

SPRING

(COOK BOOK)

Ce cook book décrit les étapes pour créer un projet Spring :

- embarquant un serveur Tomcat qui écoute sur le port 8280
- embarquant une base de données H2 en mémoire nommée calendrier_gif
- utilisant des JSPs pour le front avec la bibliothèque JSTL : <https://jakarta.ee/specifications/tags/>
- basé sur le modèle en 5 couches comme suit :

Classes Métier (ce que le client connaît) Package : fr.asl.calendrier_gif.business Exemples : classes Emotion, Jour,Reaction,Theme,Utilisateur, Gif, GifDistant, GifTeleverse @Entity	Vues Dossier : src/main/webapp/WEB-INF Exemple : Index.jsp
	Contrôleurs @Controller Package : fr.asl.calendrier_gif.controller Exemple : Classes GifController
	Service @Service Package : fr.asl.calendrier_gif.service Exemple : Classes GifService
	DAO @Repository Package : fr.asl.calendrier_gif.dao Exemple : Classes GifDao

Couche de Persistance

(Base de données Relationnelle)

Exemples : MariaDB, MySQL, Oracle, SQL Server, PostgreSQL

SPRING

(COOK BOOK)

Avec ce modèle en 5 couches :

- les dépendances entre les couches se matérialisent par des directives d'import
- la programmation par contrat est mise en œuvre pour réduire le couplage : la communication entre les couches se base sur la notion d'interface.

Exemple : le contrôleur *UtilisateurController* a besoin d'un service *UtilisateurService*, il déclare cette dépendance en faisant référence à l'interface et non à l'implémentation de service (*UtilisateurServiceImpl*) :

```
@Controller
@AllArgsConstructor
public class UtilisateurController {

    // Attributs (toujours au début de la classe)
    private final ThemeService themeService;
    private final UtilisateurService utilisateurService;
    private final HttpSession httpSession;
```

1) Générer un projet à l'aide de Spring Initializer : <https://start.spring.io>

 **spring** initializr

Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Maven

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.0.2 (SNAPSHOT) ☒ 3.0.1 ☐ 2.7.8 (SNAPSHOT) ☐ 2.7.7

Project Metadata

Group

fr.m2i

Artifact

calendrier

Name

calendrier

Description

Package name

fr.m2i.calendrier

Packaging

☒ Jar ☐ War

Java

☐ 19 ☒ 17 ☐ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

MariaDB Driver

SQL

MariaDB JDBC and R2DBC driver.

Spring Data JPA

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Spring Boot DevTools

DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Validation

MG

Bean Validation with Hibernate validator.

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Rest Repositories

WEB

Exposing Spring Data repositories over REST via Spring Data REST.

Spring Boot Actuator

OPS

Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

Lombok

DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

H2 Database

SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

SPRING

(COOK BOOK)

A savoir :

- Ce site évolue aussi vite que Spring Boot :
<https://github.com/spring-projects/spring-boot/milestones>

En sélectionnant Maven, le nouveau projet aura pour parent le projet spring-boot-starter-parent. La version de chaque dépendance (du projet parent) est précisée sur cette page :

<https://docs.spring.io/spring-boot/docs/current/reference/html/dependency-versions.html>

Alternativement, pour générer le projet Maven :

- avec **Eclipse** et le **plugin Spring Tool Suite** :
File / New / Spring Boot / Spring Starter Project
- avec **IntelliJ** et les **plugins Spring** et **Spring Boot** :
File / New / Project / Spring Initializr

DEPENDANCES :

- **MariaDB Driver & H2 Database** : Drivers JDBC à sélectionner SI le projet utilise une base de données. (H2 est une base de données embarquée)

- **Spring Web** : permet à l'application d'embarquer un serveur Tomcat géré par Spring.

- **Spring Data JPA** : permet d'obtenir les classes persistantes, autrement dit annotées avec @Entity de JPA (Jakarta Persistence API : <https://jakarta.ee/specifications/persistence/2.2/>)

- **Spring Boot DevTools** : permet à l'application de redémarrer dès qu'elle détecte un changement dans les fichiers du projet.

- **Validation** : permet d'utiliser les annotations de validation (@NotNull, @NotBlank, @Size..)

- **Lombok** : permet d'utiliser les annotations pour laisser Spring gérer la construction de la classe (@Setter, @Getter, @NoArgsConstructor, @ToString..)

- **Rest Repositories** : permet l'utilisation de Spring DATA REST (facilite la création de services Web REST avec les référentiels Spring DATA)

- **Spring Boot Actuator** : permet d'obtenir des mesures de santé et de surveillance à partir d'applications prêtes pour la production.

Cliquer sur **Generate**

2) Décompresser le fichier .zip dans votre workspace

3) Dans Eclipse, importer le projet Maven en utilisant l'assistant :
File / Import / Existing Maven Projects

SPRING

(COOK BOOK)

4) Ajouter dans le fichier pom.xml, au niveau de la balise dependencies, les trois balises suivantes :

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
  <version>8.0.33</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
<dependency>
  <groupId>jakarta.servlet.jsp.jstl</groupId>
  <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
  <version>3.0.0</version>
</dependency>
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>jakarta.servlet.jsp.jstl</artifactId>
  <version>3.0.1</version>
</dependency>
```

5) Ajouter dans le fichier *src/main/resources/application.properties* les lignes suivantes :

```
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true

spring.mvc.view.suffix=.jsp
spring.mvc.view.prefix=/WEB-INF/
```

A) Pour exposer les **beans** présents dans le conteneur de Spring :

```
management.endpoint.info.enabled=true
management.endpoints.web.base-path=/
management.endpoints.web.exposure.include=beans
```

Ce faisant la liste des beans contenus dans le conteneur de Spring sera accessible par l'URL : <http://localhost:8080/beans>

B) Pour demander à Spring la gestion d'un fichier de journalisation :

```
logging.level.root=INFO
logging.level.org.springframework=INFO
logging.file.name=log/calendrier_gif_log
logging.pattern.console= %d %p %c{1.} [%t] %m%n
```

Lien vers la document de Log4J concernant les patterns :

<https://logging.apache.org/log4j/2.x/manual/layouts.html>

Par défaut Spring crée un nouveau fichier pour chaque jour ou dès que le fichier actuel de log atteint 10.5 Mo.

SPRING

(COOK BOOK)

Se référer à la documentation officielle pour bien comprendre comment modifier le fichier `application.properties` :

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

OPTIONS DE BDD :

1) Pour une base **H2 en mémoire**, le fichier de configuration doit également inclure :

```
spring.datasource.url=jdbc:h2:mem:calendrier_gif  
spring.datasource.driver-class-name=org.h2.Driver
```

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
```

Attention :

Si une base H2 est utilisée en production, vérifier :

- que la balise `<scope>test</scope>` n'est pas présente dans la dépendance vers H2 du fichier `pom.xml`.
- que la propriété `spring.h2.console.enabled` est bien définie à vrai :

```
spring.h2.console.enabled=true
```

Ce faisant, la console H2 sera accessible à cette URL :
`http://localhost:8080/h2-console`

Voici la page de connexion de la console H2 :

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:enquetes

User Name: sa

Password:

Connect Test Connection

Par défaut le username est **sa**. Le champ Password peut être laissé vide.

SPRING

(COOK BOOK)

2) Pour une base **H2 stockée sur le disque dur**, le fichier de configuration doit également inclure :

```
spring.datasource.url=jdbc:h2:~/calendrier_gif
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
```

3) Pour une base **PostgreSQL**, le fichier de configuration doit également inclure :

```
spring.datasource.url=jdbc:postgresql://localhost:5432/calendrier_gif
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.
PostgreSQL95Dialect
```

4) Pour une base **MySQL** nommée calendrier_gif, le fichier de configuration doit également inclure :

```
spring.datasource.url=jdbc:mysql://localhost:3306/calendrier_gif
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

6) Créer un package business :

6.1) Écrire / Générer les classes business :

6.1.1) Soit en **Code First** :

Pour chaque classes métier :

1 – ajouter un **constructeur vide**, sinon on obtient l'exception :

[org.hibernate.InstantiationException: No default constructor](#)

2 – ajouter les **getter**(*accesseur*) et **setter**(*mutateur*) pour chaque attribut privé

3 – une méthode **toString()** : Spring va se servir de cette méthode pour générer les formulaires HTML utilisant les balises <form:form> et donner à chaque élément du formulaire la bonne valeur par défaut

4 – Annoter les classes business avec les annotations Hibernate (se référer au mémento Annotations)

Exemple :

@Entity

```
public class Emotion {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
    @NotBlank(message = "Merci de donner un nom à l'émotion")
```

```
    private String nom;
```

```
    private String code;
```

```
    public Emotion(String nom, String code) {
```

```
        super();
```

```
        this.nom = nom;
```

```
        this.code = code;
```

```
    }
```

```
}
```

SPRING

(COOK BOOK)

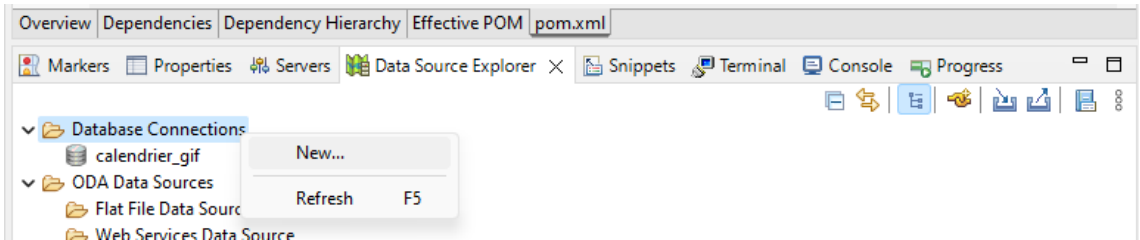
Lombok peut être utilisé pour ne plus avoir à écrire *les constructeurs, les getters, les setters, la méthode hashCode, equals ainsi que toString* :

```
@Entity
@NoArgsConstructor
@Getter
@Setter
@ToString
public class Emotion {

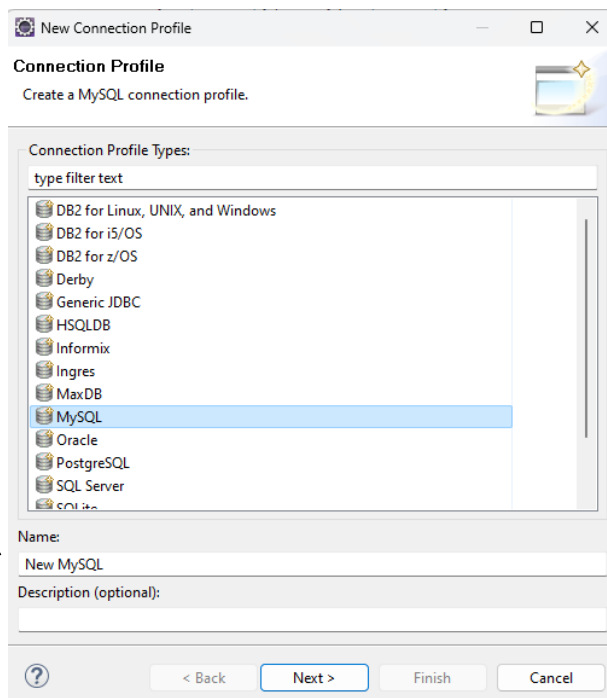
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    [...]
    public Emotion(String nom, String code) {
        super();
        this.nom = nom;
        this.code = code;
    }
}
```

6.2) Soit en **Database First** :

- 1 – concevoir les tables avec MySQL Workbench, Jmerise ou looping
- 2 – ajouter une connexion à la base de donnée

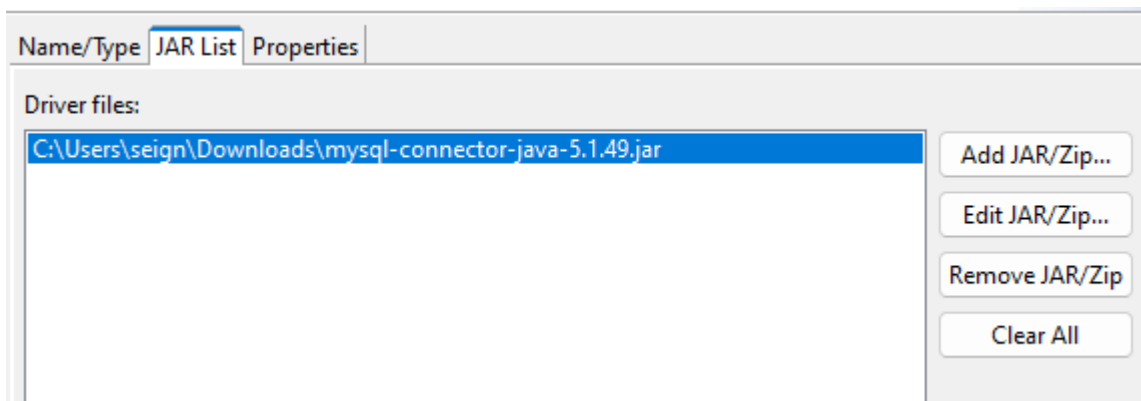
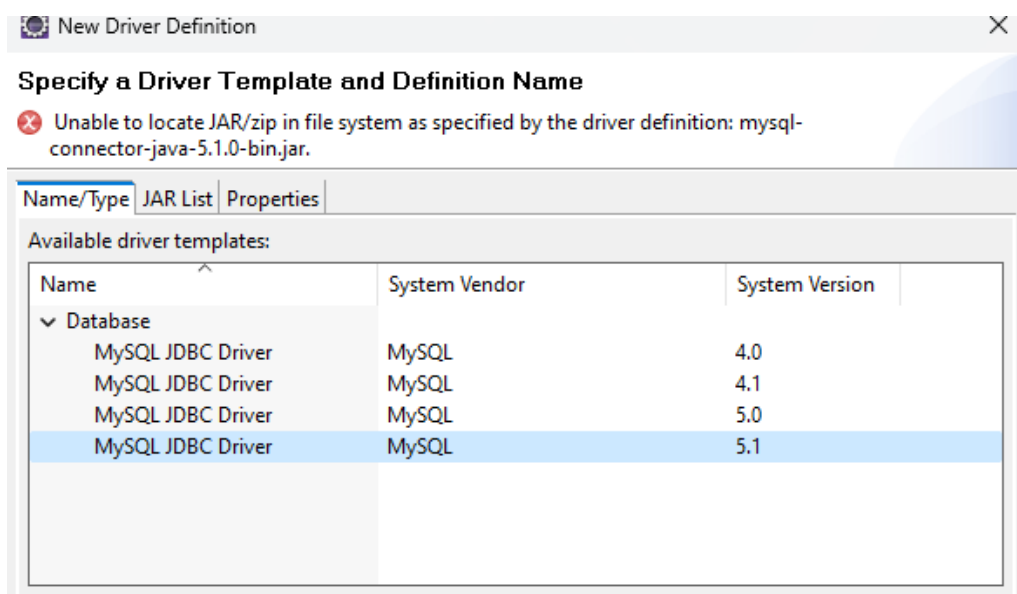
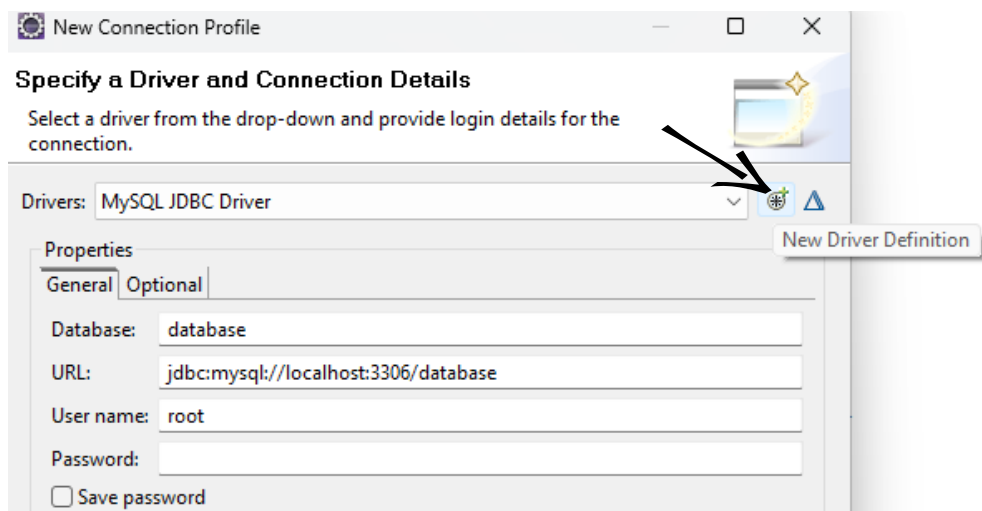


Ici nommer la connexion
puis
Next >



SPRING

(COOK BOOK)



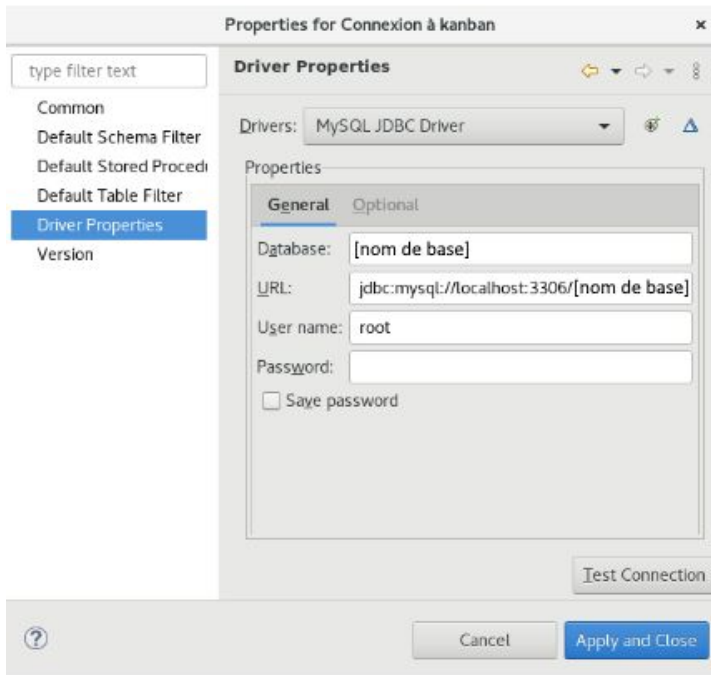
Enfin selectionner le Jar téléchargé ajouter le avec Add JAR puis faite OK

Apply and Close

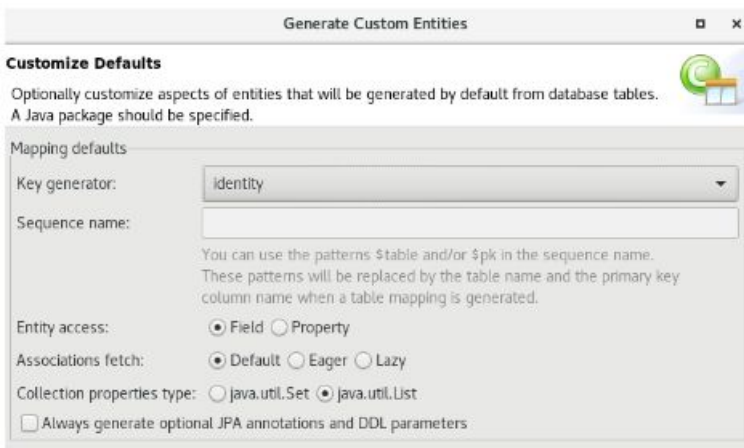
Puis un double clic sur la connection nouvellement créé

SPRING

(COOK BOOK)



- 3 – Clic droit sur le projet : *configure / convert to JPA project*
- 4 – JPA est coché, puis Next
- 5 – Choisir *Generic 2.1*, disable *user library*, puis cliquer sur *Finish*
- 6 – Clic droit sur le projet *JPA Tools / Generate Entities from Tables*
- 7 – Sélectionner les tables, cliquer sur Next 2 fois
- 8 – Sur la fenêtre « *Customize Defaults* » choisir *identity* comme Key generator et comme package le dossier *business*



Alternativement, les classes métier peuvent être générées en choisissant *File/ New / JPA Entities from Tables*

SPRING

(COOK BOOK)

7) Générer le diagramme de classes métier avec reverse engineer de StarUML et placer le fichier .mdj ainsi qu'une version PNG du diagramme dans un dossier nommé doc.

8) Écrire les interfaces DAO en sélectionnant dans le File / New / Interface. Chaque interface hérite de JpaRepository :

The screenshot shows the 'New Java Interface' dialog box. The title bar says 'New Java Interface'. Below the title bar, it says 'Java Interface' and 'Create a new Java interface.' There is a purple 'I' icon in the top right corner. The dialog has several input fields and buttons:

- Source folder:** A text field containing 'calendrier_gif_m2i_spring_hibernate/src/main/java' and a 'Browse...' button.
- Package:** A text field containing 'fr.m2i.asl.calendrier_gif.dao' and a 'Browse...' button.
- Enclosing type:** A text field containing 'fr.m2i.asl.calendrier_gif.dao.EmotionDao' and a 'Browse...' button.
- Name:** A text field containing 'EmotionDao'.
- Modifiers:** Radio buttons for 'public' (selected), 'package', 'private', and 'protected'.
- Extended interfaces:** A list box containing 'org.springframework.data.jpa.repository.JpaRepository<Emotion, long>'. There are 'Add...' and 'Remove' buttons next to it.
- Do you want to add comments?** A checkbox labeled 'Generate comments'.
- Buttons:** 'Finish' and 'Cancel' buttons at the bottom right.

Exemple :

```
public interface EmotionDao extends JpaRepository<Emotion, Long> {
```

Javadoc des interfaces Repository de Spring Data JPA :

<https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>

L'interface JpaRepository hérite de l'interface PagingAndSortingRepository :

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html>

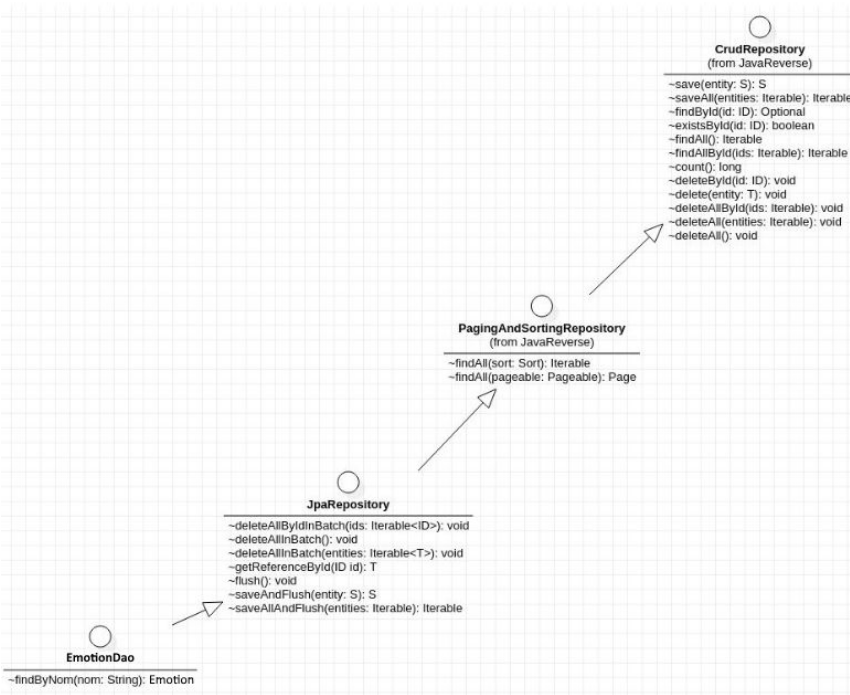
L'interface PagingAndSortingRepository hérite de l'interface CrudRepository :

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>

SPRING

(COOK BOOK)

Voici un diagramme de classes présentant l'héritage entre les 4 interfaces :



Dans chaque interface du package dao, des méthodes annotées `@Query` ou des méthodes requêtes peuvent être déclarées.

Par défaut l'annotation `@Query` attend une **requête HQL** :

```
@Query("FROM Emotion WHERE nom LIKE 's%'")
List<Emotion> findEmotionsHavingNameStartingWithS();
```

Se référer à la documentation officielle pour rédiger la requête HQL :

https://docs.jboss.org/hibernate/stable/orm/userguide/html_single/Hibernate_User_Guide.html

L'annotation `@Query` peut aussi accueillir une requête SQL grâce à l'attribut `nativeQuery`

```
@Query(value="SELECT * FROM Enquete WHERE theme_id=:idTheme",
nativeQuery=true)
List<Enquete> findByIdTheme(@Param("idTheme") Long idTheme);
```

Une alternative à l'annotation `@Query` est l'écriture de requête par dérivation, en anglais « query-method ». Le nom de la méthode est interprété par Spring Data et traduit en HQL.
Exemple :

```
List<Enquete> findByTheme(Theme theme);
```

Les mots clés autorisés dans le nom des méthodes sont résumés sur la table suivante :

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation>

SPRING

(COOK BOOK)

9) Écrire les packages service.

9.1) Écrire les classes de Service. Annoter chaque classe service avec le stéréotype Spring `@Service` et demander à Spring l'injection des DAO par l'écriture d'un constructeur ayant en paramètre les objets que Spring doit injecter dans le service.

Exemple :

```
@Service
public class JourServiceImpl implements JourService {

    private JourDao jourDao;

    private static Random random;

    public JourServiceImpl(JourDao jourDao) {
        this.jourDao = jourDao;
        random = new Random();
    }
    [...]
}
```

10) Écrire le ou les contrôleurs Spring.

Annoter chaque classe **contrôleur** avec `@Controller` ou `@RestController`.

10.1) (manière dépréciée) Injecter des objets de type Service dans les contrôleurs avec l'annotation `@Autowired`.

NB : Chaque objet de type **Service** doit être **annoté** `@Autowired`.

Exemple :

```
@Controller
public class EnqueteController {

    @Autowired
    private EnqueteService enqueteService;

    @Autowired
    private QuestionService questionService;
```

10.2) (manière moderne, à préférer) Ajouter un **constructeur** dans le **contrôleur** avec en **paramètre tous les objets que Spring doit injecter** dans le contrôleur.

Exemple :

```
@Controller
public class EnqueteController {

    private final EnqueteService enqueteService;
    private final QuestionService questionService;

    public EnqueteController(EnqueteService enqueteService,
        QuestionService questionService) {
        super();
        this.enqueteService = enqueteService;
        this.questionService = questionService;
    }
}
```

SPRING

(COOK BOOK)

10.3) (manière encore plus moderne, à préférer) Ajouter l'annotation **@AllArgsConstructor** de **Lombok** qui **va ajouter à la volée un constructeur** avec en paramètre tous les objets que Spring doit injecter dans le contrôleur.

Exemple :

```
@Controller
@AllArgsConstructor
public class UtilisateurController {

    // Attributs (toujours au début de la classe)
    private final ThemeService themeService;
    private final UtilisateurService utilisateurService;
    private final HttpSession httpSession;

    [...]

}
```

10.4) Ajouter les **méthodes nécessaires** pour traiter toutes les **requêtes HTTP** (chacune de ces méthodes doit renvoyer un **objet de type ModelAndView**), ajouter si besoin la méthode annotée **@PostConstruct** :

Exemple :

```
[...]
@GetMapping("/inscription")
public ModelAndView inscriptionGet(@ModelAttribute
Utilisateur utilisateur) {
    ModelAndView mav = new ModelAndView("inscription");
    mav.addObject("themes",
themeService.recupererThemes());
    return mav;
}

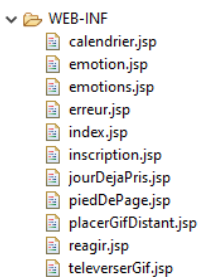
[...]

@GetMapping("calendrier/deconnexion")
public ModelAndView deconnexion() {
    httpSession.invalidate();
    return new ModelAndView("redirect:/index");
}

}
```

11) Ajouter un dossier nommé **src/main/webapp/WEB-INF**

12) Écrire les JSPs dans le dossier **src/main/webapp/WEB-INF** en cliquant-droit sur ce dossier, New / JSP File. En plaçant les JSPs dans ce dossier, elles ne sont pas accessibles publiquement, seuls les contrôleurs peuvent les utiliser.



SPRING

(COOK BOOK)

Pour éficer des balises JSTL (Java Standard Tag Library) ajouter la directive suivante en ligne 3 de la JSP :

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
```

Grâce à cette bibliothèque de balises, l'écriture de la vue est simplifiée :

- `<c:if>` pour ajouter un branchement.
Exemple : on teste si l'attribut erreur n'est pas nul. Dans l'affirmative on affiche cet attribut dans une balise `h2` :

```
<c:if test="${param.notification ne null}"><h2>${param.notification}</h2></c:if>
```

NB : le mot clé **ne** signifie « not equals ». Il rend la condition du test plus lisible. Il existe également : **eq**, **gt**, **ls**, **ge** et **le**.

- `<c:forEach>` pour parcourir une collection. Cette balise doit obligatoirement avoir deux attributs `items` et `var`. L'attribut `items` correspond à la collection à parcourir. L'attribut `var` correspond au nom de la variable de boucle i.e. la variable locale à la boucle.

Exemple 1 :

```
<ul>
  <c:forEach items="${utilisateurs}" var="utilisateur">
    <li>${utilisateur.prenom}</li>
  </c:forEach>
</ul>
```

Exemple 2 :

```
<select name="ID_EMOTION" required>
  <option value="">Merci de choisir une émotion</option>
  <c:forEach items="${emotions}" var="emotion">
    <option value="${emotion.id}">${emotion.nom}</option>
  </c:forEach>
</select>
```

- `<c:choose>`, `<c:when>` et `<c:otherwise>` pour les branchements complexes

Exemple :

```
<c:choose>
  <c:when test="${jour.gif eq null}">
    <td colspan="3">${jour.nbPoints} points<br>
      <a href="calendrier/placerGifDistant?ID_JOUR=${jour.date}">Placer
un Gif distant</a><br>
      <a href="calendrier/televerserGif?ID_JOUR=${jour.date}">Téléverser
un Gif</a>
    </td>
  </c:when>
  <c:when test="${jour.gif ne null}">
    <td>
      <c:if test="${jour.gif.legende ne null && jour.gif.legende ne
''}"><h2>${jour.gif.legende}</h2></c:if>
      <c:if test="${jour.gif.getClass().getSimpleName eq 'GifDistant'}"></c:if>
      <c:if test="${jour.gif.getClass().getSimpleName eq
'GifTeleverse'}"></c:if>
    </td>
    <td>${jour.gif.utilisateur.prenom}</td>
    <td>
      <c:forEach items="${jour.gif.reactions}" var="reaction">
        ${reaction.emotion.code} ${reaction.utilisateur.prenom}<br>
      </c:forEach>
      <a href="calendrier/reagir?ID_GIF=${jour.gif.id}">Réagir</a></td>
    </c:when>
  </c:choose>
```

SPRING

(COOK BOOK)

- `<c:set>` pour définir des variables locales

Exemple :

```
<c:set var="msFin" value="${dateFin.getTime()}" scope="page" />
```

13) Ajouter un fichier `application.properties` dans un nouveau dossier `src/test/resources`

Ce fichier peut contenir des propriétés qui configurent une base H2 en mémoire :

```
spring.datasource.url=jdbc:h2:mem:calendrier_gif_test
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
```

Cette configuration servira exclusivement pendant la phase de tests.

Sans la présence du fichier `src/test/resources/application.properties`, Maven utilisera le fichier `src/main/resources/application.properties` pour lancer les tests.

14) Écrire les tests dans le dossier `src/test/java`

Le package `dao` accueillera les tests sur les interfaces DAO. Ces classes de test pourront être annotées `@SpringBootTest` ou `@DataJpaTest`.

Le package `service` accueillera les tests sur les classes de la couche service. Ces classes de test seront annotées `@SpringBootTest`.

Le package `controller` accueillera les tests sur les classes de la couche controller. Ces classes de test seront annotées `@SpringBootTest`.

15) Générer la documentation de l'API et la page Swagger-UI

<https://swagger.io/tools/swagger-ui/>

Ajouter dans le fichier `pom.xml`, au niveau de la balise `dependencies`, la balise suivante :

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.6</version>
</dependency>
```

Redémarrer l'application Spring Boot manuellement. La documentation Swagger est désormais accessible à partir de l'URL suivante :

<http://localhost:8080/swagger-ui/index.html>

Cette page présente les ressources de l'API de manière très lisible :

PUT	/api/villes/{id}/{nom}	▼
GET	/api/villes	▼
POST	/api/villes	▼
POST	/api/villes/{nom}	▼
GET	/api/villes/{id}	▼
DELETE	/api/villes/{id}	▼

SPRING

(COOK BOOK)

16) Confier la génération de la Javadoc et des informations sur le projet Maven en ajoutant dans le fichier pom.xml la balise reporting :

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</
artifactId>
      <version>2.6</version>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.3.2</version>
    </plugin>
  </plugins>
  <outputDirectory>doc</outputDirectory>
</reporting>
```

Puis lancer maven site.

17) Lancer l'application avec le goal Maven : ./mvnw spring-boot-run

Annexes :

A) Pour obtenir un script de création/suppression des tables en base, il est nécessaire d'ajouter ces deux lignes dans le fichier src/main/resources/application.properties :

```
spring.jpa.properties.java.persistence.schema-generation.scripts.action=create
spring.jpa.properties.java.persistence.schema-generation.scripts.create-target
=src/main/resources/script.sql
```

A noter : si ces lignes sont présentes dans le fichier de configuration, les ordres de création de tables ne seront plus envoyés à la base par Spring.

B) Pour modifier la stratégie de nommage des tables et des colonnes en base, il suffit d'ajouter la ligne suivante :

```
spring.jpa.hibernate.naming.physical-strategy
=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

Avec cette stratégie de nommage :

- les noms de tables auront une majuscule à chaque mot : exemple : TypeClient
- le nom des colonnes sera identique au nom des attributs de la classe : dateHeureCreation

C) Pour autoriser le téléversement de fichiers (dont la taille est supérieure à 1 Mo) sur le serveur, ces deux lignes sont indispensables :

```
spring.servlet.multipart.max-file=8MB
spring.servlet.multipart.max-request-size=10MB
```

D) Pour modifier le nombre de connexions créées entre l'application et la base (via le connection pool Hikari présent par défaut dans l'application Spring Boot), il faut écrire :

```
spring.datasource.hikari.maximum-pool-size=15
```


SPRING

(COOK BOOK)

E) Pour intégrer Spring Security au projet, ajouter la dépendance associée :

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Security

SECURITY

Highly customizable authentication and access-control framework for Spring applications.

Le fichier pom.xml contiendra la dépendance ci-dessous :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Dans la classe exécutable ajouter un bean chargé de chiffrer les mots de passe avec Bcrypt :

```
@Bean
public PasswordEncoder encoder() {
    return new BCryptPasswordEncoder();
}
```

Un des services doit implémenter l'interface UserDetailsService :

<https://docs.spring.io/spring-security/site/docs/current/api/org.springframework.security.core.userdetails.UserDetailsService.html>

Ajouter une classe de configuration qui hérite de WebSecurityConfigurerAdapter :

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    private final UserDetailsService userDetailsService;
    private final PasswordEncoder passwordEncoder;

    public SecurityConfig(UserDetailsService userDetailsService,
        PasswordEncoder passwordEncoder) {
        this.userDetailsService = userDetailsService;
        this.passwordEncoder = passwordEncoder;
    }

    // Authentification utilisant la DAO
    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder);

        return authProvider;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder
        authenticationManagerBuilder) throws Exception {
        authenticationManagerBuilder.userDetailsService(userDetailsService);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf()
            .disable()
            .formLogin()
            .loginPage("/index")
            .loginProcessingUrl("/login")
            .defaultSuccessUrl("/enquetes")
            .permitAll();
    }
}
```

SPRING

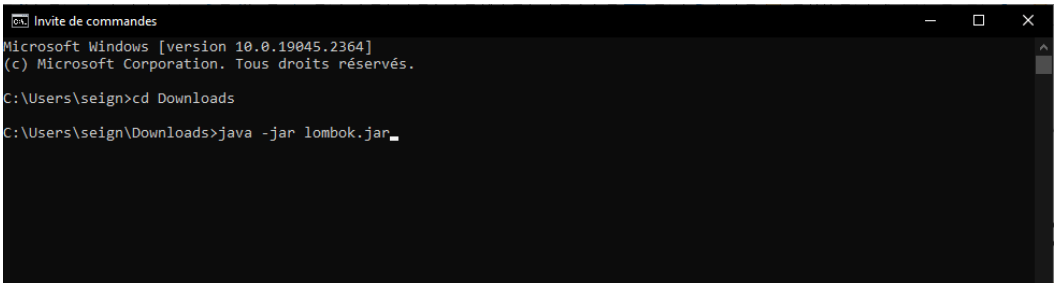
(COOK BOOK)

F) Pour que Spring lance des tâches programmées, la classe contenant la méthode main doit être annotée `@EnableScheduling`. Chaque méthode que Spring doit invoquer automatiquement doit être annotée `@Scheduled`

G) Pour modifier le port sur lequel le serveur Tomcat écoute :

```
server.port=8280
```

H) Pour intégrer Lombok dans l'IDE. Quitter l'IDE puis exécuter le jar de lombok disponible dans `.m2/repository/org/projectlombok/lombok/1.18.24/` : `java -jar lombok.jar`



```
Microsoft Windows [version 10.0.19045.2364]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\seign>cd Downloads
C:\Users\seign\Downloads>java -jar lombok.jar
```



Si Lombok ne détecte pas l'IDE.

Relancer l'IDE.

Le chemin où Eclipse est installé est précisé sur Help / About Eclipse / Installation Details.

SPRING

(COOK BOOK)

I) Exemple de fichier application.properties

```
server.port=8280

spring.datasource.url=jdbc:mysql://localhost:3306/calendrier_gif?
useSSL=false
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Diale
ct
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true

management.endpoint.info.enabled=true
management.endpoints.web.base-path=/
management.endpoints.web.exposure.include=beans

spring.mvc.view.suffix=.jsp
spring.mvc.view.prefix=/WEB-INF/

logging.level.root=INFO
logging.level.org.springframework=INFO
logging.file.name=log/calendrier_gif_log
logging.pattern.console= %d %p %c{1.} [%t] %m%n

spring.data.rest.detection-strategy=annotated

server.error.path=/erreur

spring.servlet.multipart.max-file-size=8MB
spring.servlet.multipart.max-request-size=10MB
```

Sources :

<https://projectlombok.org>

<https://spring.io/>

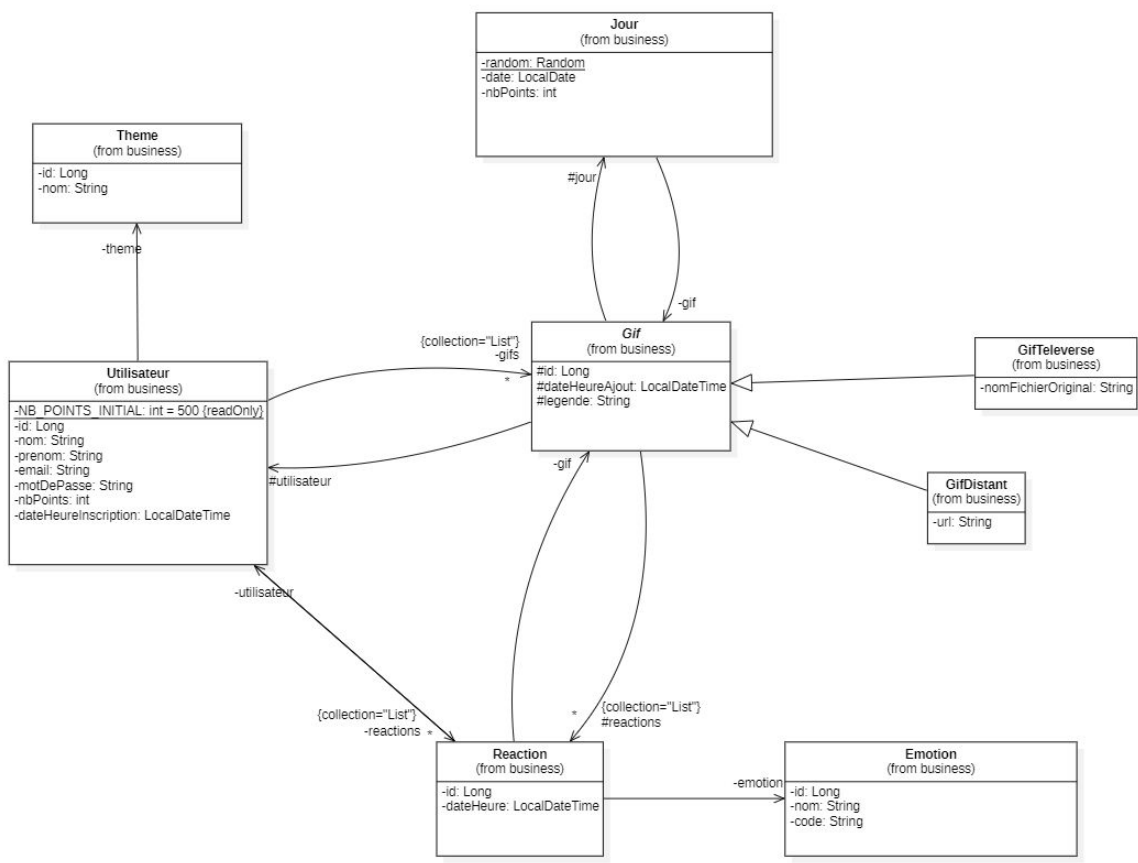
<https://www.clelia.fr/formations.html>

Francois Xavier Cote

SPRING

(COOK BOOK)

UML Business



SPRING

(COOK BOOK)

UML Dao


GifDao
(from dao)

~findTopByReactions(): Gif[*]
~findByUtilisateur(utilisateur: Utilisateur): Gif[*]
~findByUtilisateurId(id: Long): Gif[*]
~findByUtilisateurNom(nom: String): Gif[*]
~findLast1ByJour(jour: Jour): Gif
~findByLegendeContaining(legende: String): Gif[*]


ThemeDao
(from dao)

~findByNom(nom: String): Theme
~findByNomStartingWith(debutNomTheme: String): Theme[*]
~findAllOrderedRandomly(): Theme[*]
~findByNomHQL(nom: String): Theme


GifTeleverseDao
(from dao)


GifDistantDao
(from dao)

~deleteByUrlLike(source: String): long


EmotionDao
(from dao)

~findFirst2ByNomContaining(nom: String): Emotion[*]
~findEmotionsHavingNameStartingWithS(): Emotion[*]
~findByNom(nom: String): Emotion