Installazione server

Per installare l'infrastruttura server in locale è necessario clonare il seguente repository:
https://github.com/DarioTin/Tesi-Server-Setup
Dopo aver clonato il repository è necessario aprire un terminale all'interno della directory creata ed inviare all'interno del terminale
docker-compose up
Ottenendo un risultato del genere:

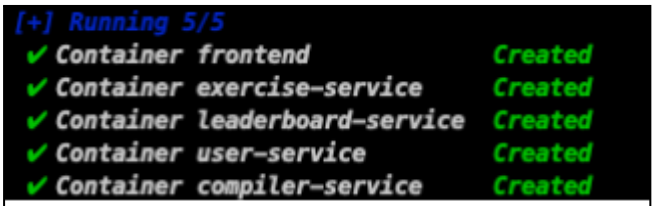
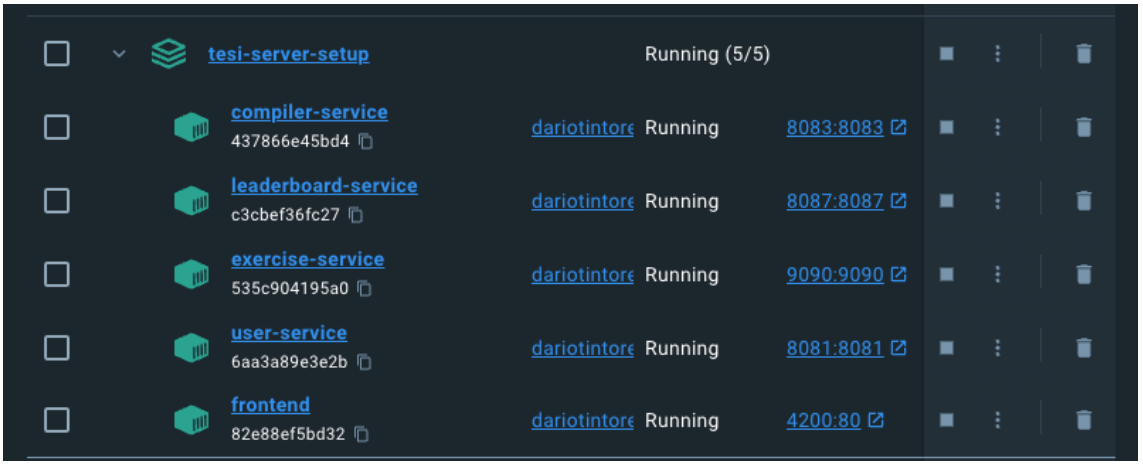


Figura 26: Risultato docker-compose up

In questo scenario avverrà la creazione di tutti i container necessari alla creazione dell'infrastruttura necessaria ad ospitare una sessione di gioco.



In seguito all'avvio di tutti i container è possibile accedere all'applicativo, partendo dal calcolatore server, al seguente indirizzo
http://localhost:4200/
In questo scenario avremo quindi installato correttamente il server in locale.

Installazione client

Per installare il client in locale è necessario clonare la seguente repository oppure scaricare l'ultima release dal seguente indirizzo:
https://github.com/DarioTin/Tesi-Client-Setup/releases
Dopo aver estratto il tutto, è necessario aprire un terminale all'interno della directory creata ed inviare all'interno del terminale
npm install
In questo caso, siccome l'installazione è indipendente dal sistema operativo, il nome della cartella e l'estensione dell'eseguibile che verrà creato varierà in base alla piattaforma su cui il comando sarà avviato.
Al termine dell'installazione verrà installato all'interno della directory "DarioTintore-Tesi-..." l'applicativo in locale, che potrà essere avviato con la seguente procedura:

1) Aprire un terminale all'interno della directory "DarioTintore-Tesi-..."
2) Eseguire il comando di avvio per l'eseguibile creato all'interno della directory "DarioTintore-Tesi"
3) Inizia a giocare 🙂

Installazione esercizi

L’aggiunta degli esercizi può essere effettuata su due piattaforme, su GitHub oppure tramite l’exercise service.
Gli esercizi recuperabili da github possono essere recuperati solo tramite la desktop application, mentre gli esercizi inseriti sull’exercise service possono essere recuperati da tutte le versioni dell’applicazione, ma richiedono che il microservizio “exercise service” sia attivo.
Sia nel caso dell’aggiunta di esercizi sulla repository GitHub che dell’aggiunta di esercizi sul microservizio, le regole da seguire sono identiche, l’unica cosa che cambia è il luogo in cui gli esercizi risiedono.

6.3.1 GitHub

Per aggiungere un nuovo esercizio recuperabile tramite github è necessario creare una nuova repository, in cui ad ogni esercizio è associata una cartella, ed in ogni cartella ci sono 3 file, dove i file java devono avere lo stesso nome dell’esercizio in questione.

6.3.2 ExerciseRepository

Per aggiungere un nuovo esercizio recuperabile tramite microservizio Exercise Repository è necessario creare una cartella all’interno della cartella ExerciseDB, in cui ad ogni esercizio è associata una cartella, ed in ogni cartella ci sono 3 file, dove i file java devono avere lo stesso nome dell’esercizio in questione

6.3.3 Struttura esercizio

Ad esempio se stiamo considerando di aggiungere un nuovo esercizio nominato “StringFormatter” avremo una struttura del genere

Per aggiungere un esercizio al microservizio ExerciseRespository è necessario

StringTester (Nome cartella)

- StringFormatter.Java (Nome file di produzione)
- StringFormatterTest.java (Nome file di testing originale)

StringFormatterConfig.json (Nome file di configurazione)

Quindi si richiede una configurazione del genere

{Nome Esercizio} (Directory)

{Nome Esercizio}.java (File java di produzione)

{Nome Esercizio}Test.java (File java di test)

{Nome Esercizio}Config.java (File di configurazione)

La struttura dei file è la seguente:

Nel file di produzione e di test deve esserci sempre il seguente package:

```
package com.dariotintore.tesi;
```

All'interno dei file di configurazione devono esserci sempre definite le seguenti chiavi:

```
{
 "refactoring_game_configuration": {
   "dependencies": [
     ""
   ],
   "refactoring_limit": number,
   "smells_allowed": number
 },
 "check_game_configuration": {
   "questions": [
     {
       "questionTitle": "",
       "questionCode": "",
       "answers": [
         {
           "answerText": "",
           "isCorrect": true/false
         }
       ]
     }
 },
 "auto_valutative": true/false
}
```

Le chiavi di refactoring_game riguardano le impostazioni riguardo la modalità refactoring game, mentre le chiavi di check_game riguardano la modalità di check smell. Nel momento in cui ci saranno chiavi mancanti in uno dei due casi l'esercizio potrebbe non funzionare correttamente.
Per ogni esercizio è quindi necessario configurare correttamente, come nell'esempio proposto in precedenza, tutte le configurazioni necessarie.
La spiegazione delle chiavi è riportata al capitolo "Opzioni esercizio".

Utilizzo database

I database presenti nei vari microservizi, come già citato in precedenza, sono dei database in memory, il che vuol dire che avviando il microservizio verrà avviato anche il rispettivo database a cui il microservizio è legato.

Per accedere al database di uno specifico microservizio è necessario considerare la seguente regola:

http://{urlmicroservizio}:{portamicroservizio}/h2

Se ad esempio stiamo considerando di avviare il microservizio user-service all'url 192.168.1.1 con porta 8081, per raggiungere il database sarà necessario aprire il browser ed inserire il seguente url:

http://192.168.1.1:8081/h2

Una volta raggiunto il seguente indirizzo avremo la seguente schermata:

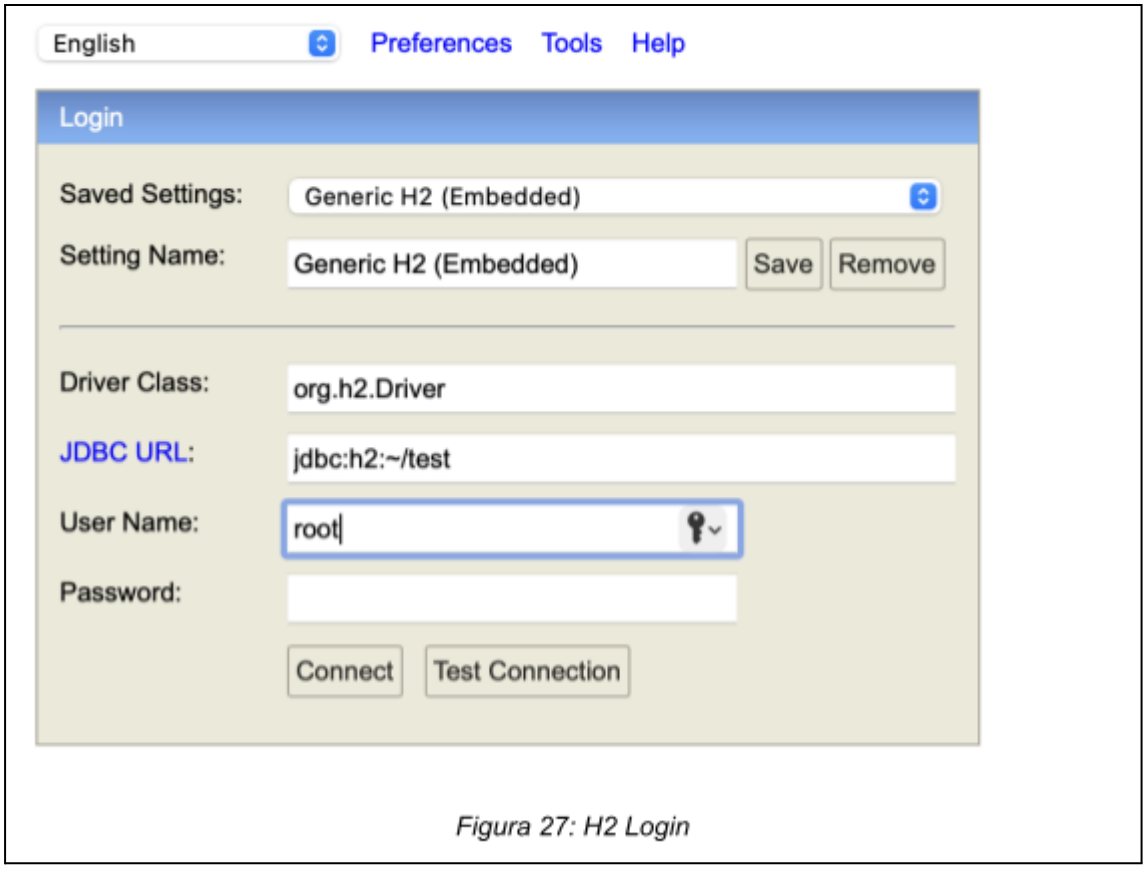


Figura 27: H2 Login

Per ottenere le informazioni necessarie, ovvero il JDBC URL, lo username e la password dobbiamo considerare il microservizio a cui ci vogliamo connettere, ed aprire il file application.properties presente nella cartella resources

All’interno di questo file troveremo tutte le informazioni necessarie da inserire per accedere al database. Considerando lo scenario in cui vogliamo accedere al microservizio user-service avremo la seguente configurazione:

```
# H2
spring.datasource.url=jdbc:h2:file:./database/userdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=root
spring.datasource.password=password
```

Quindi una volta inseriti i dati:

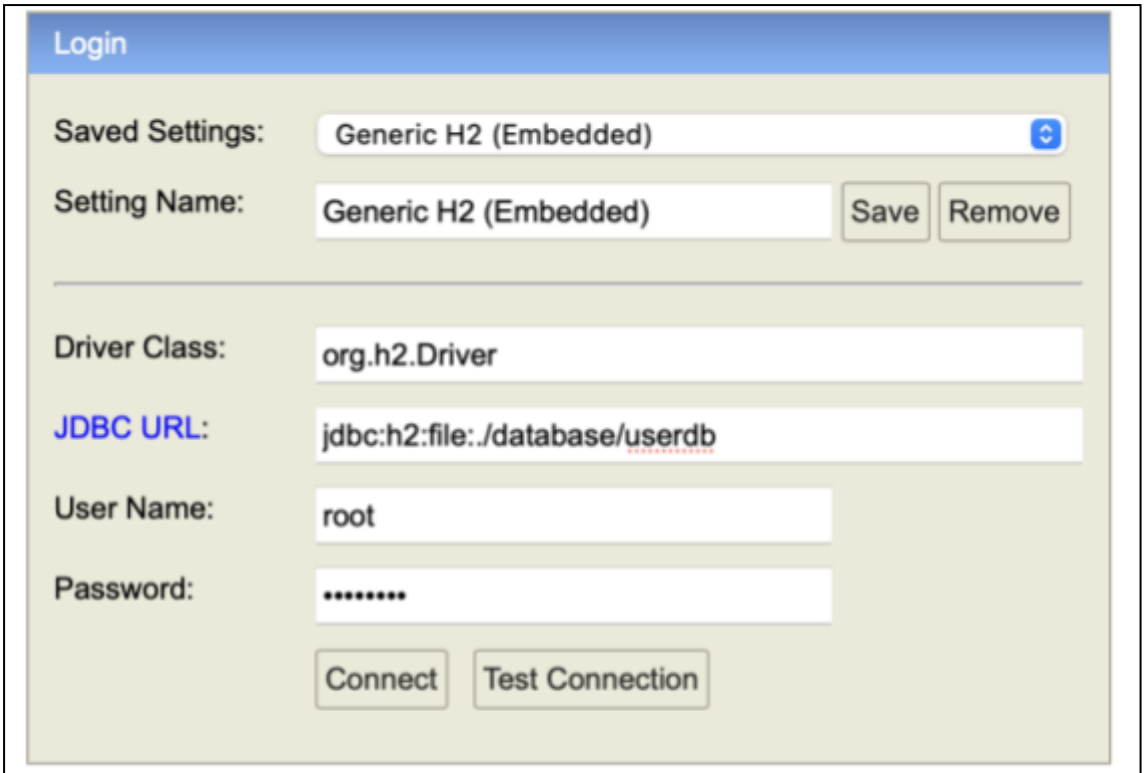


Figura 28: H2 Login (2)

Possiamo accedere al DBMS interno

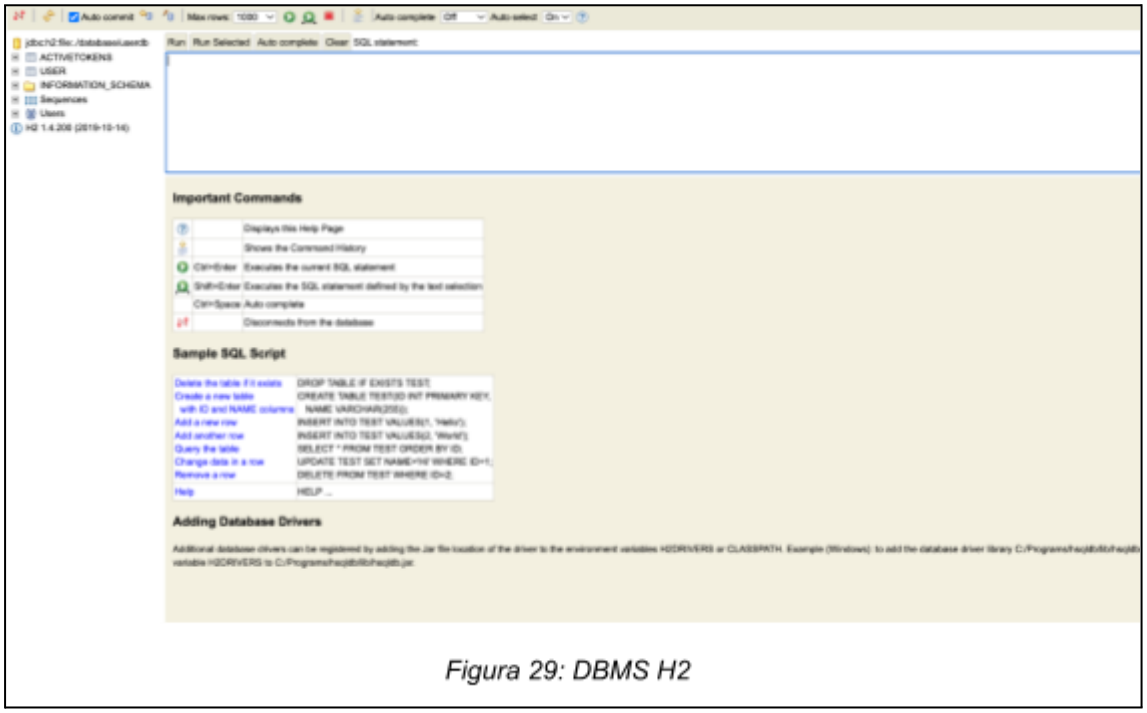


Figura 29: DBMS H2

Repositories

User Service	**https://github.com/DarioTin/Tesi_User_service**
Exercise Service	**https://github.com/DarioTin/Tesi_Exercise_service**
Compiler Service	**https://github.com/DarioTin/Tesi_Compiler_service**
Leaderboard Service	**https://github.com/DarioTin/Tesi_Leaderboard_service**
Frontend Online	**https://github.com/DarioTin/Tesi_Frontend**
Server-Setup	**https://github.com/DarioTin/Tesi-Server-Setup**
Client-Setup	**https://github.com/DarioTin/Tesi-Client-Setup**

Ambienti di sviluppo

Gli ambienti di sviluppo utilizzati sono stati:
Intellij (User Service, Exercise Service, Leaderboard Service) (non è necessario alcuno script)
WebStorm (Compiler Service, Client-Setup, Frontend Online)

Comandi di installazione

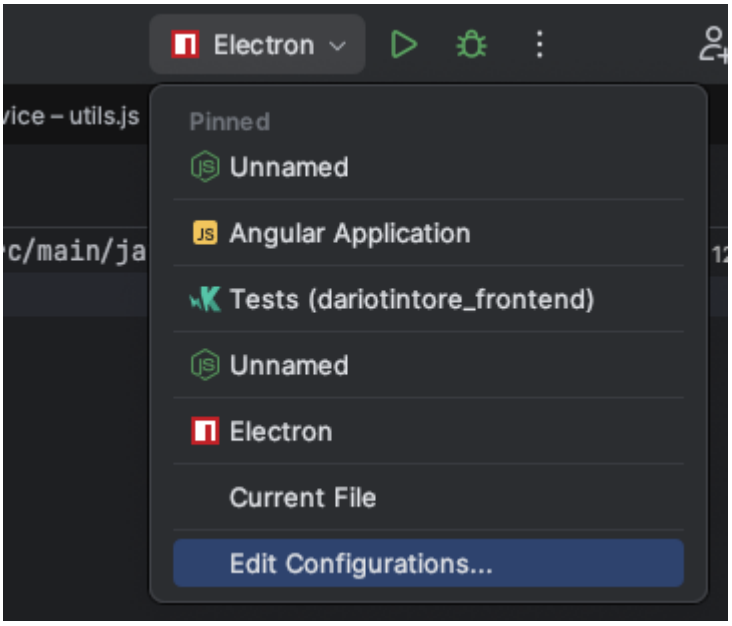
Prima di avviare in locale tutti i vari progetti è necessario avviare i comandi di compilazione che possono essere diversi da progetto a progetto.

User Service	**mvn clean install**
Exercise Service	**mvn clean install**
Compiler Service	**mvn clean install**
Leaderboard Service	**mvn clean install**
Frontend Online	**npm install**
Server-Setup	**docker-compose up**
Client-Setup	**npm install // npm run electron (per avviare electron)**

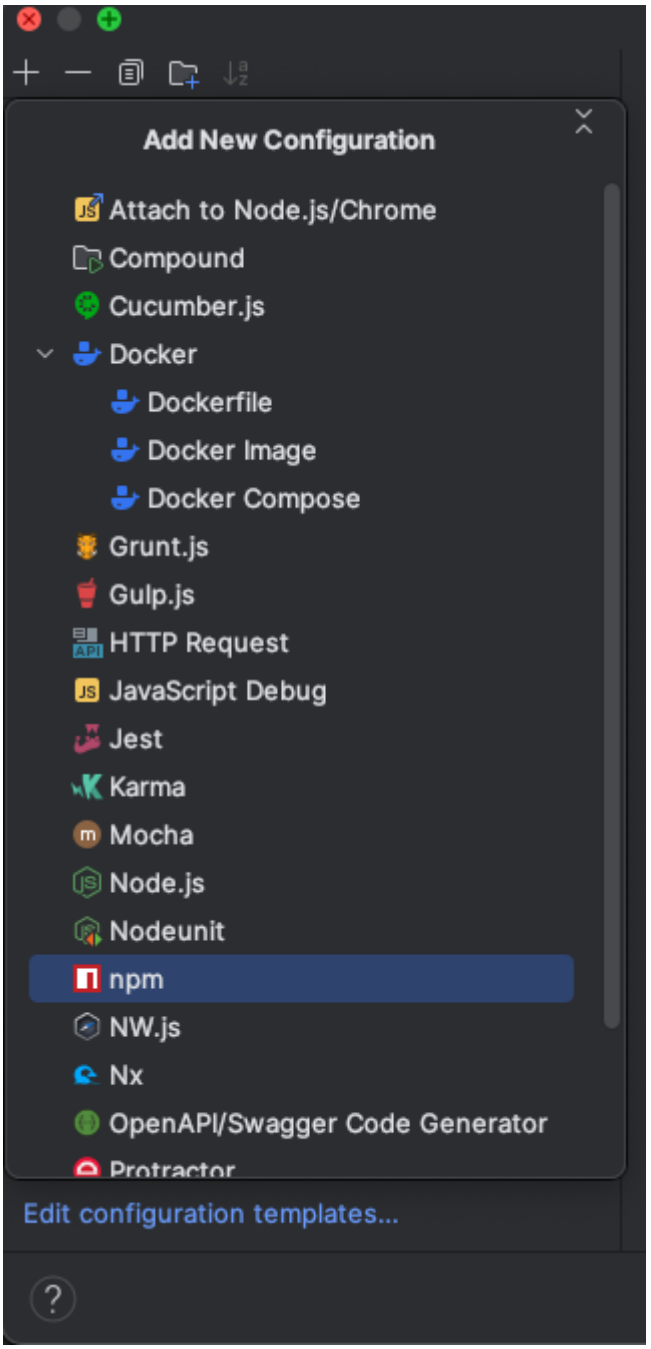
Configurazione Client-Setup

Per avviare il client setup su Webstorm è necessario effettuare i seguenti passaggi:

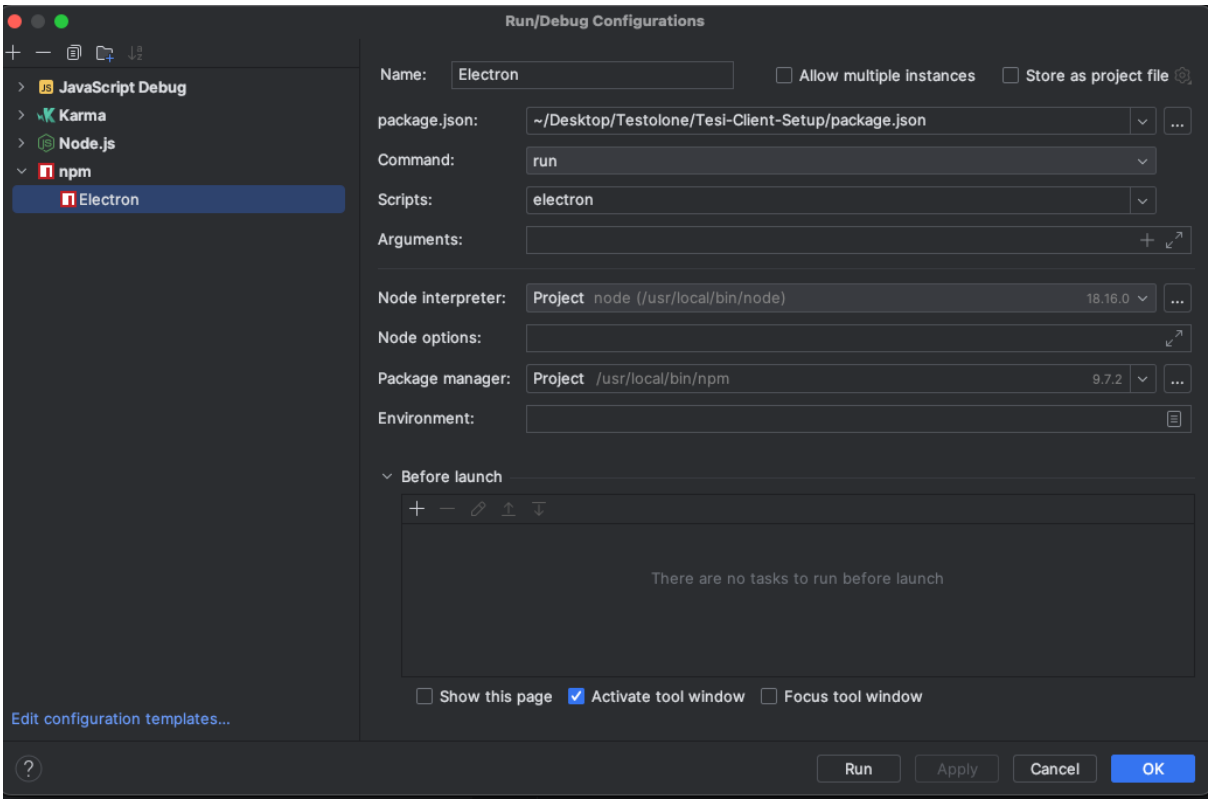
1) Creare una nuova configurazione su Webstorm


Electron
vice – utils.js
c/main/ja
Pinned
Unnamed
Angular Application
Tests (dariotintore_frontend)
Unnamed
Electron
Current File
Edit Configurations...

2) Premere il tasto + e selezionare npm



3) La configurazione deve avere le seguenti impostazioni



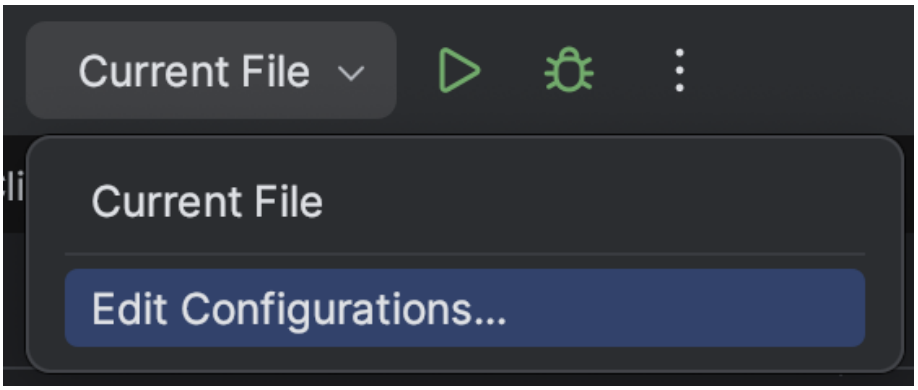
Command : run
Scripts: electron

In questo modo avremo la configurazione (che nel mio caso si chiamerà electron ma che è possibile rinominare)

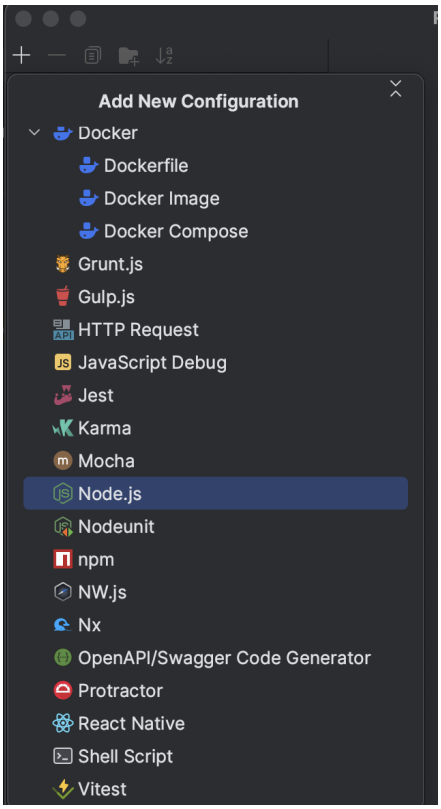
che potremo avviare da Webstorm direttamente e verificare le modifiche effettuate

Configurazione Compiler Service

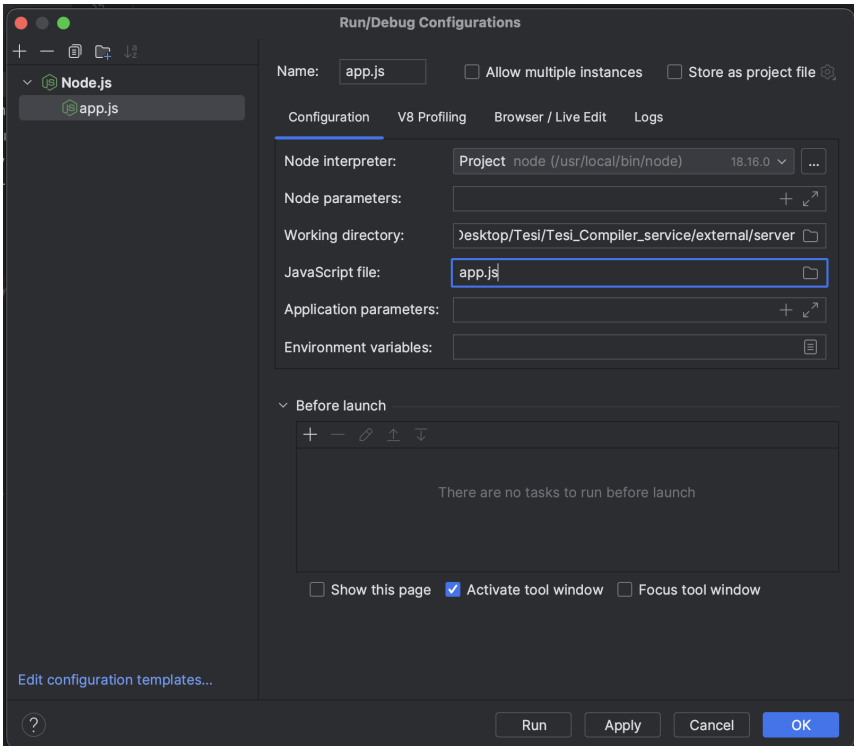
1)



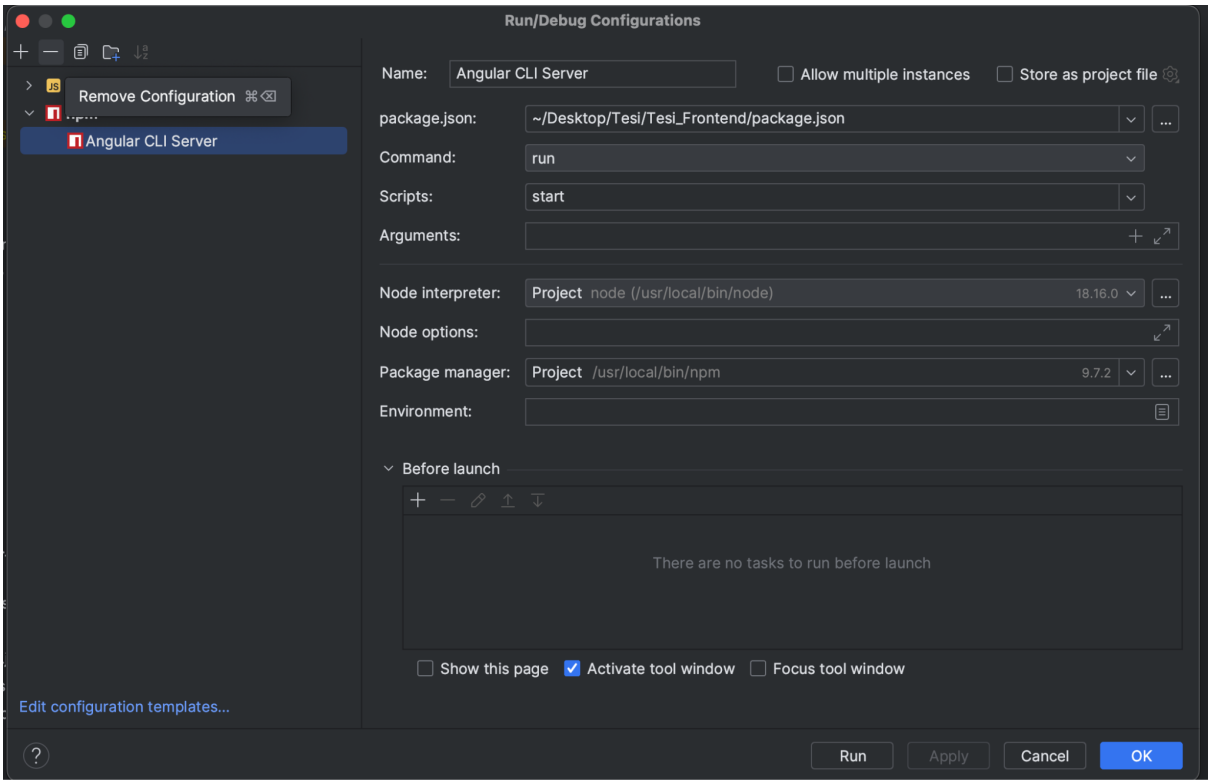
2)



3) Inserire la working direcotry “external/server”
con javascript file : app.js



Configurazione Frontend Online



Docker

Nel momento in cui sarà necessario pushare su docker una nuova immagine di uno dei vari microservizi, sarà necessario creare una repository su dockerhub e cambiare i riferimenti sul docker-compose in modo tale da effettuare il pull delle nuove immagini modificate e non le mie.
I docker file sono presenti all'interno delle repository quindi non sarà necessario crearne di nuovi, l'unica cosa necessaria sarà:

1) Effettuare le modifiche
2) Creare una nuova immagine
3) Pushare le immagini su un dockerhub personale
4) Cambiare i riferimenti al docker-compose
5) effettuare il docker-compose up

Esempio di script utilizzato per effettuare il build ed il push

```
docker build . -t compiler-service-node

docker tag compiler-service-node
dariotintore/compiler-service-node
docker push dariotintore/compiler-service-node
```

Entrypoint Progetti

User Service	UserServiceApplication.java
Compiler Service	CompilerServiceApplication.java
Leaderboard Service	LeaderboardServiceApplication.java
Exercise Service	ExerciseServiceApplication.java

Endpoint

User Service	http://localhost:8081
Compiler Service	http://localhost:8083
Leaderboard Service	http://localhost:8087
Exercise Service	http://localhost:9090
Frontend Online	http://localhost:4200