Système de Gestion des Achats et des Fournisseurs - Guide Complet

Ce guide détaillé vous fournira toutes les étapes pour développer un système complet de gestion des achats et fournisseurs utilisant Angular pour le frontend, Spring Boot pour le backend et MySQL comme base de données.

Architecture du Projet

Le projet sera divisé en deux parties principales:

1. Backend: Spring Boot application (développée avec STS)
2. Frontend: Angular application

Partie 1: Backend Spring Boot

Étape 1: Configuration initiale

1. **Créer un nouveau projet Spring Boot dans STS**
   - Utilisez Spring Initializr (https://start.spring.io/)
   - Sélectionnez:
     - Project: Maven Project
     - Language: Java
     - Spring Boot: Version stable (ex: 3.1.0)
   - Dependencies:
     - Spring Web
     - Spring Data JPA
     - MySQL Driver
     - Lombok (pour réduire le code boilerplate)
2. **Structure des packages**

Copy

Download

```
com.gestionachat
├── config
```

```
├── controller
├── dto
├── entity
├── exception
├── repository
├── service
└── GestionAchatApplication.java
```

Étape 2: Configuration de la base de données

1. **Ajouter la configuration MySQL dans** `application.properties`

```properties
spring.datasource.url=jdbc:mysql://localhost:3306/gestion_achat?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.properties.hibernate.format_sql=true

server.port=8080
```

Étape 3: Entités JPA

1. **Fournisseur.java**

```java
package com.gestionachat.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.ArrayList;
import java.util.List;

@Entity
```

```java
@Data @AllArgsConstructor @NoArgsConstructor
public class Fournisseur {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String contact;
    private String qualiteService;
    private double note;

    @OneToMany(mappedBy = "fournisseur", fetch = FetchType.LAZY)
    private List<CommandeAchat> commandes = new ArrayList<>();

    @OneToMany(mappedBy = "fournisseur", fetch = FetchType.LAZY)
    private List<HistoriqueAchats> historiqueAchats = new ArrayList<>();
}
```

2. CommandeAchat.java

java

Copy

Download

```java
package com.gestionachat.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Date;
import java.util.List;

@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class CommandeAchat {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    private Fournisseur fournisseur;

    private Date date;
    private String statut;
    private double montant;

    @OneToMany(mappedBy = "commande", cascade = CascadeType.ALL, orphanRemo
val = true)
    private List<LigneCommandeAchat> lignesCommande;
```

```
}
```

3. LigneCommandeAchat.java

```
java
Copy
Download
package com.gestionachat.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class LigneCommandeAchat {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    private CommandeAchat commande;

    private String produit;
    private int quantite;
    private double prixUnitaire;
}
```

4. HistoriqueAchats.java

```
java
Copy
Download
package com.gestionachat.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class HistoriqueAchats {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    @ManyToOne
    private Fournisseur fournisseur;

    private String produit;
    private int quantite;
    private int delaiLivraison; // en jours
}
```

Étape 4: Repositories

1. FournisseurRepository.java

```
java
Copy
Download
package com.gestionachat.repository;

import com.gestionachat.entity.Fournisseur;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface FournisseurRepository extends JpaRepository<Fournisseur, L
ong> {
}
```

2. CommandeAchatRepository.java

```
java
Copy
Download
package com.gestionachat.repository;

import com.gestionachat.entity.CommandeAchat;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CommandeAchatRepository extends JpaRepository<CommandeAcha
t, Long> {
}
```

3. LigneCommandeAchatRepository.java

```
java
Copy
```

Download

```
package com.gestionachat.repository;

import com.gestionachat.entity.LigneCommandeAchat;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface LigneCommandeAchatRepository extends JpaRepository<LigneCo
mmandeAchat, Long> {
}
```

4. HistoriqueAchatsRepository.java

java

Copy

Download

```
package com.gestionachat.repository;

import com.gestionachat.entity.HistoriqueAchats;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface HistoriqueAchatsRepository extends JpaRepository<Historiqu
eAchats, Long> {
}
```

Étape 5: Services

1. FournisseurService.java

java

Copy

Download

```
package com.gestionachat.service;

import com.gestionachat.dto.FournisseurDTO;
import com.gestionachat.entity.Fournisseur;
import com.gestionachat.exception.FournisseurNotFoundException;
import java.util.List;

public interface FournisseurService {
    Fournisseur createFournisseur(FournisseurDTO fournisseurDTO);
    Fournisseur updateFournisseur(Long id, FournisseurDTO fournisseurDTO) t
hrows CournisseurNotFoundException;
```

```
    List<Fournisseur> getAllFournisseurs();
    Fournisseur getFournisseurById(Long id) throws CournisseurNotFoundExcep
tion;
    void deleteFournisseur(Long id) throws CournisseurNotFoundException;
    double evaluerFournisseur(Long id) throws CournisseurNotFoundException;
    List<Fournisseur> comparerFournisseursParProduit(String produit);
}
```

2. FournisseurServiceImpl.java

java

Copy

Download

```
package com.gestionachat.service;

import com.gestionachat.dto.FournisseurDTO;
import com.gestionachat.entity.Fournisseur;
import com.gestionachat.entity.HistoriqueAchats;
import com.gestionachat.exception.CournisseurNotFoundException;
import com.gestionachat.repository.FournisseurRepository;
import com.gestionachat.repository.HistoriqueAchatsRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
@AllArgsConstructor
public class FournisseurServiceImpl implements FournisseurService {

    private final FournisseurRepository fournisseurRepository;
    private final HistoriqueAchatsRepository historiqueAchatsRepository;

    @Override
    public Fournisseur createFournisseur(FournisseurDTO fournisseurDTO) {
        Fournisseur fournisseur = new Fournisseur();
        fournisseur.setNom(fournisseurDTO.getNom());
        fournisseur.setContact(fournisseurDTO.getContact());
        fournisseur.setQualiteService(fournisseurDTO.getQualiteService());
        fournisseur.setNote(0); // Initial note

        return fournisseurRepository.save(fournisseur);
    }

    @Override
```

```
    public Fournisseur updateFournisseur(Long id, FournisseurDTO fournisseu
rDTO) throws CournisseurNotFoundException {
        Optional<Fournisseur> optionalFournisseur = fournisseurRepository.f
indById(id);
        if (optionalFournisseur.isEmpty()) {
            throw new CournisseurNotFoundException("Fournisseur non trouvé
avec l'ID: " + id);
        }

        Fournisseur fournisseur = optionalFournisseur.get();
        fournisseur.setNom(fournisseurDTO.getNom());
        fournisseur.setContact(fournisseurDTO.getContact());
        fournisseur.setQualiteService(fournisseurDTO.getQualiteService());

        return fournisseurRepository.save(fournisseur);
    }

    @Override
    public List<Fournisseur> getAllFournisseurs() {
        return fournisseurRepository.findAll();
    }

    @Override
    public Fournisseur getFournisseurById(Long id) throws CournisseurNotFou
ndException {
        return fournisseurRepository.findById(id)
                .orElseThrow(() -> new CournisseurNotFoundException("Fourni
sseur non trouvé avec l'ID: " + id));
    }

    @Override
    public void deleteFournisseur(Long id) throws CournisseurNotFoundExcept
ion {
        if (!fournisseurRepository.existsById(id)) {
            throw new CournisseurNotFoundException("Fournisseur non trouvé
avec l'ID: " + id);
        }
        fournisseurRepository.deleteById(id);
    }

    @Override
    public double evaluerFournisseur(Long id) throws CournisseurNotFoundExc
eption {
        Fournisseur fournisseur = getFournisseurById(id);

        // Calcul de la note basée sur l'historique des achats
```

```
        List<HistoriqueAchats> historique = historiqueAchatsRepository.find
ByFournisseur(fournisseur);

        if (historique.isEmpty()) {
            return fournisseur.getNote();
        }

        double noteMoyenneLivraison = historique.stream()
                .mapToInt(HistoriqueAchats::getDelaiLivraison)
                .average()
                .orElse(0);

        // Plus le délai est court, meilleure est la note (sur 10)
        double noteLivraison = Math.max(0, 10 - (noteMoyenneLivraison / 5))
;

        // Note finale (50% qualité service, 50% performance livraison)
        double noteFinale = (noteLivraison * 0.5) + (fournisseur.getQualite
Service().equals("excellent") ? 5 :
                fournisseur.getQualiteService().equals("bon") ? 3 : 1));

        fournisseur.setNote(noteFinale);
        fournisseurRepository.save(fournisseur);

        return noteFinale;
    }

    @Override
    public List<Fournisseur> comparerFournisseursParProduit(String produit)
{
        List<HistoriqueAchats> achats = historiqueAchatsRepository.findByPr
oduit(produit);
        return achats.stream()
                .map(HistoriqueAchats::getFournisseur)
                .distinct()
                .collect(Collectors.toList());
    }
}
```

3. CommandeAchatService.java

```
java
Copy
Download
package com.gestionachat.service;
```

```java
import com.gestionachat.dto.CommandeAchatDTO;
import com.gestionachat.entity.CommandeAchat;
import com.gestionachat.exception.CommandeNotFoundException;
import com.gestionachat.exception.FournisseurNotFoundException;
import java.util.List;

public interface CommandeAchatService {
    CommandeAchat createCommande(CommandeAchatDTO commandeDTO) throws Fourn
isseurNotFoundException;
    CommandeAchat updateCommande(Long id, CommandeAchatDTO commandeDTO) thr
ows CommandeNotFoundException;
    List<CommandeAchat> getAllCommandes();
    CommandeAchat getCommandeById(Long id) throws CommandeNotFoundException
;
    void deleteCommande(Long id) throws CommandeNotFoundException;
    CommandeAchat updateStatutCommande(Long id, String statut) throws Comma
ndeNotFoundException;
    List<CommandeAchat> getCommandesByFournisseur(Long fournisseurId) throw
s FournisseurNotFoundException;
}
```

4. CommandeAchatServiceImpl.java

java

Copy

Download

```java
package com.gestionachat.service;

import com.gestionachat.dto.CommandeAchatDTO;
import com.gestionachat.dto.LigneCommandeDTO;
import com.gestionachat.entity.*;
import com.gestionachat.exception.CommandeNotFoundException;
import com.gestionachat.exception.FournisseurNotFoundException;
import com.gestionachat.repository.CommandeAchatRepository;
import com.gestionachat.repository.FournisseurRepository;
import com.gestionachat.repository.LigneCommandeAchatRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import java.util.Date;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
@AllArgsConstructor
public class CommandeAchatServiceImpl implements CommandeAchatService {
```

```
    private final CommandeAchatRepository commandeRepository;
    private final FournisseurRepository fournisseurRepository;
    private final LigneCommandeAchatRepository ligneCommandeRepository;

    @Override
    public CommandeAchat createCommande(CommandeAchatDTO commandeDTO) throw
s FournisseurNotFoundException {
        CommandeAchat commande = new CommandeAchat();

        Fournisseur fournisseur = fournisseurRepository.findById(commandeDT
O.getFournisseurId())
                .orElseThrow(() -> new FournisseurNotFoundException("Fourni
sseur non trouvé"));

        commande.setFournisseur(fournisseur);
        commande.setDate(new Date());
        commande.setStatut("EN_COURS");

        // Calcul du montant total
        double montantTotal = commandeDTO.getLignesCommande().stream()
                .mapToDouble(l -> l.getQuantite() * l.getPrixUnitaire())
                .sum();

        commande.setMontant(montantTotal);

        CommandeAchat savedCommande = commandeRepository.save(commande);

        // Sauvegarde des lignes de commande
        List<LigneCommandeAchat> lignes = commandeDTO.getLignesCommande().s
tream()
                .map(l -> {
                    LigneCommandeAchat ligne = new LigneCommandeAchat();
                    ligne.setCommande(savedCommande);
                    ligne.setProduit(l.getProduit());
                    ligne.setQuantite(l.getQuantite());
                    ligne.setPrixUnitaire(l.getPrixUnitaire());
                    return ligne;
                })
                .collect(Collectors.toList());

        ligneCommandeRepository.saveAll(lignes);
        savedCommande.setLignesCommande(lignes);

        return savedCommande;
    }
```

```
    @Override
    public CommandeAchat updateCommande(Long id, CommandeAchatDTO commandeD
TO) throws CommandeNotFoundException {
        CommandeAchat commande = commandeRepository.findById(id)
                .orElseThrow(() -> new CommandeNotFoundException("Commande
non trouvée"));

        // Mise à jour des lignes de commande
        ligneCommandeRepository.deleteAll(commande.getLignesCommande());

        List<LigneCommandeAchat> newLignes = commandeDTO.getLignesCommande(
).stream()
                .map(l -> {
                    LigneCommandeAchat ligne = new LigneCommandeAchat();
                    ligne.setCommande(commande);
                    ligne.setProduit(l.getProduit());
                    ligne.setQuantite(l.getQuantite());
                    ligne.setPrixUnitaire(l.getPrixUnitaire());
                    return ligne;
                })
                .collect(Collectors.toList());

        ligneCommandeRepository.saveAll(newLignes);

        // Calcul du nouveau montant
        double montantTotal = newLignes.stream()
                .mapToDouble(l -> l.getQuantite() * l.getPrixUnitaire())
                .sum();

        commande.setMontant(montantTotal);
        commande.setLignesCommande(newLignes);

        return commandeRepository.save(commande);
    }

    @Override
    public List<CommandeAchat> getAllCommandes() {
        return commandeRepository.findAll();
    }

    @Override
    public CommandeAchat getCommandeById(Long id) throws CommandeNotFoundEx
ception {
        return commandeRepository.findById(id)
```

```
                .orElseThrow(() -> new CommandeNotFoundException("Commande
non trouvée"));
    }

    @Override
    public void deleteCommande(Long id) throws CommandeNotFoundException {
        if (!commandeRepository.existsById(id)) {
            throw new CommandeNotFoundException("Commande non trouvée");
        }
        commandeRepository.deleteById(id);
    }

    @Override
    public CommandeAchat updateStatutCommande(Long id, String statut) throw
s CommandeNotFoundException {
        CommandeAchat commande = getCommandeById(id);
        commande.setStatut(statut);

        // Si la commande est livrée, mettre à jour l'historique
        if ("LIVREE".equals(statut)) {
            updateHistoriqueAchats(commande);
        }

        return commandeRepository.save(commande);
    }

    @Override
    public List<CommandeAchat> getCommandesByFournisseur(Long fournisseurId
) throws FournisseurNotFoundException {
        Fournisseur fournisseur = fournisseurRepository.findById(fournisseu
rId)
                .orElseThrow(() -> new FournisseurNotFoundException("Fourni
sseur non trouvé"));
        return commandeRepository.findByFournisseur(fournisseur);
    }

    private void updateHistoriqueAchats(CommandeAchat commande) {
        commande.getLignesCommande().forEach(ligne -> {
            HistoriqueAchats historique = new HistoriqueAchats();
            historique.setFournisseur(commande.getFournisseur());
            historique.setProduit(ligne.getProduit());
            historique.setQuantite(ligne.getQuantite());
            // Ici, vous devriez calculer le délai réel de livraison
            historique.setDelaiLivraison(5); // Exemple: délai fixe de 5 jo
urs
        });
```

```
    }
}
```

Étape 6: Contrôleurs

1. FournisseurController.java

java
Copy
Download

```java
package com.gestionachat.controller;

import com.gestionachat.dto.FournisseurDTO;
import com.gestionachat.entity.Fournisseur;
import com.gestionachat.exception.FournisseurNotFoundException;
import com.gestionachat.service.FournisseurService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/fournisseurs")
public class FournisseurController {

    private final FournisseurService fournisseurService;

    public FournisseurController(FournisseurService fournisseurService) {
        this.fournisseurService = fournisseurService;
    }

    @PostMapping
    public ResponseEntity<Fournisseur> createFournisseur(@RequestBody Fourn
isseurDTO fournisseurDTO) {
        Fournisseur newFournisseur = fournisseurService.createFournisseur(f
ournisseurDTO);
        return new ResponseEntity<>(newFournisseur, HttpStatus.CREATED);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Fournisseur> updateFournisseur(@PathVariable Long
id, @RequestBody FournisseurDTO fournisseurDTO) {
        try {
            Fournisseur updatedFournisseur = fournisseurService.updateFourn
isseur(id, fournisseurDTO);
            return new ResponseEntity<>(updatedFournisseur, HttpStatus.OK);
```

```
        } catch (FournisseurNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @GetMapping
    public ResponseEntity<List<Fournisseur>> getAllFournisseurs() {
        List<Fournisseur> fournisseurs = fournisseurService.getAllFournisse
urs();
        return new ResponseEntity<>(fournisseurs, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Fournisseur> getFournisseurById(@PathVariable Lon
g id) {
        try {
            Fournisseur fournisseur = fournisseurService.getFournisseurById
(id);
            return new ResponseEntity<>(fournisseur, HttpStatus.OK);
        } catch (FournisseurNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteFournisseur(@PathVariable Long id) {
        try {
            fournisseurService.deleteFournisseur(id);
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        } catch (FournisseurNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @GetMapping("/{id}/evaluation")
    public ResponseEntity<Double> evaluerFournisseur(@PathVariable Long id)
{
        try {
            double note = fournisseurService.evaluerFournisseur(id);
            return new ResponseEntity<>(note, HttpStatus.OK);
        } catch (FournisseurNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @GetMapping("/comparaison/{produit}")
```

```
    public ResponseEntity<List<Fournisseur>> comparerFournisseurs(@PathVariable String produit) {
        List<Fournisseur> fournisseurs = fournisseurService.comparerFournisseursParProduit(produit);
        return new ResponseEntity<>(fournisseurs, HttpStatus.OK);
    }
}
```

2. CommandeAchatController.java

java

Copy

Download

```
package com.gestionachat.controller;

import com.gestionachat.dto.CommandeAchatDTO;
import com.gestionachat.entity.CommandeAchat;
import com.gestionachat.exception.CommandeNotFoundException;
import com.gestionachat.exception.FournisseurNotFoundException;
import com.gestionachat.service.CommandeAchatService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/commandes")
public class CommandeAchatController {

    private final CommandeAchatService commandeService;

    public CommandeAchatController(CommandeAchatService commandeService) {
        this.commandeService = commandeService;
    }

    @PostMapping
    public ResponseEntity<CommandeAchat> createCommande(@RequestBody CommandeAchatDTO commandeDTO) {
        try {
            CommandeAchat newCommande = commandeService.createCommande(commandeDTO);
            return new ResponseEntity<>(newCommande, HttpStatus.CREATED);
        } catch (FournisseurNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
        }
    }
```

```
    @PutMapping("/{id}")
    public ResponseEntity<CommandeAchat> updateCommande(@PathVariable Long
id, @RequestBody CommandeAchatDTO commandeDTO) {
        try {
            CommandeAchat updatedCommande = commandeService.updateCommande(
id, commandeDTO);
            return new ResponseEntity<>(updatedCommande, HttpStatus.OK);
        } catch (CommandeNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @GetMapping
    public ResponseEntity<List<CommandeAchat>> getAllCommandes() {
        List<CommandeAchat> commandes = commandeService.getAllCommandes();
        return new ResponseEntity<>(commandes, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<CommandeAchat> getCommandeById(@PathVariable Long
id) {
        try {
            CommandeAchat commande = commandeService.getCommandeById(id);
            return new ResponseEntity<>(commande, HttpStatus.OK);
        } catch (CommandeNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteCommande(@PathVariable Long id) {
        try {
            commandeService.deleteCommande(id);
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        } catch (CommandeNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @PutMapping("/{id}/statut/{statut}")
    public ResponseEntity<CommandeAchat> updateStatutCommande(
            @PathVariable Long id, @PathVariable String statut) {
        try {
            CommandeAchat commande = commandeService.updateStatutCommande(i
d, statut);
```

```
            return new ResponseEntity<>(commande, HttpStatus.OK);
        } catch (CommandeNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @GetMapping("/fournisseur/{fournisseurId}")
    public ResponseEntity<List<CommandeAchat>> getCommandesByFournisseur(@P
athVariable Long fournisseurId) {
        try {
            List<CommandeAchat> commandes = commandeService.getCommandesByF
ournisseur(fournisseurId);
            return new ResponseEntity<>(commandes, HttpStatus.OK);
        } catch (FournisseurNotFoundException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }
}
```

Étape 7: DTOs et Exceptions

1. FournisseurDTO.java

java

Copy

Download

```
package com.gestionachat.dto;

import lombok.Data;

@Data
public class FournisseurDTO {
    private String nom;
    private String contact;
    private String qualiteService;
}
```

2. CommandeAchatDTO.java

java

Copy

Download

```
package com.gestionachat.dto;

import lombok.Data;
import java.util.List;
```

```
@Data
public class CommandeAchatDTO {
    private Long fournisseurId;
    private List<LigneCommandeDTO> lignesCommande;
}
```

3. LigneCommandeDTO.java

```
java
Copy
Download
package com.gestionachat.dto;

import lombok.Data;

@Data
public class LigneCommandeDTO {
    private String produit;
    private int quantite;
    private double prixUnitaire;
}
```

4. FournisseurNotFoundException.java

```
java
Copy
Download
package com.gestionachat.exception;

public class FournisseurNotFoundException extends Exception {
    public FournisseurNotFoundException(String message) {
        super(message);
    }
}
```

5. CommandeNotFoundException.java

```
java
Copy
Download
package com.gestionachat.exception;

public class CommandeNotFoundException extends Exception {
    public CommandeNotFoundException(String message) {
        super(message);
```

```
    }
}
```

Étape 8: Configuration CORS

WebConfig.java

java

Copy

Download

```
package com.gestionachat.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("http://localhost:4200")
                .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
                .allowedHeaders("*")
                .allowCredentials(true);
    }
}
```

Partie 2: Frontend Angular

Étape 1: Configuration initiale

1. **Créer une nouvelle application Angular**

bash

Copy

Download

```
ng new gestion-achat-frontend
cd gestion-achat-frontend
```

2. **Installer les dépendances nécessaires**

bash

Copy

Download

```
npm install bootstrap @ng-bootstrap/ng-bootstrap font-awesome
```

3. Configurer Angular.json

```
json
Copy
Download
"styles": [
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "node_modules/font-awesome/css/font-awesome.min.css",
  "src/styles.css"
],
"scripts": [
  "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
]
```

4. Structure des dossiers

```
Copy
Download
src/app/
├── components/
│   ├── fournisseur/
│   ├── commande/
│   ├── shared/
├── models/
├── services/
├── app-routing.module.ts
├── app.module.ts
```

Étape 2: Modèles (Models)

1. fournisseur.model.ts

```
typescript
Copy
Download
export interface Fournisseur {
  id: number;
  nom: string;
  contact: string;
  qualiteService: string;
  note: number;
}
```

2. commande-achat.model.ts

```typescript
Copy
Download
import { Fournisseur } from './fournisseur.model';

export interface LigneCommandeAchat {
  id: number;
  produit: string;
  quantite: number;
  prixUnitaire: number;
}

export interface CommandeAchat {
  id: number;
  fournisseur: Fournisseur;
  date: Date;
  statut: string;
  montant: number;
  lignesCommande: LigneCommandeAchat[];
}
```

3. historique-achats.model.ts

```typescript
Copy
Download
import { Fournisseur } from './fournisseur.model';

export interface HistoriqueAchats {
  id: number;
  fournisseur: Fournisseur;
  produit: string;
  quantite: number;
  delaiLivraison: number;
}
```

Étape 3: Services

1. fournisseur.service.ts

```typescript
Copy
Download
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
```

```typescript
import { Fournisseur } from '../models/fournisseur.model';

@Injectable({
  providedIn: 'root'
})
export class FournisseurService {
  private apiUrl = 'http://localhost:8080/api/fournisseurs';

  constructor(private http: HttpClient) { }

  createFournisseur(fournisseur: Fournisseur): Observable<Fournisseur> {
    return this.http.post<Fournisseur>(this.apiUrl, fournisseur);
  }

  updateFournisseur(id: number, fournisseur: Fournisseur): Observable<Fourn
isseur> {
    return this.http.put<Fournisseur>(`${this.apiUrl}/${id}`, fournisseur);
  }

  getAllFournisseurs(): Observable<Fournisseur[]> {
    return this.http.get<Fournisseur[]>(this.apiUrl);
  }

  getFournisseurById(id: number): Observable<Fournisseur> {
    return this.http.get<Fournisseur>(`${this.apiUrl}/${id}`);
  }

  deleteFournisseur(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`);
  }

  evaluerFournisseur(id: number): Observable<number> {
    return this.http.get<number>(`${this.apiUrl}/${id}/evaluation`);
  }

  comparerFournisseurs(produit: string): Observable<Fournisseur[]> {
    return this.http.get<Fournisseur[]>(`${this.apiUrl}/comparaison/${produ
it}`);
  }
}
```

2. commande-achat.service.ts

typescript

Copy

Download

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { CommandeAchat } from '../models/commande-achat.model';

@Injectable({
  providedIn: 'root'
})
export class CommandeAchatService {
  private apiUrl = 'http://localhost:8080/api/commandes';

  constructor(private http: HttpClient) { }

  createCommande(commande: CommandeAchat): Observable<CommandeAchat> {
    return this.http.post<CommandeAchat>(this.apiUrl, commande);
  }

  updateCommande(id: number, commande: CommandeAchat): Observable<CommandeA
chat> {
    return this.http.put<CommandeAchat>(`${this.apiUrl}/${id}`, commande);
  }

  getAllCommandes(): Observable<CommandeAchat[]> {
    return this.http.get<CommandeAchat[]>(this.apiUrl);
  }

  getCommandeById(id: number): Observable<CommandeAchat> {
    return this.http.get<CommandeAchat>(`${this.apiUrl}/${id}`);
  }

  deleteCommande(id: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/${id}`);
  }

  updateStatutCommande(id: number, statut: string): Observable<CommandeAcha
t> {
    return this.http.put<CommandeAchat>(`${this.apiUrl}/${id}/statut/${stat
ut}`, {});
  }

  getCommandesByFournisseur(fournisseurId: number): Observable<CommandeAcha
t[]> {
    return this.http.get<CommandeAchat[]>(`${this.apiUrl}/fournisseur/${fou
rnisseurId}`);
  }
}
```

Étape 4: Composants Fournisseur

1. **fournisseur-list.component.ts**

typescript

Copy

Download

```typescript
import { Component, OnInit } from '@angular/core';
import { Fournisseur } from '../../models/fournisseur.model';
import { FournisseurService } from '../../services/fournisseur.service';

@Component({
  selector: 'app-fournisseur-list',
  templateUrl: './fournisseur-list.component.html',
  styleUrls: ['./fournisseur-list.component.css']
})
export class FournisseurListComponent implements OnInit {
  fournisseurs: Fournisseur[] = [];
  isLoading = true;

  constructor(private fournisseurService: FournisseurService) { }

  ngOnInit(): void {
    this.loadFournisseurs();
  }

  loadFournisseurs(): void {
    this.fournisseurService.getAllFournisseurs().subscribe(
      data => {
        this.fournisseurs = data;
        this.isLoading = false;
      },
      error => {
        console.error('Error loading fournisseurs', error);
        this.isLoading = false;
      }
    );
  }

  deleteFournisseur(id: number): void {
    if (confirm('Êtes-vous sûr de vouloir supprimer ce fournisseur ?')) {
      this.fournisseurService.deleteFournisseur(id).subscribe(
        () => {
          this.fournisseurs = this.fournisseurs.filter(f => f.id !== id);
        },
        error => console.error('Error deleting fournisseur', error)
```

```
      );
    }
  }

  evaluerFournisseur(id: number): void {
    this.fournisseurService.evaluerFournisseur(id).subscribe(
      note => {
        const index = this.fournisseurs.findIndex(f => f.id === id);
        if (index !== -1) {
          this.fournisseurs[index].note = note;
        }
      },
      error => console.error('Error evaluating fournisseur', error)
    );
  }
}
```

2. fournisseur-list.component.html

html

Copy

Download

Run

```
<div class="container mt-4">
  <h2 class="mb-4">Liste des Fournisseurs</h2>

  <div class="text-end mb-3">
    <a routerLink="/fournisseurs/add" class="btn btn-primary">
      <i class="fa fa-plus"></i> Ajouter un Fournisseur
    </a>
  </div>

  <div *ngIf="isLoading" class="text-center">
    <div class="spinner-border text-primary" role="status">
      <span class="visually-hidden">Loading...</span>
    </div>
  </div>

  <div *ngIf="!isLoading && fournisseurs.length === 0" class="alert alert-i
nfo">
    Aucun fournisseur trouvé.
  </div>

  <table *ngIf="!isLoading && fournisseurs.length > 0" class="table table-s
triped table-hover">
    <thead class="table-dark">
```

```
      <tr>
        <th>ID</th>
        <th>Nom</th>
        <th>Contact</th>
        <th>Qualité Service</th>
        <th>Note</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let fournisseur of fournisseurs">
        <td>{{ fournisseur.id }}</td>
        <td>{{ fournisseur.nom }}</td>
        <td>{{ fournisseur.contact }}</td>
        <td>{{ fournisseur.qualiteService }}</td>
        <td>
          <span class="badge bg-{{ getNoteClass(fournisseur.note) }}">
            {{ fournisseur.note | number:'1.1-1' }}
          </span>
        </td>
        <td>
          <div class="btn-group" role="group">
            <a [routerLink]="['/fournisseurs', fournisseur.id]" class="btn
btn-sm btn-info">
              <i class="fa fa-eye"></i>
            </a>
            <a [routerLink]="['/fournisseurs/edit', fournisseur.id]" class=
"btn btn-sm btn-warning">
```