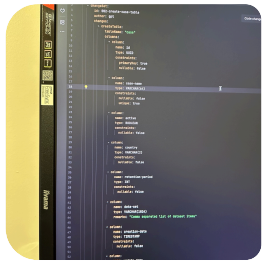
Crud entités case config




La relation entre config et case esr la suivante case peux avoir plusieurs config et un config a un seul case

Ajouter ce qui est nécessaire pour le liquibase et fait le crud de les deux entité

Crud complet entité doa controller service repo ect et n'oublie pas les log je veux un travail profisionelle de dev expert de spring

Je travail avec la version 3.5.3 de soringboot et java 21

Excellent 👌 tu veux donc :

- définir la **relation entre** `case` **et** `config` dans Liquibase (1 `case` → plusieurs `config`),
- et générer **un CRUD complet professionnel** (Spring Boot 3.5.3 / Java 21) pour les deux entités (`Case` et `Config`), avec :

- `Entity`,
- `Repository`,
- `Service`,
- `Controller`,
- logs structurés (via SLF4J),
- code propre et conforme aux bonnes pratiques Spring.

🔧 1. Liquibase : Relation `case` ↔ `config`

Tu dois ajouter une **colonne** `case_id` dans la table `config` avec une **foreign key** vers la table `case`.

Voici la mise à jour du fichier `tb_config-create.yaml` :

yaml

```yaml
databaseChangeLog:
  - changeSet:
      id: 001-create-config-table
      author: gpt
      changes:
        - createTable:
            tableName: config
            columns:
              - column:
                  name: id
                  type: UUID
```

```
      constraints:
        primaryKey: true
        nullable: false

  - column:
      name: notification
      type: VARCHAR(20)

  - column:
      name: callback-url
      type: VARCHAR(255)

  - column:
      name: oauth-token-url
      type: VARCHAR(255)

  - column:
      name: topic-name
      type: VARCHAR(100)

  - column:
      name: algorithm
      type: VARCHAR(50)

  - column:
      name: mode
      type: VARCHAR(20)

  - column:
```

```
      name: case_id
      type: UUID
      constraints:
        nullable: false
        foreignKeyName: fk_config_case
        references: case(id)

  - column:
      name: creation-date
      type: TIMESTAMP
      constraints:
        nullable: false

  - column:
      name: update-date
      type: TIMESTAMP
      constraints:
        nullable: false
```

🧩 2. Entités JPA

Case.java

java

```
package com.example.demo.entity;
```

```java
import jakarta.persistence.*;
import lombok.*;
import java.time.Instant;
import java.util.List;
import java.util.UUID;

@Entity
@Table(name = "case")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Case {

    @Id
    @GeneratedValue
    private UUID id;

    @Column(name = "case_name", nullable = false, unique = true, length = 64)
    private String caseName;

    @Column(nullable = false)
    private boolean active;

    @Column(length = 2, nullable = false)
    private String country;

    @Column(name = "retention_period", nullable = false)
```

```
    private int retentionPeriod;

    @Column(name = "data_set", length = 1024)
    private String dataSet;

    @Column(name = "creation_date", nullable = false)
    private Instant creationDate;

    @Column(name = "update_date", nullable = false)
    private Instant updateDate;

    @OneToMany(mappedBy = "caseEntity", cascade = CascadeType.ALL,
orphanRemoval = true, fetch = FetchType.LAZY)
    private List<Config> configs;
}
```

Config.java

java

```
package com.example.demo.entity;

import jakarta.persistence.*;
import lombok.*;
import java.time.Instant;
import java.util.UUID;

@Entity
@Table(name = "config")
```

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Config {

    @Id
    @GeneratedValue
    private UUID id;

    @Column(length = 20)
    private String notification;

    @Column(name = "callback_url", length = 255)
    private String callbackUrl;

    @Column(name = "oauth_token_url", length = 255)
    private String oauthTokenUrl;

    @Column(name = "topic_name", length = 100)
    private String topicName;

    @Column(length = 50)
    private String algorithm;

    @Column(length = 20)
    private String mode;
```

```
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "case_id", nullable = false)
    private Case caseEntity;

    @Column(name = "creation_date", nullable = false)
    private Instant creationDate;

    @Column(name = "update_date", nullable = false)
    private Instant updateDate;
}
```

🧱 3. Repository

CaseRepository.java

java

```
package com.example.demo.repository;

import com.example.demo.entity.Case;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;
import java.util.UUID;

public interface CaseRepository extends JpaRepository<Case, UUID> {
```

```
    Optional<Case> findByCaseName(String caseName);
}
```

ConfigRepository.java

```java
package com.example.demo.repository;

import com.example.demo.entity.Config;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;
import java.util.UUID;

public interface ConfigRepository extends JpaRepository<Config, UUID> {
    List<Config> findByCaseEntity_Id(UUID caseId);
}
```

⚙️ 4. Service Layer

CaseService.java

```java
package com.example.demo.service;
```

```java
import com.example.demo.entity.Case;
import com.example.demo.repository.CaseRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.time.Instant;
import java.util.List;
import java.util.UUID;

@Service
@RequiredArgsConstructor
@Slf4j
public class CaseService {

    private final CaseRepository caseRepository;

    public List<Case> getAllCases() {
        log.info("Fetching all cases");
        return caseRepository.findAll();
    }

    public Case getCaseById(UUID id) {
        log.info("Fetching case by id: {}", id);
        return caseRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Case not found"));
    }

    public Case createCase(Case caseEntity) {
```

```
        log.info("Creating new case: {}", caseEntity.getCaseName());
        caseEntity.setCreationDate(Instant.now());
        caseEntity.setUpdateDate(Instant.now());
        return caseRepository.save(caseEntity);
    }

    public Case updateCase(UUID id, Case updatedCase) {
        log.info("Updating case: {}", id);
        Case existing = getCaseById(id);
        existing.setCaseName(updatedCase.getCaseName());
        existing.setActive(updatedCase.isActive());
        existing.setCountry(updatedCase.getCountry());
        existing.setRetentionPeriod(updatedCase.getRetentionPeriod());
        existing.setDataSet(updatedCase.getDataSet());
        existing.setUpdateDate(Instant.now());
        return caseRepository.save(existing);
    }

    public void deleteCase(UUID id) {
        log.warn("Deleting case with id: {}", id);
        caseRepository.deleteById(id);
    }
}
```

ConfigService.java

```
java
```

```java
package com.example.demo.service;

import com.example.demo.entity.Config;
import com.example.demo.repository.ConfigRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.time.Instant;
import java.util.List;
import java.util.UUID;

@Service
@RequiredArgsConstructor
@Slf4j
public class ConfigService {

    private final ConfigRepository configRepository;

    public List<Config> getAllConfigs() {
        log.info("Fetching all configs");
        return configRepository.findAll();
    }

    public List<Config> getConfigsByCase(UUID caseId) {
        log.info("Fetching configs for case {}", caseId);
        return configRepository.findByCaseEntity_Id(caseId);
    }
```

```
    public Config getConfigById(UUID id) {
        log.info("Fetching config by id: {}", id);
        return configRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Config not found"));
    }

    public Config createConfig(Config config) {
        log.info("Creating new config for case {}",
config.getCaseEntity().getId());
        config.setCreationDate(Instant.now());
        config.setUpdateDate(Instant.now());
        return configRepository.save(config);
    }

    public Config updateConfig(UUID id, Config updatedConfig) {
        log.info("Updating config {}", id);
        Config existing = getConfigById(id);
        existing.setNotification(updatedConfig.getNotification());
        existing.setCallbackUrl(updatedConfig.getCallbackUrl());
        existing.setOauthTokenUrl(updatedConfig.getOauthTokenUrl());
        existing.setTopicName(updatedConfig.getTopicName());
        existing.setAlgorithm(updatedConfig.getAlgorithm());
        existing.setMode(updatedConfig.getMode());
        existing.setUpdateDate(Instant.now());
        return configRepository.save(existing);
    }

    public void deleteConfig(UUID id) {
```

```
        log.warn("Deleting config {}", id);
        configRepository.deleteById(id);
    }
}
```

🌐 5. REST Controllers

CaseController.java

java

```java
package com.example.demo.controller;

import com.example.demo.entity.Case;
import com.example.demo.service.CaseService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.UUID;

@RestController
@RequestMapping("/api/cases")
@RequiredArgsConstructor
@Slf4j
public class CaseController {
```

```
    private final CaseService caseService;

    @GetMapping
    public List<Case> getAll() {
        log.debug("GET /api/cases");
        return caseService.getAllCases();
    }

    @GetMapping("/{id}")
    public Case getById(@PathVariable UUID id) {
        log.debug("GET /api/cases/{}", id);
        return caseService.getCaseById(id);
    }

    @PostMapping
    public Case create(@RequestBody Case caseEntity) {
        log.debug("POST /api/cases");
        return caseService.createCase(caseEntity);
    }

    @PutMapping("/{id}")
    public Case update(@PathVariable UUID id, @RequestBody Case updatedCase) {
        log.debug("PUT /api/cases/{}", id);
        return caseService.updateCase(id, updatedCase);
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable UUID id) {
```

```
        log.debug("DELETE /api/cases/{}", id);
        caseService.deleteCase(id);
    }
}
```

ConfigController.java

java

```java
package com.example.demo.controller;

import com.example.demo.entity.Config;
import com.example.demo.service.ConfigService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.UUID;

@RestController
@RequestMapping("/api/configs")
@RequiredArgsConstructor
@Slf4j
public class ConfigController {

    private final ConfigService configService;

    @GetMapping
```

```
    public List<Config> getAll() {
        log.debug("GET /api/configs");
        return configService.getAllConfigs();
    }

    @GetMapping("/case/{caseId}")
    public List<Config> getByCase(@PathVariable UUID caseId) {
        log.debug("GET /api/configs/case/{}", caseId);
        return configService.getConfigsByCase(caseId);
    }

    @GetMapping("/{id}")
    public Config getById(@PathVariable UUID id) {
        log.debug("GET /api/configs/{}", id);
        return configService.getConfigById(id);
    }

    @PostMapping
    public Config create(@RequestBody Config config) {
        log.debug("POST /api/configs");
        return configService.createConfig(config);
    }

    @PutMapping("/{id}")
    public Config update(@PathVariable UUID id, @RequestBody Config
updatedConfig) {
        log.debug("PUT /api/configs/{}", id);
        return configService.updateConfig(id, updatedConfig);
    }
```

```
    @DeleteMapping("/{id}")
    public void delete(@PathVariable UUID id) {
        log.debug("DELETE /api/configs/{}", id);
        configService.deleteConfig(id);
    }
}
```

Souhaites-tu que je t'ajoute aussi :

- ✅ Les **DTOs + MapStruct Mapper** (pour éviter d'exposer les entités directement en REST),
- ✅ La **validation (Jakarta Validation)** et gestion des erreurs personnalisée (`@ControllerAdvice`),
- ✅ Ou bien que je garde le CRUD brut comme ici ?

Ainsi, je peux te produire un projet Spring Boot complet et professionnel avec tout configuré.