contatti um membro dello sviluppo.