Nom de naissance	‣	05/11/1982
Nom d'usage	‣	
Prénom	‣	Davy
Adresse	‣	

Titre professionnel visé

Développeur Logiciel - Niveau III

SOMMAIRE

INTRODUCTION

SIMPLON'CLICK

"Teaching By Teaching"

Nous connaissons la devise des **Concepteurs/Développeurs**: "Google est ton ami !".

Cette notion est nécessaire afin de favoriser l'autonomie de tout développeur (l'un des principes guidant notamment la formation Simplon tel que la débrouillardise, la proactivité, la persévérance, la curiosité, la réflexion et le partage avec la communauté).

Cependant, durant cette formation, le **travail en groupe**, **favorisant** aussi bien la **compréhension** que les **échanges**, est aussi **très valorisé**.

L'humain est remis au coeur de la performance de l'entreprise.

De plus, n'oublions pas que le fait de **partager ses propres connaissances et compétences**, notamment pour des apprenants, **peut conforter**, **compléter** ou encore **corriger** certains **"acquis"** ***("Teaching by teaching")***.

L'échange de connaissance en face à face (direct) est alors **gain de productivité** et **source d'expériences** (en accord avec les attentes des entreprises actuelles de développement).

C'est pourquoi mon idée a été de créer une **application web** (accessible via un PC, une tablette ou un smartphone) **consistant à mettre à profit toutes les ressources humaines** d'une même école (ici Simplon) **en terme de compétences et de connaissances**.

Via des propositions de soutien et des demandes d'aides explicites, tous les acteurs, précisant leur degré de compétence (niveau), leur lieu d'affectation (site) et leurs disponibilités, pourront **favoriser une rencontre** afin d'échanger leurs connaissances et savoir-faire dans un domaine de compétences choisi et ce, en face à face.

*Dans une version **V2**, je souhaiterai ajouter à mon application :*

- *un **forum** afin de compléter son utilité*
- *un système d'envoi de **mail(s)** afin de signaler l'inscription d'un passeur ou d'un receveur dans le "savoir" intéressé*
- ***restreindre** par mesure de sécurité les **habilitations** (modifications ou ajouts liés au contenu de l'application) et les transmettre à l'administrateur*
- *donner la possibilité à l'**administrateur** de **gérer** aussi les **catégories de savoir***

RÉSUMÉ DU PROJET EN ANGLAIS

LISTE DES COMPÉTENCES

MAQUETTER UNE APPLICATION :

Le maquettage de l'application a été réalisé à partir du logiciel Balsamiq.
Un logiciel simple d'utilisation et parfaitement adapté à mes besoins.

CONCEVOIR UNE BASE DE DONNÉES :

Le choix du logiciel de réalisation de diagramme (ici le diagramme des classes) a été porté sur draw.io, pour ensuite utiliser MySQL Workbench afin de modéliser la base de données.

METTRE EN PLACE UNE BASE DE DONNÉES :

La base de données SQL a été réalisée à partir de sa modélisation, ainsi toujours à l'aide de MySQL Workbench.

DÉVELOPPER UNE INTERFACE UTILISATEUR :

Le Front-End a été développé via le Framework Typesript, Angular V.

DÉVELOPPER DES COMPOSANTS D'ACCÈS AUX DONNÉES :

Les composants d'accès aux données (Back-End) ont été développés avec Springboot, Framework Java.

DÉVELOPPER DES PAGES WEB EN LIEN AVEC UNE BASE DE DONNÉES :

Mon application est directement liée à une base données, permettant ainsi à l'utilisateur d'ajouter, de consulter, de modifier et enfin de supprimer des données dans la base.

DÉVELOPPER UNE APPLICATION SIMPLE DE MOBILITÉ NUMÉRIQUE :

Mon application a été pensée en premier lieu pour les mobiles (accessibilité adaptée).
Cependant, elle reste responsive et s'adapte à tous les formats (pc, tablette et ainsi mobile).

UTILISER L'ANGLAIS DANS SON ACTIVITÉ PROFESSIONNELLE EN INFORMATIQUE :

Les méthodes et verbes "généraux" (ex: get, add, update...) ont été conservés en anglais dans le code.
Aussi, toutes les recherches sont prioritairement faites via des sites en anglais (ex: Stack Overflow).
Ce n'est que par vérification ou apprentissage qu'une traduction est utilisée.

CAHIER DES CHARGES OU EXPRESSION DES BESOINS DE L'APPLICATION SIMPLON'CLICK

1. Objectifs

Basé sur le principe du "*Teaching By Teaching*", le logiciel de **partage de connaissances et de compétences** Simplon'click permettra aux utilisateurs inscrits d'échanger facilement sur leurs attentes et leurs besoins favorisant leur évolution dans la conception et le développement informatique :

- Chaque adhérent est décrit principalement par ses besoins (**receveur**), sa spécialité (**passeur**), sa **disponibilité** et son **lieu de présence**.
- Chaque besoin y est détaillé et laisse place à la notion de "savoir".
- Chaque savoir peut prétendre à des ressources libres (lien hypertexte).
- Chaque page de l'application doit allier simplicité (interface épurée) et efficacité.

Rappelons que la volonté de faire preuve de pédagogie est un exercice formateur à la fois pour celui qui écoute que pour celui qui enseigne.

2. Glossaire et définitions

Simplon'click

L'association Simplon ayant créé le sigle "Simplonline" (parcours en ligne de monter en compétences rapide mis au point pas Simplon.co et OpenClassrooms), et faire part d'un besoin devant être aussi simple qu'un clic (d'où la conception souhaitée d'une interface épuré et efficace), ma pensée était de créer un sigle alliant alors la simplicité ("Simpl") et le code (l'événement "onclick").

Administrateur

Utilisateur gestionnaire de tous les membres.

Membre

Utilisateur adhérent aux fonctionnalités de l'application web (Membre de la communauté Simplon).

Savoir

Un savoir est un terme générique qui regroupe les notions de compétences, de langages informatiques, de concepts informatiques et autres.

Catégorie de savoir

Classement des savoirs.

Passeur

Le membre qui apporte un enseignement, le donneur de savoir.

Receveur

Le membre qui exprime le besoin d'approfondir ses connaissances, le receveur de savoir.

Ressource

Lien hypertexte redirigeant l'utilisateur vers une source web en rapport avec un savoir particulier.

3. Fonctionnalités de l'application

a) Description et enchaînement logique des fonctionnalités

Pour cette première version de l'application, tous les membres sont habilités à gérer les savoirs et leurs ressources.
L'utilisateur (*membre*) :

- s'inscrit à Simplon'click et renseigne les informations obligatoires pour l'ouverture de son compte
- crée/consulte/modifie/supprime éventuellement un(des) savoir(s)
- crée/consulte/modifie/supprime éventuellement une(des) ressource(s) liée(s) à un savoir
- Fait part éventuellement d'un ou plusieurs besoin(s) lié(s) à un(des) savoir(s)
- Propose éventuellement son aide pour un(des) savoir(s)

b) Règles de gestion associées

Un administrateur du site :

- peut consulter la liste des membres
- peut créer/consulter/modifier/supprimer/désactiver un membre
- peut créer/consulter/modifier/supprimer un(des) savoir(s)
- peut créer/consulter/modifier/supprimer une(des) ressource(s) liée(s) à un savoir

Un utilisateur (membre) :

- peut créer/consulter/modifier/supprimer son compte
- peut consulter les comptes des autres membres
- peut créer/consulter/modifier/supprimer un(des) savoir(s)
- peut créer/consulter/modifier/supprimer une(des) ressource(s) liée(s) à un savoir
- peut créer/consulter/modifier/supprimer un besoin (lié à son compte)
- peut créer/consulter/modifier/supprimer une proposition de soutien (liée à son compte)
- détient une ou plusieurs inscription(s)

Un savoir :

- doit avoir un nom unique
- ne peut être supprimé s'il est utilisé par un membre
- contient zéro, un ou plusieurs ressource(s)
- appartient obligatoirement à une catégorie de savoir

Une catégorie de savoir :

- doit avoir un nom unique
- contient un ou plusieurs savoir(s)

Une ressource :

- est défini par une adresse web (URL)
- peut être nommée
- est lié obligatoirement à un savoir

Un type d'inscription :

- défini le passeur ou le receveur
- est lié obligatoirement à une ou des inscription(s)

Une inscription :

- comprend un type d'inscription unique
- comprend un niveau de savoir unique
- est lié obligatoirement à un membre

Un niveau de savoir :

- défini le niveau subjectif de connaissance du passeur ou du receveur pour un savoir
- est lié obligatoirement à une ou des inscription(s)

4. Interface utilisateur

a) Ergonomie

- L'interface utilisateur doit être simple et épurée afin de gagner en efficacité
- Tout membre doit pouvoir s'y connecter et ce, peu importe le support utilisé (application web responsive)
- La page d'accueil doit résumer le statut et les attendus de l'utilisateur
- Les fonctionnalités peuvent se réaliser en suivant un parcours simple depuis la page d'accueil

b) Accessibilité

- Il est prévu de rendre l'application compatible avec les standards de l'accessibilité du web

c) Charte graphique

- Les couleurs dominantes de l'application sont le rouge, le noir et le blanc (en rapport avec la charte graphique du site de Simplon.co)

SPÉCIFICATIONS FONCTIONNELLES

L'application Simplon'click est une application web de type Multi Page Application.

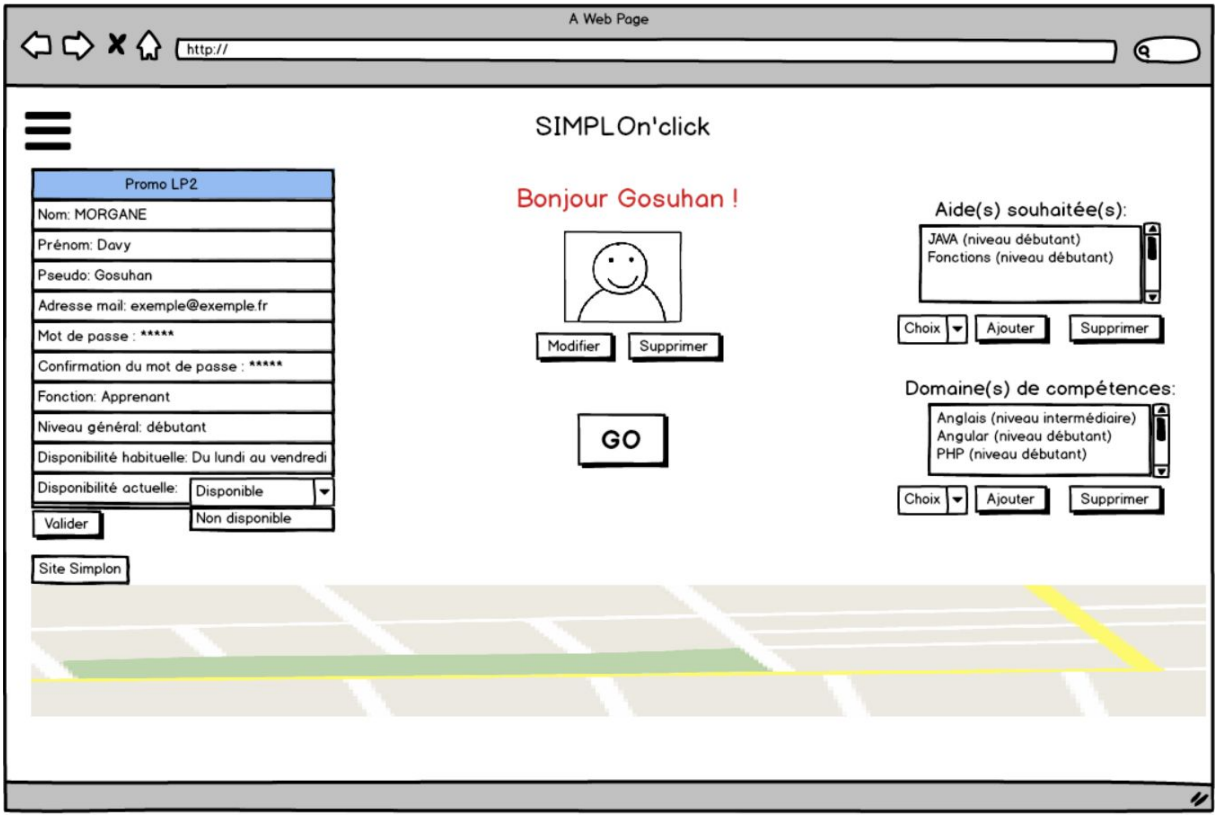
1. CONNEXION ET INSCRIPTION

2. PAGE DE L'ADMINISTRATEUR

3. PAGE D'ACCUEIL

La page d'accueil contient les données de l'**utilisateur connecté**.

C'est sur cette page que l'utilisateur vérifie son statut et ses attendus.



La partie située à gauche de l'écran résume les données personnelles de l'utilisateur, alors que la partie de droite résume ses attendus.

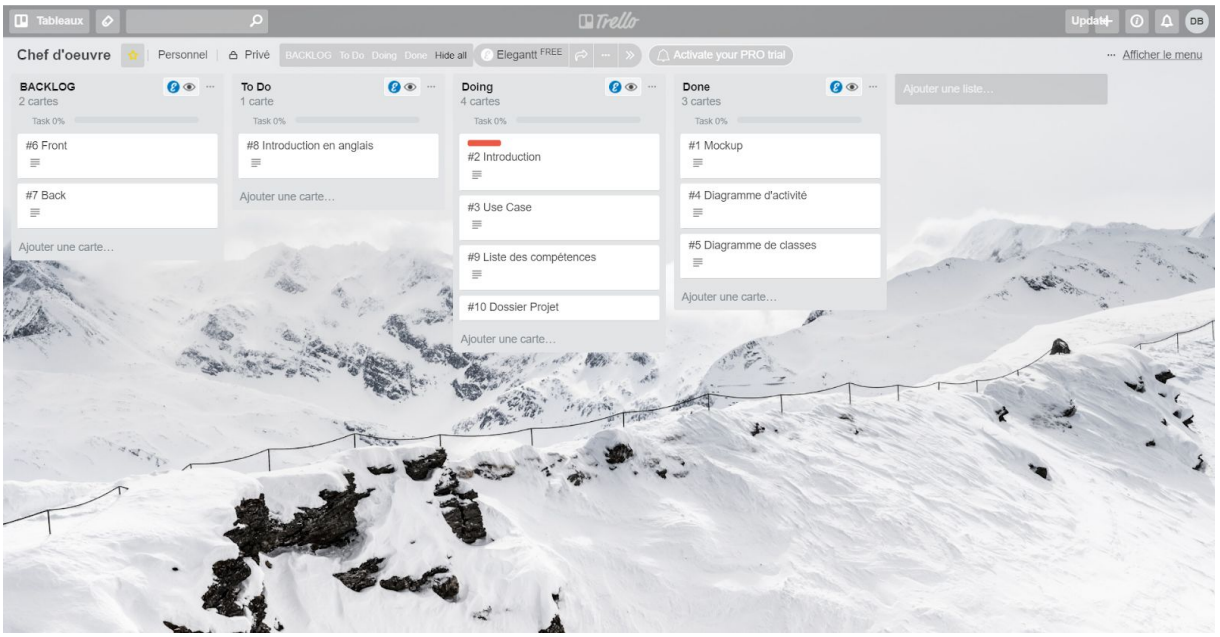
4. PAGE DES BESOINS/SOUHAITS

RÉALISATION DE L'APPLICATION SIMPLON'CLICK

1. ORGANISATION

a) Planification des tâches

Trello

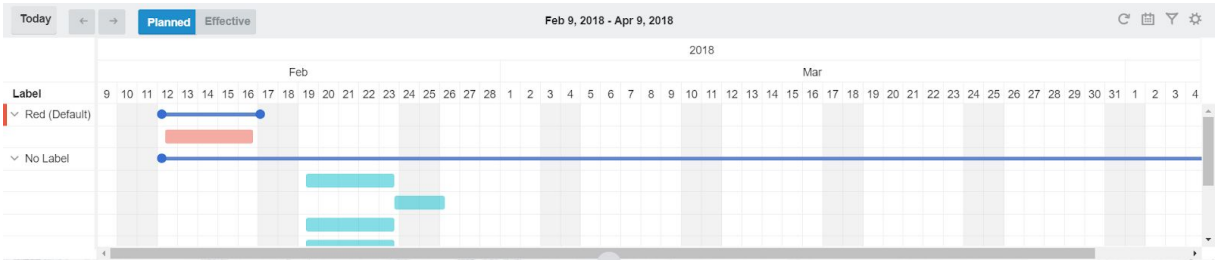


L'organisation et la planification du projet a été géré par l'outil en ligne Trello (https://trello.com). Un outil utilisé notamment durant la formation Simplon.

Cette outil permet notamment de gérer des fiches qui peuvent être :

- ➢ assignées (particulièrement utile dans le cas d'un projet de groupe)
- ➢ affectées à une catégorie
- ➢ affectées à une date d'échéance

La planification paraît alors claire et précise notamment grâce à la possibilité de générer un ***Diagramme de Gantt*** avec l'extension ***Elegantt*** :

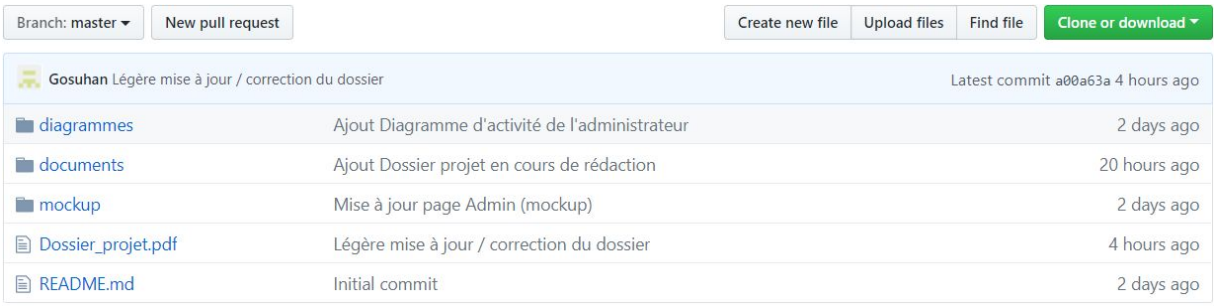


b) Sauvegarde et communication

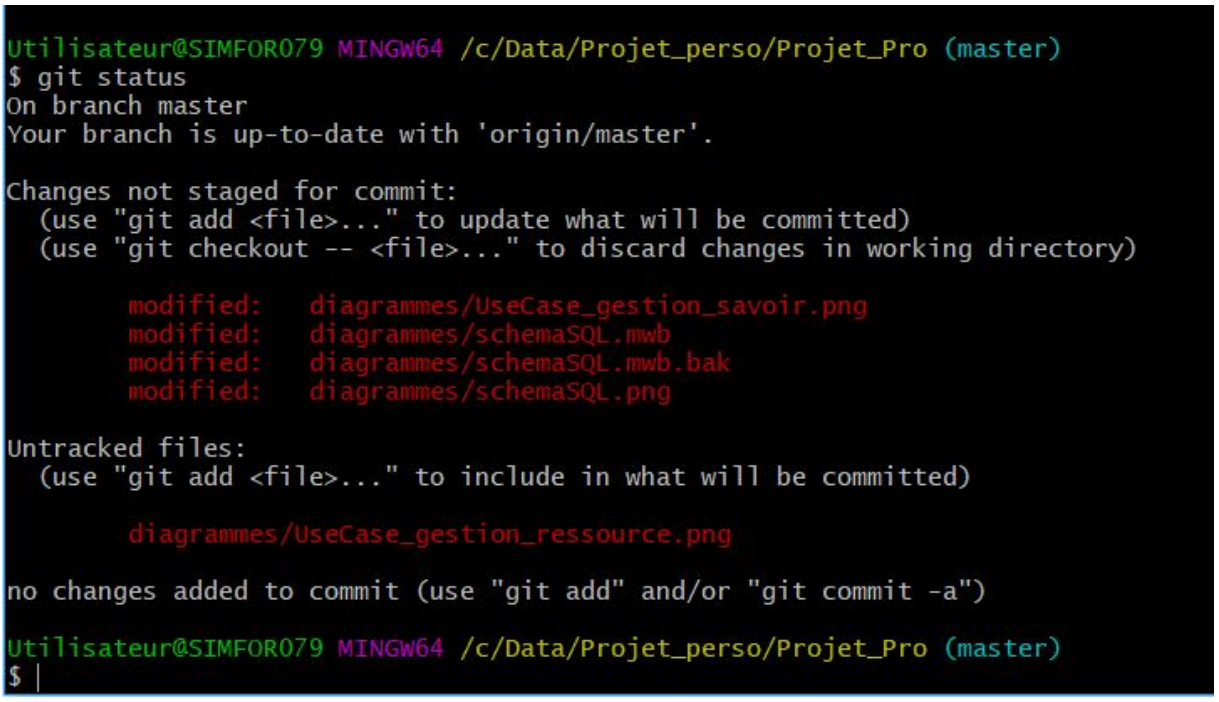
La plateforme open source de gestion de versions et de collaboration ***GitHub*** (https://github.com) a été très utiles afin de permettre aux formateurs (et tuteur) d'avoir une visibilité sur le projet en cas de besoin, tout en sauvegardant efficacement (décentralisation) les informations et documents.

Le Back-End et le Front-End ont chacun bénéficiés d'un dépôt distinct.

Ci-après, le ***dépôt*** du Back-End :



Les interactions avec ces dépôts distants ont notamment été gérées en ***ligne de commande Git*** :

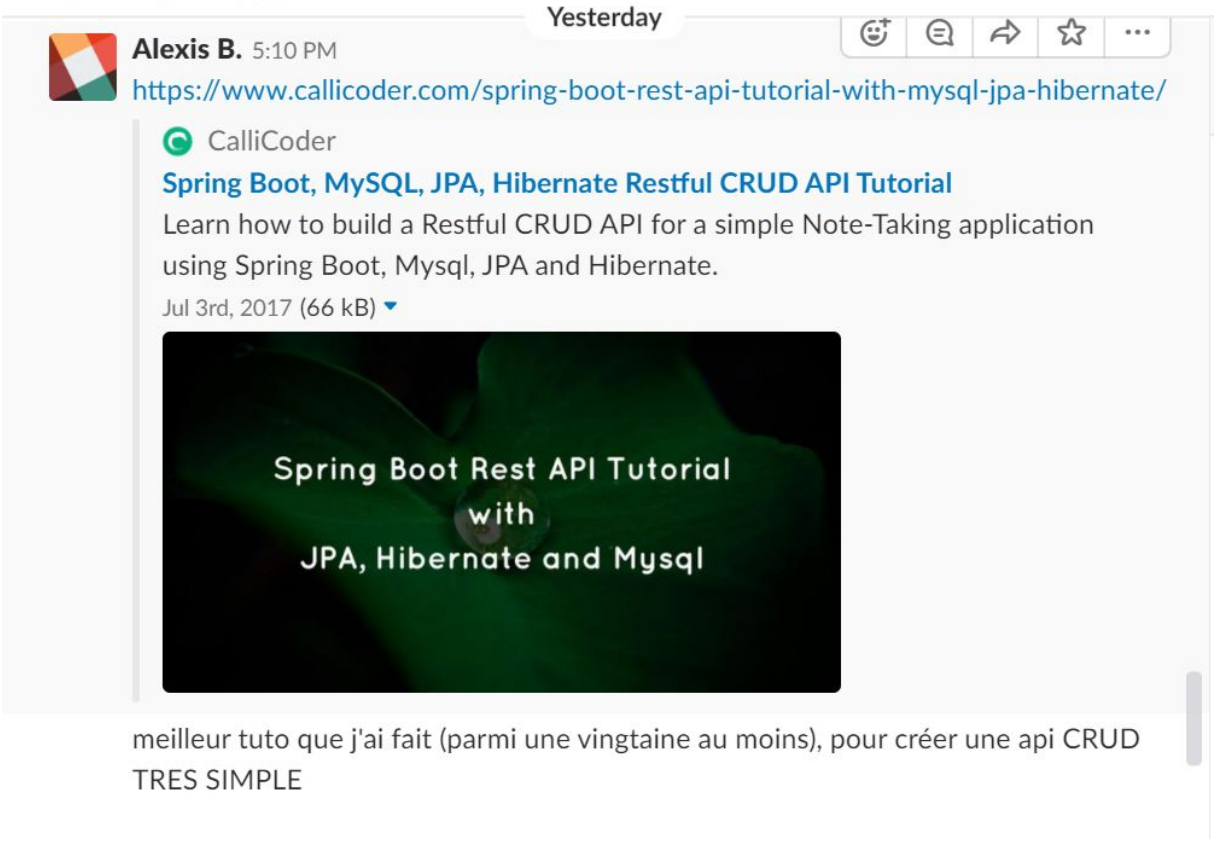


Google Drive

Les sauvegardes instantanées ont pu être gérées par Google Drive avec en plus l'utilisation des services associés comme Google Docs ou encore Google Slides.

Slack

Enfin, l'outil Slack a permi un contact quotidien avec les formateurs de Simplon et toute la 2ème Promo de La Poste ainsi que d'autres apprenants.



Cette outil sera d'ailleur pris en compte (communication des pseudos) dans l'application Simplon'click en attendant un forum intégré dans une prochaine version.

2. CONCEPTION DE L'APPLICATION

Le projet Simplon'click a été pensé suite à un dispositif mis en place durant un cours en début d'année où chaque apprenant de notre groupe a eu la possibilité d'exprimer ses forces et faiblesses via un système de post-it et ce, dans le but d'échanger et valoriser les compétences.

L'idée était alors de dématérialiser ce concept et le rendre ouvert à tout une communauté.

a) Diagrammes UML

Le langage UML (Unified Modeling Language) a été adopté pour schématiser la structure, les fonctionnalités et le comportement de l'application, en particulier ces trois types de digrammes réalisés par l'outil *draw.io* :

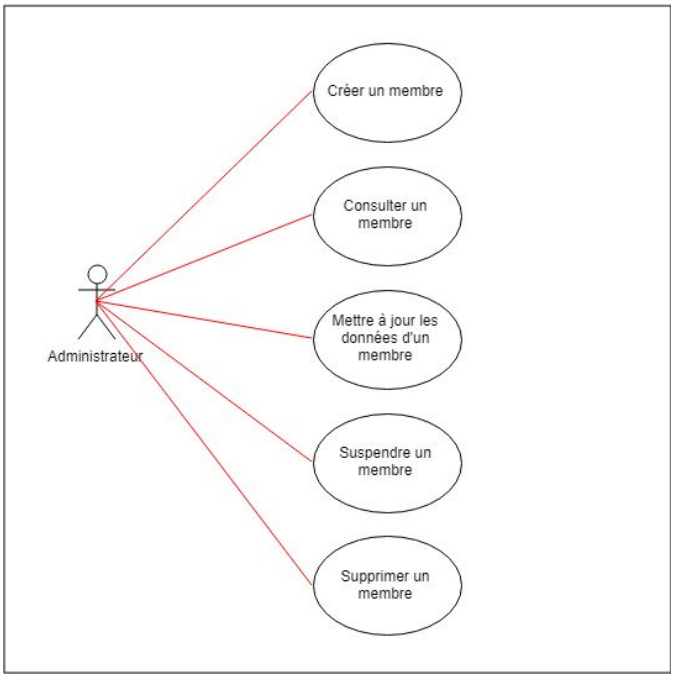
❖ *Diagramme des cas d'utilisation*

Ce diagramme décrit les possibilités offertes par chaque utilisateur.
Dans le cas présent, cinq cas d'utilisation ont été créés :

- ➢ un diagramme pour la gestion des membres (administrateur)
- ➢ deux diagrammes pour la gestion des savoirs (administrateur et membres)
- ➢ deux diagrammes pour la gestion des ressources (administrateurs et membres)

Pour exemple, ci-après, le diagramme "Gestion des membres" d'un administrateur :

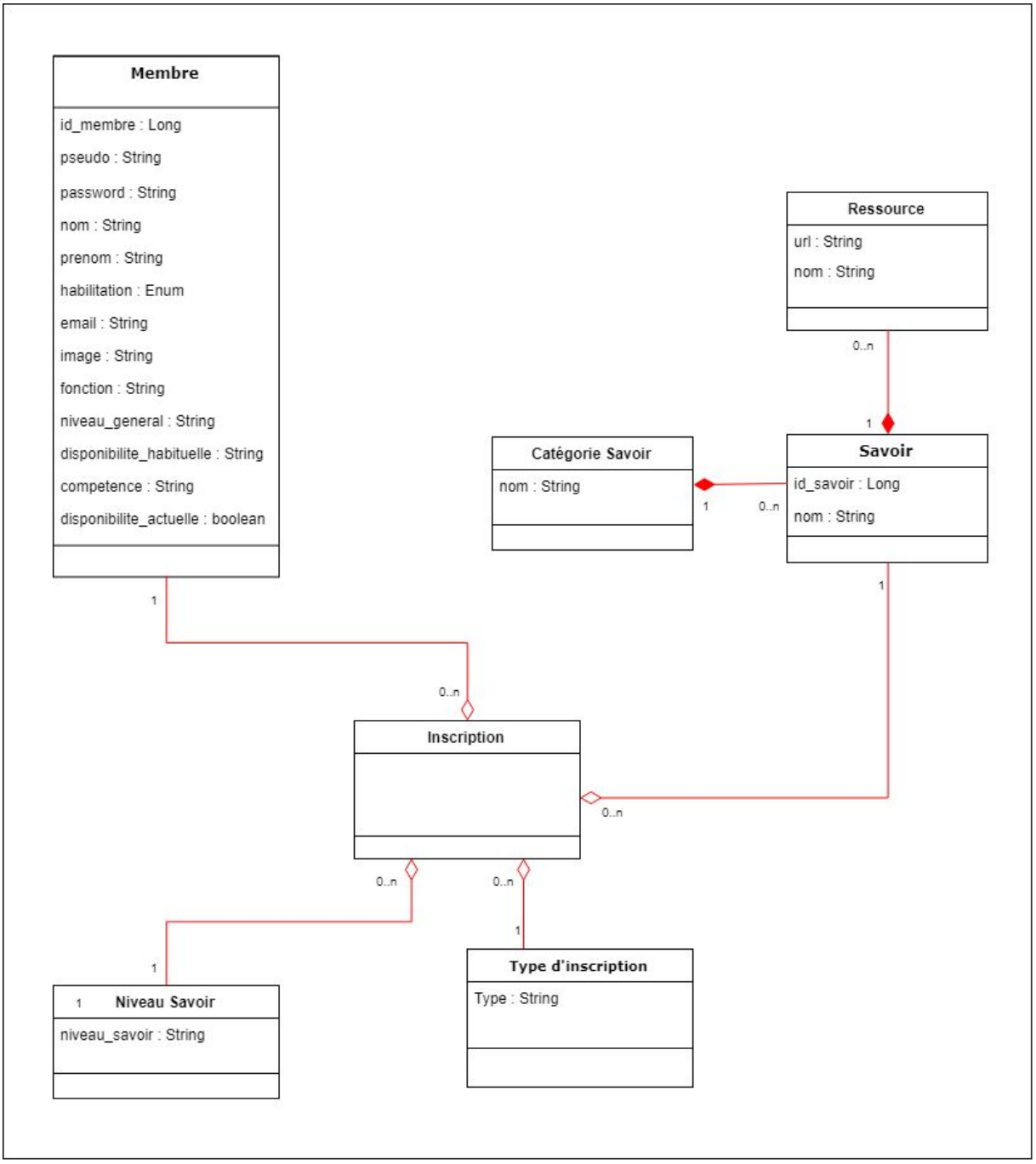
Gestion des membres



❖ *Diagramme de séquences*

❖ *Diagramme des classes*

Le diagramme des classes représente les différents objets ou entités qui constituent l'application, ainsi que leurs relations :

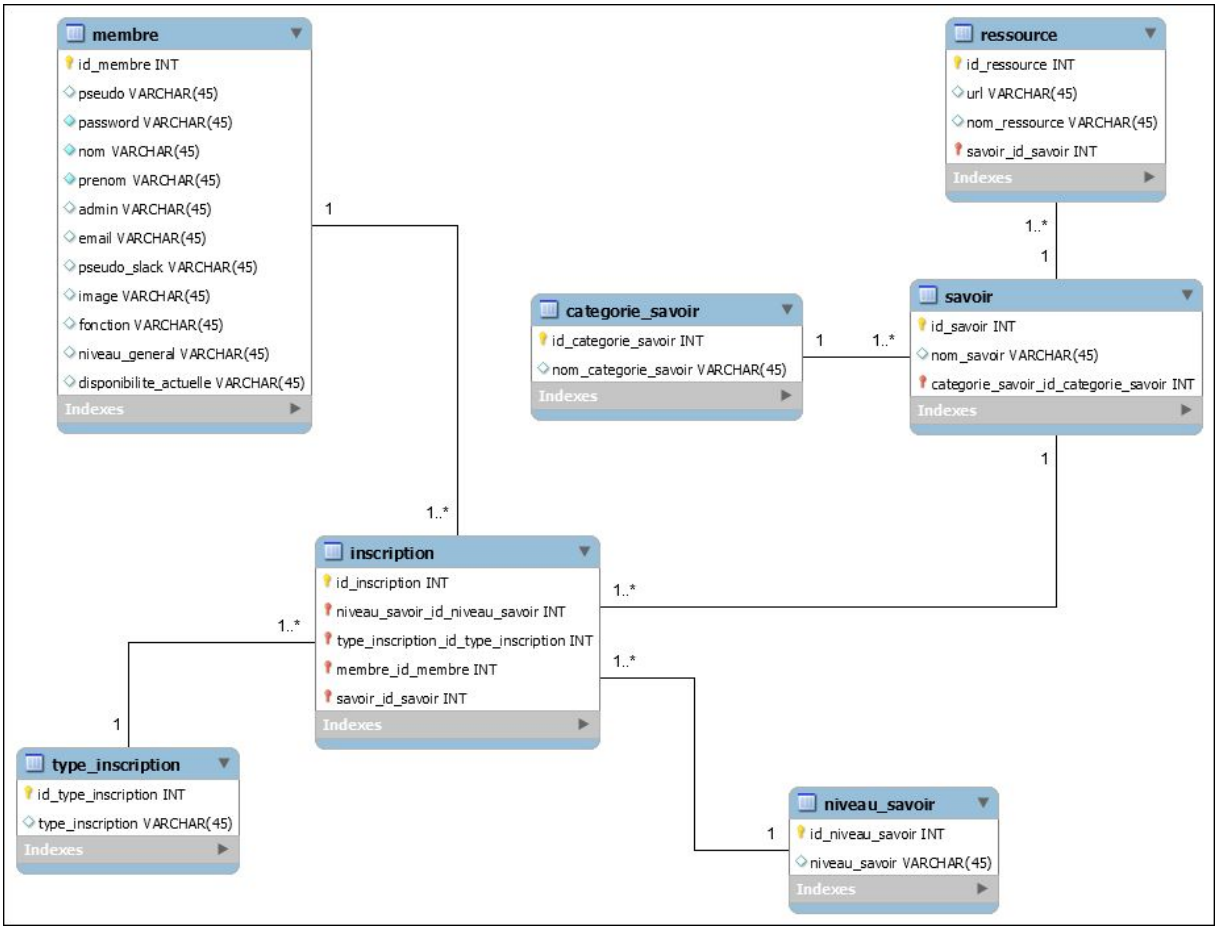


Les différents types de relation sont :

- L'*Agrégation* (losange vide) : C'est une association avec une relation de subordination (si A contient B, la destruction de A n'entraîne pas la destruction de B)
- La *Composition* (losange plein) : C'est une agrégation avec un cycle de vie dépendant (si A contient B, la destruction de A entraîne alors la destruction de B)
- La *Cardinalité* (1, 0..n) : C'est le nombre d'objets possibles de chaque côté de la relation

b) Base de données

La base de données a été modélisée via le logiciel MySQL Workbench en prenant référence sur le diagramme des classes :



Ce diagramme ou modèle ER (Entity-Relationship ou Entité-Association) décrit en détail la structure des tables de la base et leurs relations.

c) Maquettes de l'interface

3. MISE EN PLACE DE LA BASE DE DONNÉES

a) Structure

b) Données

4. DÉVELOPPEMENT DU BACK-END

Le choix personnel du langage de programmation a été porté sur **Java**. Ce langage (étudié durant le temps de formation) n'étant pas utilisé par mon entreprise, il m'a semblé intéressant de l'utiliser afin

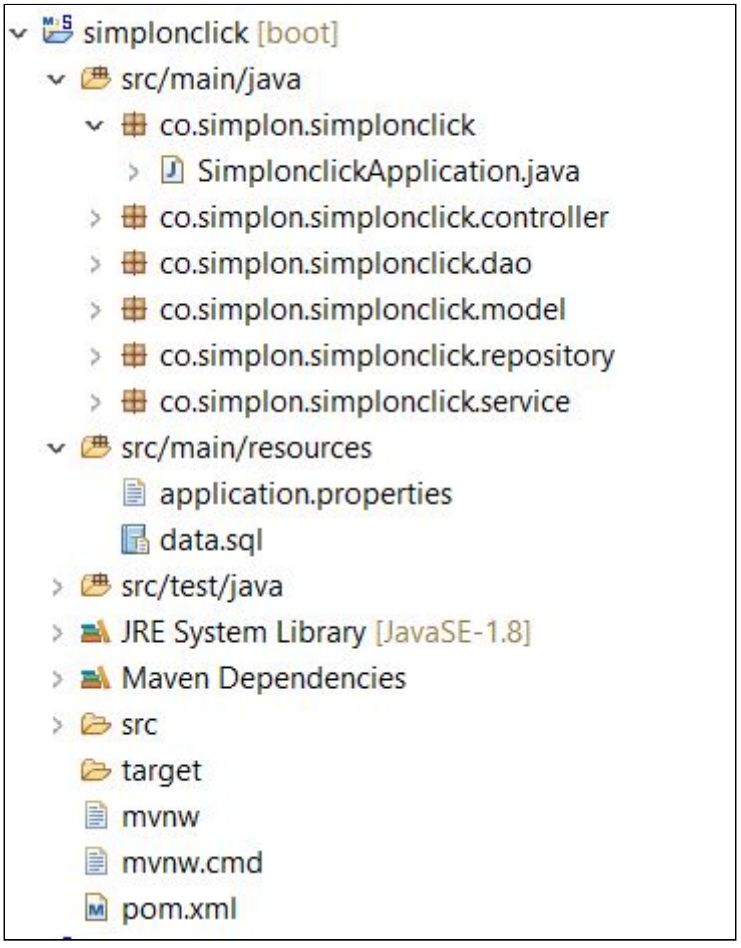
de conserver et d'approfondir mes connaissances et compétences dans celui-ci lorsque des modifications seront apportées par mes soins et ce, après le délai de formation.
L'**IDE** *(Environnement de Développement Intégré)* utilisé est alors **Eclypse** avec l'outil **Maven** pour la gestion des dépendances et le build de l'application. Outre les raisons précisées en amont, le choix d'utiliser Eclypse afin de développer en Java est aussi le résultat d'un raisonnement logique suite à son utilisation durant les périodes de formations, avec une connaissance approfondi de son fonctionnement et de son adaptation au langage.

a) Spring Boot

Spring Boot est un framework léger, il permet de démarrer une application Spring MVC auto-configurée d'après les bonnes pratiques standardisées. Il contient aussi son propre serveur **Tomcat**.

Initialisation
Spring fournit un initialiseur sur le site https://start.spring.io/:
On choisit les dépendances et un zip contenant les fichiers de base du projet est généré.

Organisation de l'application



- src/main/java contient les classes java (controller, DAO, model, repository et service)
- Le fichier *SimplonclickApplication.java* permet de lancer l'application
- *src/main/resources* contient le fichier application.properties (propriétés de l'application comme les codes d'accès à la base de données)
- *src/test/java* contient les classes test
- *target* contiendra le build de l'application pour la livraison
- *pom.xml* contient la configuration des dépendances, importées dans l'application par Maven

Ci-dessous une partie du fichier **pom.xml :**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>co-simplon</groupId>
    <artifactId>simplonclick</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>simplonclick</name>
    <description>Projet professionnel</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.10.BUILD-SNAPSHOT</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>

        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>mssql-jdbc</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
```

Dépendances

- **JPA** *(Java Persistence API) :* Inclus dans la plateforme *Java EE*, JPA est une spécification qui définit un ensemble de règles permettant la gestion de la correspondance entre des objets Java et les données d'une base de données (= gestion de la persistance).

- **Hibernate** (framework ORM *(Object-Relational Mapping)) :* JPA et Hibernate se chargent de la persistance des données (mappage entre les données en base de données et les classes java) et rendent l'application indépendante de la base de données (le code reste inchangé et ce, même en cas de modification de type de base). De plus, Hibernate protège l'application des injections de code lors des requêtes SQL.

- **Spring Security** : Framework qui fournit les classes nécessaires à la sécurisation des accès à l'application.

- **Maven** : Outil précieux qui sert par exemple à gérer automatiquement les dépendances, compiler le code java, etc.

- **Lombok** : Bibliothèque qui permet de générer automatiquement les accesseurs (getter ou setter), ainsi que les méthodes toString(), hashCode() et equals() d'une classe.

Composants

Dans Spring Boot (et Spring MVC), le traitement des demandes du Frontend nécessite plusieurs composants :

- **Model / Entity** : Gère la structure, le modèle des données et (Entity) les objets mappés par le Repository.
- **Repository** : Gère la persistance des données, à la demande du Service.
- **Service** (couche métier) : Traite les demandes du Controller et lui fournira le résultat.
- **Controller** (couche de présentation) : Reçoit les actions de l'utilisateur et lui renverra la réponse.
- **DAO** : Gère le résultat souhaité depuis la base de donnée (via des requêtes SQL).

Des annotations permettent de différencier ces composants et surtout leur rôle (*@Component, @Controller, etc).*

b) Présentation en détail du module "Membre" à titre d'exemple

Controller (REST)
Classe identifiable par l'annotation *@RestController.*

Un *@RestController* renvoie comme réponse les données au format Json (application REST) contrairement à un *@Controller* renvoyant l'url de la page qui affichera les données.

```java
package co.simplon.simplonclick.controller;

import java.util.List;

@RestController
@RequestMapping("/api")
@CrossOrigin(origins = "*")

public class MembreController {

    @Autowired
    private MembreService membreService;
    @Autowired
    private MembreDAO membreDAO;

    //INSERT INTO `simplonclick`.`membre` (`id_membre`, `pseudo`, `password`,`nom`, `prenom`, `admin`, `email`
    @PostMapping(path = "/membre")
    Membre addMembre(@Valid @RequestBody Membre membre) throws Exception {
        return membreService.addMembre(membre);
    }

    //SELECT * FROM membre;
    @GetMapping(path = "/membres")
    public @ResponseBody Iterable<Membre> getAllMembres() throws Exception {
        return membreService.getAllMembres();
    }

    //SELECT * FROM membre WHERE `id_membre`=id;
    @GetMapping(path = "/membre/{id}")
    ResponseEntity<Membre> getMembre(@PathVariable(value = "id") long id) throws Exception {
        Membre membre = membreService.getMembre(id);
```

Le contrôleur contient la table de routage de l'application :

- *@RequestMapping* contient la base de l'URI (Uniform Resource Identifier) : '/api'
- *@GetMapping*('/membre') indique la méthode déclenchée par un GET

On remarque que les méthodes addMembre(), getAllMembres() et getMembre() font appel à des méthodes du Service.

Service

Classe identifiable par l'annotation @Service.

```java
package co.simplon.simplonclick.service;

import org.springframework.beans.factory.annotation.Autowired;

@Service
public class MembreService {

    @Autowired
    private MembreRepository membreRepository;

    public Iterable<Membre> getAllMembres() throws Exception {
        return membreRepository.findAll();
    }

    public Membre getMembre(Long id) throws Exception {
        return membreRepository.findOne(id);
    }

    public void deleteMembre(Long id) {
        membreRepository.delete(id);
    }

    public Membre addMembre(Membre membre) throws Exception {
        return membreRepository.save(membre);
    }

    public Membre editMembre(Long id, Membre membre) throws Exception {
        return membreRepository.save(membre);
    }

    public void clearMembreTable() {
        membreRepository.deleteAll();
    }

}
```

Le Service contient la couche métier où sont effectués les traitements des données.
Les méthodes sont assez simples et explicites, faisant appellent à des méthodes toutes aussi explicites du Repository.

On remarque le bloc de commentaires en bleu /** ... */, qui permettra de générer automatiquement la documentation de l'application : la javadoc.

Repository

L'annotation *@Repository* peut être omise, il est suffisant que la classe hérite de JpaRepository pour que Spring Boot puisse l'identifier.

```java
package co.simplon.simplonclick.repository;

import org.springframework.data.jpa.repository.JpaRepository;

@Repository
public interface MembreRepository extends JpaRepository<Membre, Long> {

}
```

Le Repository contient les méthodes nécessaires à la communication avec la base de données. Grâce à l'héritage, notre **repository possède toutes les méthodes de sa classe mère** : *findAll(), save(), etc.*
On peut aussi déclarer des ***Query Methods*** (héritées de la classe Repository) comme *findBySavoirIdAndCategorieSavoirsId()*, afin d'utiliser toute la puissance d'Hibernate (notamment le ***HQL*** *Hibernate Query Language*) et rendre possible la modification de la Base de données sans aucune refonte du code (contrairement à l'utilisation de requêtes SQL) et ainsi établir un code "propre" (en accord avec le framework utilisé).

La communication avec la base de données s'effectue grâce à la déclaration des identifiants de connexion et le nom de la base dans le fichier *applications.properties* placé dans le dossier *src/main/resources* :

```
# connection base
spring.datasource.url=jdbc:mysql://localhost/simplonclick?useSSL=false
spring.datasource.username=admin
spring.datasource.password=admin
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

Entity
Classe identifiable par l'annotation @Entity.

```java
package co.simplon.simplonclick.model;

import java.util.HashSet;

@Entity
@Table(name = "membre")

public class Membre {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator="native")
    @GenericGenerator(name = "native", strategy = "native")

    private Long id_membre;
    private String pseudo;
    private String password;
    private String nom;
    private String prenom;
    private boolean admin = false;
    private String email;
    private String pseudo_slack;
    private String image;
    private String fonction;
    private String niveau_general;
    private String disponibilite_habituelle;
    private boolean disponibilite_actuelle = false;

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy = "membre")
    @JsonManagedReference
    /*
     * Pour que Jackson fonctionne bien, l'un des deux côtés de la relation ne doit pas être sérialisé,
     * afin d'éviter la boucle infinie qui provoque l'erreur stackoverflow (Postman)
     */
    private Set<Inscription> inscriptions = new HashSet<>();

    public Membre() {

    }
```

Cette classe représente le Bean, c'est à dire un objet sérialisable (= qui peut être rendu persistant). À chaque enregistrement d'une table de la base de données correspond une instance du bean.
Les annotations Hibernate comme *@Id, @GeneratedValue, @GenericGenerator @OneToMany, @JsonManagedReference, @JoinColumn, etc*. servent au mappage des données.

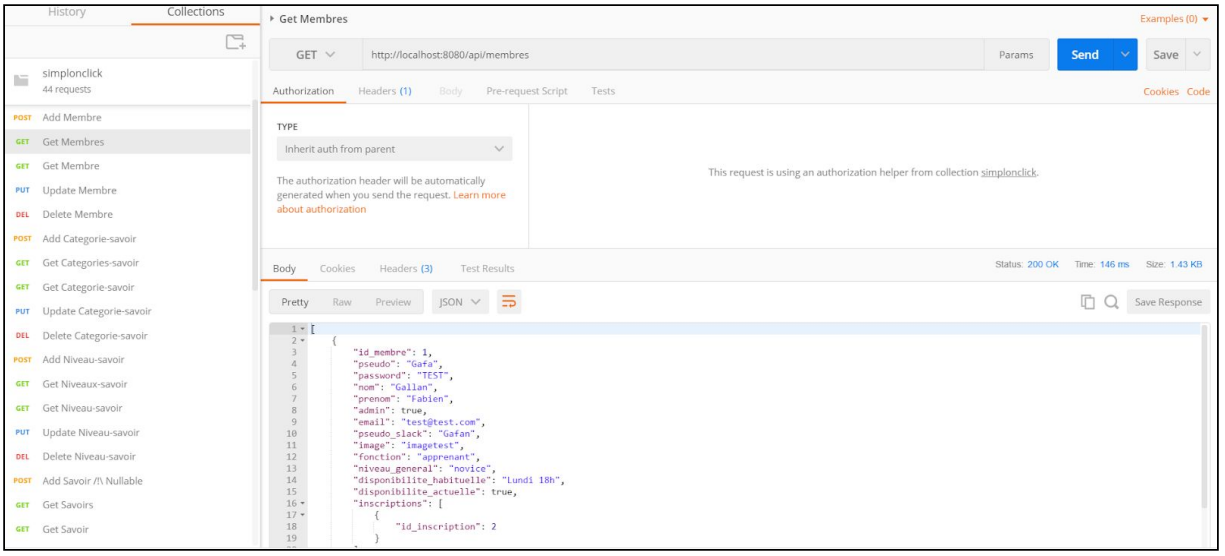
c) Gestion des erreurs

d) Gestion de la sécurité

e) Tests

Tests de l'API

Postman a été un outil extrêmement utile afin de tester l'**API** *(Application Programming Interface)* et d'en vérifier les interactions avec la base de données.



Tests unitaires

5. DÉVELOPPEMENT DU FRONT-END

6. LIVRAISON DE L'APPLICATION

CONCLUSION

REMERCIEMENTS

LEXIQUE