



Manuale Tecnico

Tirocinio Sync Lab

Studente:

Alessandro Tigani Sava

Azienda:

Sync Lab s.r.l

29 luglio 2024

Indice

1	Introduzione	2
1.1	Contatti	2
1.2	Scopo del documento	2
1.3	Informazioni sul progetto	2
1.4	Funzionalità	2
2	Setup dell'ambiente di sviluppo	3
2.1	Prerequisiti	3
2.2	Configurazione del Progetto	3
3	Architettura del sistema	4
3.1	Struttura del progetto	4
4	Scelte implementative	5
4.1	Framework e librerie	5
4.2	Pattern di progettazione	6
4.3	Gestione della persistenza	6
4.4	Gestione della sicurezza	6
5	Deploy dell'applicazione	6

1 Introduzione

1.1 Contatti

Studente: Alessandro Tigani Sava, alessandro.tiganisava@studenti.unipd.it

Tutor aziendale: Fabio Pallaro, f.pallaro@syncclab.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 interno 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=xVQ9J3BcTTJIQqA3r8xd1JEe6jrTlyWtzJZt5cZtn10=
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
```

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.

```
triphippie
├── resources
│   ├── user_id
│   └── ...
├── src
│   ├── main
│   └── test
└── target
```

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.

```
main
├── java
│   └── com.synclab.triphippie
├── resources
│   ├── .env
│   ├── application.properties
│   └── compose-dev.yaml
```

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 principali, 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.

```
com.synclab.triphippie
├── controller
├── dto
├── exception
├── model
├── repository
├── service
└── util
```

4 Scelte implementative

4.1 Framework e librerie

Il back-end dell'applicazione è stato sviluppato utilizzando 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.

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 sviluppo

- **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.

Vista la necessità di mantenere bassa la complessità dell'applicazione, l'azienda ospitante ha richiesto che non venisse implementato il modulo Spring Security.

4.2 Pattern di progettazione

Alcuni pattern implementati sono:

- **Singleton**: assicura che una classe abbia solo un'istanza e fornisce un punto di accesso globale a essa, le classi annotate con `@Component`, `@Service`, `@Repository`, e `@Configuration` sono gestite come singleton di default da Spring.
- **Repository**: fornisce un'interfaccia per accedere ai dati e nasconde la logica di accesso ai dati, utilizzato per creare interfacce che estendono `JpaRepository`.

4.3 Gestione della persistenza

Descrivi l'approccio alla gestione della persistenza dei dati, inclusi i dettagli sulla configurazione di JPA/Hibernate, e le relazioni tra le entità principali.

4.4 Gestione della sicurezza

Spiega come viene gestita la sicurezza (autenticazione, autorizzazione), utilizzando ad esempio Spring Security.

5 Deploy dell'applicazione

Per effettuare la compilazione sarà sufficiente spostarsi nella cartella del progetto ed utilizzare il comando:

```
mvn clean package
```

Tale comando eseguirà anche i test presenti all'interno del progetto. A questo punto nella cartella `target` sarà presente un file con estensione `.jar`.