Manuale Tecnico

Tirocinio Sync Lab

Studente:
Alessandro Tigani Sava

Azienda:
Sync Lab s.r.l

31 luglio 2024

Indice

1 Introduzione

1.1 Contatti

Studente: Alessandro Tigani Sava, alessandro.tiganisava@studenti.unipd.it
Tutor aziendale: Fabio Pallaro, f.pallaro@synclab.it

1.2 Scopo del documento

La seguente documentazione fornisce istruzioni per l'utilizzo software prodotto, durante il tirocinio curricolare, presso l'azienda Sync Lab.

1.3 Informazioni sul progetto

Lo scopo del progetto è la realizzazione del back-end relativo all'applicazione denominata Trip Hippie, il progetto è stato svolto simultaneamente ad altri studenti che si sono occupati della realizzazione del front-end e di un servizio relativo alla chat.
Il prodotto da me realizzato è disponibile nella seguente repository: Progetto Sync Lab. Al suo intenro sono presenti i file relativi alla documentazione Stopligh, la cartella contenente il codice del programma, il file necessario per generare il database ed avviare l'applicazione tramite Docker.
Il progetto si compone di due parti principali, una incentrata sugli utenti ed una sulla gestione viaggi. Non si è scelto di utilizzare una architettura a microservizi per questioni relative a necessità di semplicità applicativa e di tempistiche, si dispone quindi di un unico progetto che include tutte le funzionalità.

1.4 Funzionalità

Il back-end da me realizzato permette di effettuare operazioni inerenti la gestione di utenti e viaggi. Le funzionalità inerenti le chat sono state sviluppate da uno studente differente e non sono state implementate nel presente progetto.
Le funzionalità sono state raggruppate in due gruppi principali, User e Trip. La documentazione relativa a tutte le operazioni da rendere disponibili mi è stata consegnata utilizzando StopLight. Pur non avendo creato un progetto pubblico è possibile importare la documentazione tramite due file presenti nel root del Repository, denominati:

- Trip.json
- User.json

Possono esserci differenze tra la versione presente nella mia repository e quella in possesso di altri studenti, in quanto durante il mio tirocinio mi è stato richiesto di aggiungere anche un servizio necessario alla registrazione delle preferenze dei singoli utenti.

2 Setup dell'ambiente di sviluppo

2.1 Prerequisiti

Per lo sviluppo del back-end è stata utilizzata la JDK 21 di Java, con framework Spring Boot nella versione 3.3.2, mentre per la gestione di progetto è stato utilizzato Maven.
Per la creazione del database viene utilizzato Docker con immagine postgres:alpine.

L'IDE utilizzato per lo sviluppo è IntelliJ IDEA, si consiglia inoltre l'utilizzo di pgAdmin per l'interazione diretta con il database.

2.2 Configurazione del Progetto

Una volta soddisfatti i prerequisiti, è possibile effettuare la clonazione da GitHub del Progetto.
Per avviare il progetto è necessario prima creare il container relativo al solo database di sviluppo, è possibile farlo eseguendo il file `compose-dev.yml` presente nella cartella `triphippie/src/main/resources` del repository. Tale file preleva i dati di configurazione relativi al database dal file `.env` presente nella stessa cartella. Al suo interno sono presenti le informazioni in merito al database, alla chiave per la generazione dei token JWT ed il path relativo alla cartella che dovrà contenere le immagini degli utenti. Un esempio di compilazione del file è il seguente:

```
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_DB=triphippie
JWT_SECRET_KEY=xVQ9J3BcTTJIQqA3r8xdlJEe6jrTlyWtzJZt5cZtnl0=
IMAGE_PATH=C:/Dev/ProgettoStageSyncLab/triphippie/resources
```

Eventuali modifiche apportate al file `.env` dovranno essere riflesse anche nel file `application.properties` presente nella stessa directory.
Si può infine procedere ad installare le dipendenze con il comando:

```
mvn clean install
```

Ora il progetto può essere avviato con il comando:

```
mvn spring-boot:run
```

Per completare la configurazione è necessario inserire manualmente nel database, preferibilmente tramite pgAdmin, i valori corrispondenti ai tag e ai veicoli da utilizzare nell'applicazione.
I valori previsti sono elencati nel contratto presente su StopLight.

3 Architettura del sistema

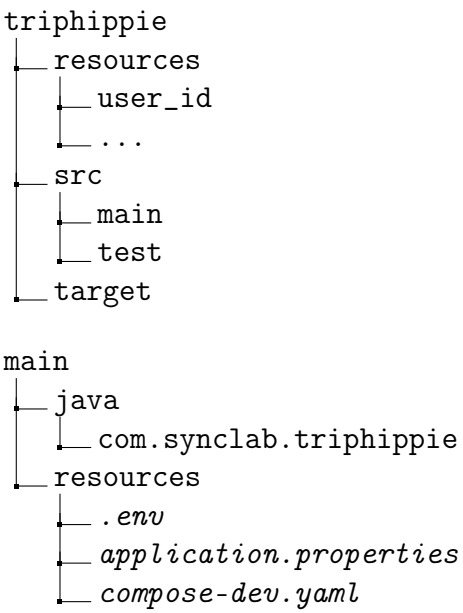
3.1 Struttura del progetto

Il codice del progetto è disponibile all'interno della cartella `triphippie`, al suo interno vi sono tre sotto-cartelle principali:

- **src**: contenente il codice Java;
- **target**: contenente i file di compilazione e l'eseguibile;
- **resources**: contenente le immagini relative al profilo di ogni utente.

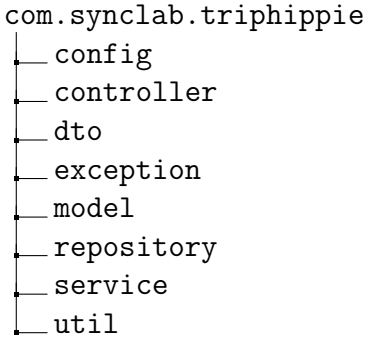
La cartella `resources` contiene al suo interno una sotto-cartella per ogni utente registrato, al suo interno saranno inserite tutte le risorse relative a tale utente.
La cartella `src` contiene sia il codice dell'applicazione che quello relativo ai test.
Nella cartella `main` sono presenti le sotto-cartelle:

- **java**: contenente i pacchetti in cui è stata suddivisa l'applicazione;



- **resources**: contiene i file di configurazione come `.env`, `application.properties` ed il file `compose-dev.yaml` per la creazione del database di sviluppo.

Nella cartella `java` sono presenti i package in cui è stato suddiviso il progetto. Il pacchetto principale è `com.synclab.triphippie`, al suo interno sono presenti ulteriori pacchetti che racchiudono le diverse tipologie di classi create.



I pacchetti suddividono il progetto individuandone le componenti proncipali, ossia controller, servizi, DTO (Data Transfer Object), repository.
Il modello viene usato per definire la struttura del database SQL, dunque non si ha un file `schema.sql`.
Ulteriori pacchetti sono:

- `exception`: definisce eccezioni personalizzate ed un `GlobalExceptionHandler`;
- `util`: definisce i mapper per i DTO, le classi per la gestione dell'hashing delle password e per la generazione dei token JWT;
- `config`: contiene classi di configurazione, in particolare quella dedicata alla gestione del Cors Mapping.

4 Scelte implementative

4.1 Framework e librerie

Il back-end dell'applicazione è stato sviluppato utilzzando Java e Spring Boot, entrambe richieste avanzate dall'azienda ospitante. In particolare, si è usata la versione 3.3.1 di Spring Boot e la JDK 21 di Java.
Sono stati poi utilizzati strumenti quali:

- **Maven**: tool di gestione del progetto utilizzato per la compilazione, il testing e il packaging dell'applicazione;
- **Docker**: per la containerizzazione, l'immagine Docker postgres:alpine è utilizzata per eseguire il database PostgreSQL in un contenitore.
- **PostgreSQL**: il database relazionale scelto per l'archiviazione dei dati dell'applicazione;
- **SonarLint**: integrato direttamente nell'IDE IntelliJ IDEA, utilizzato per l'analisi statica del codice.

Librerie base

Relativamente a Spring Boot sono state inserite le seguenti librerie:

- **spring-boot-starter-actuator**: per il monitoraggio e la gestione dell'applicazione, come metriche, controlli di stato e informazioni sugli endpoint;
- **spring-boot-starter-data-jdbc**: supporto per l'accesso ai dati utilizzando JDBC;
- **spring-boot-starter-data-jpa**: integrazione con JPA (Java Persistence API) per l'accesso ai dati in modo ORM (Object-Relational Mapping);
- **spring-boot-starter-validation**: validazione dei dati delle richieste utilizzando le annotazioni di Java Bean Validation;
- **spring-boot-starter-web**: dipendenze necessarie per costruire applicazioni web RESTful.

Librerie di utilità

- **io.jsonwebtoken**: insieme alle relative dipendenze è utilizzata per la gestione dei JSON Web Tokens (JWT), che sono utilizzati per l'autenticazione e l'autorizzazione;
- **io.github.cdimascio**: permette di caricare variabili d'ambiente da un file .env, semplificando la configurazione dell'applicazione;
- **org.projectlombok**: utilizzato per ridurre il boilerplate del codice Java attraverso l'uso di annotazioni, in particolare evita la scrittura manuale di getter, setter e costruttori.

Strumenti di svilupppo

- **spring-boot-devtools**: fornisce funzionalità di sviluppo come il live reload;
- **spring-boot-starter-test**: include librerie di testing come JUnit, Mockito e AssertJ per supportare lo sviluppo di test unitari e di integrazione.

4.2 Gestione della persistenza

Per la gestione della persistenza si è utilizzato JPA e la sua implementazione Hibernate. Questo ha permesso di lavorare con oggetti Java (entità) che vengono poi mappati alle tabelle del database PostgreSQL.

La configurazione delle proprietà del database avviene nel file `application.properties`.

```
# Configurazione di esempio del datasource
spring.datasource.url=jdbc:postgresql://localhost:5432/triphippie
spring.datasource.username=USERNAME
spring.datasource.password=PASSWORD
spring.datasource.driver-class-name=org.postgresql.Driver

# Configurazione JPA/Hibernate
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Per quanto riguarda la configurazione di JPA si usano le proprietà:

- **spring.jpa.database-platform**: specifica il dialect di Hibernate da usare per il database PostgreSQL;
- **spring.jpa.hibernate.ddl-auto**: controlla il comportamento di Hibernate per la gestione dello schema del database. L'opzione update aggiorna lo schema del database all'avvio dell'applicazione per riflettere i cambiamenti nelle entità JPA. Altre opzioni possibili includono none, validate, create, create-drop;
- **spring.jpa.show-sql**: abilita la stampa delle istruzioni SQL generate da Hibernate nella console, utile per scopi di debug.

Le entità che definiscono le tabelle del database vengono definite nell'apposito package `model`, si utilizzano le annotazioni di Lombok per velocizzare la scrittura di getter, setter e costruttori. Le relazioni tra le entità sono realizzate tramite annotazioni, allo stato attuale le relazioni presenti sono:

- **UserProfile - PreferenceTag**: relazione ManyToMany per la realizzazione della profilazione delle preferenze utente in merito alla tipologia di vacanza;
- **UserProfile - PreferenceVehicle**: relazione ManyToMany per la realizzazione della profilazione delle preferenze utente in merito al mezzo di trasporto preferito;
- **Trip - UserProfile**: relazione ManyToOne che definisce l'utente creatore del viaggio;
- **Trip - PreferenceTag**: relazione ManyToOne che definisce il tag da associare al viaggio;
- **Trip - PreferenceVehicle**: relazione ManyToOne che definisce il veicolo utilizzato per il viaggio;
- **Trip - Destination**: relazione OneToOne che definisce il punto di arrivo o di partenza di un viaggio;
- **Journey - Trip**: relazione ManyToOne che associa la tappa ad un viaggio;

- **Journey - Destination**: relazione OneToOne che associa la tappa ad una località.

Il repository è un'interfaccia che estende JpaRepository e JpaSpecificationExecutor, fornendo metodi CRUD e capacità di query avanzate. Vengono definiti all'interno del package `repository`. Ho cercato di limitare il più possibile l'utilizzo di query scritte manualmente, sfruttando il framework a disposizione.

4.3 Gestione della sicurezza

Vista la necessità di mantenere bassa la complessità dell'applicazione, l'azienda ospitante ha richiesto che non venisse implementato il modulo Spring Security.
Le operazioni di autorizzazione ed autenticazione vengono implementate manualmente all'interno della classe JwtUtil, che tramite `jsonwebtoken` si occupa della generazione e validazione dei token JWT.
In base alla tipologia di operazione richiesta il token viene validato considerando solo la sua data di scadenza, oppure assicurandosi che il token corrisponda all'identificativo dell'utente su cui si vogliono apportare modifiche.

5 Deploy dell'applicazione

Se si sono effettuate modifiche al codice dell'applicazione sarà necessario effettuare nuovamente la compilazione, a tal proposito sarà sufficiente spostarsi nella cartella del progetto ed utilizzare il comando:

```
mvn clean package
```

La compilazione eseguirà anche i test definiti all'interno del codice e provvederà a generare, nella cartella `target`, un file eseguibile con estensione `.jar`. Sarà sufficiente modificare in modo opportuno il file `.env` presente nella root del progetto ed eseguire il file `compose-deploy.yaml` per effettuare la creazione dei container relativi al back-end e al nuovo database.