Exposer un «Web service» à travers une interface REST

Objectif:

Apprendre comment developper une application SPA (Single Page Application), et qui offre un Web Service, en utilisant Angular 8 comme interface frontale (front-end) et Spring boot API (Application Programming Interface) CRUD (Create Read Update Delete) RESTful comme backend.

Quelques définitions:

---**API:**

L'API, pour Application Programming Interface, est la partie du programme qu'on expose officiellement au monde extérieur pour manipuler celui-ci. L'API est au développeur ce que l'User Interface est à l'utilisateur. Cette dernière permet d'entrer des données et de les récupérer la sortie d'un traitement. Initialement, une API regroupe un ensemble de fonctions ou méthodes, leurs signatures et ordre d'usage pour obtenir un résultat.

La mise en place d'une API permet d'opérer une séparation des responsabilités entre le client et le serveur. Cette séparation permet donc une portabilité et évolutivité grandement améliorées. Chaque composant peut évoluer séparément car il n'y a aucun logique du côté du serveur. Ains on peut imaginer une refonte totale de la charte graphique du site web sans devoir modifier le code côté serveur ou sur les autres clients (mobiles par exemple).

Réf:

https://www.supinfo.com/articles/single/5642-qu-est-ce-qu-une-api-rest-restful

---**CRUD:**

Le terme CRUD est étroitement lié avec la gestion des données numériques. Plus précisément, CRUD est un acronyme des noms des quatre opérations de base de la gestion de la persistance des données et applications :

- Create (créer),
- Read ou Retrieve (lire),
- Update (mettre à jour),
- Delete ou Destroy (supprimer).

Plus simplement, le terme CRUD résume les fonctions qu'un utilisateur a besoin d'utiliser pour créer et gérer des données. Divers processus de gestion des données sont basés sur CRUD, cependant les opérations sont spécifiquement adaptées aux besoins des systèmes et des utilisateurs, que ce soit dans la gestion des bases de données ou pour l'utilisation des applications. Ainsi, les opérations sont des outils d'accès classiques et indispensables avec lesquels les experts, peuvent par exemple vérifier les problèmes de base de données. Tandis que pour un utilisateur, CRUD signifie la création d'un compte (create), l'utilisation à tout moment (read), la mise à jour (update) ou encore la suppression (delete).
Réf:
https://www.ionos.fr/digitalguide/sites-internet/developpement-web/crud-les-operations-de-base-de-donnees-les-plus-importantes/

--- **Web Service**:
Un service web (ou Web service) est un médium standardisé permettant la communication entre les applications clients et serveur sur le World Wide Web. Il s'agit d'un module logiciel conçu pour effectuer certaines tâches.
Une fois invoqué, un service web est en mesure de fournir ses fonctionnalités au client qui l'invoque. Le client invoque une série d'appels de service web par le biais de requêtes envoyées au serveur qui héberge le service. Ces requêtes sont effectuées par le biais d'appels de procédure distante (Remote Procedure Calls).
Par exemple, Amazon propose un service web fournissant les prix pour des produits vendus en ligne via Amazon.com. Le front end ou la couche de présentation peuvent être en .Net ou en Java, mais ces deux langages de programmation auront la capacité de communiquer avec le service web.
Le principal composant d'un service web sont les données transférées entre le client et le serveur. Ces données sont en XML (Extensible Markup Language). Le XML est la contrepartie du HTML. Pour faire simple, on peut le décrire comme un langage intermédiaire compris par la plupart des langages de programmation. Ainsi, les applications communiquent entre elles en XML.
Réf: https://www.lebigdata.fr/services-web-definition

---**API REST ou API SOAP?**

Il existe actuellement deux types d'architecture très utilisées pour les API : Simple Object Access Protocol (SOAP)et Representational State Transfer (REST). SOAP et REST sont deux solutions permettant à un client d'accéder à des services web. Le choix d'abord peut sembler facile, mais parfois il peut être étonnamment difficile. D'un côté, SOAP, initialement développé par Microsoft, est un protocole d'accès aux services Web qui existe depuis un certain temps. De l'autre, l'architecture REST est la nouvelle venue. Elle vise à résoudre certains problèmes rencontrés avec SOAP et donner la possibilité de mettre en place une méthode vraiment simple afin d'accéder à des services web.

Les deux techniques ont des problèmes à prendre en compte au moment de décider quel protocole utiliser. Avant d'aller plus loin, il est important de préciser que même si SOAP et REST présentent des similitudes en utilisant le protocole HTTP, SOAP est un ensemble plus rigide que REST. REST a une architecture qui ne nécessite pas de traitement et qui est naturellement plus flexible. SOAP et REST reposent sur des règles bien établies que tout le monde a accepté de respecter dans l'intérêt de l'échange d'informations.

Réf:

https://www.supinfo.com/articles/single/5642-qu-est-ce-qu-une-api-rest-restful

--- **API RESTfull:**

REST (Representational State Transfer) ou RESTful est un style d'architecture permettant de construire des applications (Web, Intranet, Web Service). Il s'agit d'un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie à part entière. L'architecture REST utilise les spécifications originelles du protocole HTTP, plutôt que de réinventer une surcouche (comme le fait SOAP). Pour créer une APi RESTfull, on doit respecter les règles suivantes:

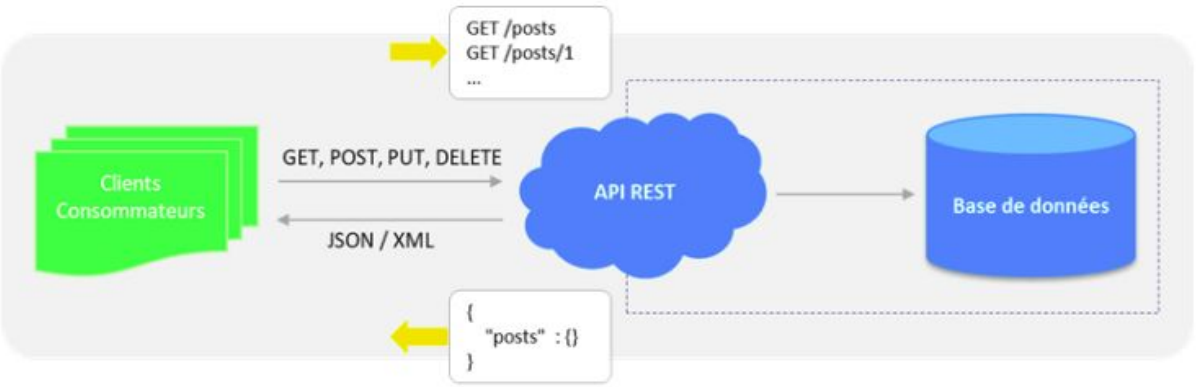
Règle n°1 : l'URI comme identifiant des ressources.

Règle n°2 : les verbes HTTP comme identifiant des opérations.

Règle n°3 : les réponses HTTP comme représentation des ressources.

Règle n°4 : les liens comme relation entre ressources.

Règle n°5 : un paramètre comme jeton d'authentification.



Architecture REST

Réf:
https://blog.nicolashachet.com/developpement-php/larchitecture-rest-expliquee-en-5-regles/

Projets à construire:

Pour atteindre votre objectif, vous allez créer deux projets:

1) api-rest-crud-backend-springboot: Ce projet est utilisé pour développer des APIs CRUD RESTFul pour un système de gestion de comptes bancaires simple en utilisant Spring Boot 2, JPA et MySQL comme base de données.
2) angular8-frontend-springboot-client: Ce projet est utilisé pour développer une application de page unique (SPA) en utilisant Angular 8 comme technologie frontale. Cette application Angular 8 utilise les API CRUD Restful développées et exposées par un api-rest-crud-backend-springboot-jpa.

Spécifications des fonctionnalités:

+ Créez un compte bancaire,
+ Mettre à jour un compte bancaire,
+ Voir un compte bancaire,
+ Supprimer un compte bancaire,
+ Lister tous les comptes bancaires créés.

Outils et technologies à utiliser:

➔ Technologies côté serveur (backend):

- Spring Boot (2.0.5.RELEASE): Spring Boot est un sous projet de Spring qui vise à rendre Spring plus facile à utiliser en éliminant plusieurs étapes de configuration. L'objectif de Spring Boot est de permettre aux développeurs de se concentrer sur les tâches techniques et non des tâches de configurations, de déploiements, etc. Ce qui a pour conséquences un gain de temps et de productivité
- JDK (1.8 ou version ultérieure): Le kit de développement Java (JDK: Java Development Kit) est un environnement de développement logiciel utilisé pour développer des applications Java. Il comprend l'environnement d'exécution Java (JRE: Java Runtime Environment), un interpréteur (java), un compilateur (javac), un archiveur (jar), un générateur de documentation (javadoc) et d'autres outils nécessaires au développement Java.
- Spring Framework (5.0.8 RELEASE): Spring est un framework très riche permettant de structurer, d'améliorer et de simplifier l'écriture d'application Java EE. Les applications Spring sont faiblement couplées grâce à l'injection de dépendance. Il fournit des modèles pour JDBC, Hibernate, JPA, etc. Donc, il n'y a pas besoin d'écrire trop de code. Avec Spring, le test des applications est devenu plus facile en utilisant l'injection de dépendance.
- Spring Data JPA (version 2+): Spring Data JPA met en place une surcouche d'accès à JPA (Java Persistence API), et fournit donc ainsi un ensemble cohérent de fonctionnalités avancées sur lequel s'appuyer pour bâtir et consolider ses applications.

➔ Technologies côté front-end:

- Angular (8.0.0 RELEASE): Angular est un framework Javascript côté client qui permet de réaliser des applications de type "Single Page Application". Il est basé sur le concept de l'architecture MVC (Model View Controller) qui permet de séparer les données, les vues et les différentes actions que vous pouvez effectuer.
- Bootstrap 4: Bootstrap est un framework CSS, mais pas seulement, puisqu'il embarque également des composants HTML et JavaScript. Il comporte un système de grille simple et efficace pour mettre en ordre l'aspect visuel d'une page web. Il apporte du style

pour les boutons, les formulaires, la navigation... Il permet ainsi de concevoir un site web rapidement et avec peu de lignes de code ajoutées.

- NPM (6.9.0 RELEASE): NPM (Node Package Manager) est un package manager spécialement conçu pour Node.js. Il vous permet de partager des packages (aussi nommés modules) pour qu'ils soient accessible publiquement, permettant alors à d'autres utilisateurs de les installer simplement dans leur projet.

➔ Outils:

- Maven (3.2+ RELEASE): Maven est un outil permettant d'automatiser la gestion de projets Java. Parmi les fonctionnalités qu'il offre: la compilation et le déploiement des applications Java, la gestion des librairies requises par l'application, l'exécution des tests unitaires et l'intégration du projet Java dans différents IDE.
- Visual Studio Code IDE (possible pour le développement de deux projets)
- Angular CLI (Command Line Interface).

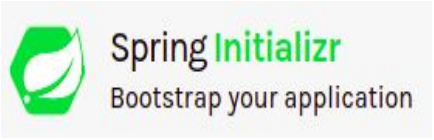
Développement des APIs Spring Boot CRUD REST:

Le tableau suivant résume les cinq APIs REST que vous allez développer pour la ressource “compte bancaire”:

num API	nom API	méthode HTTP	URI	code statut	description
1	GET accounts	GET	/api/v1/accounts	200 (OK)	tous les comptes sont récupérés.
2	POST account	POST	/api/v1/accounts	201 (Created)	nouveau compte est créé.
3	GET account	GET	/api/v1/accounts /{id}	200 (OK)	un seul compte est récupéré selon l’identifiant mentionné.
4	PUT account	PUT	/api/v1/accounts /{id}	200 (OK)	un compte est mis à jour selon l’identifiant

					mentionné.
5	DELETE account	DELETE	/api/v1/accounts /{id}	204 (No Content)	un compte est supprimé selon l'identifiant mentionné.

Création et importation du projet "api-rest-crud-backend-springboot":



Il existe plusieurs façons pour créer une application Spring Boot. Le moyen le plus simple est d'utiliser Spring Initializr sur le lien "http://start.spring.io/". C'est un générateur d'application Spring Boot en ligne.

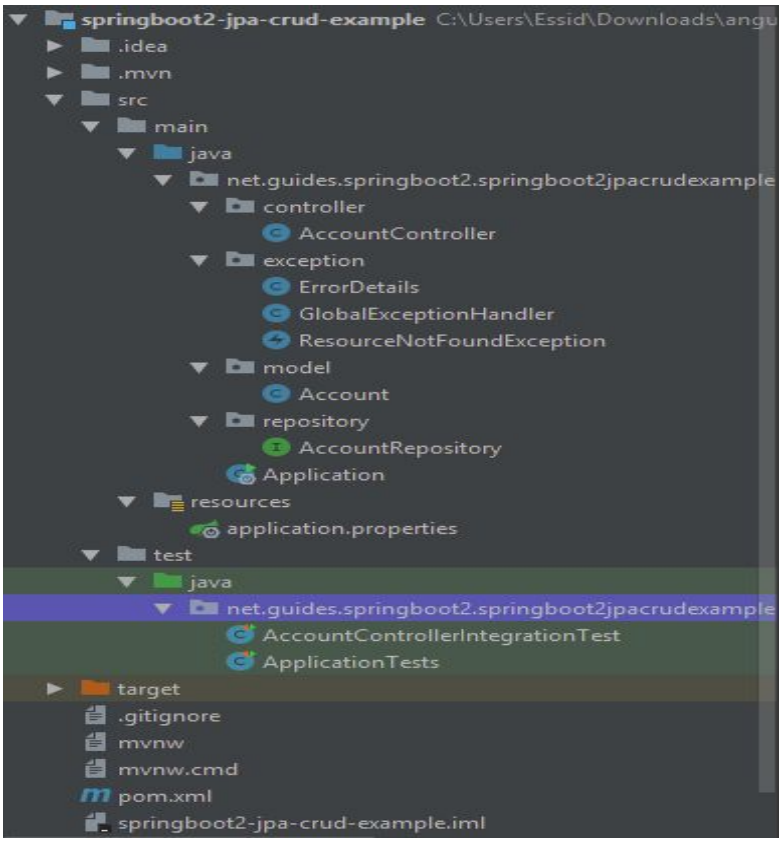
Pour toutes les applications Spring, vous devez commencer par Spring Initializr, car il offre un moyen rapide d'extraire toutes les dépendances dont vous avez besoin pour une application et fait une grande partie de la configuration pour vous.

Pour créer le projet, remplissez les champs proposés comme suit:

Generate: Maven Project
Java Version: 1.8 (Default)
Spring Boot:2.0.4
Group: net.guides.springboot2
Artifact: springboot2-jpa-crud-example
Name: springboot2-jpa-crud-example
Description: Rest API for a Simple Account Management Application
Package Name : net.guides.springboot2.springboot2jpacrudexample
Packaging: jar (This is the default value)
Dependencies: Web, JPA, MySQL, DevTools

Une fois que tous les détails sont entrés, cliquez sur le bouton "Generate Project" pour générer un projet Spring Boot avec les dépendances que vous avez ajouté et le télécharger. Ensuite, décompressez le fichier zip téléchargé et importez-le dans votre Visual Studio Code IDE.

Structure de Packaging du système de gestion de comptes bancaires:



Le fichier pom.xml:

Le projet est géré par Maven comme vous avez choisi dans la création avec Spring Initializr. Il est nécessaire de fournir à Maven une description du projet sous la forme d'un document XML nommé pom.xml dans spring boot et situé à la racine du répertoire contenant le projet. Ce fichier contient aussi toutes les dépendances de l'application comme il est indiqué dans l'extrait suivant:

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
```

Quelques explications des dépendances:

- La dépendance “spring-boot-starter-parent” permet de rapatrier la plupart des dépendances du projet. Sans elle, le fichier pom.xml serait plus complexe.
- La dépendance “spring-boot-starter-web” indique à Spring Boot qu'il s'agit d'une application web, ce qui permet à Spring Boot de rapatrier les dépendances comme SpringMVC, SpringContext, et même le serveur d'application Tomcat, etc. .
- La dépendance “spring-boot-starter-data-jpa” est nécessaire pour le développement de la couche de persistance. Les autres dépendances seront ajoutées notamment pour les besoins de tests.
- Il faut noter aussi l'absence de versions dans les dépendances. Ceci est dû au fait que Spring Boot gère de manière très autonome les versions et nous n'avons plus besoin de déclarer.

Configuration de la base de données MySQL:

Il faut configurer le fichier “application.properties” pour vous connecter à votre base de données MySQL. Ouvrez le fichier “application.properties” et y ajoutez la configuration de base de données suivante. Notez également que vous avez ajouté la dépendance MySQL à “pom.xml” afin que Spring Boot configure automatiquement tous les beans et les configurations liés à la base de données en interne. N’oubliez pas de modifier la configuration ci-dessous telle que l'URL JDBC, le nom d'utilisateur et le mot de passe selon votre environnement.

```
spring.datasource.url =
jdbc:mysql://localhost:3306/Accounts?serverTimezone=UTC&useSSL=false
spring.datasource.username = root
spring.datasource.password =

## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

Création de l’entité JPA “Account.java”:

Les entités dans JPA ne sont rien d'autre que des POJO (plain old Java object) représentant des données qui peuvent être conservées dans la base de données. Une entité représente une table stockée dans une base de données. Chaque instance d'une entité représente une ligne du tableau.
Dans cette application vous allez créer une seule table qui va stocker les comptes bancaires. Donc, vous allez créer “an Account Entity” qui va faire la correspondance avec la base des données. L’entité Account possède 7 attributs. Suivez le code suivant:

```
package net.guides.springboot2.springboot2jpacrudexample.model;

import javax.persistence.Column;
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "accounts")
public class Account {

   private long id;
   private String firstName;
   private String lastName;
   private String cin;
   private String accountNumber;
   private String agency;
   private long amount;

   public Account() {

   }

   public Account(final String firstName, final String lastName, final String cin, final String
accountNumber,
         final String agency, final long amount) {
      this.firstName = firstName;
      this.lastName = lastName;
      this.cin = cin;
      this.accountNumber = accountNumber;
      this.agency = agency;
      this.amount = amount;
   }
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   public long getId() {
      return id;
```

```
}

public void setId(final long id) {
    this.id = id;
}

@Column(name = "first_name", nullable = false)
public String getFirstName() {
    return firstName;
}

public void setFirstName(final String firstName) {
    this.firstName = firstName;
}

@Column(name = "last_name", nullable = false)
public String getLastName() {
    return lastName;
}

public void setLastName(final String lastName) {
    this.lastName = lastName;
}

@Column(name = "cin", nullable = false)
public String getCin() {
    return cin;
}

public void setCin(final String cin) {
    this.cin = cin;
}

@Column(name = "account_number", nullable = false)
public String getAccountNumber() {
```

```
        return accountNumber;
    }

    public void setAccountNumber(final String accountNumber) {
        this.accountNumber = accountNumber;
    }

    @Column(name = "agency", nullable = false)
    public String getAgency() {
        return agency;
    }

    public void setAgency(final String agency) {
        this.agency = agency;
    }

    @Column(name = "amount", nullable = false)
    public long getAmount() {
        return amount;
    }

    public void setAmount(final long amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return "Account [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ",
cin=" + cin + ", accountNumber=" + accountNumber + ", agency=" + agency + ", amount=" +
amount + "]";
    }

}
```

Création du référentiel de données (repository) Spring "AccountRepository.java":

La mise en œuvre de la couche d'accès aux données d'une application est fastidieuse depuis longtemps. Trop de code a dû être écrit. Les classes de domaine étaient anémiques et n'étaient pas conçues de manière orientée objet. L'objectif de l'abstraction du référentiel de Spring Data est de réduire considérablement les efforts requis pour implémenter les couches d'accès aux données.

La dépendance "spring-boot-starter-data-jpa" que vous aviez ajoutée dans le pom.xml permet d'utiliser "SpringData" qui est une implémentation de JPA. "SpringData" implémente toutes méthodes du CRUD (Create, Read, Update, Delete) quand on hérite de JpaRepository ou de CrudRepository. Suivez le code suivant afin de créer une couche d'abstraction pour l'entité Account:

```
package net.guides.springboot2.springboot2jpacrudexample.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import net.guides.springboot2.springboot2jpacrudexample.model.Account;

@Repository
public interface AccountRepository extends JpaRepository<Account, Long>{

}
```

Création de contrôleurs Spring Rest "AccountController.java":

Il s'agit de développer dans cette partie la couche de contrôle. C'est cette couche qui intercepte et filtre toutes les requêtes utilisateurs. Chaque contrôleur dispose d'un service pour traiter les requêtes: c'est le service REST. Chaque service est implémenté par un contrôleur. Suivez le code suivant pour créer les 5 services CRUD REST offerts par l'application:

```
package net.guides.springboot2.springboot2jpacrudexample.controller;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.validation.Valid;

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

import
net.guides.springboot2.springboot2jpacrudexample.exception.ResourceNotFoundException;
import net.guides.springboot2.springboot2jpacrudexample.model.Account;
import net.guides.springboot2.springboot2jpacrudexample.repository.AccountRepository;

@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping("/api/v1")

public class AccountController {
    @Autowired
    private AccountRepository accountRepository;

    @GetMapping("/accounts")
    public List<Account> getAllAccounts() {
        return accountRepository.findAll();
    }
```

```
  @GetMapping("/accounts/{id}")
  public ResponseEntity<Account> getAccountById(@PathVariable(value = "id") Long
accountId)
      throws ResourceNotFoundException {
      Account account = accountRepository.findById(accountId)
        .orElseThrow(() -> new ResourceNotFoundException("Account not found for this id :: "
+ accountId));
      return ResponseEntity.ok().body(account);
  }

  @PostMapping("/accounts")
  public Account createAccount(@Valid @RequestBody Account account) {
      return accountRepository.save(account);
  }

  @PutMapping("/accounts/{id}")
  public ResponseEntity<Account> updateAccount(@PathVariable(value = "id") Long
accountId,
       @Valid @RequestBody Account accountDetails) throws ResourceNotFoundException {
      Account account = accountRepository.findById(accountId)
      .orElseThrow(() -> new ResourceNotFoundException("Account not found for this id :: " +
accountId));

      account.setCin(accountDetails.getCin());
      account.setAccountNumber(accountDetails.getAccountNumber());
      account.setAgency(accountDetails.getAgency());
      account.setAmount(accountDetails.getAmount());
      account.setLastName(accountDetails.getLastName());
      account.setFirstName(accountDetails.getFirstName());
      final Account updatedAccount = accountRepository.save(account);
      return ResponseEntity.ok(updatedAccount);
  }

  @DeleteMapping("/accounts/{id}")
```

```
    public Map<String, Boolean> deleteAccount(@PathVariable(value = "id") Long accountId)
        throws ResourceNotFoundException {
        Account accounts = accountRepository.findById(accountId)
       .orElseThrow(() -> new ResourceNotFoundException("Account not found for this id :: " +
accountId));

        accountRepository.delete(accounts);
        Map<String, Boolean> response = new HashMap<>();
        response.put("deleted", Boolean.TRUE);
        return response;
    }
}
```

N'oubliez pas d'activer le partage de ressources inter-origines (Cross) sur le serveur en ajoutant l'annotation "@CrossOrigin" au "AccountController". Cette annotation permet de favoriser une communication distante entre le client et le serveur, c'est-à-dire lorsque le client et le serveur sont déployés dans deux serveurs distincts, ce qui permet d'éviter des problèmes réseau.

```
@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping("/api/v1")
```

Exécution de l'application:

Cette application Spring Boot possède une classe Java appelée Application.java qui contient la méthode "public static void main (String [] args)", que vous devez exécuter pour démarrer l'application.

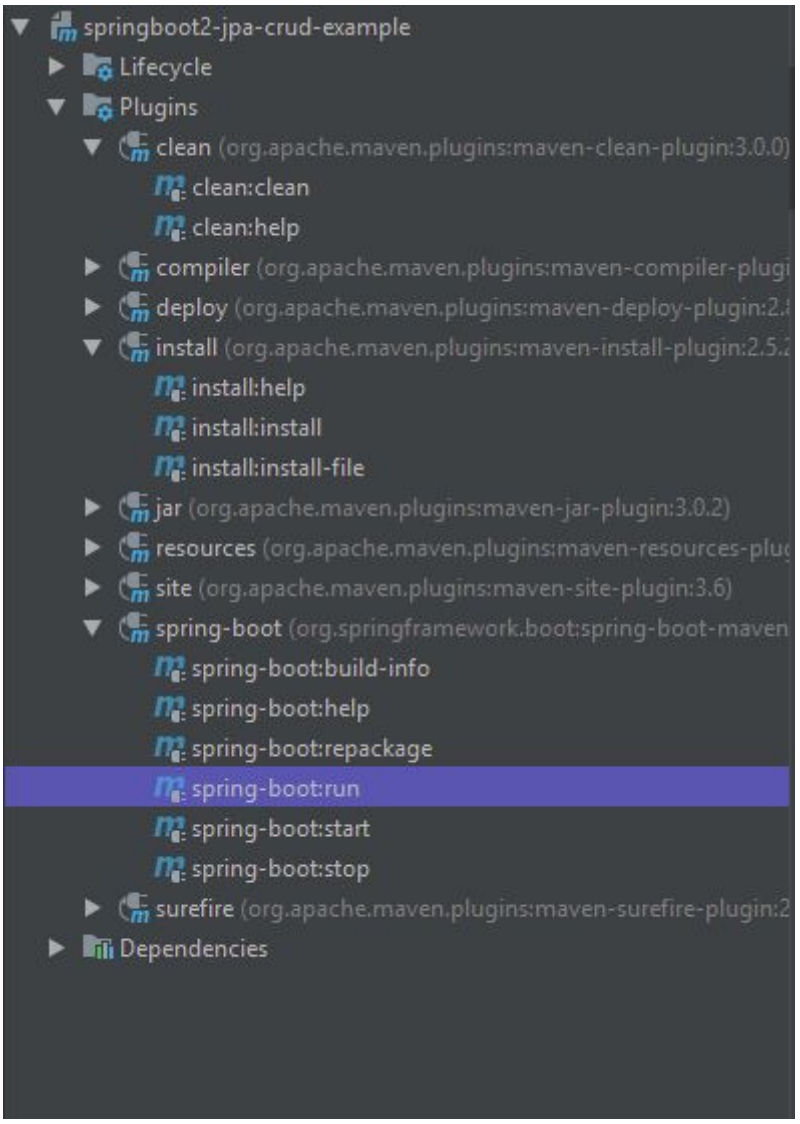
La méthode "main ()" utilise la méthode "SpringApplication.run ()" de Spring Boot pour lancer une application, ou bien, vous pouvez démarrer l'application via la ligne de commande à l'aide de la commande:

- mvnw spring-boot:run

NB: Avant d'exécuter l'application, lancez toujours la commandes suivante pour vous assurer de la compilation correcte de l'application:

- mvnw clean install

Et surtout n'oubliez pas de se connecter à la base de données avant de lancer l'application.



<u>Développement de l'application client Angular 8:</u>

Maintenant, vous allez développer l'application Web CRUD étape par étape en utilisant Angular 8 qui consomme l'api REST CRUD, que vous avez déjà créé. Tout d'abord, vous allez vérifier les versions de Node.js et NPM. Ouvrez le terminal ou la cmd, puis tapez les commandes suivantes:

- C:\Angular>node -v
- C:\Angular>npm -v

<u>Installation de la dernière version d'Angular CLI:</u>

Pour installer ou mettre à jour Angular 8 CLI, tapez cette commande dans le terminal ou la cmd:

- npm install -g @angular/cli

Maintenant, vérifiez que vous avez la dernière version de Angular 8 CLI:

- C:\angular>ng --version

Création de l'application client Angular 8 à l'aide de Angular CLI:

Angular CLI est un outil d'interface de ligne de commande que vous utilisez pour initialiser, développer, échafauder et maintenir des applications angulaires. Utilisez la commande ci-dessous pour générer une application client:

- ng new angular8-springboot-client

Identification des composants, des services et des modules Angular:

Tout d'abord, vous devez identifier les composants, les services et les modules que nous allez créer dans cette application. Utilisez Angular CLI pour générer les composants et les services, car elle suit les meilleures pratiques et fait gagner beaucoup de temps.

Les composants sont:

- create-account
- account-list
- account-details

Les services sont:

- account.service.ts - Service for Http Client methods

Les modules sont:

- FormsModule
- HttpClientModule
- AppRoutingModule

Identification de la classe Account. C'est une classe Typescript:

- account.ts: class Account (id, firstName, lastName, cin, accountNumber, agency, amount)

Création du service et des composants à l'aide de Angular CLI:

Générez automatiquement le service et les composants à l'aide de Angular CLI. Changez votre répertoire sous le projet Angular que vous avez créé et exécutez les commandes suivantes:

- ng g s account
- ng g c create-account

- ng g c account-details
- ng g c account-list

Intégration de JQuery et Bootstrap avec Angular:

Utilisez NPM pour télécharger Bootstrap et JQuery. Ils seront installés dans le dossier “node_modules”:

- npm install bootstrap jquery --save

Configurez Bootstrap et JQuery installés dans le fichier “angular.json” comme suit:

```
...

"styles": [
  "src/styles.css",
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
  "node_modules/jquery/dist/jquery.min.js",
  "node_modules/bootstrap/dist/js/bootstrap.min.js"
]

...
```

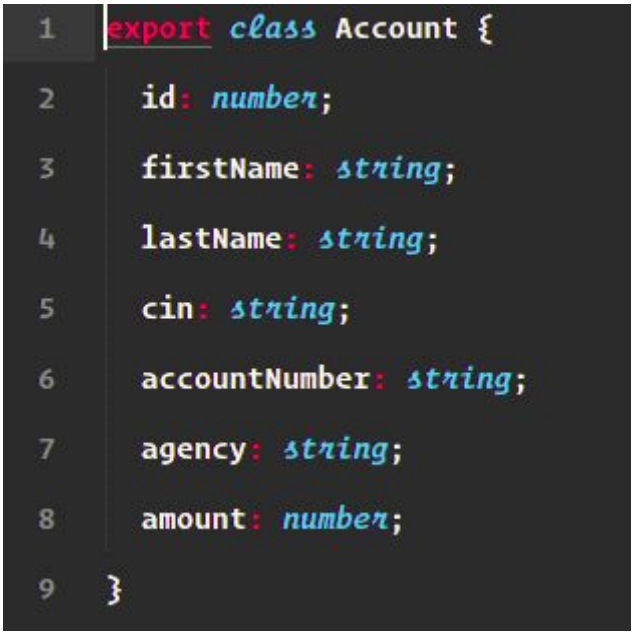
Si Bootstrap ne fonctionne pas, essayez d'importer le CSS bootstrap dans le fichier “style.css” comme suit:

```
@import '~bootstrap/dist/css/bootstrap.min.css';

.footer {
    position: absolute;
    bottom: 0;
    width:100%;
    height: 70px;
    background-color: blue;
    text-align: center;
    color: white;
}
```

Configuration du modèle Account:

Définissez une classe Account pour déclarer les comptes bancaires. Créez un nouveau fichier “account.ts” dans le dossier “src/app” et y ajoutez le code suivant:



```
export class Account {
  id: number;
  firstName: string;
  lastName: string;
  cin: string;
  accountNumber: string;
  agency: string;
  amount: number;
}
```

Configuration du composant liste des comptes bancaires:

Trouvez le fichier sous le chemin suivant:

- src/app/account-list/account-list.component.ts

Pour créer le composant “AccountListComponent”, qui sera utilisé pour afficher une liste des comptes bancaires, créer un nouveau compte et supprimer un compte, suivez le code suivant:

```
import { Component, OnInit } from '@angular/core';
import { AccountDetailsComponent } from '../account-details/account-details.component';
import { Observable } from "rxjs";
import { AccountService } from "../account.service";
import { Account } from "../account";
import { Router } from '@angular/router';
@Component({
  selector: "app-account-list",
  templateUrl: "./account-list.component.html",
  styleUrls: ["./account-list.component.css"]
})
```

```
export class accountListComponent implements OnInit {
  accounts: Observable<Account[]>;

  constructor(private accountService: AccountService,
    private router: Router) {}

  ngOnInit() {
    this.reloadData();
  }

  reloadData() {
    this.accounts = this.accountService.getAccountsList();
  }

  deleteAccount(id: number) {
    this.accountService.deleteAccount(id)
      .subscribe(
        data => {
          console.log(data);
          this.reloadData();
        },
        error => console.log(error));
  }

  accountDetails(id: number){
    this.router.navigate(['details', id]);
  }
}
```

Configuration du modèle liste des comptes bancaires:

Trouvez le fichier sous le chemin suivant:

- src/app/account-list/account-list.component.html

Ajoutez au fichier “account-list.component.html” le code suivant:

```
<p>account-list works!</p>
```

```
<div class="panel panel-primary">
   <div class="panel-heading">
    <h2>Account List</h2>
   </div>
   <div class="panel-body">
    <table class="table table-striped">
     <thead>
      <tr>
       <th>Id</th>
       <th>Firstname</th>
       <th>Lastname</th>
       <th>Cin</th>
       <th>Accountnumber</th>
       <th>Agency</th>
       <th>Amount</th>
      </tr>
     </thead>
     <tbody>
      <tr *ngFor="let account of accounts | async">
       <td>{{account.id}}</td>
       <td>{{account.firstName}}</td>
       <td>{{account.lastName}}</td>
       <td>{{account.cin}}</td>
       <td>{{account.accountNumber}}</td>
       <td>{{account.agency}}</td>
       <td>{{account.amount}}</td>
       <td><button (click)="deleteAccount(account.id)" class="btn
btn-danger">Delete</button>
          <button (click)="accountDetails(account.id)" class="btn btn-info"
style="margin-left: 10px">Details</button>
       </td>
      </tr>
     </tbody>
    </table>
```

```
  </div>
 </div>
```

Configuration du composant compte bancaire:

Trouvez le fichier sous le chemin suivant:

- src/app/create-account/create-account.component.ts

Modifiez le fichier “create-account.component.ts” avec le code suivant:

```
import { AccountService } from '../account.service';
import { Account } from '../account';
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';

@Component({
  selector: 'app-create-account',
  templateUrl: './create-account.component.html',
  styleUrls: ['./create-account.component.css']
})
export class CreateAccountComponent implements OnInit {

  account: Account = new Account();
  submitted = false;

  constructor(private accountService: AccountService,
    private router: Router) { }

  ngOnInit() {
  }

  newAccount(): void {
    this.submitted = false;
    this.account = new Account();
  }

  save() {
```

```
    console.log(this.account)
    this.accountService.createAccount(this.account)
      .subscribe(data => console.log(data), error => console.log(error));
    this.account = new Account();
    this.gotoList();
  }

  onSubmit() {
    this.submitted = true;
    this.save();
  }

  gotoList() {
    this.router.navigate(['/accounts']);
  }
}
```

Création du modèle compte bancaire:

Trouvez le fichier sous le chemin suivant:

- src/app/create-account/create-account.component.html

Modifiez le fichier “create-account.component.html” avec le code suivant:

```
<p>create-account works!</p>
<h3>Create Account</h3>
<div [hidden]="submitted" style="width: 400px;">
  <form (ngSubmit)="onSubmit()">
    <div class="form-group">
      <label for="name">Id</label>
      <input type="text" class="form-control" id="id" required [(ngModel)]="account.id"
name="id">
    </div>

    <div class="form-group">
      <label for="name">First Name</label>
```

```
    <input type="text" class="form-control" id="firstName" required
[(ngModel)]="account.firstName" name="firstName">
  </div>

  <div class="form-group">
    <label for="name">Last Name</label>
    <input type="text" class="form-control" id="lastName" required
[(ngModel)]="account.lastName" name="lastName">
  </div>

  <div class="form-group">
    <label for="name">Cin</label>
    <input type="text" class="form-control" id="cin" required [(ngModel)]="account.cin"
name="id">
  </div>

  <div class="form-group">
    <label for="name">Account Number</label>
    <input type="text" class="form-control" id="accountNumber" required
[(ngModel)]="account.accountNumber" name="id">
  </div>

  <div class="form-group">
    <label for="name">Agency</label>
    <input type="text" class="form-control" id="agency" required
[(ngModel)]="account.agency" name="id">
  </div>

  <div class="form-group">
    <label for="name">Amount</label>
    <input type="text" class="form-control" id="amount" required
[(ngModel)]="account.amount" name="id">
  </div>

  <button type="submit" class="btn btn-success">Submit</button>
```

```
  </form>
</div>

<div [hidden]="!submitted">
  <h4>You submitted successfully!</h4>
  <!-- <button class="btn btn-success" (click)="newAccount()">Add</button> -->
</div>
```

Mise à jour du composant compte bancaire:

Trouvez le fichier sous le chemin suivant:

- src/app/update-account/update-account.component.ts

"UpdateAccountComponent" est utilisé pour mettre à jour un compte existant. Vous obtenez d'abord l'objet compte à l'aide de l'API REST et remplissez au format HTML via la liaison de données. Les utilisateurs peuvent modifier les données du formulaire compte et soumettre le formulaire.

Modifier le fichier "update-account.component.ts" avec le code suivant:

```
import { Component, OnInit } from '@angular/core';
import { Account } from '../account';
import { ActivatedRoute, Router } from '@angular/router';
import { AccountService } from '../account.service';

@Component({
  selector: 'app-update-account',
  templateUrl: './update-account.component.html',
  styleUrls: ['./update-account.component.css']
})
export class UpdateAccountComponent implements OnInit {

  id: number;
  account: Account;

  constructor(private route: ActivatedRoute,private router: Router,
    private accountService: AccountService) { }
```

```
 ngOnInit() {
  this.account = new Account();

  this.id = this.route.snapshot.params['id'];

  this.accountService.getAccount(this.id)
    .subscribe(data => {
     console.log(data)
     this.account = data;
    }, error => console.log(error));
 }

 updateAccount() {
  this.accountService.updateAccount(this.id, this.account)
    .subscribe(data => console.log(data), error => console.log(error));
  this.account = new Account();
  this.gotoList();
 }

 onSubmit() {
  this.updateAccount();
 }

 gotoList() {
  this.router.navigate(['/accounts']);
 }
}
```

Mise à jour du modèle compte bancaire:

Trouvez le fichier sous le chemin suivant:

- src/app/update-account/update-account.component.html

Modifiez le fichier “update-account.component.html” avec le code suivant:

```
<p>update-account works!</p>
<h3>Update Account</h3>
```

```
<div [hidden]="submitted" style="width: 400px;">
<!--   <form (ngSubmit)="onSubmit()">
    <div class="form-group">

      <label for="name">First Name</label>
      <input type="text" class="form-control" id="firstName"
required [(ngModel)]="account.firstName" name="firstName">
    </div>

    <div class="form-group">
      <label for="name">Last Name</label>
      <input type="text" class="form-control" id="lastName"
required [(ngModel)]="account.lastName" name="lastName">
    </div>

    <div class="form-group">
      <label for="name">Cin</label>
      <input type="text" class="form-control" id="cin"
required [(ngModel)]="account.cin" name="cin">
    </div>

    <div class="form-group">
      <label for="name">Account Number</label>
      <input type="text" class="form-control"
id="accountNumber" required
[(ngModel)]="account.accountNumber" name="accountNumber">
    </div>

    <div class="form-group">
      <label for="name">Agency/label>
      <input type="text" class="form-control" id="agency"
required [(ngModel)]="account.agency" name="agency">
    </div>

    <div class="form-group">
```

```
      <label for="name">Amount</label>
      <input type="text" class="form-control" id="amount"
required [(ngModel)]="account.amount" name="amount">
    </div>

    <button type="submit" class="btn
btn-success">Submit</button>
  </form> -->
</div>
```

Création du modèle et du composant View des détails des comptes bancaires:

Ici, vous allez créer la fonctionnalité view des détails des comptes. Créez un modèle HTML et un composant de la fonctionnalité view des détails des comptes.

Composant View des détails d'un compte bancaire:

Trouvez le fichier sous le chemin suivant:

- src/app/account-details/account-details.component.ts

Le composant "AccountDetailsComponent" est utilisé pour afficher les détails d'un compte particulier.

Modifiez le fichier "account-details.component.ts" avec le code suivant:

```
import { Account } from '../account';
import { Component, OnInit, Input } from '@angular/core';
import { AccountService } from '../account.service';
import { accountListComponent } from '../account-list/account-list.component';
import { Router, ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-acccount-details',
  templateUrl: './account-details.component.html',
  styleUrls: ['./account-details.component.css']
})
export class AccountDetailsComponent implements OnInit {
```

```
  id: number;
  account: Account;

  constructor(private route: ActivatedRoute,private router: Router,
    private accountService: AccountService) { }

  ngOnInit() {
    this.account = new Account();

    this.id = this.route.snapshot.params['id'];

    this.accountService.getAccount(this.id)
      .subscribe(data => {
        console.log(data)
        this.account = data;
      }, error => console.log(error));
  }

  list(){
    this.router.navigate(['accounts']);
  }
}
```

Modèle du composant View des détails d'un compte bancaire:

Trouvez le fichier sous le chemin suivant:

- src/app/account-details/account-details.component.html

Le fichier "account-details.component.html" affiche un détail d'un compte particulier.

Modifiez le fichier "account-details.component.html" avec le code suivant:

```
<p>account-details works!</p>
<h2>Account Details</h2>

<hr/>
<div *ngIf="account">
```

```
 <div>
  <label><b>Id: </b></label> {{account.id}}
 </div>
 <div>
  <label><b>First Name: </b></label> {{account.firstName}}
 </div>
 <div>
  <label><b>Last Name: </b></label> {{account.lastName}}
 </div>
 <div>
  <label><b>Cin: </b></label> {{account.cin}}
 </div>
 <div>
  <label><b>Number Account: </b></label> {{account.numberAccount}}
 </div>
 <div>
  <label><b>Agency: </b></label> {{account.agency}}
 </div>
 <div>
  <label><b>Amount: </b></label> {{account.amount}}
 </div>
</div>
<br>
<br>
<button (click)="list()" class="btn btn-primary">Back to Account List</button><br>
```

Configuration du service du compte bancaire:

Créez le fichier sous le chemin suivant:

- src/app/account.service.ts

“AccountService” sera utilisé pour obtenir les données du backend en appelant les APIs Spring Boot. Mettez à jour le fichier “account.service.ts” dans le répertoire “src/app” avec le code suivant:

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

```
import { Observable } from 'rxjs';

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

  private baseUrl = 'http://localhost:8080/api/v1/accounts';

  constructor(private http: HttpClient) { }

  getAccount(id: number): Observable<any> {
    return this.http.get(`${this.baseUrl}/${id}`);
  }

  createAccount(account: Object): Observable<Object> {
    return this.http.post(`${this.baseUrl}`, account);
  }

  updateAccount(id: number, value: any): Observable<Object> {
    return this.http.put(`${this.baseUrl}/${id}`, value);
  }

  deleteAccount(id: number): Observable<any> {
    return this.http.delete(`${this.baseUrl}/${id}`, { responseType: 'text' });
  }

  getAccountsList(): Observable<any> {
    return this.http.get(`${this.baseUrl}`);
  }
}
```

Configuration de l'application:

Vous avez terminé le développement des opérations CRUD du compte bancaire à l'aide d'Angular 8. Dans cette partie, vous allez poursuivre la configuration

d'application Web Angular 8, la création du composant App, des modules App, etc.

Configuration des dépendances:

Le fichier “package.json” contient des informations de configuration de projet, y compris les dépendances de package qui seront installées lorsque vous exécutez l'installation de npm. L’exécution de la commande “npm install” va apporter toutes les dépendances du projet.

Configuration du module de routage d'application:

Le routage pour l'application Angular est configuré comme un tableau de routes, chaque composant est mappé sur un chemin afin que le routeur d’Angular sache quel composant à afficher en fonction de l'URL dans la barre d'adresse du navigateur.

Trouvez le fichier sous le chemin suivant:

- /src/app/app-routing.module.ts

Modifiez le fichier “app-routing.module.ts” avec le code suivant:

```
import { AccountDetailsComponent } from './account-details/account-details.component';
import { CreateAccountComponent } from './create-account/create-account.component';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { accountListComponent } from './account-list/account-list.component';
import { UpdateAccountComponent } from './update-account/update-account.component';

const routes: Routes = [
  { path: '', redirectTo: 'account', pathMatch: 'full' },
  { path: 'accounts', component: accountListComponent },
  { path: 'add', component: CreateAccountComponent },
  { path: 'update/:id', component: UpdateAccountComponent },
  { path: 'details/:id', component: AccountDetailsComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
```

```
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Configuration du composant d'application:

Le composant d'application est le composant racine de l'application, il définit la balise racine de l'application avec la propriété selector du décorateur @Component.

Trouvez le fichier sous le chemin suivant:

- /src/app/app.component.ts

Modifiez le fichier “app.component.ts” avec le code suivant:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Accounts Manager';
}
```

Configuration du modèle de composant d'application:

Le modèle de composant d’application définit le modèle HTML associé au composant “AppComponent” racine.

Trouvez le fichier sous le chemin suivant:

- /src/app/app.component.html

Modifiez le fichier “app.component.html” avec le code suivant:

```
<nav class="navbar navbar-expand-sm bg-primary navbar-dark">
  <!-- Links -->
  <ul class="navbar-nav">
    <li class="nav-item">
      <a routerLink="accounts" class="nav-link" routerLinkActive="active">Account List</a>
```

```
    </li>
    <li class="nav-item">
      <a routerLink="add" class="nav-link" routerLinkActive="active">Add Account</a>
    </li>
  </ul>
</nav>
<div class="container">
  <br>
  <h2 style="text-align: center;">{{title}}</h2>
  <hr>
  <div class="card">
    <div class="card-body">
  <router-outlet></router-outlet>
    </div>
  </div>
</div>

<footer class="footer">
</footer>
```

Configuration du module d'application:

Le module d'application définit le module racine, nommé "AppModule", qui indique à Angular comment assembler l'application. Initialement, il déclare uniquement "AppComponent". Et lorsque vous ajoutez d'autres composants à l'application, ils doivent être déclarés ici.

Trouvez le fichier sous le chemin suivant:

- /src/app/app.module.ts

Modifiez le fichier "app.module.ts" avec le code suivant:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CreateAccountComponent } from './create-account/create-account.component';
```

```
import { AccountDetailsComponent } from './account-details/account-details.component';
import { accountListComponent } from './account-list/account-list.component';
import { HttpClientModule } from '@angular/common/http';
import { UpdateAccountComponent } from './update-account/update-account.component';
@NgModule({
  declarations: [
    AppComponent,
    CreateAccountComponent,
    AccountDetailsComponent,
    accountListComponent,
    UpdateAccountComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Configuration du module d'application:

Le fichier “index.html” est la page initiale chargée par le navigateur qui lance tout. Webpack regroupe tous les fichiers TypeScript et les injecte dans le corps de la page “index.html” afin que les scripts soient chargés et exécutés par le navigateur.

Trouvez le fichier sous le chemin suivant:

- /src/index.html

Modifiez le fichier “index.html” avec le code suivant:

```
<!doctype html>
<html lang="en">
<head>
```

```
  <meta charset="utf-8">
  <title>Angular8SpringbootClient</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Configuration du fichier principal (Bootstrapper l'application):

Le fichier principal est le point d'entrée utilisé par Angular pour lancer et amorcer l'application.

Trouvez le fichier sous le chemin suivant:

- /src/main.ts

Modifiez le fichier "main.ts" avec le code suivant:

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Configuration du Polyfills:

Certaines fonctionnalités utilisées par Angular 8 ne sont pas encore prises en charge nativement par tous les principaux navigateurs, les polyfills sont utilisés

pour ajouter la prise en charge des fonctionnalités si nécessaire afin que votre application Angular 8 fonctionne sur tous les principaux navigateurs.
Trouvez le fichier sous le chemin suivant:

- /src/polyfills.ts

Modifiez le fichier “polyfills.ts” avec le code suivant:

```
import 'core-js/features/reflect';
import 'zone.js/dist/zone';
```

Configuration du fichier “tsconfig.json”:

Le fichier “tsconfig.json” configure la façon dont le compilateur TypeScript convertira TypeScript en JavaScript compris par le navigateur.
Chemin du fichier:

- /tsconfig.json

Modifiez le fichier “tsconfig.json” avec le code suivant:

```
{
 "compileOnSave": false,
 "compilerOptions": {
  "baseUrl": "./",
  "outDir": "./dist/out-tsc",
  "sourceMap": true,
  "declaration": false,
  "downlevelIteration": true,
  "experimentalDecorators": true,
  "module": "esnext",
  "moduleResolution": "node",
  "importHelpers": true,
  "target": "es2015",
  "typeRoots": [
   "node_modules/@types"
  ],
  "lib": [
   "es2018",
   "dom"
  ]
 },
```

```
  "angularCompilerOptions": {
    "fullTemplateTypeCheck": true,
    "strictInjectionParameters": true
  }
}
```

Exécution de l'application client Angular 8:

Pour exécuter l'application Angular 8 que vous avez créé, lancez la commande suivante:

- ng serve

Par défaut, l'application Angular s'exécute sur le port 4200 mais vous pouvez modifier le port par défaut avec la commande suivante:

- ng serve --port 42100 (par exemple)

Pour utilisez l'application, écrivez ce lien "http://localhost:4200" dans le navigateur et commencez à tester les fonctionnalitées de votre app.

Quelques ScreenShots:

Accounts Manager

Accounts Add

Accounts

Firstname	Lastname	CIN	Account Number	Agency	Amount	
Client1	Client1	11500400	555555555	Agency1	500000	Delete
Client2	Client2	11900235	222222222	Agency2	6000000	Delete
Client3	Client3	11200600	888888888	Agency3	300000	Delete