Comment développer et créer une application angulaire avec le backend Java

Apprenez comment vous développez et construisez avec un exemple de projet

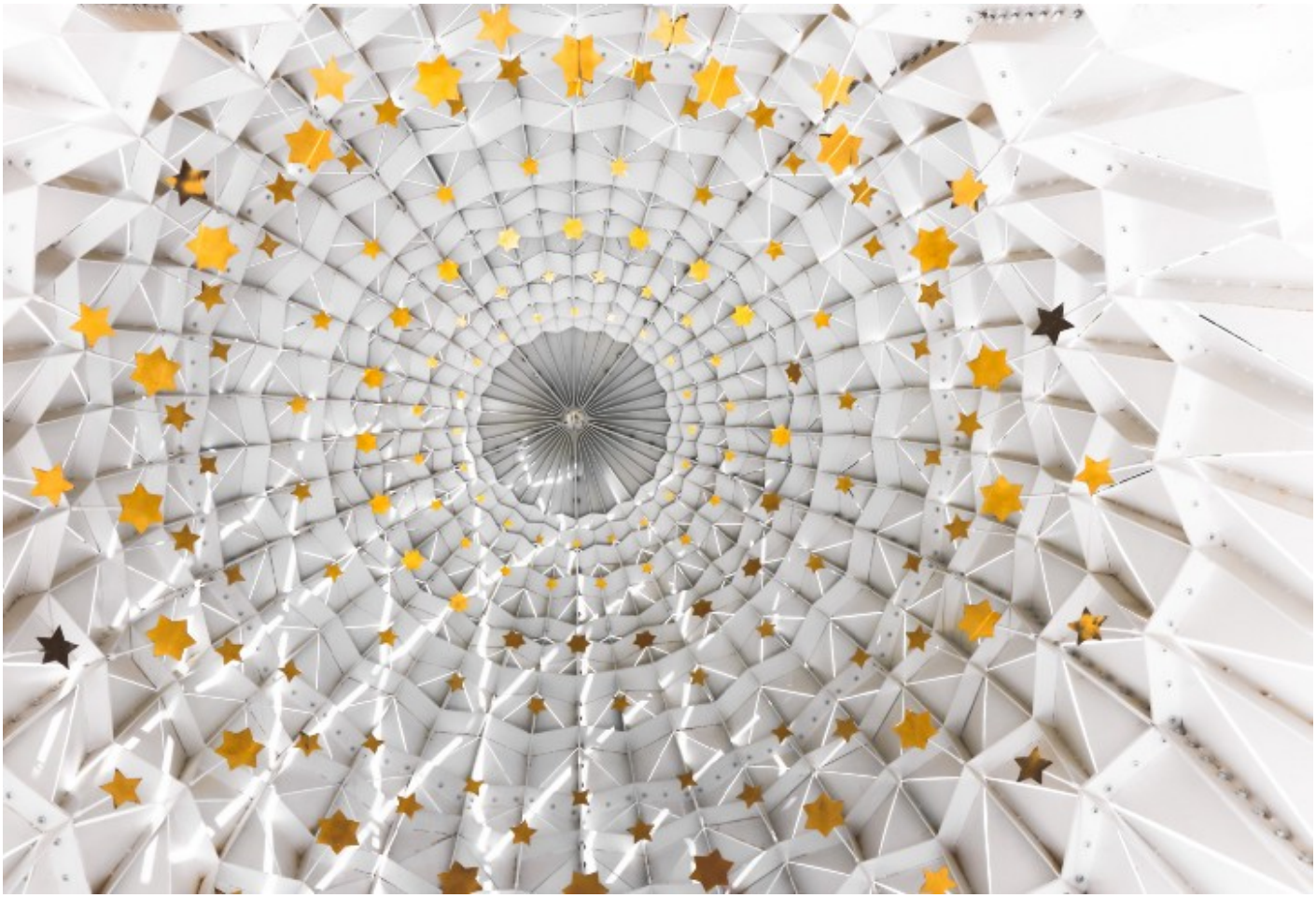


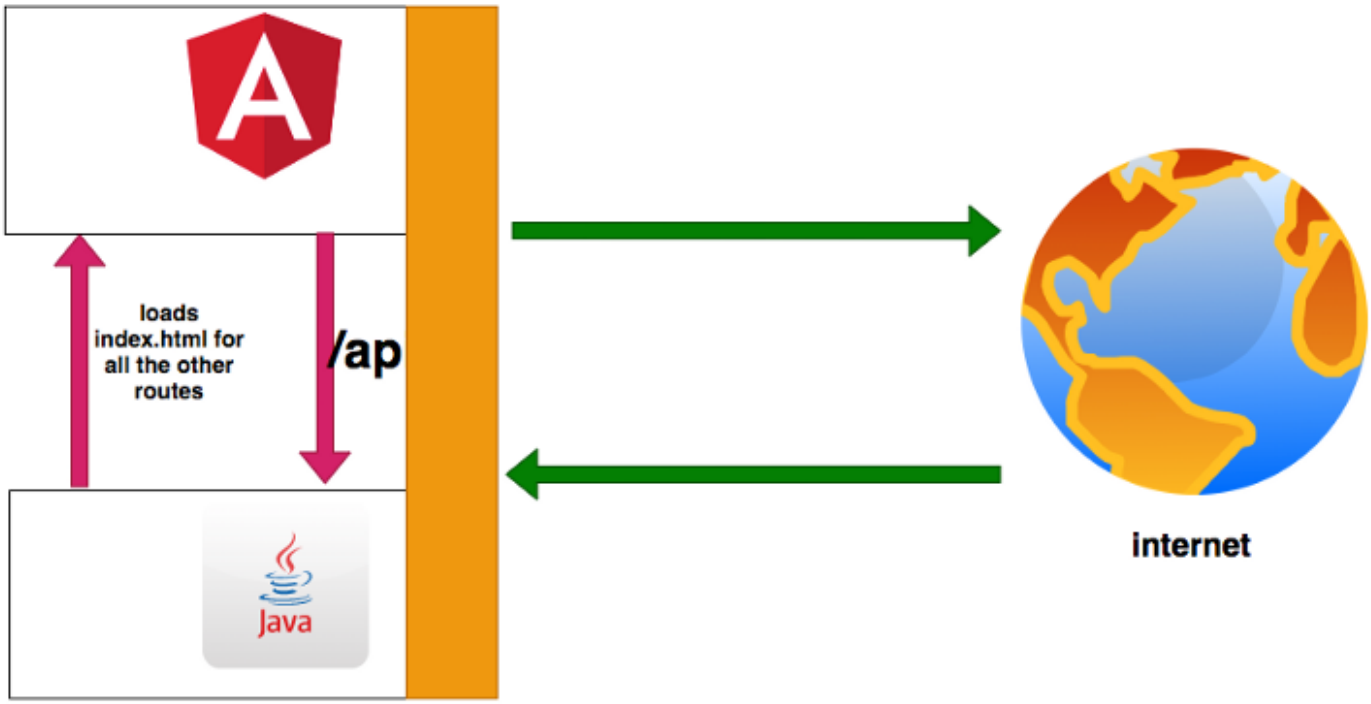
Photo de Martin Adams sur Unsplash

Il existe de nombreuses façons de créer des applications angulaires et de les expédier pour la production. Une façon consiste à créer Angular avec NodeJS ou Java et une autre consiste à créer l'angular et à servir ce contenu statique avec le serveur Web NGINX. Avec Java, nous devons également gérer le code du serveur, par exemple, vous devez charger la page index.html avec java.

Dans cet article, nous verrons les détails et l'implémentation avec Java. Nous allons parcourir étape par étape avec un exemple.

- ***introduction***
- ***Conditions préalables***

Angular est un framework javascript pour créer des applications Web et il ne se charge pas dans le navigateur. Nous avons besoin d'une sorte de mécanisme qui charge l'index.html (page unique) d'Angular avec toutes les dépendances (fichiers CSS et js) dans le navigateur. Dans ce cas, nous utilisons java comme serveur Web qui charge les actifs Angular et accepte tous les appels d'API de l'application Angular.



Angulaire avec Java

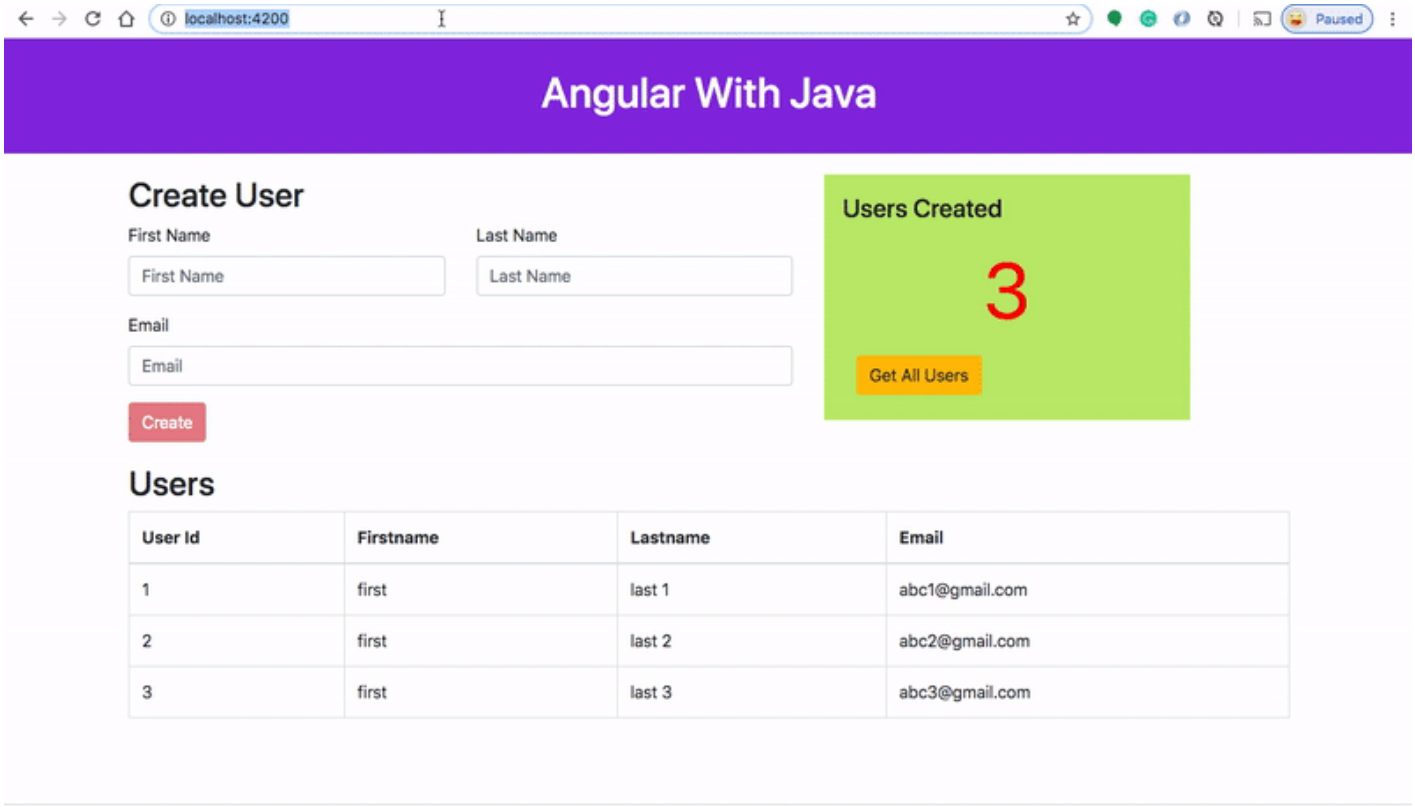
Si vous regardez le diagramme ci-dessus, toutes les requêtes Web sans **/ api** iront au routage angulaire. Tous les chemins contenant **/ api** seront gérés par le serveur Java.

Conditions préalables

Il existe quelques prérequis pour cet article. Vous devez avoir installé java sur votre ordinateur portable et comment fonctionne http. Si vous voulez vous entraîner et l'exécuter sur votre ordinateur portable, vous devez les avoir sur votre ordinateur portable.

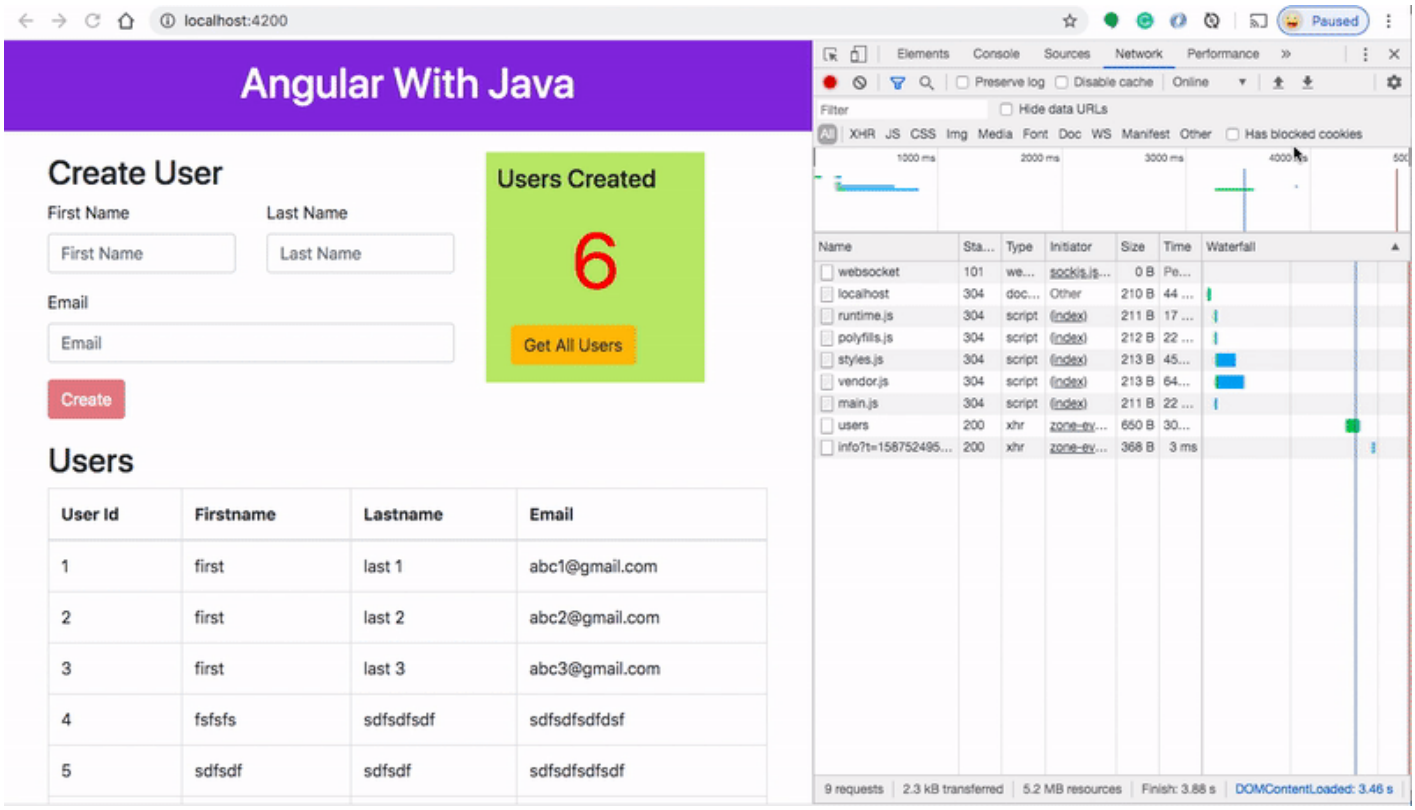
- Java
- CLI angulaire
- Manuscrit
- VSCode
- ngx-bootstrap
- Maven

Il s'agit d'un projet simple qui montre le développement et l'exécution d'une application Angular avec Java. Nous avons une application simple dans laquelle nous pouvons ajouter des utilisateurs, les compter et les afficher sur le côté, et les récupérer quand vous le souhaitez.



Exemple de projet

Lorsque vous ajoutez des utilisateurs, nous faisons un appel API au serveur Java pour les stocker et obtenir les mêmes données du serveur lorsque nous les récupérons. Vous pouvez voir les appels réseau dans la vidéo suivante.



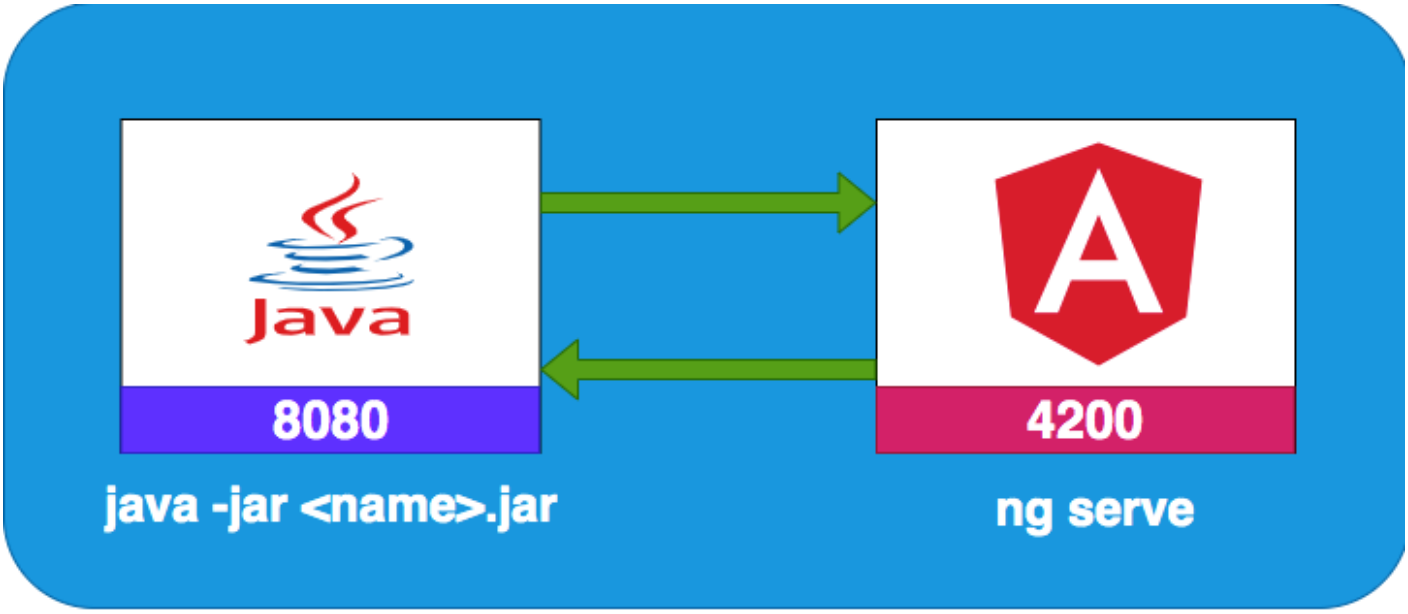
Appels réseau

Voici un lien Github vers ce projet. Vous pouvez le cloner et l'exécuter sur votre machine.

```
// clone the project
git clone https://github.com/bbachi/angular-java-example.git
// Run Angular on port 4200
cd /src/main/ui
npm install
npm start
// Run Java Code on 8080
mvn clean install
java -jar target/users-0.0.1-SNAPSHOT.jar
```

Habituellement, la façon dont vous développez et la façon dont vous créez et exécutez en production sont complètement différentes. C'est pourquoi, je voudrais définir deux phases: la phase de développement et la phase de production.

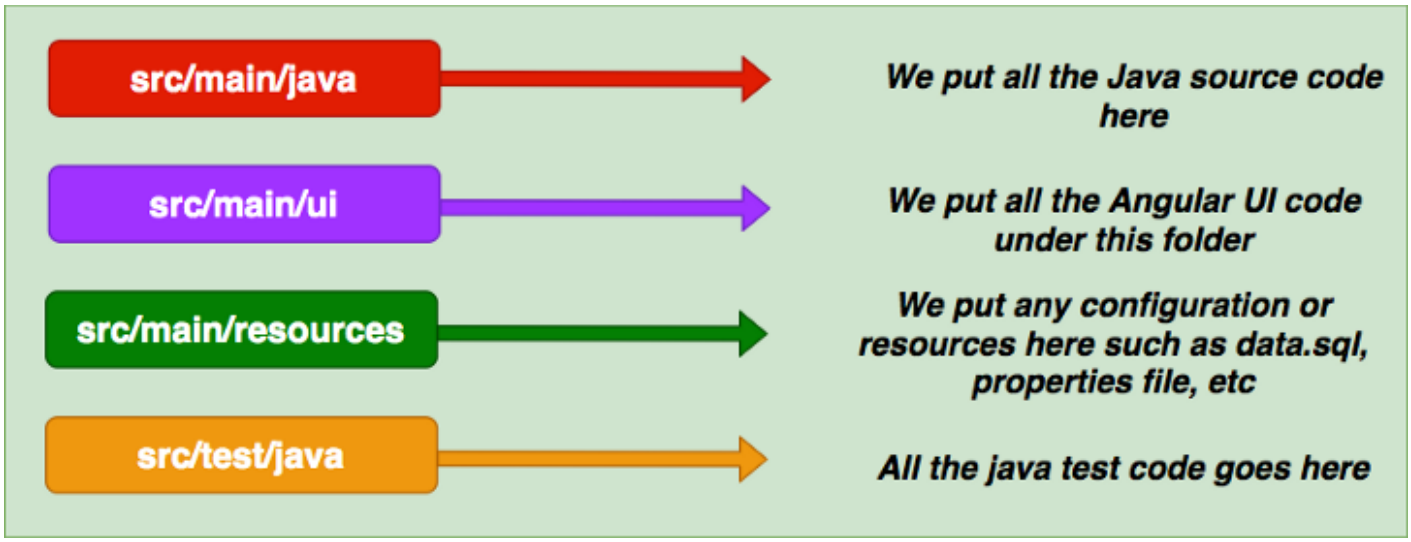
Dans la phase de développement, nous exécutons le serveur java et l'application Angular sur des ports complètement différents. C'est plus facile et plus rapide à développer de cette façon. Si vous regardez le diagramme suivant, l'application Angular s'exécute sur le port **4200** à l'aide d'un serveur de développement Webpack et le serveur Java s'exécute sur le port **8080** .



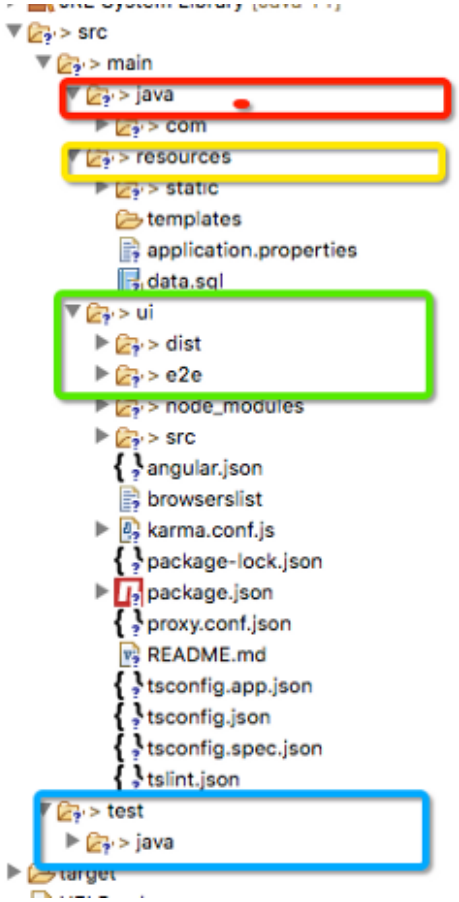
Environnement de développement

Structure du projet

Comprenons la structure du projet pour ce projet. Nous avons besoin de deux dossiers complètement différents pour java et angular. Il est toujours préférable d'avoir des dossiers complètement différents pour chacun. De cette façon, vous aurez une architecture propre ou tout autre problème concernant la fusion de fichiers.



Structure du projet



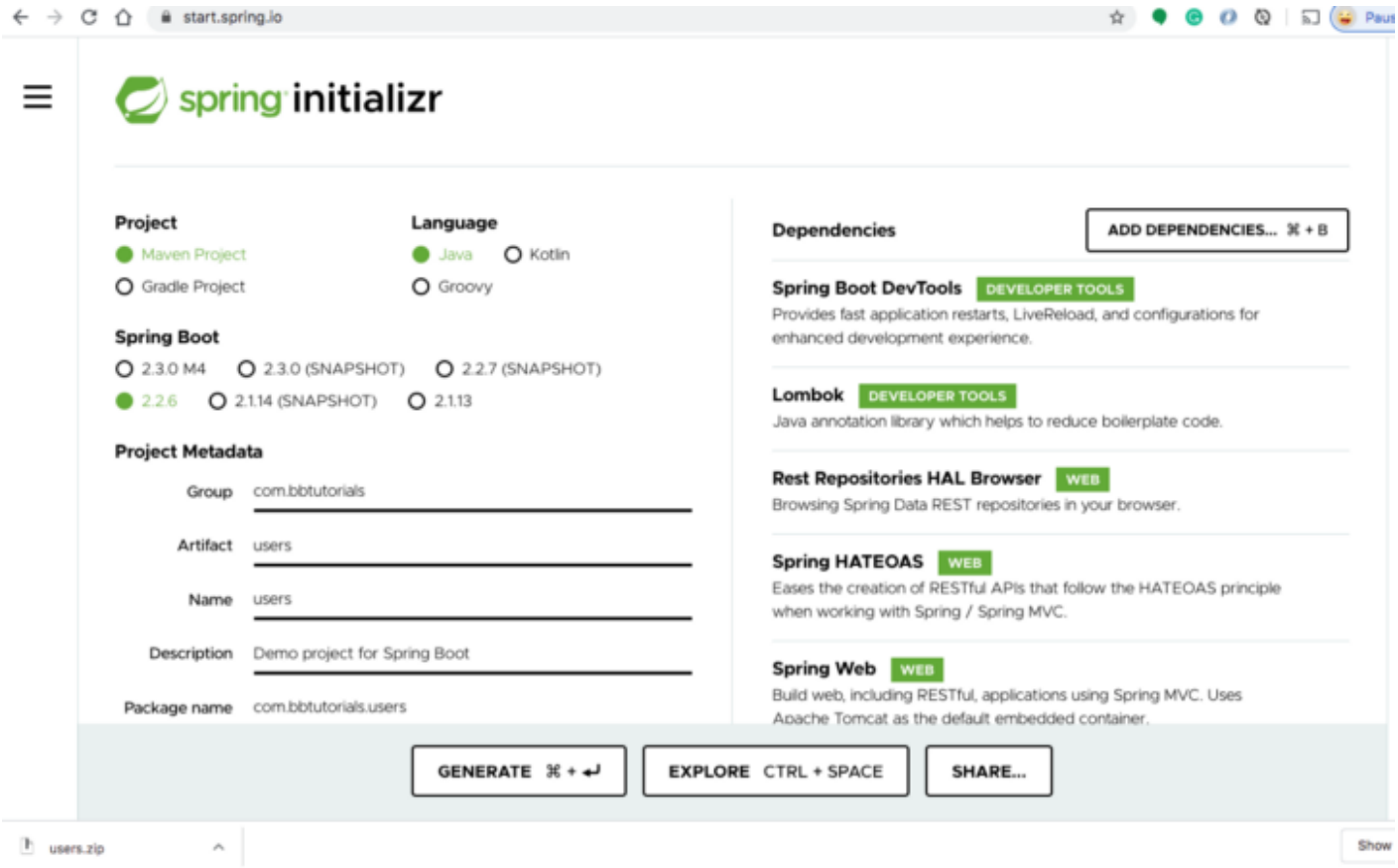
Structure du projet

Si vous regardez la structure du projet ci-dessus, toute l'application Angular réside dans le dossier ***src / main / ui*** et le code Java se trouve dans le dossier ***src / main / java*** . Toutes les ressources se trouvent dans le dossier ***/ src / main / resources*** telles que les propriétés, les actifs statiques, etc.

API Java

Nous utilisons Spring Boot et de nombreux autres outils tels que Spring Devtools, Spring Actuator, etc. sous le parapluie à ressort. De nos jours, presque toutes les applications ont un démarrage à ressort et il s'agit d'un framework open-source basé sur Java utilisé pour créer un micro service. Il est développé par l'équipe Pivotal et est utilisé pour créer des applications de **ressort** autonomes et prêtes pour la production .

Nous commençons par Spring initializr et sélectionnons toutes les dépendances et générons le fichier zip.



Une fois que vous avez importé le fichier zip dans eclipse ou dans tout autre IDE en tant que projet Maven, vous pouvez voir toutes les dépendances dans le **pom.xml.** Vous trouverez ci-dessous la section des dépendances de pom.xml.

```
	<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-actuator</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-jpa</artifactId>
	</dependency>
	<dependency>
		<groupId>com.h2database</groupId>
		<artifactId>h2</artifactId>
		<scope>runtime</scope>
		<version>1.4.199</version>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-rest</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-hateoas</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-rest-hal-browser</artifactId>
    </dependency>
    <!-- QueryDSL -->
    <dependency>
        <groupId>com.querydsl</groupId>
        <artifactId>querydsl-apt</artifactId>
    </dependency>
    <dependency>
        <groupId>com.querydsl</groupId>
        <artifactId>querydsl-jpa</artifactId>
    </dependency>

    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    </dependencies>
```

pom.xml hosted with ❤ by GitHub view raw

Voici le fichier de démarrage de printemps et le contrôleur avec deux routes, l'une avec la requête GET et l'autre avec la requête POST.

```java
package com.bbtutorials.users;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class UsersApplication {

        public static void main(String[] args) {
           SpringApplication.run(UsersApplication.class, args);
        }

}
```

UsersApplication.java hosted with ❤ by GitHub view raw

```java
package com.bbtutorials.users.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.bbtutorials.users.entity.Users;
import com.bbtutorials.users.links.UserLinks;
import com.bbtutorials.users.service.UsersService;

import lombok.extern.slf4j.Slf4j;

@Slf4j
@RestController
@RequestMapping("/api/")
public class UsersController {

    @Autowired
    UsersService usersService;

    @GetMapping(path = UserLinks.LIST_USERS)
    public ResponseEntity<?> listUsers() {
        log.info("UsersController:  list users");
        List<Users> resource = usersService.getUsers();
        return ResponseEntity.ok(resource);
    }
```

```
        @PostMapping(path = UserLinks.ADD_USER)
        public ResponseEntity<?> saveUser(@RequestBody Users user) {
        log.info("UsersController:  list users");
        Users resource = usersService.saveUser(user);
        return ResponseEntity.ok(resource);
    }
}
```

UsersController.java hosted with ❤ by GitHub view raw

Configurer la base de données H2

Cette base de données H2 est uniquement destinée au développement. Lorsque vous créez ce projet pour la production, vous pouvez le remplacer par n'importe quelle base de données de votre choix. Vous pouvez exécuter cette base de données de manière autonome sans votre application. Nous verrons comment nous pouvons configurer avec Spring Boot.

Tout d'abord, nous devons ajouter des propriétés au fichier application.properties sous **/ src / main / resources**

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

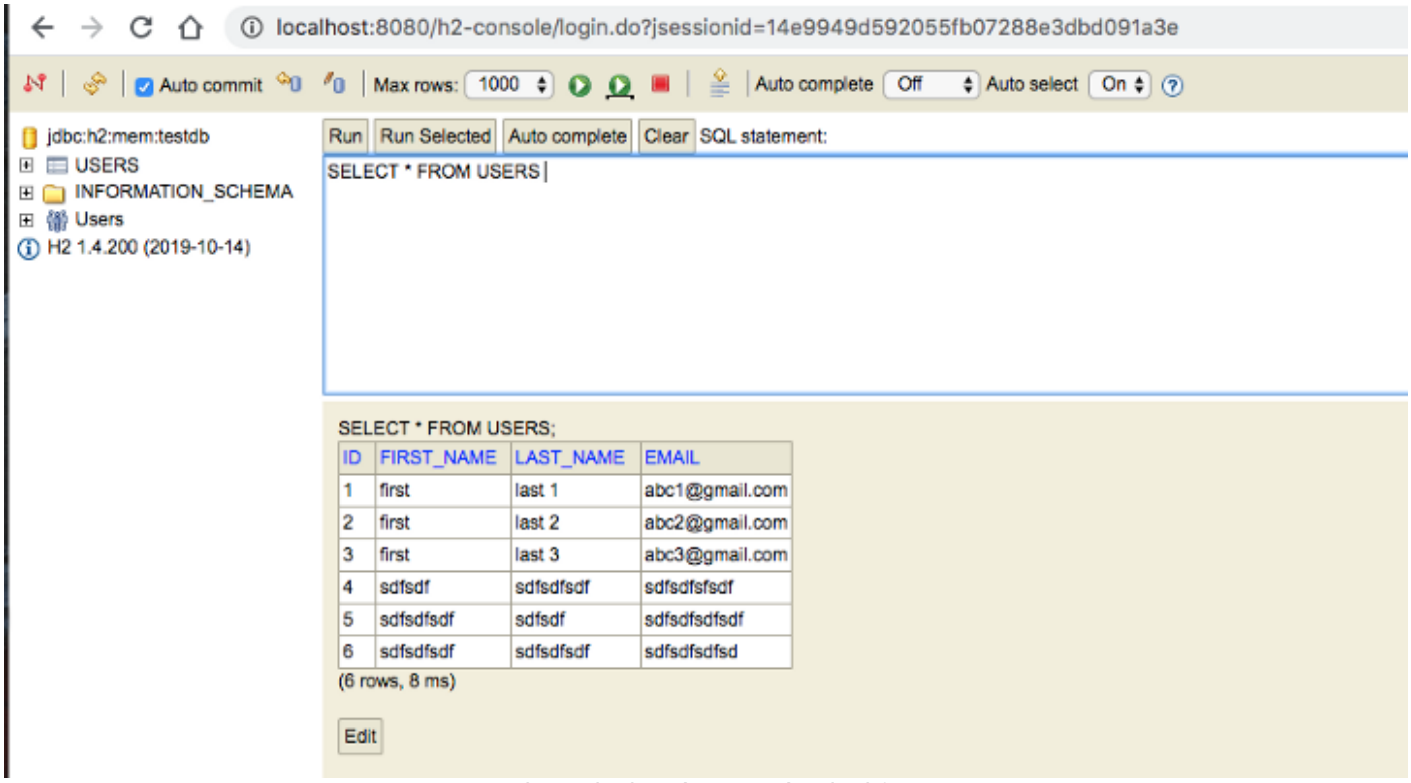
application.properties hosted with ❤ by GitHub view raw

Deuxièmement, ajoutez le fichier SQL ci-dessous au même emplacement.

```
DROP TABLE IF EXISTS users;

CREATE TABLE users (
  id INT PRIMARY KEY,
  FIRST_NAME VARCHAR(250) NOT NULL,
  LAST_NAME VARCHAR(250) NOT NULL,
  EMAIL VARCHAR(250) NOT NULL
);

INSERT INTO users (ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES
  (1, 'first', 'last 1', 'abc1@gmail.com'),
  (2, 'first', 'last 2', 'abc2@gmail.com'),
  (3, 'first', 'last 3', 'abc3@gmail.com');
```

data.sql hosted with ❤ by GitHub view raw

Troisièmement, démarrez l'application et Spring Boot crée cette table au démarrage. Une fois l'application lancée, vous pouvez accéder à cette URLhttp://localhost:8080/api/h2-consoleet accédez à la base de données sur le navigateur Web. Assurez-vous d'avoir la même URL JDBC, le même nom d'utilisateur et le même mot de passe que dans le fichier de propriétés.



base de données en mémoire h2

Ajoutons les fichiers de référentiel, les fichiers de service et les classes d'entité comme ci-dessous et démarrons l'application Spring Boot.

```
package com.bbtutorials.users.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.validation.constraints.NotNull;


import lombok.Data;

@Entity
@Data
public class Users {

        @Id
        @Column
    private long id;

    @Column
    @NotNull(message="{NotNull.User.firstName}")
    private String firstName;

    @Column
    @NotNull(message="{NotNull.User.lastName}")
    private String lastName;

    @Column
```

```
        @NotNull(message="{NotNull.User.email}")
        private String email;

    }
```

Users.java hosted with ❤ by GitHub view raw

```
package com.bbtutorials.users.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.springframework.data.querydsl.QuerydslPredicateExecutor;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import com.bbtutorials.users.entity.Users;

@RepositoryRestResource()
public interface UsersRepository extends JpaRepository<Users, Integer>, JpaSpecificationExecutor<Users>,
```

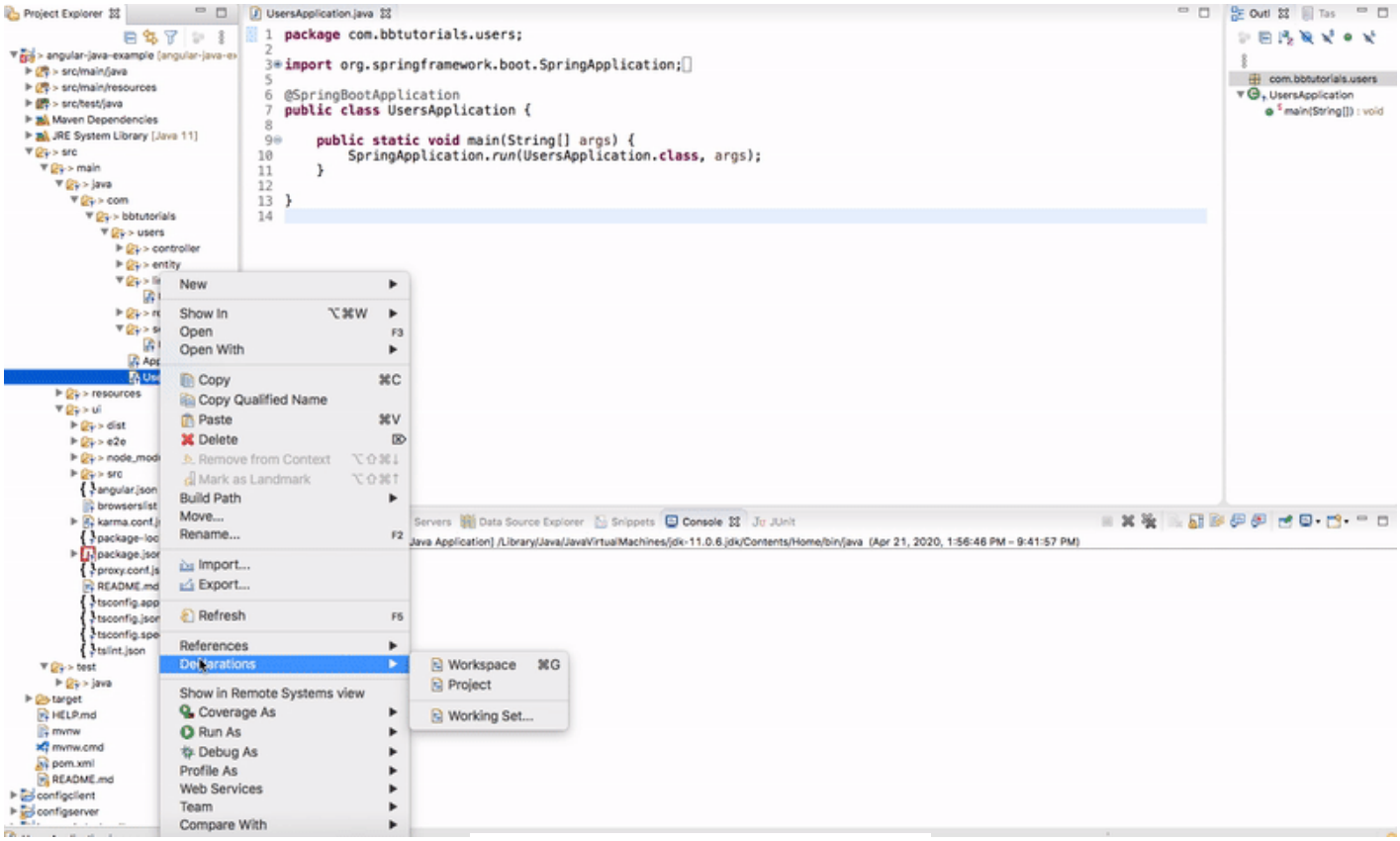
UsersRepository.java hosted with ❤ by GitHub view raw

```
package com.bbtutorials.users.service;

import java.util.List;

import org.springframework.stereotype.Component;

import com.bbtutorials.users.entity.Users;
import com.bbtutorials.users.repository.UsersRepository;

@Component
public class UsersService {

        private UsersRepository usersRepository;

    public UsersService(UsersRepository usersRepository) {
        this.usersRepository = usersRepository;
    }

    public List<Users> getUsers() {
        return usersRepository.findAll();
    }

    public Users saveUser(Users users) {
        return usersRepository.save(users);
    }

}
```

UsersService.java hosted with ❤ by GitHub view raw

Vous pouvez démarrer l'application de deux manières: vous pouvez cliquer avec le bouton droit de la souris sur UsersApplication et l'exécuter en tant qu'application java ou effectuer les étapes suivantes.

```
// mvn install
mvn clean install
// run the app
java -jar target/<repo>.war
```



starting the spring boot application

Code Java s'exécutant sur le port 8080

Application angulaire

Maintenant, le code java s'exécute sur le port **8080** . Il est maintenant temps de regarder l'application Angular. L'ensemble de l'application Angular se trouve dans le dossier ***src / main / ui*** . Vous pouvez créer avec cette commande `ng new ui.` je ne vais pas mettre tous les fichiers ici, vous pouvez regarder les fichiers entiers dans le lien Github ci-dessus ou ici.

Voyons quelques fichiers importants ici. Voici le fichier de service qui appelle l'API Java.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class AppService {

  constructor(private http: HttpClient) { }

  rootURL = '/api';

  getUsers() {
    return this.http.get(this.rootURL + '/users');
  }

  addUser(user: any, id: number) {
        user.id = id;
        return this.http.post(this.rootURL + '/user', user);
  }

}
```

app.service.ts hosted with ❤ by GitHub view raw

Voici le composant d'application qui s'abonne à ces appels et récupère les données de l'API.

```
import { Component, OnDestroy } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { AppService } from './app.service';
import { takeUntil } from 'rxjs/operators';
import { Subject } from 'rxjs';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnDestroy {

  constructor(private appService: AppService) {}
```

```
    title = 'angular-nodejs-example';

    userForm = new FormGroup({
      firstName: new FormControl('', Validators.nullValidator && Validators.required),
      lastName: new FormControl('', Validators.nullValidator && Validators.required),
      email: new FormControl('', Validators.nullValidator && Validators.required)
    });

    users: any[] = [];
    userCount = 0;

    destroy$: Subject<boolean> = new Subject<boolean>();

    onSubmit() {
      this.appService.addUser(this.userForm.value, this.userCount + 1).pipe(takeUntil(this.destroy$)).subscr
        console.log('message::::', data);
        this.userCount = this.userCount + 1;
        console.log(this.userCount);
        this.userForm.reset();
      });
    }

    getAllUsers() {
      this.appService.getUsers().pipe(takeUntil(this.destroy$)).subscribe((users: any[]) => {
                  this.userCount = users.length;
          this.users = users;
      });
    }

    ngOnDestroy() {
      this.destroy$.next(true);
      this.destroy$.unsubscribe();
    }

    ngOnInit() {
          this.getAllUsers();
    }
  }
```

app.component.ts hosted with ❤ by GitHub view raw

Interaction entre l'API Angular et Java

Dans la phase de développement, l'application Angular s'exécute sur le port **4200** à l'aide d'un serveur de développement Webpack et d'une API Java s'exécutant sur le port **8080** .

Il devrait y avoir une certaine interaction entre ces deux. Nous pouvons proxy tous les appels d'API vers l'API Java. Angular fournit une méthode de proxy intégrée. Tout d' abord, nous devons définir les éléments suivants **proxy.conf.json** sous **src / main / ui** dossier.

```
{
```

```
    "/api": {
      "target": "http://localhost:8080",
      "secure": false
    }
  }
```

proxy.conf.json hosted with ❤ by GitHub view raw

Si vous regardez le fichier, tous les chemins commençant par **/ api** seront redirigés vers **http: // localhost: 8080** où l'API Java s'exécute. Ensuite, vous devez définir dans angular.json sous la partie servir avec la clé proxyConfig. Voici le fichier angular.json complet.

```
"serve": {
  "builder": "@angular-devkit/build-angular:dev-server",
  "options": {
    "browserTarget": "angular-nodejs-example:build",
    "proxyConfig": "proxy.conf.json"
  },
  "configurations": {
    "production": {
      "browserTarget": "angular-nodejs-example:build:production"
    }
  }
},
```

angular.json

Une fois cela configuré, vous pouvez exécuter l'application Angular sur le port **4200** et l'API java sur **8080** pour les faire fonctionner ensemble.

```
// java API (Terminal 1)
mvn clean install
java -jar target/<war file name>
// Angular app (Terminal 2)
npm start
```

Comme vous l'avez vu ci-dessus, nous exécutons le serveur Angular et Java sur différents ports en phase de développement. Mais, lorsque vous créez l'application pour la production, vous devez emballer votre code Angular avec Java et l'exécuter sur un port. J'ai écrit un autre article car voici le lien.

- Comment créer Angular avec le backend Java pour la production

- Il existe de nombreuses façons de créer des applications angulaires et de les expédier pour la production.
- Une façon est de construire Angular avec Java.
- Dans la phase de développement, nous pouvons exécuter Angular et Java sur des ports séparés.

- L'interaction entre ces deux se produit avec le proxy de tous les appels à l'API.
- Dans la phase de production, vous pouvez créer l'application Angular et placer tous les actifs dans le dossier dist et le charger avec le code java.
- Nous pouvons conditionner l'application de plusieurs manières.

C'est une façon de créer et de diffuser des applications angulaires. Ceci est vraiment utile lorsque vous souhaitez effectuer un rendu côté serveur ou que vous devez effectuer un traitement. Dans les prochains articles, je discuterai plus en détail de la construction pour la production et du déploiement de stratégies.