ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

# Rapport de Projet :

# Test Systèmes Distribués GLSID


**Réalisé par :**

- JERRARI Mountassir

**Encadrée par :**

Pr. Mohamed YOUSSFI


**Année Universitaire : 2025-2026**

# Sommaire

# 1. Introduction

## 1.1 Contexte du Projet

Ce projet vise à concevoir et développer un système distribué basé sur une architecture microservices permettant de gérer les cotations boursières des entreprises cotées en bourse. Le système est construit avec Spring Cloud et respecte les principes de l'architecture microservices moderne.

## 1.2 Objectifs

- **Gestion des entreprises** : Permettre l'enregistrement, la modification et la consultation des entreprises cotées
- **Gestion des cotations** : Gérer l'historique des cotations boursières avec calcul automatique des prix actuels

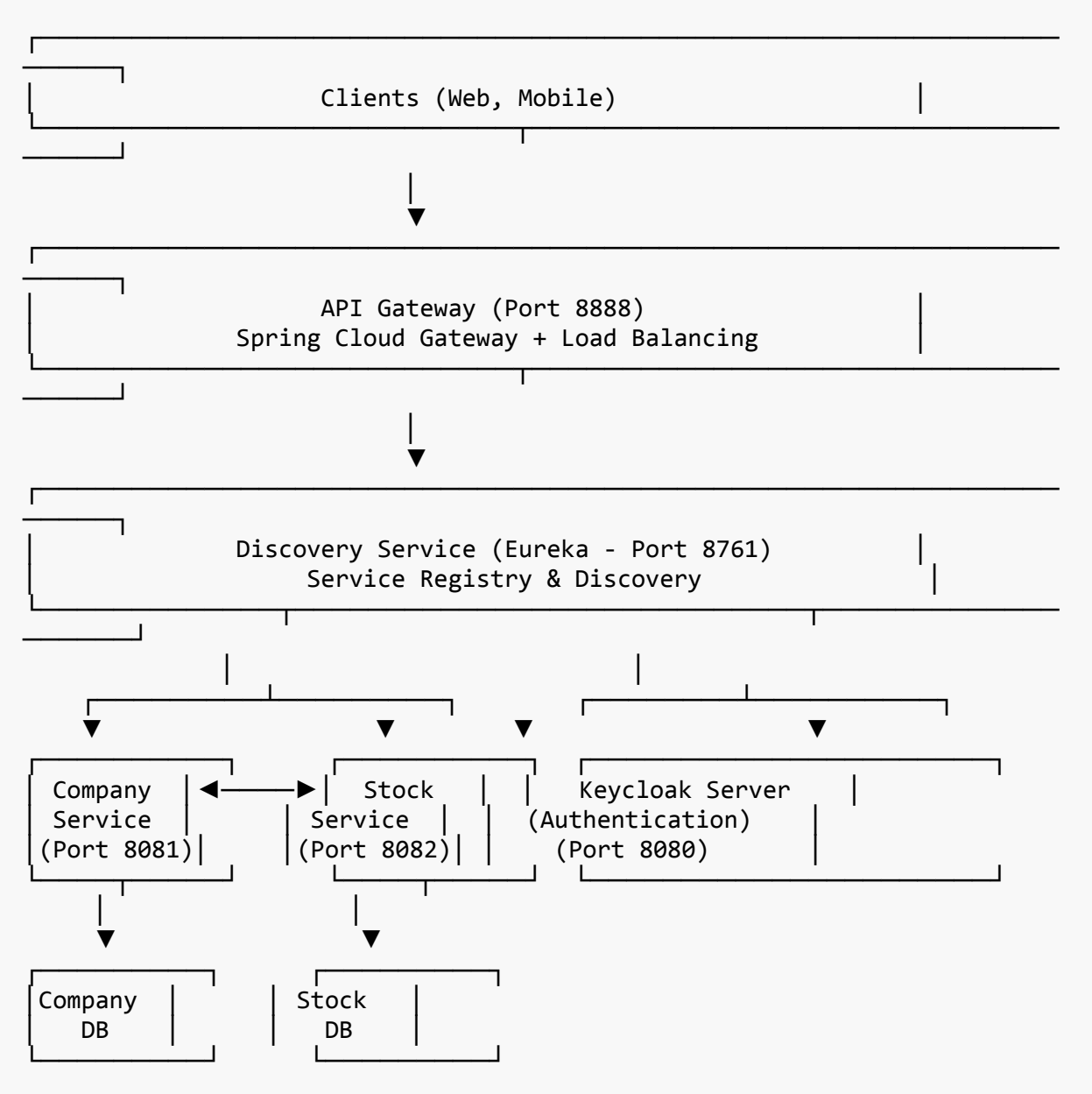
## 1.3 Technologies Utilisées

| Technologie | Usage |
|---|---|
| Spring Boot 3.x | Framework de base des microservices |
| Spring Cloud | Outils pour architecture distribuée |
| Eureka Server | Service Discovery |
| Spring Cloud Gateway | API Gateway |
| OpenFeign | Communication REST inter-services |
| Resilience4J | Tolérance aux pannes (Circuit Breaker) |
| Keycloak | Gestion de l'authentification OAuth2/OIDC |
| PostgreSQL/MySQL | Bases de données relationnelles |
| Docker | Containerisation |
| Docker Compose | Orchestration locale |
| React | Frontend Web |
| Maven | Gestionnaire de dépendances |

## 2. Architecture Globale du Système

### 2.1 Vue d'Ensemble

Le système est composé de 5 microservices principaux organisés selon une architecture en couches :




### 2.2 Principe de l'Architecture Microservices

L'architecture microservices adoptée offre plusieurs avantages :

- **Décomposition fonctionnelle** : Chaque service gère un domaine métier spécifique

- **Déploiement indépendant** : Chaque service peut être déployé séparément
- **Scalabilité horizontale** : Possibilité de dupliquer les instances selon la charge
- **Isolation des pannes** : La défaillance d'un service n'affecte pas les autres
- **Technologie hétérogène** : Possibilité d'utiliser différentes technologies par service

### 2.3 Patterns Architecturaux Utilisés

#### 2.3.1 API Gateway Pattern

- Point d'entrée unique pour tous les clients
- Routage intelligent des requêtes
- Load balancing automatique
- Gestion centralisée de la sécurité

#### 2.3.2 Service Registry Pattern

- Enregistrement automatique des services
- Découverte dynamique des instances
- Health check des services

#### 2.3.3 Circuit Breaker Pattern

- Protection contre les cascades de pannes
- Fallback automatique en cas d'erreur
- Récupération progressive

---

## 4. Microservices Techniques

### 4.1 Discovery Service (Eureka Server)

#### 4.1.1 Rôle et Responsabilités

Le Discovery Service joue un rôle central dans l'architecture microservices :

- **Service Registry** : Enregistrement de tous les microservices
- **Service Discovery** : Permet aux services de se découvrir mutuellement
- **Health Monitoring** : Surveillance de l'état de santé des services
- **Load Balancing** : Distribution de charge entre instances

#### 4.1.2 Configuration

**pom.xml**

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
```

```
<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
</dependencies>
```

**DiscoveryServiceApplication.java**

```java
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(DiscoveryServiceApplication.class, args);
    }
}
```

**application.properties**

```
spring.application.name=discovery-service
server.port=8761

# Eureka Server Configuration
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/

# Dashboard Configuration
eureka.dashboard.enabled=true
```

### 4.1.3 Fonctionnalités

- **Dashboard Web** : Interface de monitoring accessible sur http://localhost:8761
- **API REST** : Endpoints pour interroger le registre
- **Heartbeat** : Vérification périodique de la disponibilité des services (30 secondes par défaut)
- **Self-Preservation Mode** : Protection contre les faux positifs en cas de problème réseau

### 4.1.4 Captures d'Écran

Instances currently registered with Eureka

| Application | AMIs | Availability Zones | Status |
|---|---|---|---|
| COMPANY-SERVICE | n/a (1) | (1) | UP (1) - 172.16.0.148:company-service:8081 |
| STOCK-SERVICE | n/a (1) | (1) | UP (1) - 172.16.0.148:stock-service:8082 |
| UNKNOWN | n/a (1) | (1) | UP (1) - 172.16.0.148:8000 |

## 4.2 Gateway Service (Spring Cloud Gateway)

### 4.2.1 Rôle et Responsabilités

Le Gateway Service agit comme point d'entrée unique :

- **Routage dynamique** : Redirection des requêtes vers les services appropriés
- **Load Balancing** : Distribution de charge client-side
- **Filtrage** : Pre-processing et post-processing des requêtes

### 4.2.2 Configuration

**pom.xml**

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>
```

### 4.2.4 Avantages de l'API Gateway

- **Point d'entrée unique** : Simplifie l'accès client
- **Découplage** : Les clients ne connaissent pas l'infrastructure interne
- **Cross-Cutting Concerns** : Gestion centralisée de la sécurité, logging, monitoring
- **Évolutivité** : Facilite l'ajout de nouveaux services
-

# 5. Microservices Fonctionnels

## 5.1 Company Service

### 5.1.1 Modèle de Données

**Entity - Company.java**

```java
@Entity
@Table(name = "companies")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String name;

    @Column(name = "listing_date", nullable = false)
    private LocalDate listingDate;

    @Column(name = "current_stock_price")
    private Double currentStockPrice;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private Domain domain;

    @CreatedDate
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    @LastModifiedDate
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;
}

public enum Domain {
    IT, AI, BANKING, INSURANCE, HEALTHCARE, RETAIL, ENERGY, TELECOM
}
```

**DTO - CompanyRequestDTO.java**

```java
@Data
@Builder
public class CompanyRequestDTO {
    @NotBlank(message = "Company name is required")
    private String name;

    @NotNull(message = "Listing date is required")
```

```
    @PastOrPresent(message = "Listing date cannot be in the future")
    private LocalDate listingDate;

    @Positive(message = "Stock price must be positive")
    private Double currentStockPrice;

    @NotNull(message = "Domain is required")
    private Domain domain;
}
```

**DTO - CompanyResponseDTO.java**

```
@Data
@Builder
public class CompanyResponseDTO {
    private Long id;
    private String name;
    private LocalDate listingDate;
    private Double currentStockPrice;
    private Domain domain;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
}
```

### 5.1.2 Couche Repository

```
@Repository
public interface CompanyRepository extends JpaRepository<Company, Long> {

    Optional<Company> findByName(String name);

    List<Company> findByDomain(Domain domain);

    @Query("SELECT c FROM Company c WHERE c.currentStockPrice >
:minPrice")
    List<Company> findByStockPriceGreaterThan(@Param("minPrice") Double
minPrice);

    boolean existsByName(String name);
}
```

### 5.1.3 Couche Service

**CompanyService.java (Interface)**

```
public interface CompanyService {
    CompanyResponseDTO createCompany(CompanyRequestDTO companyRequestDTO);
    CompanyResponseDTO updateCompany(Long id, CompanyRequestDTO
companyRequestDTO);
    void deleteCompany(Long id);
    CompanyResponseDTO getCompanyById(Long id);
    List<CompanyResponseDTO> getAllCompanies();
    List<CompanyResponseDTO> getCompaniesByDomain(Domain domain);
    CompanyResponseDTO updateStockPrice(Long id, Double newPrice);
}
```

**CompanyServiceImpl.java**

```
@Service
@Transactional
@Slf4j
public class CompanyServiceImpl implements CompanyService {

    private final CompanyRepository companyRepository;
    private final CompanyMapper companyMapper;

    @Autowired
    public CompanyServiceImpl(CompanyRepository companyRepository,
                              CompanyMapper companyMapper) {
        this.companyRepository = companyRepository;
        this.companyMapper = companyMapper;
    }

    @Override
    public CompanyResponseDTO createCompany(CompanyRequestDTO dto) {
        log.info("Creating new company: {}", dto.getName());

        if (companyRepository.existsByName(dto.getName())) {
            throw new DuplicateCompanyException(
                "Company with name " + dto.getName() + " already exists");
        }

        Company company = companyMapper.toEntity(dto);
        Company savedCompany = companyRepository.save(company);

        log.info("Company created successfully with ID: {}",
savedCompany.getId());
        return companyMapper.toDTO(savedCompany);
    }
```

```
    @Override
    public CompanyResponseDTO updateStockPrice(Long id, Double newPrice) {
        log.info("Updating stock price for company ID: {} to {}", id,
newPrice);

        Company company = companyRepository.findById(id)
            .orElseThrow(() -> new CompanyNotFoundException(
                "Company not found with ID: " + id));

        company.setCurrentStockPrice(newPrice);
        Company updatedCompany = companyRepository.save(company);

        return companyMapper.toDTO(updatedCompany);
    }

}
```

### 5.1.4 Couche Controller

```
@RestController
@RequestMapping("/api/companies")
@Slf4j
@CrossOrigin(origins = "*")
public class CompanyController {

    private final CompanyService companyService;

    @Autowired
    public CompanyController(CompanyService companyService) {
        this.companyService = companyService;
    }

    @PostMapping
    public ResponseEntity<CompanyResponseDTO> createCompany(
            @Valid @RequestBody CompanyRequestDTO companyRequestDTO) {
        log.info("REST request to create company: {}", companyRequestDTO);
        CompanyResponseDTO response =
companyService.createCompany(companyRequestDTO);
        return ResponseEntity.status(HttpStatus.CREATED).body(response);
    }

    @GetMapping
    public ResponseEntity<List<CompanyResponseDTO>> getAllCompanies() {
        log.info("REST request to get all companies");
```

```
        List<CompanyResponseDTO> companies =
companyService.getAllCompanies();
        return ResponseEntity.ok(companies);
    }

    @GetMapping("/{id}")
    public ResponseEntity<CompanyResponseDTO> getCompanyById(@PathVariable
Long id) {
        log.info("REST request to get company by ID: {}", id);
        CompanyResponseDTO company = companyService.getCompanyById(id);
        return ResponseEntity.ok(company);
    }

    @GetMapping("/domain/{domain}")
    public ResponseEntity<List<CompanyResponseDTO>> getCompaniesByDomain(
            @PathVariable Domain domain) {
        log.info("REST request to get companies by domain: {}", domain);
        List<CompanyResponseDTO> companies =
companyService.getCompaniesByDomain(domain);
        return ResponseEntity.ok(companies);
    }

    @PutMapping("/{id}")
    public ResponseEntity<CompanyResponseDTO> updateCompany(
            @PathVariable Long id,
            @Valid @RequestBody CompanyRequestDTO companyRequestDTO) {
        log.info("REST request to update company ID: {}", id);
        CompanyResponseDTO response = companyService.updateCompany(id,
companyRequestDTO);
        return ResponseEntity.ok(response);
    }

    @PatchMapping("/{id}/price")
    public ResponseEntity<CompanyResponseDTO> updateStockPrice(
            @PathVariable Long id,
            @RequestParam Double newPrice) {
        log.info("REST request to update stock price for company ID: {}",
id);
        CompanyResponseDTO response = companyService.updateStockPrice(id,
newPrice);
        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteCompany(@PathVariable Long id) {
        log.info("REST request to delete company ID: {}", id);
        companyService.deleteCompany(id);
        return ResponseEntity.noContent().build();
```

```
    }
}
```

### 5.1.5 Configuration Application

**application.properties**

```
spring.application.name=company-service
server.port=8081

# Database Configuration
spring.datasource.url=jdbc:postgresql://localhost:5432/company_db
spring.datasource.username=postgres
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver

# JPA Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDi
alect

# Eureka Client Configuration
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
eureka.instance.prefer-ip-address=true
eureka.instance.lease-renewal-interval-in-seconds=30

# Actuator Configuration
management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=always
```

## 5.2 Stock Service

### 5.2.1 Modèle de Données

**Entity - StockMarket.java**

```
@Entity
@Table(name = "stock_markets")
@Data
@NoArgsConstructor
@AllArgsConstructor
```

```java
@Builder
public class StockMarket {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private LocalDate date;

    @Column(name = "open_value", nullable = false)
    private Double openValue;

    @Column(name = "high_value", nullable = false)
    private Double highValue;

    @Column(name = "low_value", nullable = false)
    private Double lowValue;

    @Column(name = "close_value", nullable = false)
    private Double closeValue;

    @Column(nullable = false)
    private Long volume;

    @Column(name = "company_id", nullable = false)
    private Long companyId;

    @Transient
    private CompanyResponseDTO company;

    @CreatedDate
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;
}
```

### 5.2.2 Client OpenFeign

```java
@FeignClient(name = "COMPANY-SERVICE", fallback =
CompanyRestClientFallback.class)
public interface CompanyRestClient {

    @GetMapping("/api/companies/{id}")
    CompanyResponseDTO getCompanyById(@PathVariable("id") Long id);

    @PatchMapping("/api/companies/{id}/price")
```

```
    CompanyResponseDTO updateStockPrice(
        @PathVariable("id") Long id,
        @RequestParam("newPrice") Double newPrice
    );
}

@Component
@Slf4j
public class CompanyRestClientFallback implements CompanyRestClient {

    @Override
    public CompanyResponseDTO getCompanyById(Long id) {
        log.warn("Fallback: Unable to get company with ID: {}", id);
        return CompanyResponseDTO.builder()
            .id(id)
            .name("Service Unavailable")
            .build();
    }

    @Override
    public CompanyResponseDTO updateStockPrice(Long id, Double newPrice) {
        log.warn("Fallback: Unable to update stock price for company ID:
{}", id);
        return null;
    }
}
```

### 5.2.3 Couche Service

```
@Service
@Transactional
@Slf4j
public class StockServiceImpl implements StockService {

    private final StockMarketRepository stockMarketRepository;
    private final CompanyRestClient companyRestClient;
    private final StockMapper stockMapper;

    @Override
    @CircuitBreaker(name = "companyService", fallbackMethod =
"createStockFallback")
    @Retry(name = "companyService")
    public StockResponseDTO createStock(StockRequestDTO dto) {
        log.info("Creating new stock for company ID: {}",
dto.getCompanyId());
```

```
        // Vérifier que la société existe
        CompanyResponseDTO company =
companyRestClient.getCompanyById(dto.getCompanyId());
        if (company == null) {
            throw new CompanyNotFoundException(
                "Company not found with ID: " + dto.getCompanyId());
        }

        // Créer la cotation
        StockMarket stock = stockMapper.toEntity(dto);
        StockMarket savedStock = stockMarketRepository.save(stock);

        // Mettre à jour le prix actuel de l'action dans Company Service
        updateCompanyCurrentPrice(dto.getCompanyId(),
dto.getCloseValue());

        StockResponseDTO response = stockMapper.toDTO(savedStock);
        response.setCompany(company);

        log.info("Stock created successfully with ID: {}",
savedStock.getId());
        return response;
    }

    private void updateCompanyCurrentPrice(Long companyId, Double
closeValue) {
        try {
            companyRestClient.updateStockPrice(companyId, closeValue);
            log.info("Updated current stock price for company ID: {}",
companyId);
        } catch (Exception e) {
            log.error("Failed to update company stock price: {}",
e.getMessage());
        }
    }

    public StockResponseDTO createStockFallback(StockRequestDTO dto,
Exception e) {
        log.error("Fallback triggered for createStock: {}",
e.getMessage());
        throw new ServiceUnavailableException("Company service is
currently unavailable");
    }

    // Autres méthodes...
}
```

5.2.4 Configuration

**application.properties**

```properties
spring.application.name=stock-service
server.port=8082
```

# Database Configuration

```
spring.datasource.url=jdbc:postgresql://localhost:5432/stock_db
spring.datasource.username=postgres
spring.datasource.password=password
```

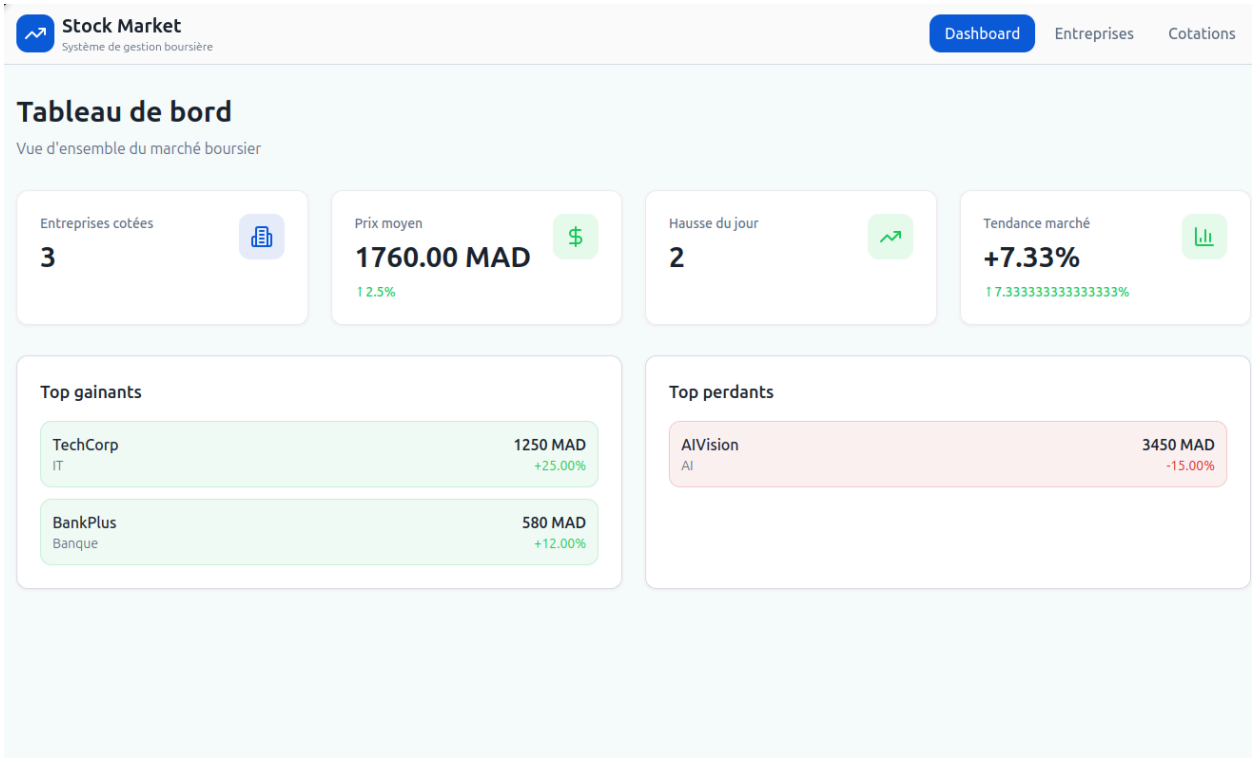
# Eureka Configuration

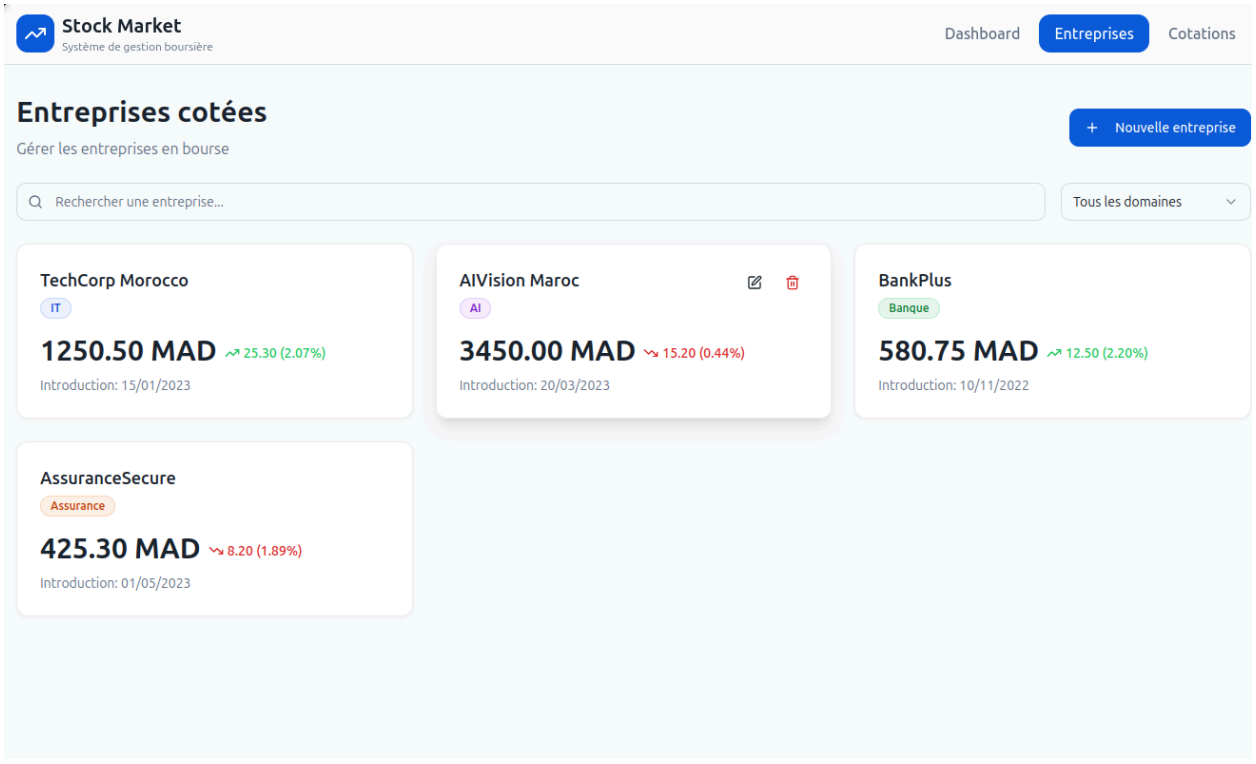
```
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
```

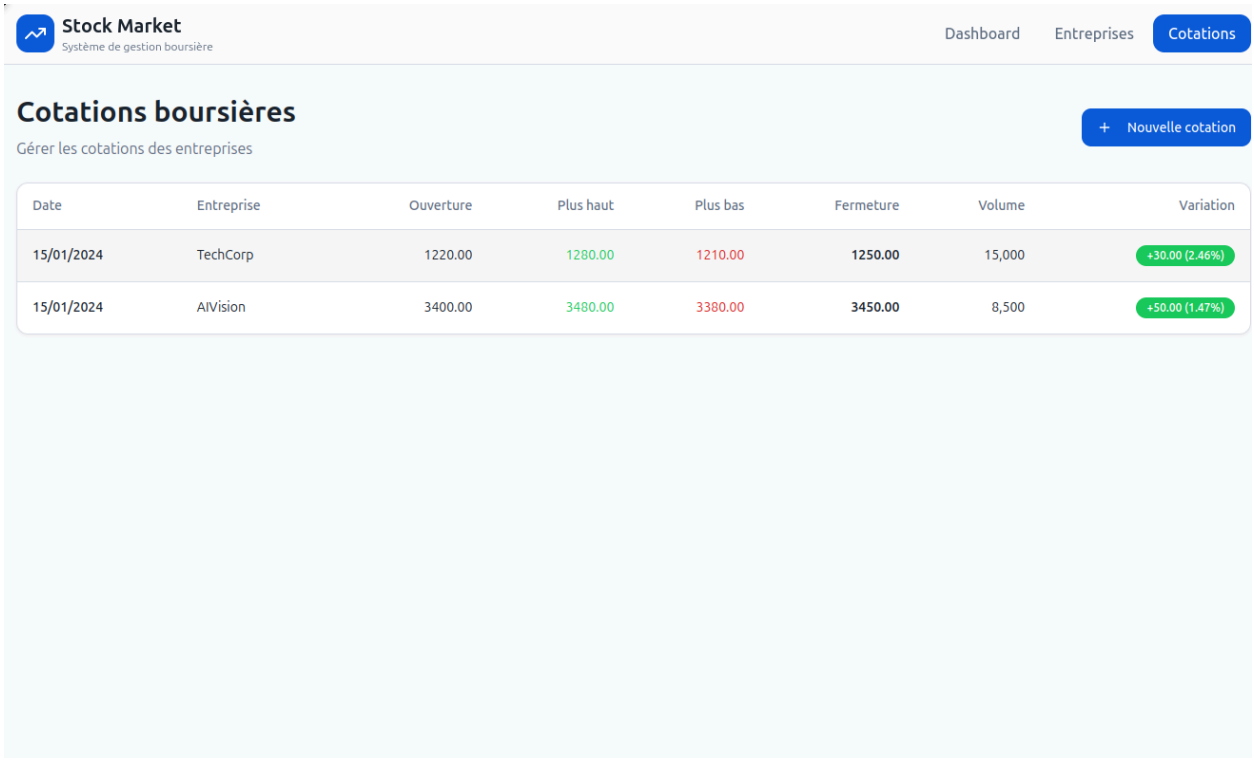
# OpenFeign Configuration

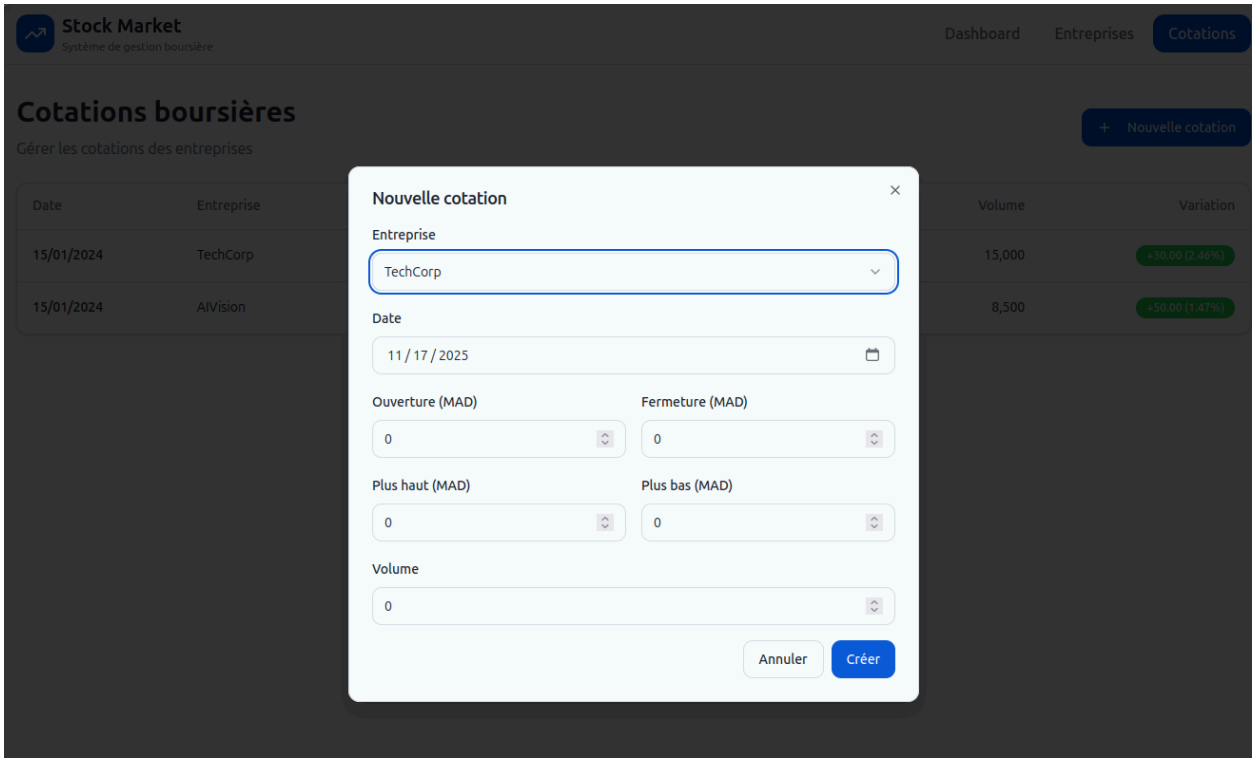
```
spring.cloud.openfeign.client.config.default.connect-timeout=5000
spring
```

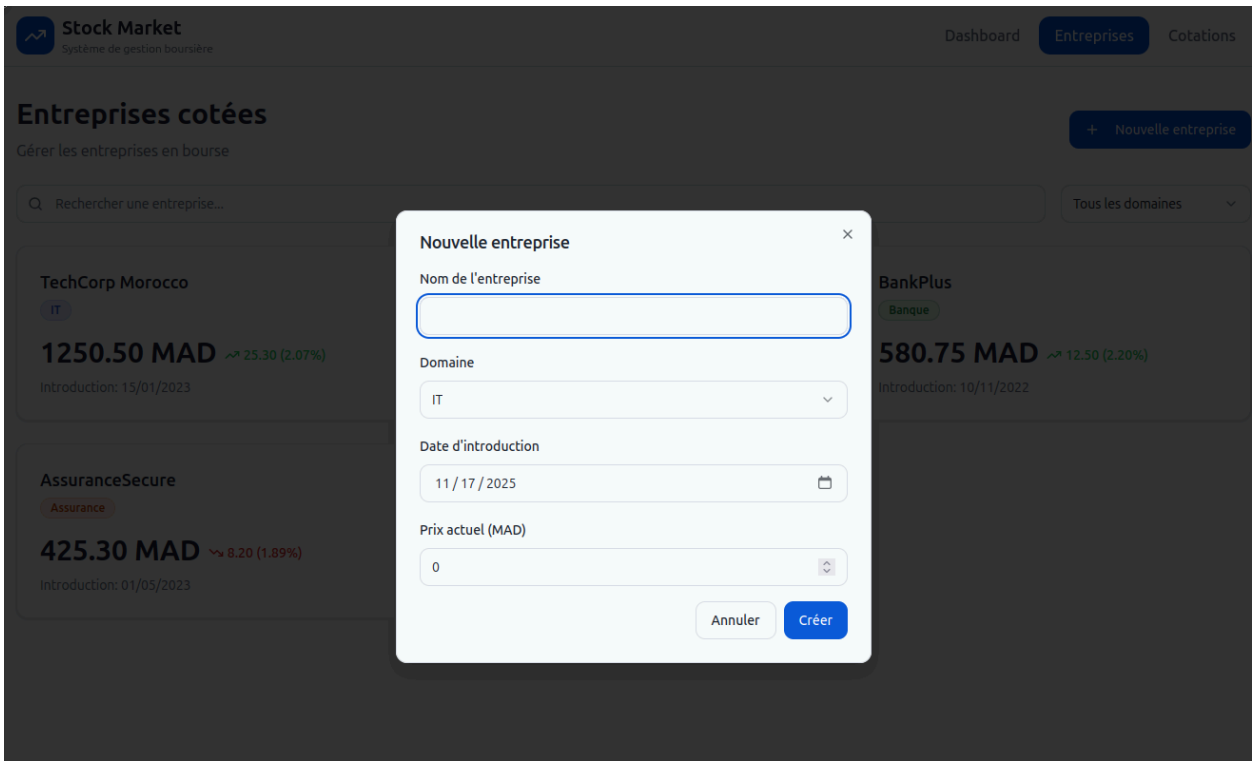
# Capture d’ecran :

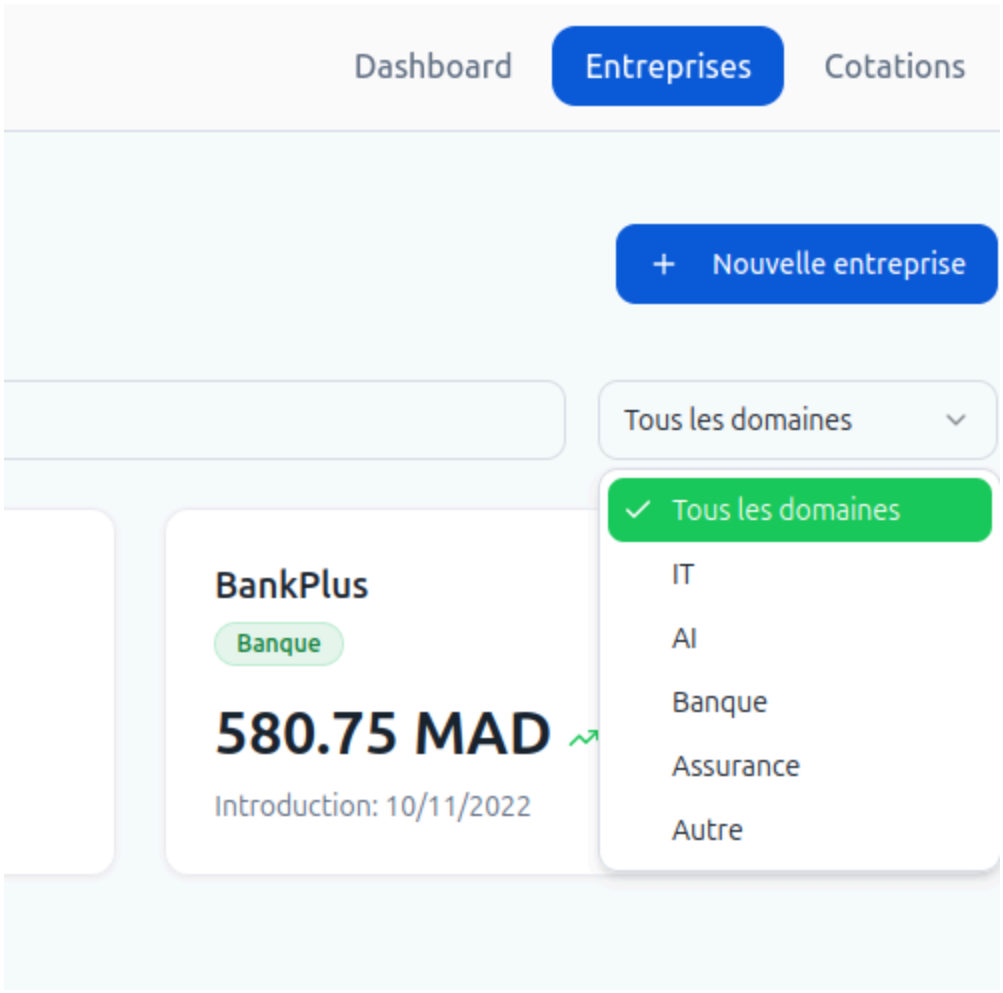

Stock Market
Système de gestion boursière
Dashboard
Entreprises
Cotations
Tableau de bord
Vue d'ensemble du marché boursier
Entreprises cotées
3
Prix moyen
1760.00 MAD
↑ 2.5%
Hausse du jour
2
Tendance marché
+7.33%
↑ 7.333333333333333%
Top gainants
TechCorp
IT
1250 MAD
+25.00%
BankPlus
Banque
580 MAD
+12.00%
Top perdants
AIVision
AI
3450 MAD
-15.00%


Stock Market
Système de gestion boursière
Dashboard
Entreprises
Cotations
Entreprises cotées
Gérer les entreprises en bourse
Nouvelle entreprise
Rechercher une entreprise...
Tous les domaines
TechCorp Morocco
IT
1250.50 MAD
25.30 (2.07%)
Introduction: 15/01/2023
AIVision Maroc
AI
3450.00 MAD
15.20 (0.44%)
Introduction: 20/03/2023
BankPlus
Banque
580.75 MAD
12.50 (2.20%)
Introduction: 10/11/2022
AssuranceSecure
Assurance
425.30 MAD
8.20 (1.89%)
Introduction: 01/05/2023


Stock Market
Système de gestion boursière
Dashboard
Entreprises
Cotations
Cotations boursières
Gérer les cotations des entreprises
+ Nouvelle cotation
Date
Entreprise
Ouverture
Plus haut
Plus bas
Fermeture
Volume
Variation
15/01/2024
TechCorp
1220.00
1280.00
1210.00
1250.00
15,000
+30.00 (2.46%)
15/01/2024
AIVision
3400.00
3480.00
3380.00
3450.00
8,500
+50.00 (1.47%)


Stock Market
Système de gestion boursière
Dashboard
Entreprises
Cotations
Cotations boursières
Gérer les cotations des entreprises
Nouvelle cotation
Date
Entreprise
Volume
Variation
15/01/2024
TechCorp
15,000
+30.00 (2.46%)
15/01/2024
AIVision
8,500
+50.00 (1.47%)
Nouvelle cotation
Entreprise
TechCorp
Date
11 / 17 / 2025
Ouverture (MAD)
0
Fermeture (MAD)
0
Plus haut (MAD)
0
Plus bas (MAD)
0
Volume
0
Annuler
Créer


Stock Market
Système de gestion boursière
Dashboard
Entreprises
Cotations
Entreprises cotées
Gérer les entreprises en bourse
Nouvelle entreprise
Rechercher une entreprise...
Tous les domaines
TechCorp Morocco
IT
1250.50 MAD
25.30 (2.07%)
Introduction: 15/01/2023
BankPlus
Banque
580.75 MAD
12.50 (2.20%)
Introduction: 10/11/2022
AssuranceSecure
Assurance
425.30 MAD
8.20 (1.89%)
Introduction: 01/05/2023
Nouvelle entreprise
Nom de l'entreprise
Domaine
IT
Date d'introduction
11 / 17 / 2025
Prix actuel (MAD)
0
Annuler
Créer




## 6. Deployment (docker compose) :

docker-compose.yml

```yaml
version: '3.8'

services:
  # Eureka Discovery Service
  discovery-service: ...

  # API Gateway
  gateway-service: ...

  # Company Service
  company-service: ...

  # Stock Service
  stock-service: ...

  # Frontend (React in development mode)
  frontend: ...

networks:
  stock-market-network:
    driver: bridge
```