Projet base des données NOSQL : gestion des employées avec MongoDB

Normes :

- Back-end :Springboot
- Front-end :Angular
- BD :MongoDB

Objectifs :

- Créer une application de gestion des données des employées avec une base de données NOSQ orientée documents

Spécifications/Tâche(s) :

-création employées
-création liste employées
-ajout employées
-suppression employées
-mis à jour des employées
- affiche détails

Conception :

database_sequences :

Nous allons créer une collection qui stockera la séquence auto-incrémentée pour les autres collections. Nous appellerons cette collection database_sequences. Il peut être créé à l'aide du shell mongo ou de MongoDB Compass. Créons une classe de modèle correspondante :

```java
package com.example.tpnosql.model;

import ...


@Document(collection = "database_sequences")
public class DatabaseSequence {

    @Id
    private String id;

    private long seq;

    public DatabaseSequence() {}

    public String getId() { return id; }

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

    public long getSeq() { return seq; }

    public void setSeq(long seq) { this.seq = seq; }
}
```

Employee

Créons maintenant le modèle Employee qui sera mappé à un document dans la base de données MongoDB. Créez un nouveau modèle :

```java
@Document (collection = "Employee")
public class Employee {

    @Transient
    public static final String SEQUENCE_NAME = "users_sequence";

    @Id
    private long id;

    @{...}
    private String firstName;
    private String lastName;

    @{...}
    private String emailId;

    public Employee() {}

    public Employee(String firstName, String lastName, String emailId) {...}

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

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

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

    public String getEmailId() { return emailId; }
```

Configuration :

Pom.xml :

```
<version>0.0.1-SNAPSHOT</version>
<name>tpnosql</name>
<description>tpnosql</description>
<properties>
    <java.version>11</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>7.0</version>
```

Créer un référentiel de données Spring - EmployeeRepository.java

Ensuite, nous devons créer EmployeeRepository pour accéder aux données de la base de données.

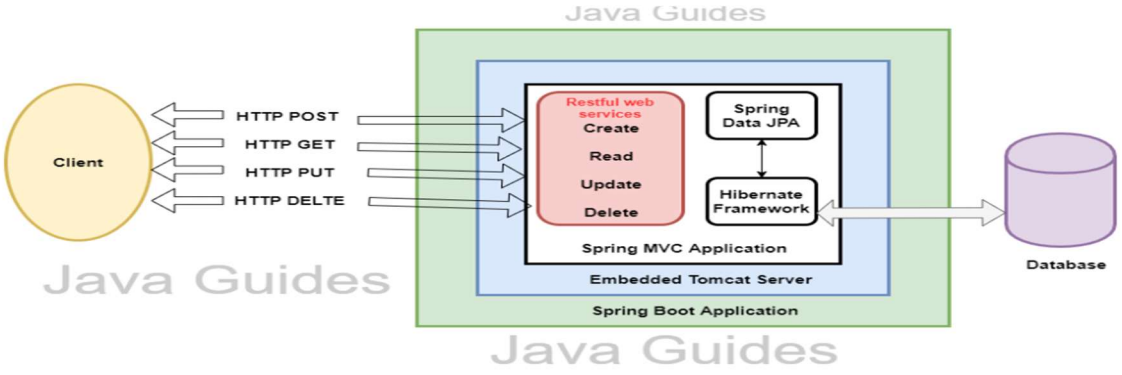
```
package com.example.tpnosql.repository;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

import  com.example.tpnosql.model.Employee;

@Repository
public interface EmployeeRepository extends MongoRepository<Employee, Long>
{

}
```

SequenceGeneratorService - Auto-Generate Sequence

```
import static org.springframework.data.mongodb.core.FindAndModifyOptions.options;
import static org.springframework.data.mongodb.core.query.Criteria.where;
import static org.springframework.data.mongodb.core.query.Query.query;

import java.util.Objects;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.stereotype.Service;

import net.guides.springboot.crud.model.DatabaseSequence;

@Service
public class SequenceGeneratorService {

    private MongoOperations mongoOperations;

    @Autowired
    public SequenceGeneratorService(MongoOperations mongoOperations) {
        this.mongoOperations = mongoOperations;
    }

    public long generateSequence(String seqName) {

        DatabaseSequence counter =
mongoOperations.findAndModify(query(where("_id").is(seqName)),
                new Update().inc("seq",1), options().returnNew(true).upsert(true),
                DatabaseSequence.class);
        return !Objects.isNull(counter) ? counter.getSeq() : 1;

    }
}
```

Création des API - EmployeeController

Maintenant, créons les API qui seront exposées aux clients. Notez l'utilisation des annotations @RestController, @RequestMapping, @GetMapping, @PostMapping et @DeleteMapping pour mapper divers URI aux méthodes du contrôleur. Notez que nous utilisons generateSequence() lors de la création d'un nouvel enregistrement :



```
package com.example.tpnosql.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 com.example.tpnosql.exception.ResourceNotFoundException;
import com.example.tpnosql.model.Employee;
import com.example.tpnosql.repository.EmployeeRepository;
import com.example.tpnosql.service.SequenceGeneratorService;

@CrossOrigin(origins = "http://localhost:4200")
@RestController
@RequestMapping("/api/v1")
public class EmployeeController {
    @Autowired
    private EmployeeRepository employeeRepository;

    @Autowired
    private SequenceGeneratorService sequenceGeneratorService;

    @GetMapping("/employees")
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    @GetMapping("/employees/{id}")
    public ResponseEntity<Employee> getEmployeeById(@PathVariable(value =
"id") Long employeeId)
            throws ResourceNotFoundException {
        Employee employee = employeeRepository.findById(employeeId)
                .orElseThrow(() -> new ResourceNotFoundException("Employee
not found for this id :: " + employeeId));
        return ResponseEntity.ok().body(employee);
    }

    @PostMapping("/employees")
    public Employee createEmployee(@Valid @RequestBody Employee employee) {

employee.setId(sequenceGeneratorService.generateSequence(Employee.SEQUENCE_N
AME));
        return employeeRepository.save(employee);
    }

    @PutMapping("/employees/{id}")
    public ResponseEntity<Employee> updateEmployee(@PathVariable(value =
"id") Long employeeId,
                                                   @Valid @RequestBody
Employee employeeDetails) throws ResourceNotFoundException {
        Employee employee = employeeRepository.findById(employeeId)
                .orElseThrow(() -> new ResourceNotFoundException("Employee
```

```
not found for this id :: " + employeeId));

        employee.setEmailId(employeeDetails.getEmailId());
        employee.setLastName(employeeDetails.getLastName());
        employee.setFirstName(employeeDetails.getFirstName());
        final Employee updatedEmployee = employeeRepository.save(employee);
        return ResponseEntity.ok(updatedEmployee);
    }

    @DeleteMapping("/employees/{id}")
    public Map<String, Boolean> deleteEmployee(@PathVariable(value = "id")
Long employeeId)
            throws ResourceNotFoundException {
        Employee employee = employeeRepository.findById(employeeId)
                .orElseThrow(() -> new ResourceNotFoundException("Employee
not found for this id :: " + employeeId));

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

front-end :

nous allons créer une simple application de gestion des employés (application FULL STACK avec Angular 10 et Spring Boot) avec les fonctionnalités suivantes :

Créer un employé

Mettre à jour un employé

Liste des employés

Supprimer l'employé

Voir l'employé

Realisation :

Dressons la liste des composants, services et modules que nous allons créer dans cette application. Nous utiliserons Angular CLI pour générer des composants, des services car Angular CLI suit les meilleures pratiques et fait gagner beaucoup de temps.

Composants

créer-employé

liste des employés

Détails de l'employé

mise à jour-employé

Prestations de service

employee.service.ts - Service pour les méthodes du client HTTP

Modules

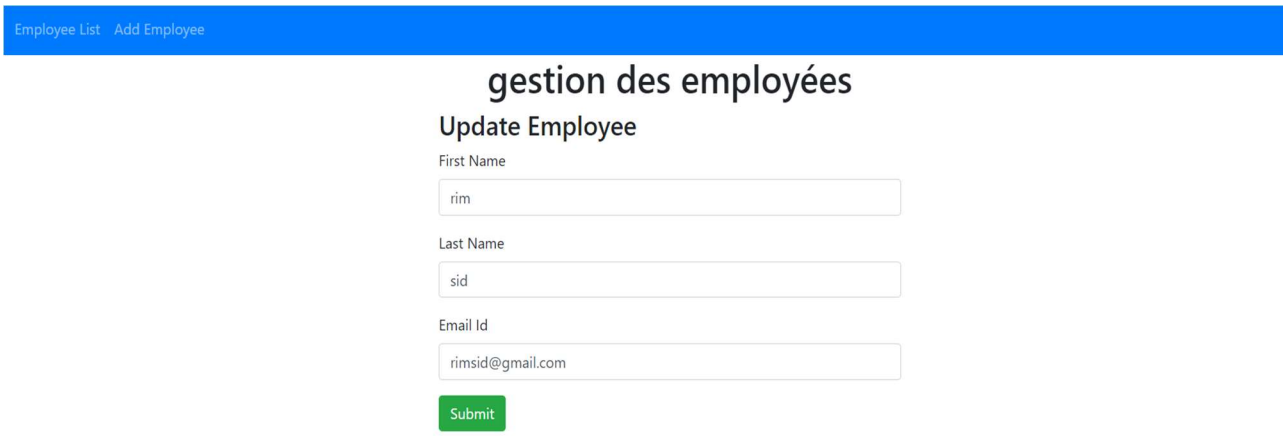
ModuleFormulaires

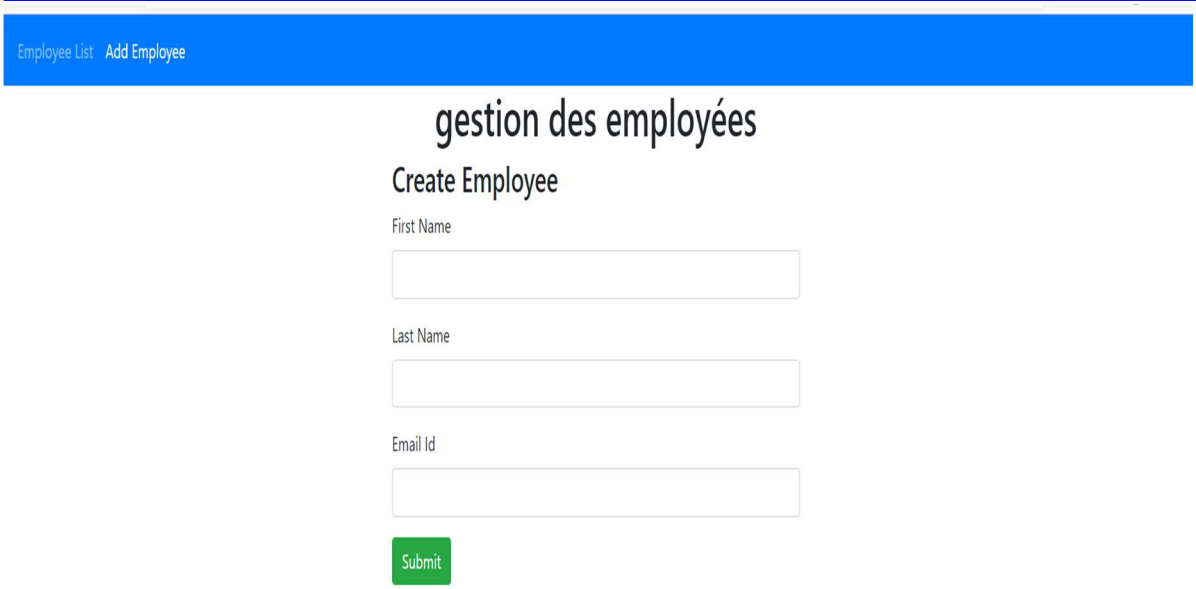
HttpClientModuleHttpClientModule

AppRoutingModule

Classe d'employés (classe dactylographiée)

employee.ts : classe Employee (id, firstName, lastName, emailId)


Employee List
Add Employee
gestion des employées
Update Employee
First Name
rim
Last Name
sid
Email Id
rimsid@gmail.com
Submit


Employee List
Add Employee
gestion des employées
Create Employee
First Name
Last Name
Email Id
Submit

Employee List Add Employee

gestion des employées

View Employee Details

First Name: amin

Last Name: ahmed

Email Id: ahmedamin@gmail.com


localhost:4200/employees
Employee List
Add Employee
gestion des employées
Employee List
First Name
Last Name
Email Id
Actions
amin
ahmed
ahmedamin@gmail.com
Update
Delete
View
rim
sid
rimsid@gmail.com
Update
Delete
View