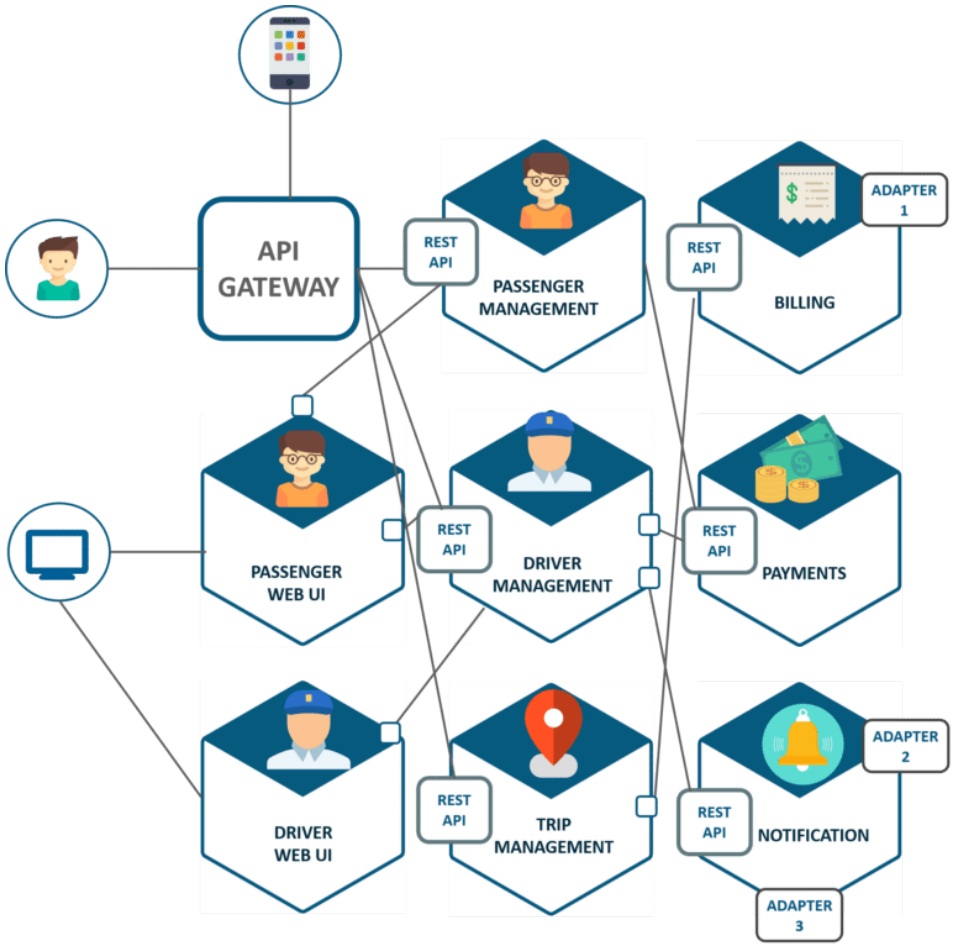
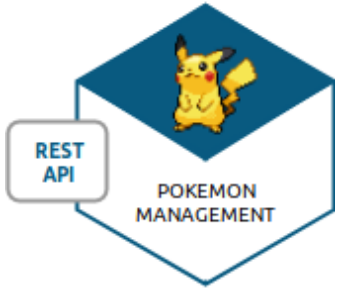
ALOM

 PERSISTANCE DE DONNÉES

UBER


API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
BILLING
ADAPTER
1
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
NOTIFICATION
ADAPTER
2
ADAPTER
3

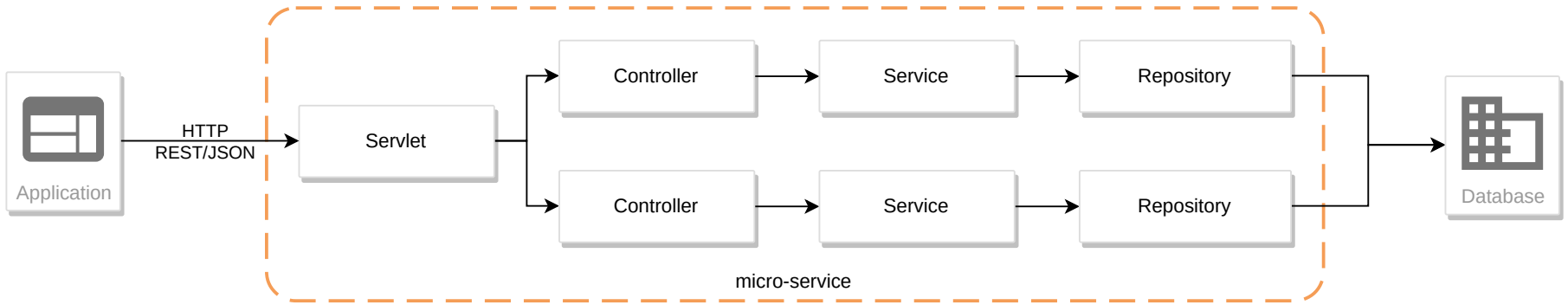


UN MICRO-SERVICE C'EST :

- Un ensemble de fonctionnalités du même domaine métier
- Un ou plusieurs canaux de communication
  - HTTP - REST/JSON
- Une source de données dédiée

🔎 UN MICRO-SERVICE JAVA

On s'appuie sur les technologies connues: les servlets !



🔎 LA SOURCE DE DONNÉES

Une application :

- fournit un service
- manipule des données

Problématique :

- comment enregistrer des données ?
- comment accéder aux données enregistrées ?

DESIGN BEST PRACTICES

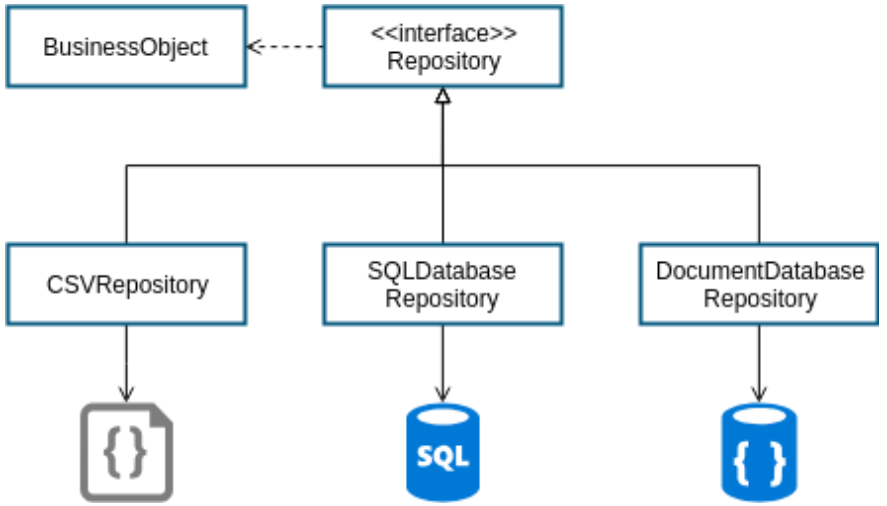
 S.O.L.I.D

DESIGN PATTERN

INJECTION DE DÉPENDANCE

DESIGN PATTERN

DAO - DATA ACCESS OBJECT / REPOSITORY



🔎 LA SOURCE DE DONNÉES

- des fichiers
- une API
- une base de données SQL
- une base de données NoSQL

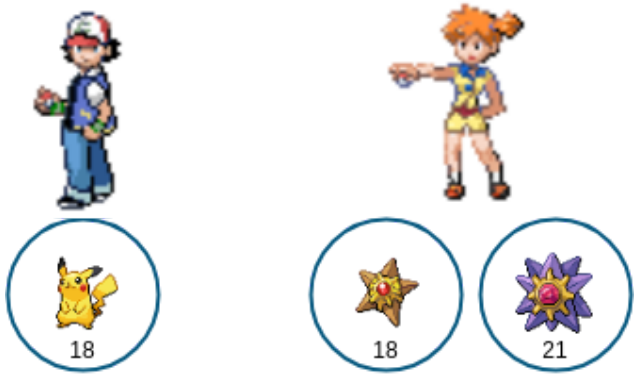
BASES DE DONNÉES SQL

Modèle relationnel (SQL)

- Stockage de l'information en tables
- Relations entre les tables (clés étrangères)
- Requêtage dans un langage dédié : SQL

BASES DE DONNÉES SQL

Comment lier le modèle BDD relationnel et le modèle objet Java?



id	name	sprite
25	pikachu	pikachu.png
120	staryu	staryu.png
121	starmie	starmie.png

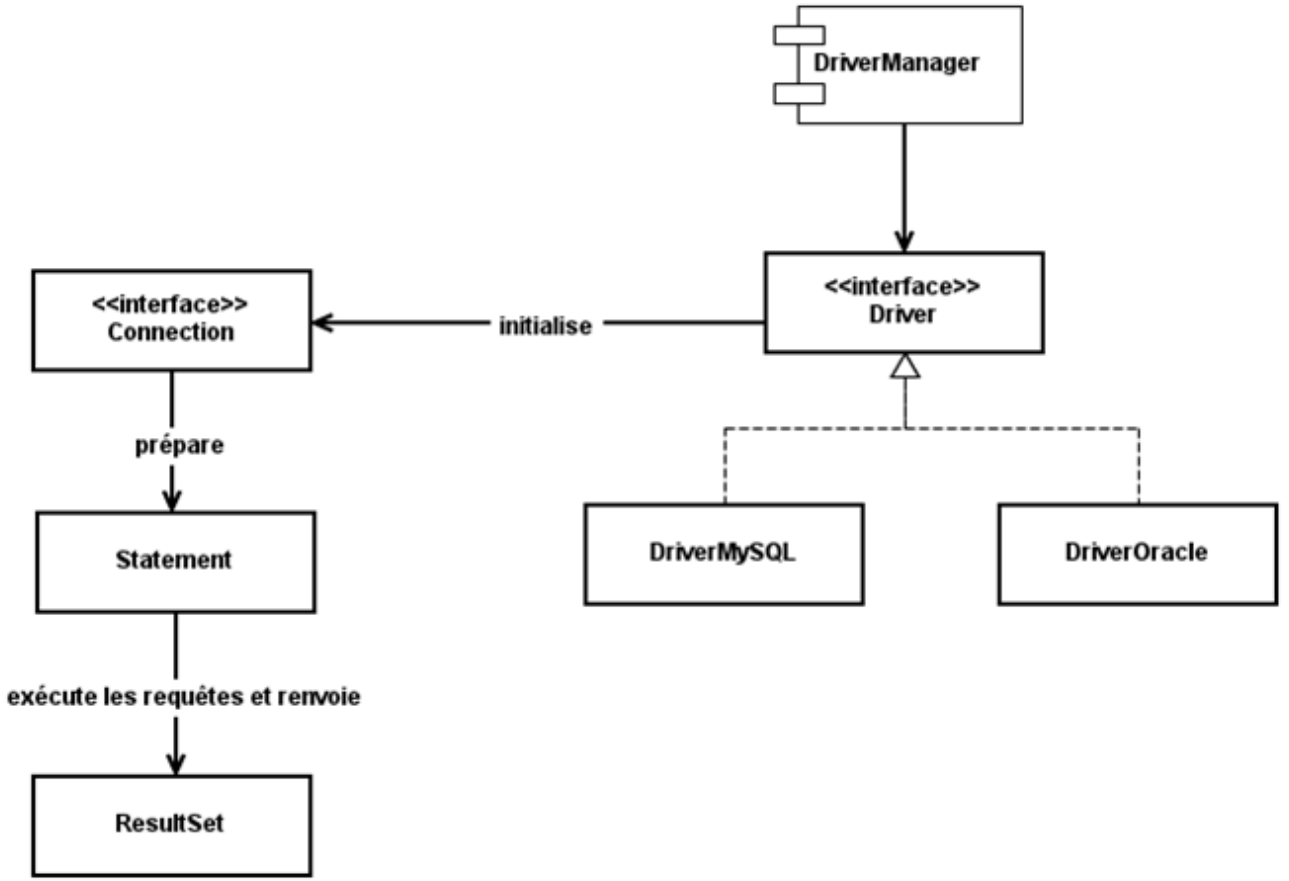
id	name
1	ash
2	misty

trainer_id	pokemon_type	level
1	25	18
2	120	18
2	121	21

À LA DURE : JDBC

```java
public List<Trainer> findAll() throws Exception{
    Class.forName("org.h2.Driver");
    var connection = DriverManager.getConnection("jdbc:h2:mem:test");
    var statement = connection.createStatement();
    var result = statement.executeQuery("select * from trainer");

    var trainers = new ArrayList<Trainer>();
    while(result.next()){
        var id = result.getInt("id");
        var name = result.getString("name");
        trainers.add(new Trainer(id, name));
    }
    statement.close();
    connection.commit();
    connection.close();

    return trainers;
}
```

FONCTIONNEMENT DE JDBC



JDBC AVEC spring by Pivotal : JDBCTEMPLATE

```java
@Repository
public class TrainerJDBCRepositoryImpl implements TrainerRepository{

    private JdbcTemplate jdbcTemplate;

    @Autowired
    public TrainerJDBCRepositoryImpl(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
    @Override
    public List<Trainer> findAll() {
        return jdbcTemplate.query("select * from trainer",  new TrainerRowMapper());
    }
    @Override
    public Trainer save(Trainer trainer) {
        return jdbcTemplate.update("insert into trainer(id,name) values (?,?)", trainer.getId(), trainer.getName());
    }

    static class TrainerRowMapper implements RowMapper<Trainer> {
        @Override
        public Trainer mapRow(ResultSet resultSet, int i) throws SQLException {
            var trainer = new Trainer();
            trainer.setId(resultSet.getInt("id"));
            trainer.setName(resultSet.getString("name"));
            return trainer;
        }
    }
}
```

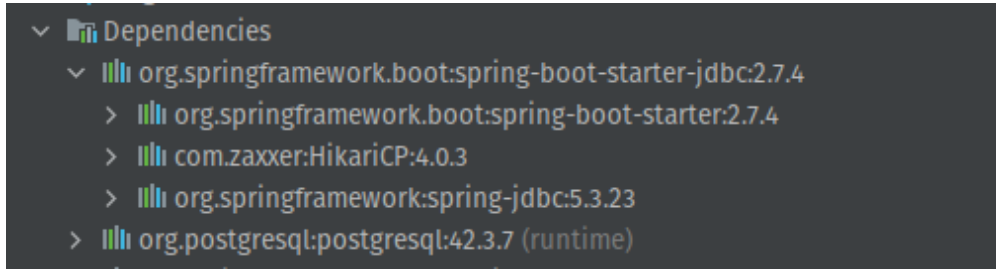

MIAGE Lille

CONFIGURATION SPRING-BOOT

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

+ un driver JDBC

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```

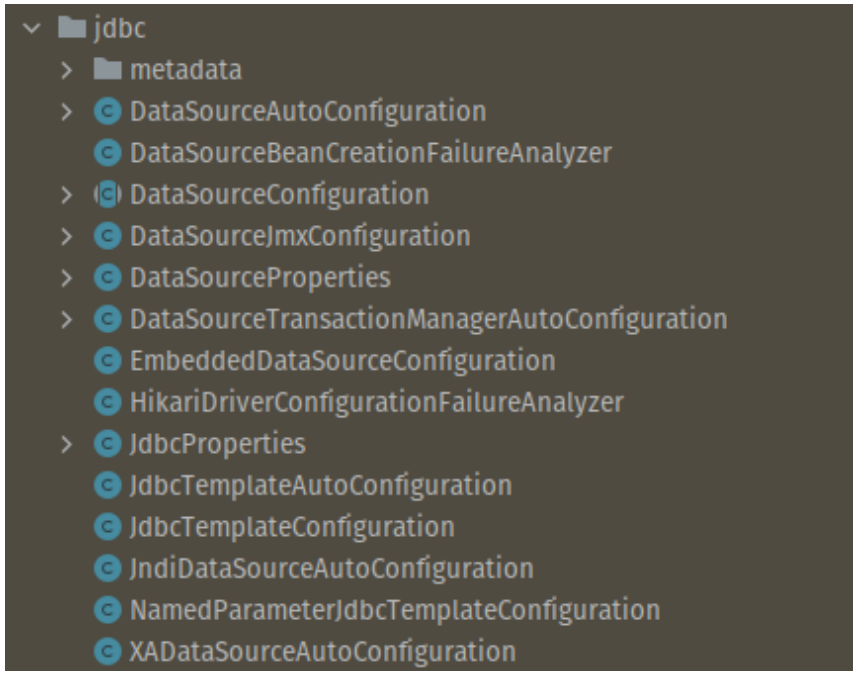


```xml
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```

Les drivers JDBC peuvent être positionnés en *scope* Maven `runtime`.

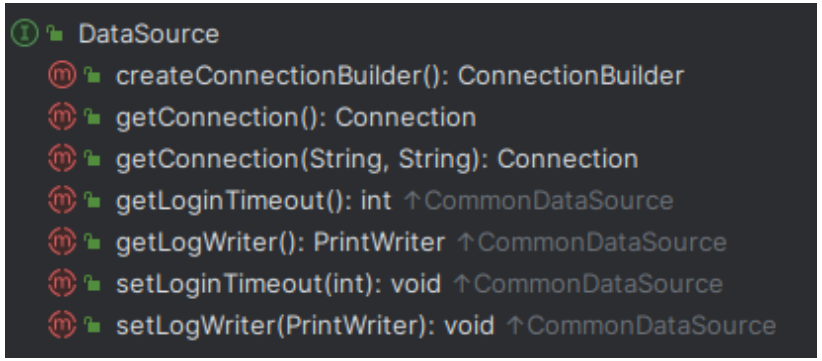
- Ils ne sont pas nécessaire à la compilation du code
- Leur *jar* doit être présent à l'exécution

🔎 JDBC AUTOCONFIGURATION



JAVAX.SQL.DATASOURCE

Interface permettant de récupérer des connections à une base de données.



DATASOURCEPROPERTIES

Permet de configurer la source de données

```
spring.datasource.url
spring.datasource.username
spring.datasource.password
```

DATASOURCEAUTOCONFIGURATION

Configure un *bean* de type `javax.sql.DataSource`.

Si `spring.datasource.url` est renseignée, elle sera utilisée. Sinon, si une base de données embarquée (*h2*, *derby*, ou *hsqldb*) est disponible, elle sera utilisée.

Utilise une implémentation avec un pool de

CAS D'USAGES

Utilisation de la base de données embarquée H2:

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Pas de properties particulières.

CAS D'USAGES

Utilisation d'une base de données postgresql:

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```

Properties avec l'url:

```
spring.datasource.url=jdbc:postgresql://<database_host>:<port>
spring.datasource.username=<user>
spring.datasource.password=<password>
```

JDBCTEMPLATECONFIGURATION

Si un *bean* de type `javax.sql.DataSource` existe,
crée un bean de type
`org.springframework.jdbc.core.JdbcTemplate`.

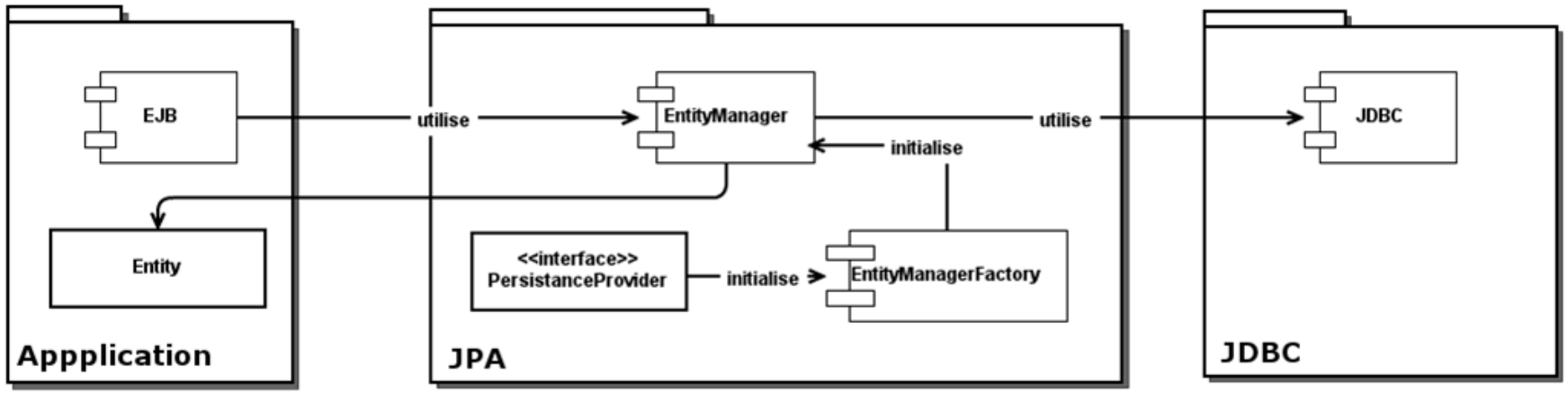
```
 1 @AutoConfiguration(after = DataSourceAutoConfiguration.clas
 2 @ConditionalOnClass({ DataSource.class, JdbcTemplate.class
 3 @ConditionalOnSingleCandidate(DataSource.class)
 4 @EnableConfigurationProperties(JdbcProperties.class)
 5 @Import({ JdbcTemplateConfiguration.class})
 6 public class JdbcTemplateAutoConfiguration {}
 7
 8 @Configuration
 9 @ConditionalOnMissingBean(JdbcTemplate.class)
10 class JdbcTemplateConfiguration {
11
12     @Bean
13     @Primary
14     JdbcTemplate jdbcTemplate(DataSource dataSource, JdbcPr
15         JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSo
```

JPA - JAVA PERSISTENCE API

Surcouche à JDBC - Fournit un moyen de mapper les tables aux objets et une abstraction de l'exécution de requêtes via les annotations

```java
@Entity
public class Trainer {

    @Id
    private int id;

    private String name;

    @OneToMany
    private List<Pokemon> pokemons;

}
```

FONCTIONNEMENT DE JPA



Repository Spring/JPA (mode DAO):

```java
@Repository
public class TrainerJPARepositoryImpl implements TrainerRepository {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public List<Trainer> findAll() {
        Query query = entityManager.createQuery("select t from Trainer t");
        return query.getResultList();
    }

    @Override
    public Trainer save(Trainer trainer) {
        return entityManager.merge(trainer);
    }
}
```

SPRING-DATA JPA

Génération automatique des méthodes au runtime à partir d'interfaces!

```java
@Repository
public interface TrainerRepository extends CrudRepository<Trainer, Integer> {
    List<Trainer> findAll();
    Trainer findById(int id);
    Trainer save(Trainer trainer);
}
```

SPRING-DATA JPA

Configuration Maven

```
<!-- spring-data JPA -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<!-- an embedded database -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

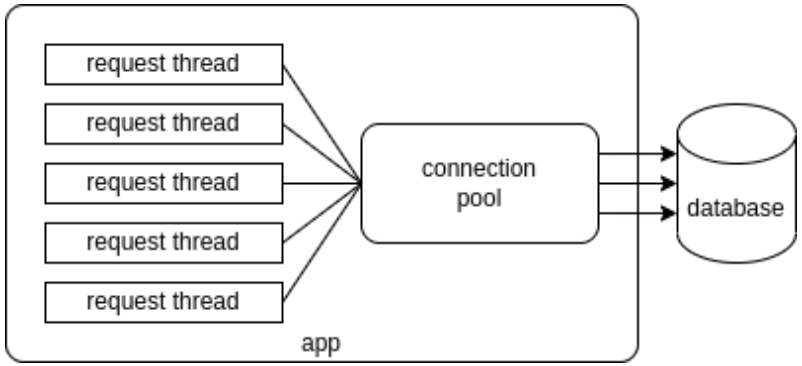
CONNECTION POOLING

⚠️ Ouvrir une connexion à une BDD est coûteux

- Ouverture d'une socket
- Authentification

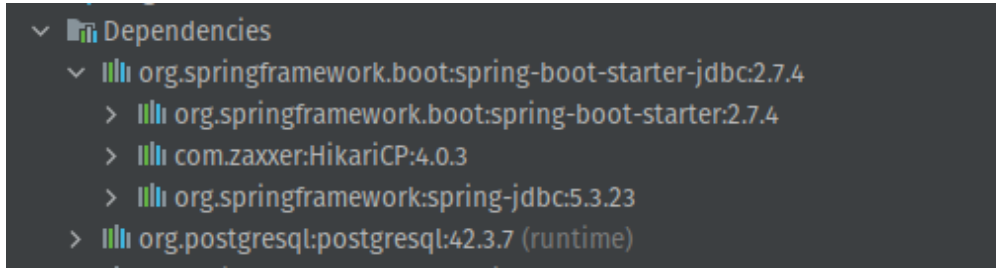
💡 Ouvrir les connexions au démarrage de l'application et maintenir les connexion ouvertes

CONNECTION POOLING



CONNECTION POOLING

Avec `Spring Boot`, utilisation de `HikariCP` :



Possibilité d'utiliser d'autres pools (doc)

CONNECTION POOLING

Configuration du pool de connexions :

Properties `spring.datasource.hikari.*`

```
# nombre de connection conservées au minimum
spring.datasource.hikari.minimum-idle=1

# nombre de connections max
spring.datasource.hikari.maximum-pool-size=10
```

TRANSACTIONS

⚠️ Effectuer plusieurs opérations en même temps

TRANSACTIONS ACID

Garantir la fiabilité d'une transaction en BDD

- Atomicité : Tout ou rien
- Cohérence : L'état base de données est valide à tout instant (contraintes)
- Isolation : Les transactions peuvent être exécutées simultanément
- Durabilité : Une transaction validée survit à une défaillance du système

TRANSACTIONS EN JAVA

```
public interface java.sql.Connection {
    Statement createStatement() throws SQLException;
    void setAutoCommit(boolean autoCommit) throws SQLException
    void commit() throws SQLException;
    void rollback() throws SQLException;
}
```

```
public interface jakarta.transaction.UserTransaction {
    void commit() throws IllegalStateException, SystemExceptio
    void rollback() throws IllegalStateException, SystemExcept
}
```

TRANSACTIONS

Avec `Spring Boot`, utilisation de l'`AOP` et de l'annotation `@Transactional` (`Spring` ou `jakarta`)

```
org.springframework.transaction.annotation.@Transactional

jakarta.transaction.@Transactional
```

Par défaut, tous les appels aux repositories sont dans une transaction :

```
package org.springframework.data.jpa.repository.support;

@Repository
@Transactional
public class SimpleJpaRepository<T, ID> implements JpaReposito
  ...
}
```

TRANSACTIONS

À réfléchir quand on développe :

- Est-ce que je dois gérer plusieurs modifications dans une même transaction ?

⚠️ L'annotation `@Transactional` se pose au niveau de la couche `service`

⚠️ Au niveau de la couche `controller`, risque d'ouvrir des transactions trop tôt (erreurs HTTP), et de consommer des ressources pour rien (connection de pool + commit / rollback)

⚠️ Au niveau de la couche `repository`, risque de

SPRING-DATA MONGODB

Configuration Maven

```
<!-- spring-data mongodb -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

SPRING-DATA MONGODB

- Instanciation d'un `MongoTemplate`
- Configuration du `MongoClient`
- Support des transactions avec `@Transactional`
- Repository interfaces avec `@EnableMongoRepositories`

SPRING-DATA MONGODB CONFIGURATION

Properties

```
spring.data.mongodb.uri=mongodb://user:secret@mongoserver1.exa

# ou

spring.data.mongodb.host=mongoserver1.example.com
spring.data.mongodb.port=27017
spring.data.mongodb.additional-hosts[0]=mongoserver2.example.c
spring.data.mongodb.database=test
spring.data.mongodb.username=user
spring.data.mongodb.password=secret
```

TP

Spring repositories !

FIN DU COURS