Nom de naissance BALMINE
Nom d'usage
Prénom Davy
Adresse 19 quai de Versailles
44000 Nantes

Titre professionnel visé

Développeur Logiciel - Niveau III

SOMMAIRE

REMERCIEMENTS 4

INTRODUCTION 5

ABSTRACT 6

LISTE DES COMPÉTENCES DU RÉFÉRENTIEL 7

CAHIER DES CHARGES OU EXPRESSIONS DES BESOINS DE L'APPLICATION SIMPLON'CLICK 8

1. Objectifs 8

2. Glossaire et définitions 8

3. Fonctionnalités de l'application 9

a) Description et enchaînement logique des fonctionnalités 9
b) Règles de gestion associées 9

4. Interface utilisateur 10

a) Ergonomie 10
b) Accessibilité 11
c) Charte graphique 11

ORGANISATION 12

1. Planification des tâches 12

2. Sauvegarde et communication 13

SPÉCIFICATIONS FONCTIONNELLES 15

1. Diagrammes UML 15

2. Maquettes de l'interface 19

SPECIFICATIONS TECHNIQUES 26

1. Architecture logique 26

a) Description des fonctionnalités 26
b) Les acteurs 26
c) Les flux 26

d) Exigences de services 26

2. Architecture technique 26

a) Description 27
b) Composants 27
c) Flux et protocoles 27

3. Mise en place de la base de données 28

a) Schéma 28
b) Structure 28
c) Données 30

4. Développement du Backend 31

a) Spring Boot 31
b) Un exemple : les Membres 34
c) Tests..... 38

5. Développement du Frontend 40

a) Angular 40
b) Un exemple : le module Membre 41

LIVRAISON DE L'APPLICATION 47

a) Ateliers utilisateurs 47
b) Build du livrable 47

CONCLUSION 49

REMERCIEMENTS

Au terme de ce travail, je tiens à exprimer ma profonde gratitude pour mon tuteur **Xavier CHAUVIERE (Chef de Projet)** pour son suivi et son énorme soutien qu'il n'a cessé de prodiguer tout au long de ma formation.

Je tiens à remercier tout particulièrement et à témoigner toute ma reconnaissance à **toute l'équipe du C3S (Centre de Solutions Siège et Support)** pour l'expérience enrichissante et pleine d'intérêt qu'ils m'ont fait vivre durant ma période de stage.

Je souhaite aussi remercier **toute la promotion du Groupe La Poste** pour leur bonne humeur, leur entraide et leur soutien moral, en particulier **Laetitia FOUCHERE**.

Je tiens à remercier **Fabrice GALLARD**, **Sylvain GERARD** et **Didier GENRE** pour leur altruisme et ainsi pour toutes les aides qu'ils ont pu m'apporter durant le développement de notre projet de groupe (Projet Fil Rouge), facilitant de ce fait par expérience la réalisation de mon application.

Mes remerciements vont aussi à **toute l'équipe de Simplon** pour leur disponibilité, en particulier **Jonathan SIFFERT** et nos formateurs **Sébastien MARTIN** et **Emmanuel LEPEVEDIC**.

Je ne laisserai pas cette occasion passer, sans remercier les **Responsables du C3S (Centre de Solutions Siège et Support)**, notamment **Olivier SIMON** et **Mickael RIGOLLET** auprès desquelles j'ai trouvé un accueil chaleureux.

J'adresse enfin mes vifs remerciements aux **membres des jurys** pour avoir bien voulu examiner et juger ce travail.

INTRODUCTION

SIMPLON'CLICK

"Learn 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" *("Learn 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 de mise en relation**, **développée en Java/Spring et Angular** (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 2**, 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 pour le "savoir" (sujet) intéressé*
- *Une **restriction,** par mesure de sécurité, des **habilitations (**modifications ou ajouts liés au contenu de l'application) et les transmettre à l'administrateur*
- *La possibilité à l'**administrateur** de **gérer** aussi les **catégories de savoir***

ABSTRACT

SIMPLON'CLICK

"Learn By Teaching"

We know the slogan of the **developers**: "Google is your friend !".

This notion is necessary to favor the autonomy of any developer (one of the principles Simplon formation such as resourcefulness, proactivity, perseverance, curiosity, reflection and sharing with the community).

However, during this training, **the group work**, **favoring** as well the **understanding** as the **exchanges**, is also very **valued**.

The human being is put back at the heart of the performance of the company.

Furthermore, let us not forget that the fact of **sharing its own knowledge and skills**, in particular for learners, **can consolidate**, **complete** or **still correct** some **"gains"** (***"Learn by teaching"***).

The exchange of face-to-face knowledge is then **productivity gain** and **source of experiences** (in agreement with the expectations of the current companies of development).

That is why my idea was to create a **connecting Web application**, **developed in Java/Spring and Angular** (accessible with a PC, a tablet or a smartphone) **consisting in taking advantage of all human resources** of the same school (here Simplon) **in term of skills and knowledge**.

With proposals of support and demands of explicit helps, all actors, specifying their degree of skill, their place of affectation and their availability, will can **favor a meeting** to exchange their knowledge and know-how in a field of expertise chosen and it is true **face-to-face**.

*In a **version 2**, I shall wish to add to my application:*

- *A **forum** to complete its utility*
- *A system of sending of **e-mail(s)** to indicate the inscription of a boatman or a conductor for the interested "knowledge" (subject)*
- *A **restriction**, by security measure, of the **authorizations** (modifications or additions of the contents of the application) and pass on them to the administrator*
- *The possibility to the **administrator** to **manage** also the **categories of knowledge***

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 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 Typescript, 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 "*Learn 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**), ses spécialités (**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 par 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 (sujet) 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 (l'enseignant).

Receveur

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

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

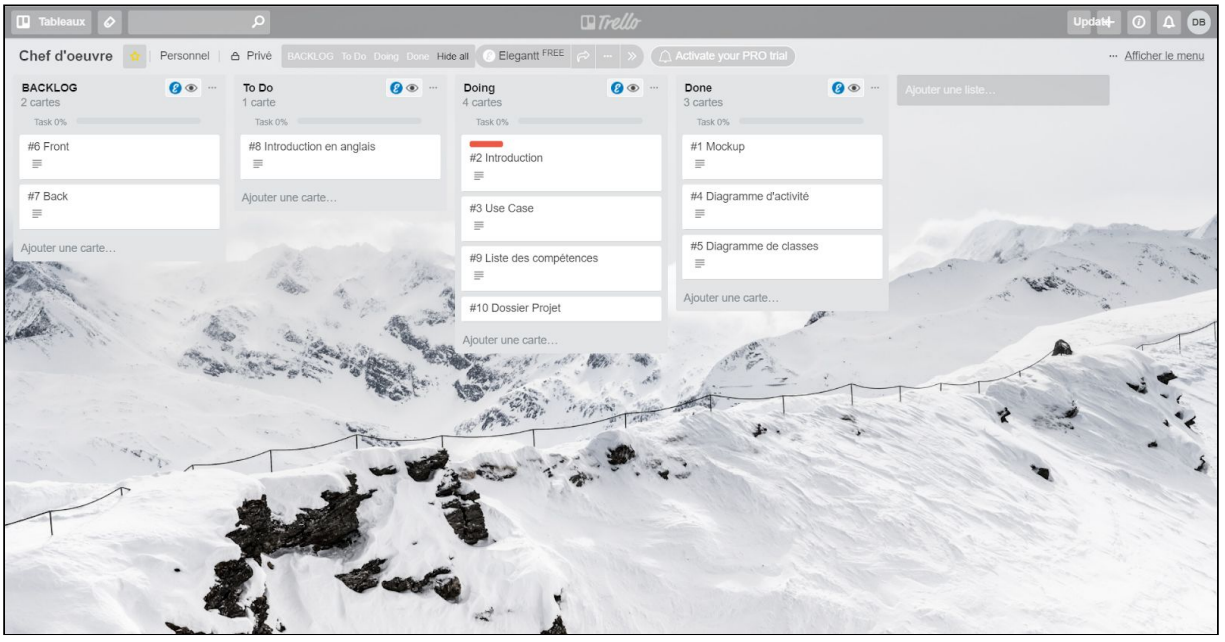
c) Charte graphique

- 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) ainsi que le bleu (rappelant les couleurs de La Banque Postale).

ORGANISATION

1. Planification des tâches

Trello

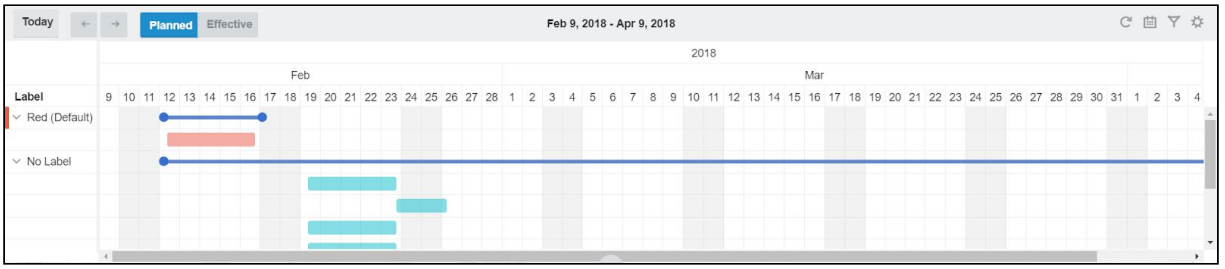


L'organisation et la planification du projet a été géré par l'outil en ligne Trello (https://trello.com). Un outil utilisé notamment durant la formation Simplon.

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*** :

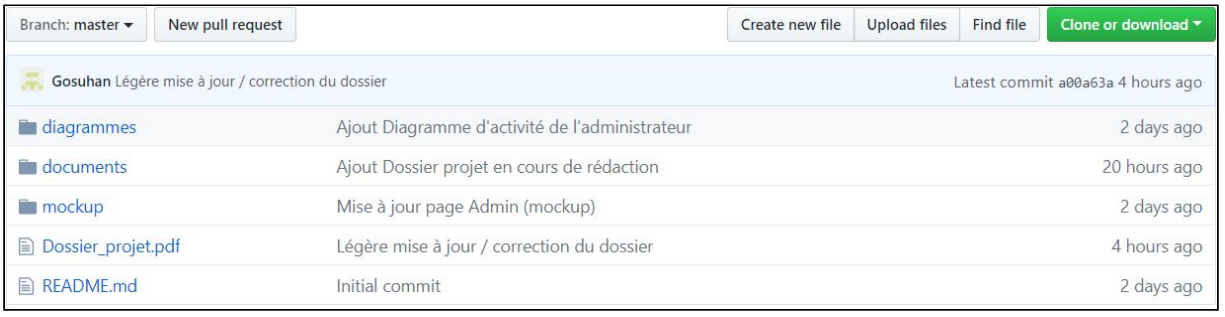


2. Sauvegarde et communication

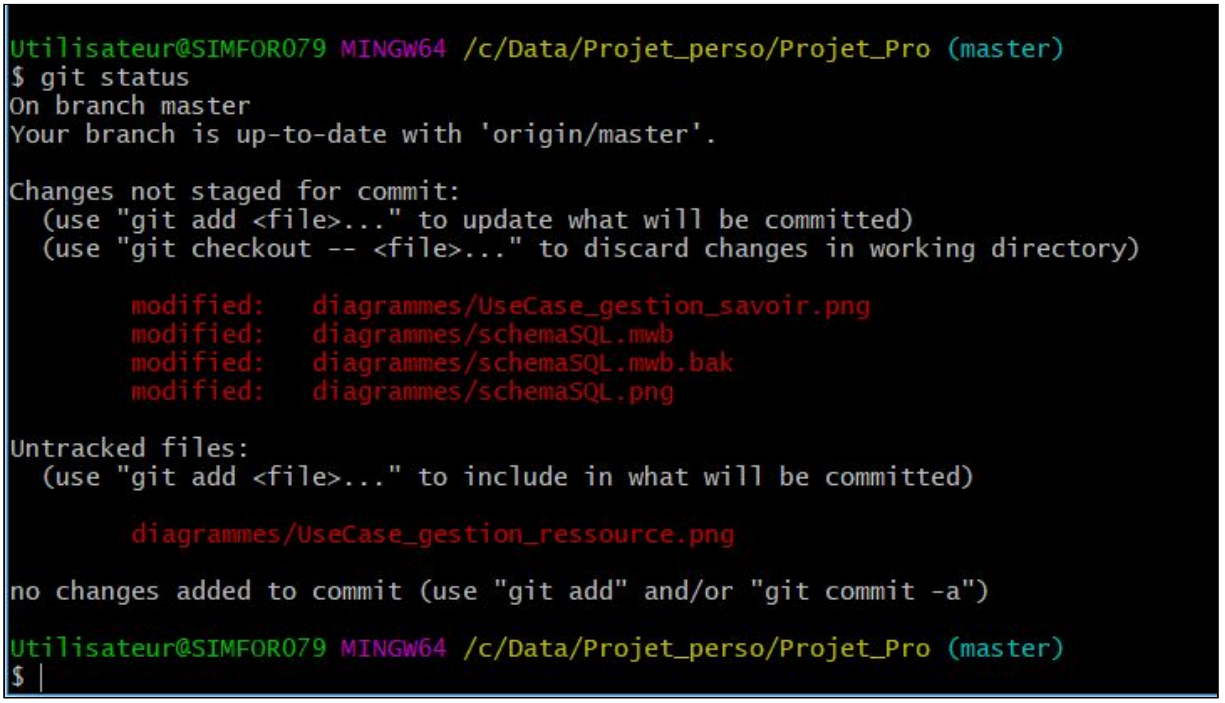
La plateforme open source de gestion de versions et de collaboration ***GitHub*** (https://github.com) a été très utiles afin de permettre aux formateurs (et tuteur) d'avoir une visibilité sur le projet en cas de besoin, tout en sauvegardant efficacement (décentralisation) les informations et documents.

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*** :

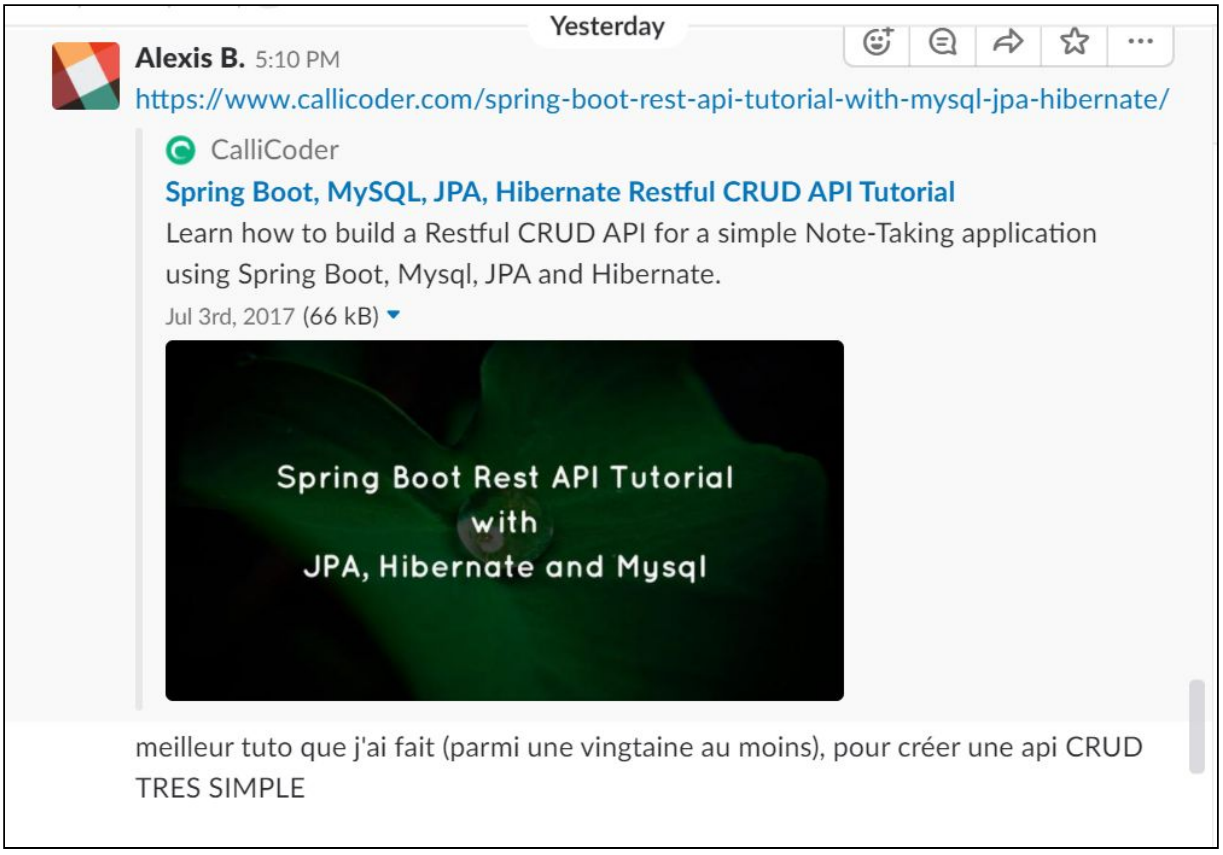


Google Drive

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

SPÉCIFICATIONS FONCTIONNELLES

➔ 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.

1. 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*

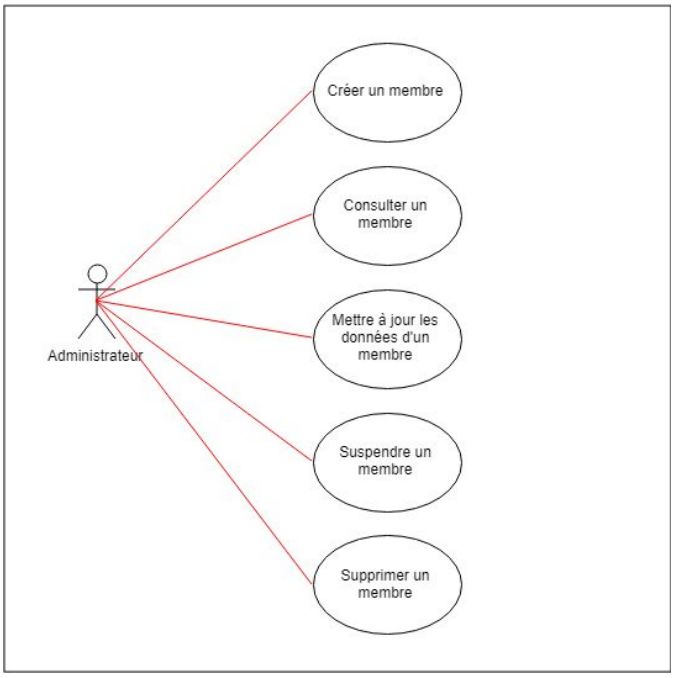
Ce diagramme décrit les possibilités offertes par chaque utilisateur.

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 :

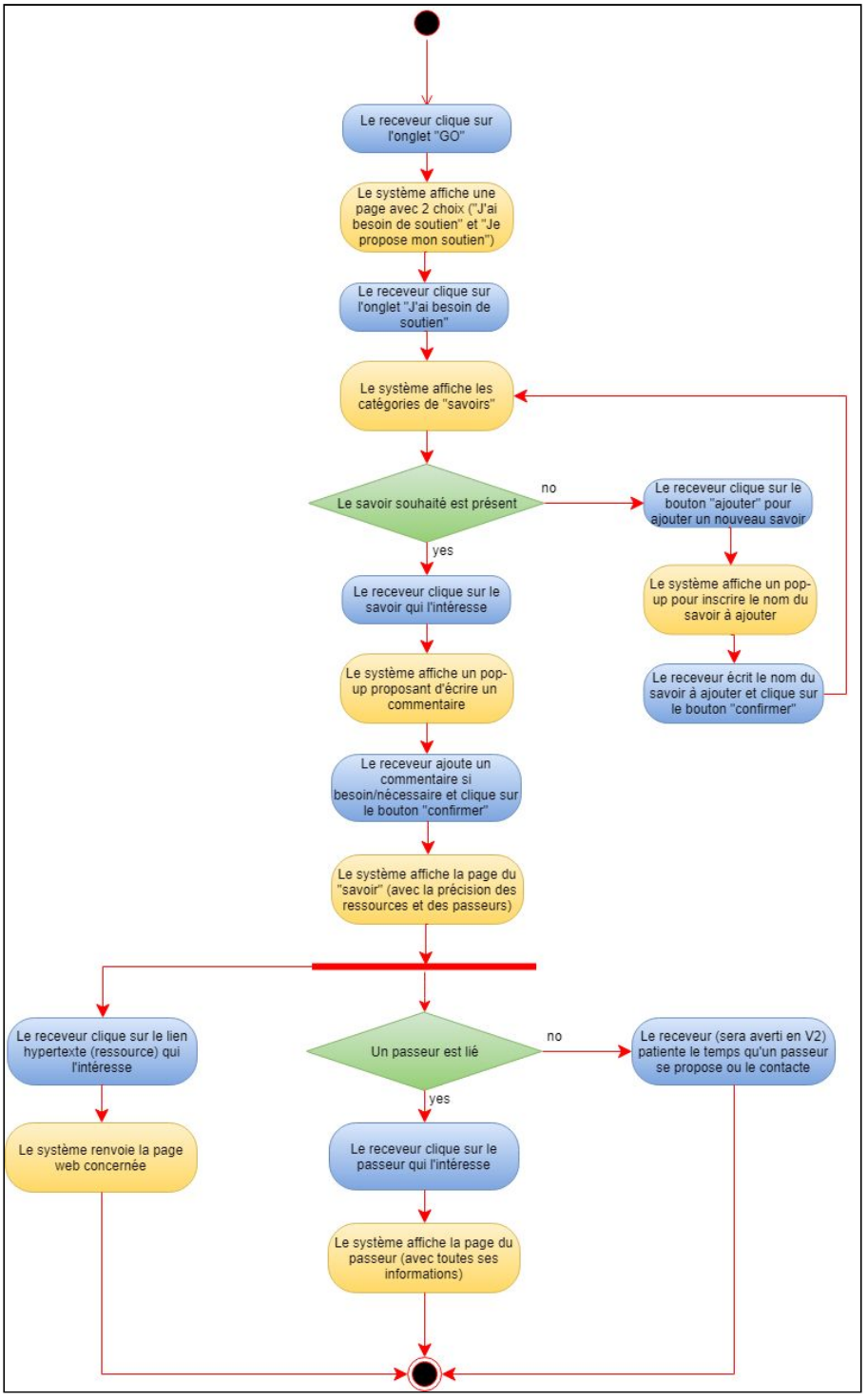
Gestion des membres



- ***Diagramme d'activité***

Le diagramme d'activité est un diagramme comportemental. Il est une formalisation graphique des actions qui sont réalisées dans un cas d'utilisation.
Le diagramme est donc organisé en actions réalisées soit par un acteur, soit par le système, relié par une flèche indiquant l'enchaînement des actions.

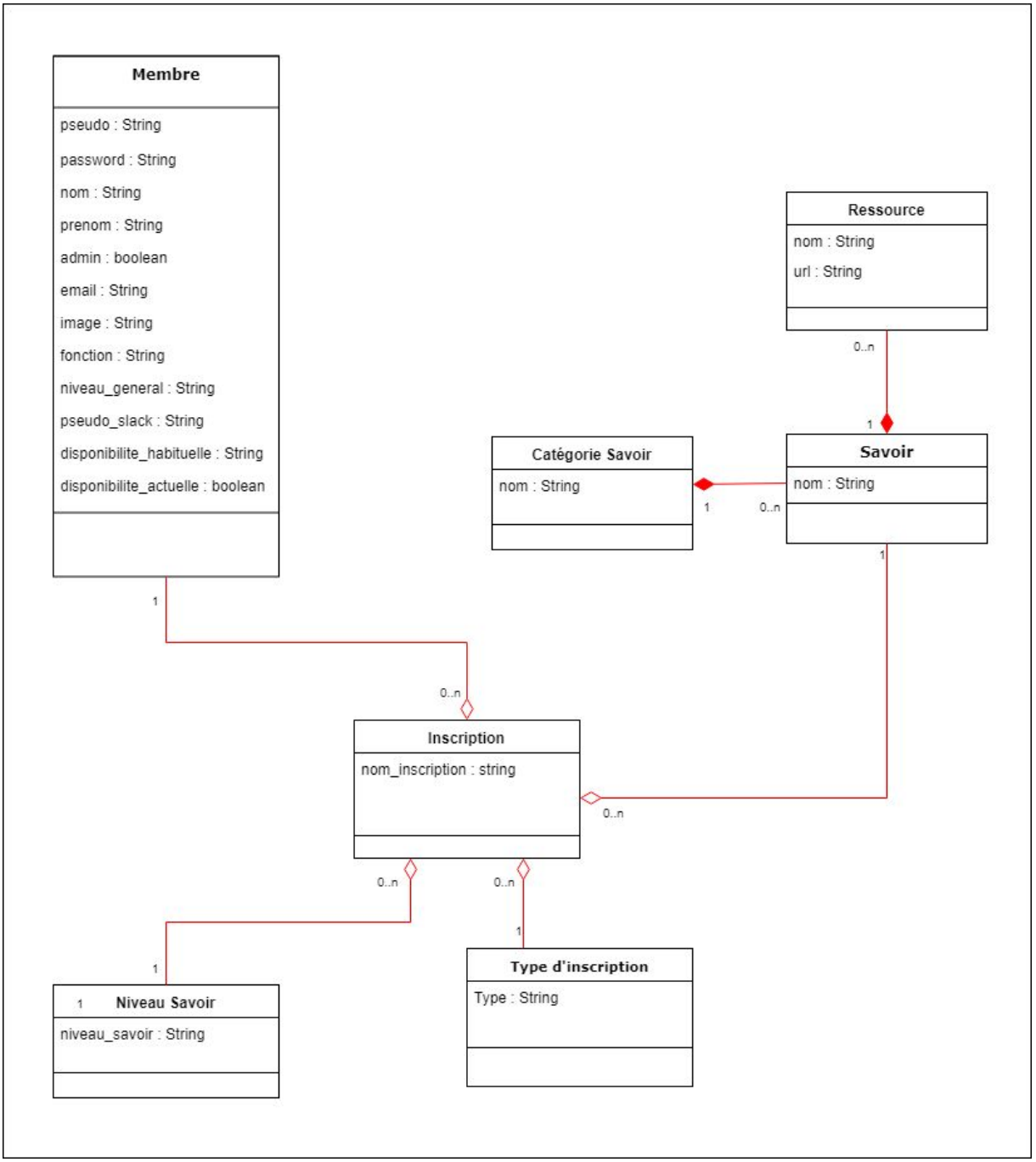
Celui-ci peut aider à y voir un peu plus clair notamment lorsque les cas d'utilisation y sont des plus complexes. Cela peut même aider à trouver de nouvelles questions auxquelles on n'avait pas pensé jusque-là.



- ➢ Le diagramme est composé d'un point de démarrage, d'un point arrêt et d'action, qui sont représentés par des cercles rouges.
- ➢ L'alternative (losange vert) permet d'indiquer les différents scénarios du cas d'utilisation dans un même diagramme.

❖ *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

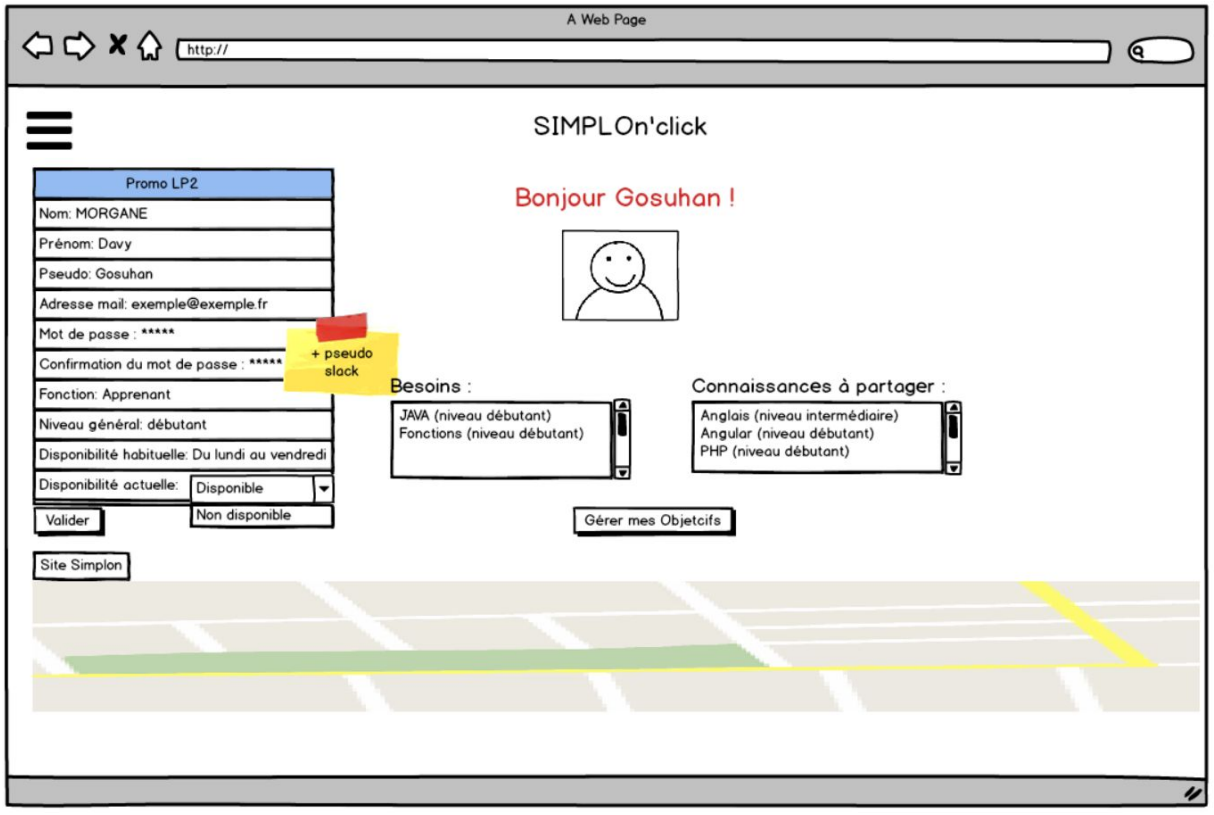
2. Maquettes de l'interface

L'application Simplon'click est une application web de type Single Page Application (application web accessible via une page web unique).

La maquette (créée via le logiciel *Balsamiq*) avait pour objectif de répondre aux exigences du cahier des charges :

- navigation simple, intuitive et adaptée à tout type d'utilisateur
- design moderne et responsive

❖ PAGE D'ACCUEIL (après authentification)

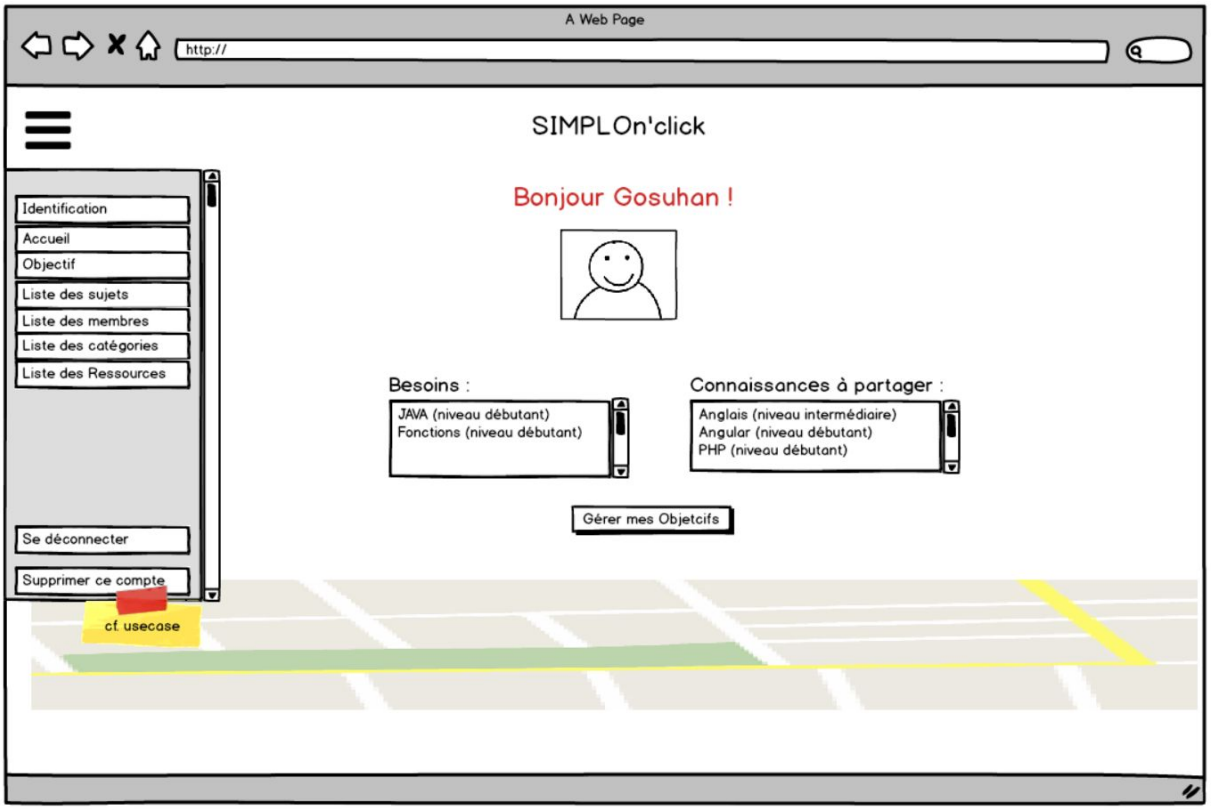


La page d'accueil contient les données de l'**utilisateur connecté** et reprend ses informations via un texte "dynamique".
C'est sur cette page que l'utilisateur vérifie son statut et ses attendus (objectifs).
Elle offre ainsi directement à l'utilisateur la possibilité de gérer ses informations personnelles, de gérer ses objectifs et enfin de visualiser de manière claire et efficace ses objectifs en cours.

a) Header

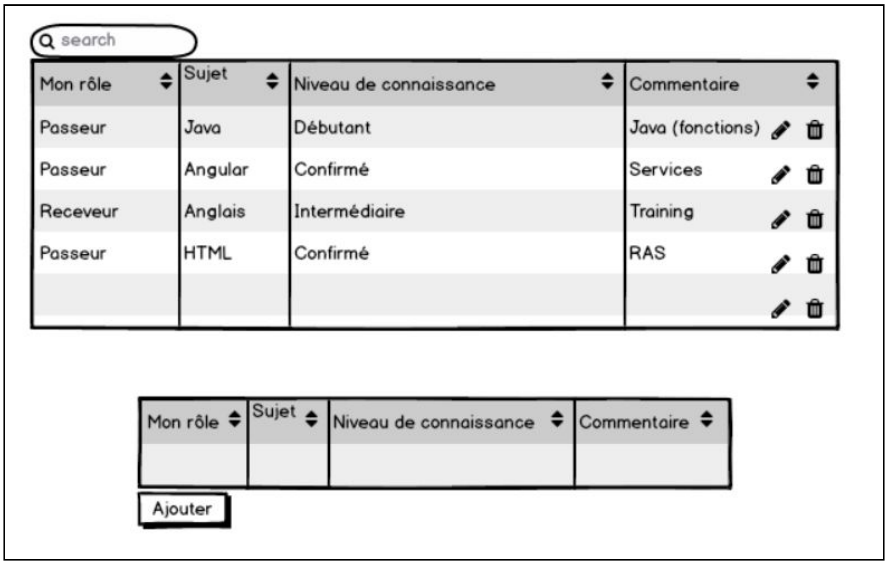
Le header contient un bouton permettant l'accès à un *menu* déroulant, le logo *Simplon'click* (cliquable pour un accès rapide vers l'accueil) et un bouton de *déconnexion*.

- *Le Menu déroulant*



Situé à gauche du header, le menu permet l'accès aux pages principales de l'application comme indiqué explicitement ci-dessus.

b) Body (bouton "Gérer mes objectifs")



Ce bouton permet, comme son nom l'indique, d'accéder à la **gestion des objectifs** de l'utilisateur.

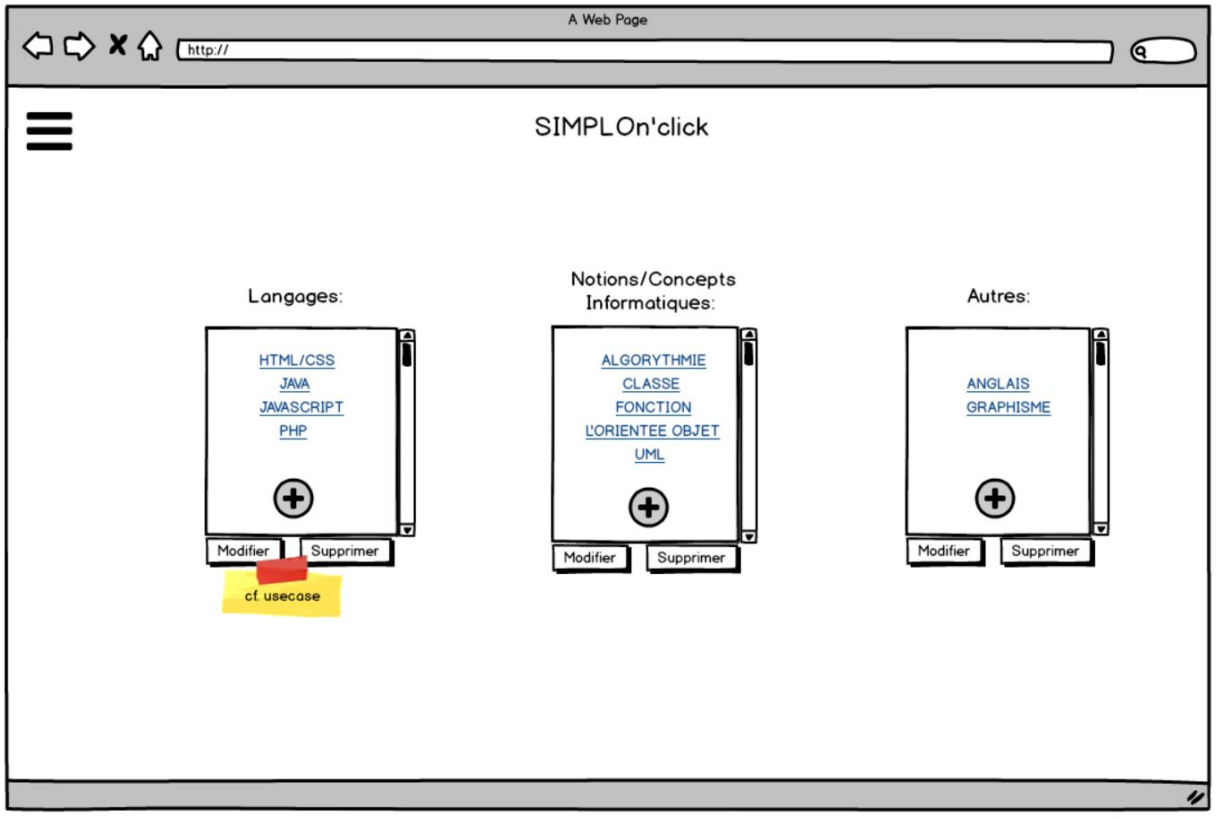
Un objectif est alors défini par :

- ➢ Le **rôle** de l'utilisateur
- ➢ Le **sujet** concerné
- ➢ Le **niveau de connaissance** attendu (passeur) ou recherché (receveur)
- ➢ Un **commentaire** (précision si nécessaire)

c) Footer

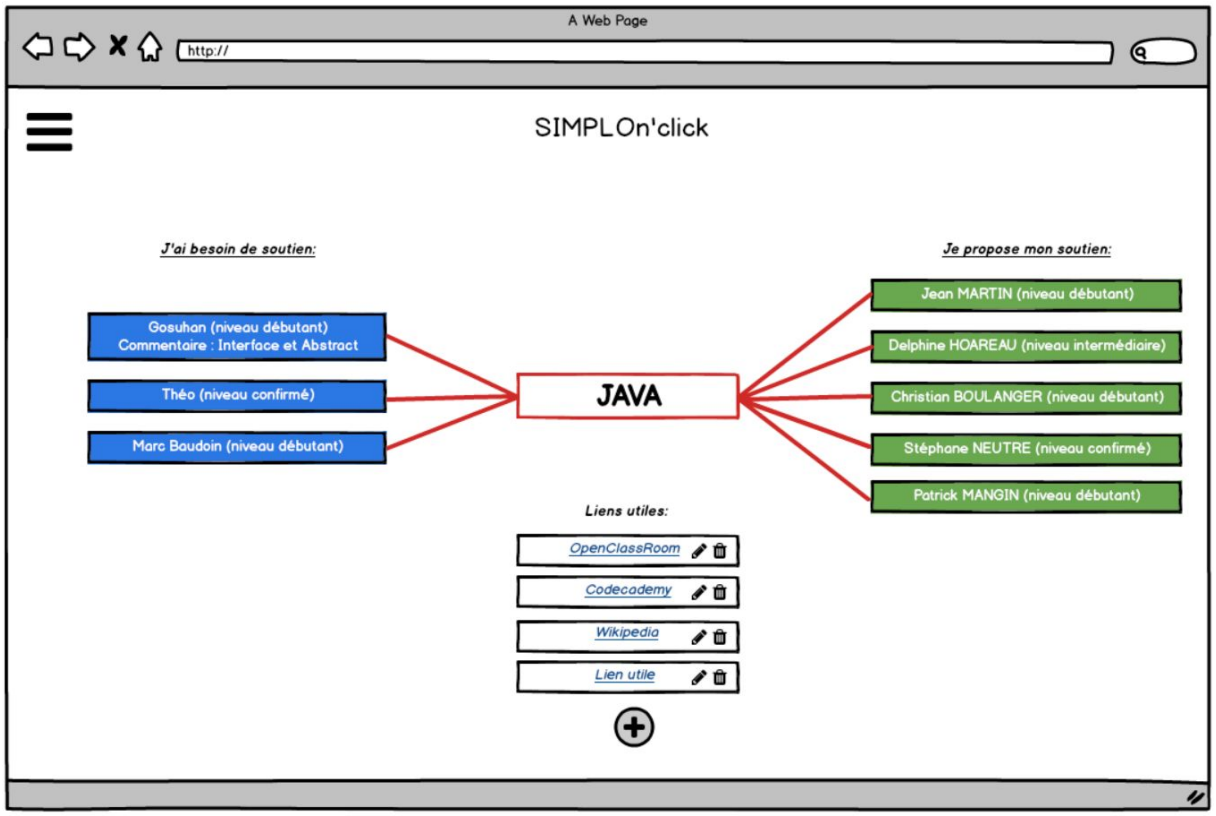
Le footer reprend simplement les institutions concernées de la formation Concepteur/Développeur.

❖ PAGE "LISTE DES SUJETS"



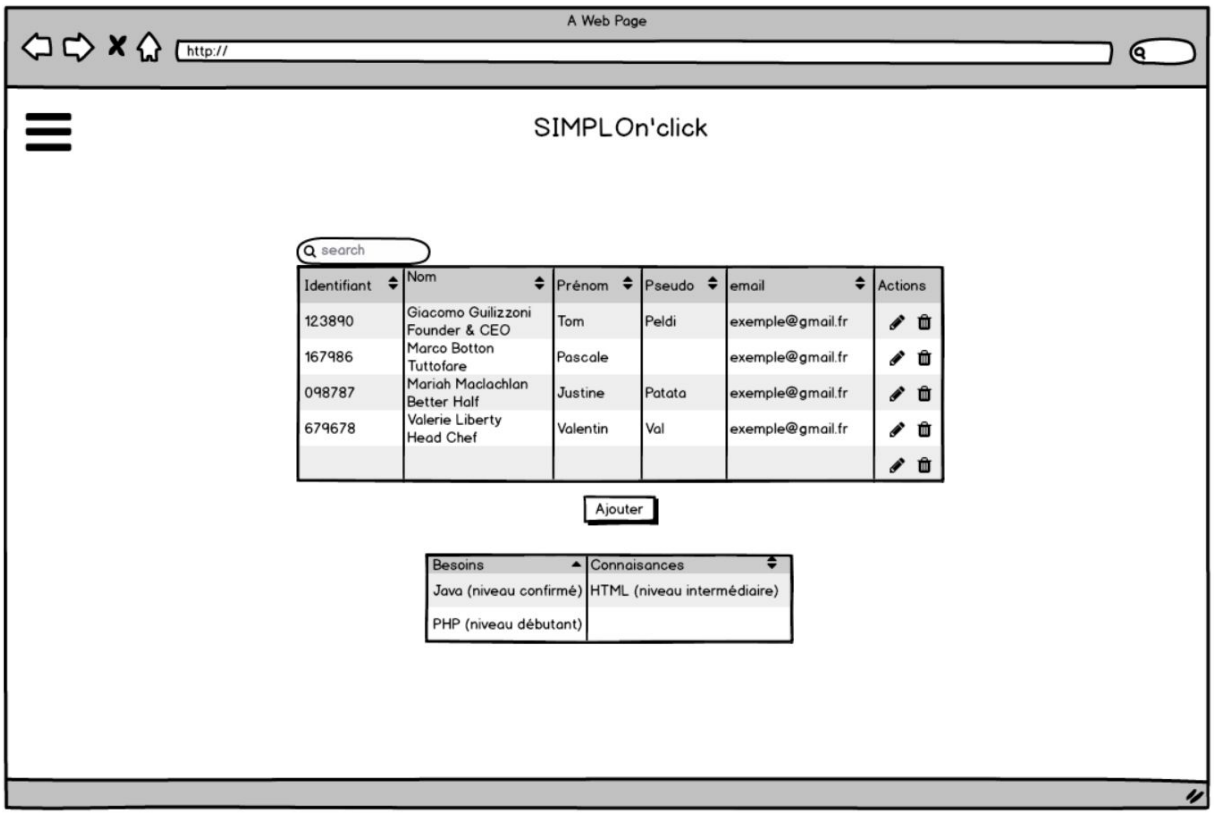
Cette page permet de **consulter** et de **gérer** la liste des **savoirs** (sujets).
On peut alors **ajouter**, **modifier** ou encore **supprimer** un **savoir**.

Chaque savoir est alors cliquable afin de consulter les membres inscrits et ce, avec la précision de leur niveau :



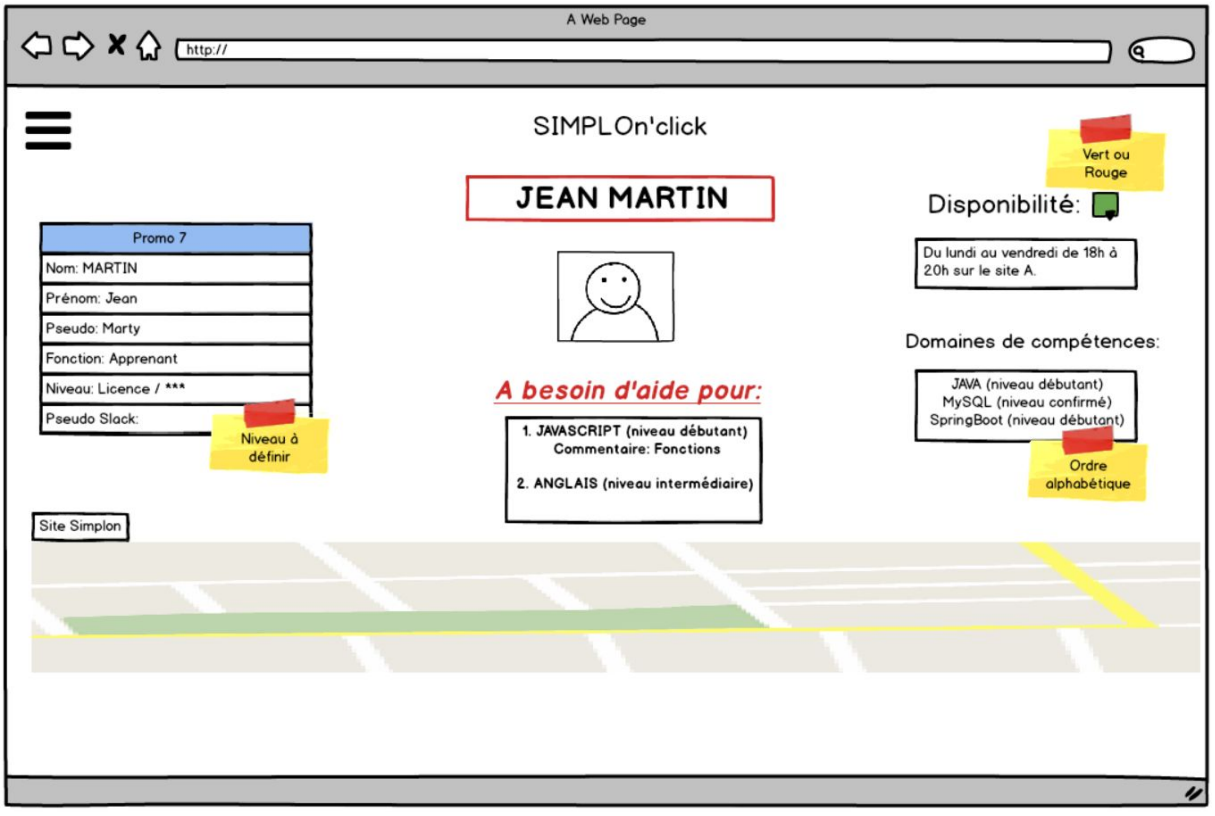
De plus, on y observe toutes les **ressources** liées au Sujet sélectionné en permettant leur **gestion (ajout, modification, suppression)** si nécessaire.

- **PAGE “LISTE DES MEMBRES”**



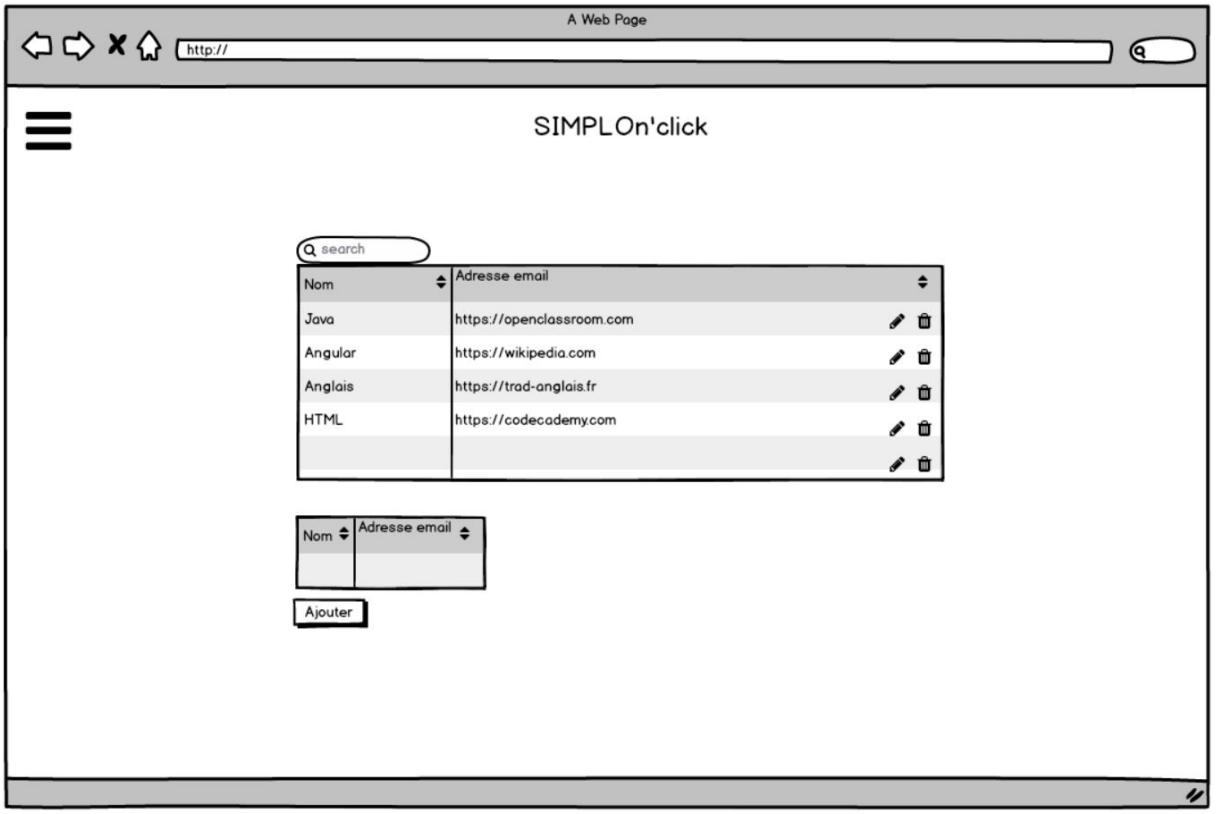
Cette page récapitule tous les membres inscrits sur Simplon-click.

A partir de là, il est possible (liens) de **consulter les détails utiles d'un membre** :



Et d'en **vérifier ses objectifs**.

- **PAGE "LISTE DES RESSOURCES"**



Cette page a pour utilité la **consultation** et la **gestion (ajout, modification, suppression)** du catalogues des **ressources**.

SPÉCIFICATIONS TECHNIQUES

1. ARCHITECTURE LOGIQUE

a) Description des fonctionnalités

L'application Simplon'click permet aux utilisateurs de se mettre facilement en relation.
C'est une application web qui doit être accessible aussi bien depuis un pc qu'un mobile.

b) Les acteurs

Tout internaute peut s'inscrire à l'application Simplon'click et, une fois connecté, utiliser les fonctions qu'elle propose.

c) Les flux

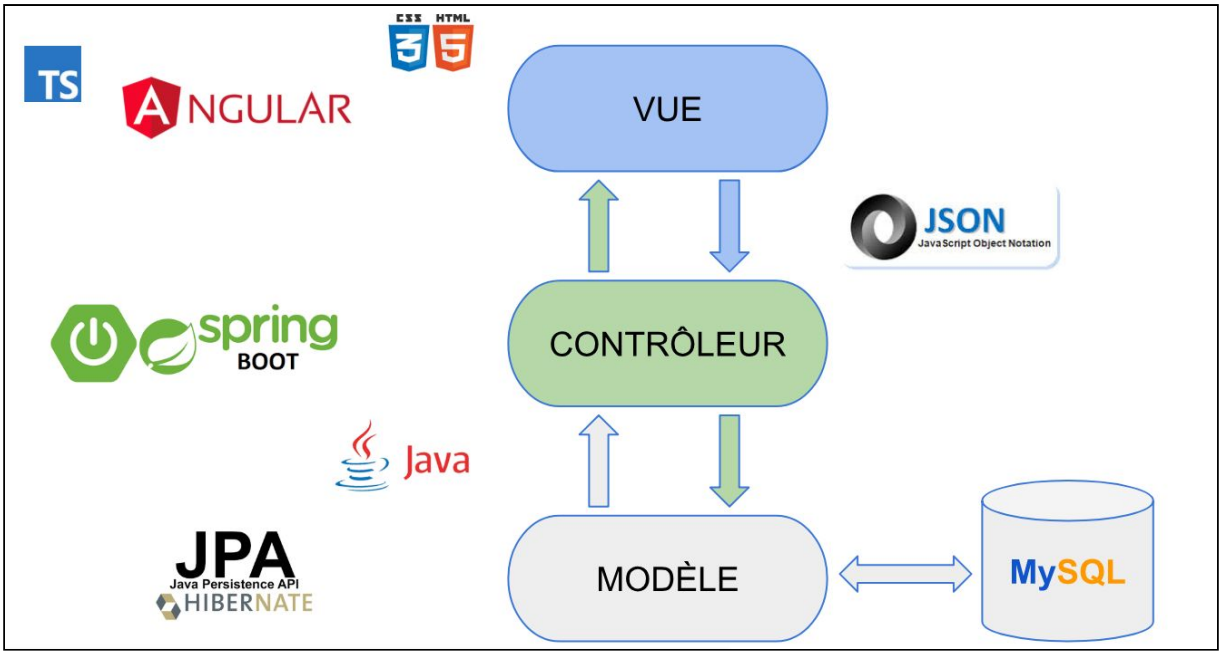
Les flux sont uniquement entrants, des utilisateurs vers l'application :

- ➢ Consultation, création, édition et suppression de données, filtrées en fonction de leur famille.

d) Exigences de services

L'application doit être disponible en 24/7, avec de bonnes performances de chargement sur mobile

2. ARCHITECTURE TECHNIQUE



a) Description

Le choix a été porté sur le développement d'une application **REST (Representational State Transfer)**. REST est une architecture logicielle très courante dans les applications web, qui utilise les standards du protocole HTTP (verbes GET, POST, PUT et DELETE).
Sa particularité est qu'elle ne garde pas les états, donc chaque requête qui lui est envoyée doit contenir toutes les informations nécessaires au traitement de la demande.

Les technologies **Java (Spring Boot)** et **Angular** ont été choisies car ces technologies répondaient à mes attentes. Spring Boot a l'avantage d'embarquer un serveur Tomcat et ainsi, permettre l'autonomie du Backend. Alors que Angular a des outils adaptés pour le côté responsive de mon application.

Rappelons que Angular est un framework **TypeScript** qui lui-même est un langage de programmation "amélioré" de Javascript. Le typage y est géré différemment et l'on observe la possibilité de gérer des classes à l'instar de Java.

Hibernate quant-à lui traduit les tables de la base de données **MySQL** en objet Java, et réciproquement.

Aussi, le transfert de données entre le Backend et le Frontend est communiqué via la structure de données "Json" (JavaScript Object Notation).

b) Composants

MVC
L'utilisation du pattern MVC (Model Vue Controller) m'est préféré, ainsi le Frontend possède un développement séparé du Backend :

- Le *modèle* : Les données ainsi que leur logique (lecture, manipulation, validation, enregistrement)
- La *vue* : Les éléments visuels de l'interface graphique et la logique permettant d'afficher des données provenant du modèle
- Le *contrôleur* : La logique permettant de traiter les actions de l'utilisateur et de modifier les données du modèle et de la vue en conséquence

Base de données

L'utilisation d'une base de données MySQL a été choisie par mes soins, étant open source et très répandue.

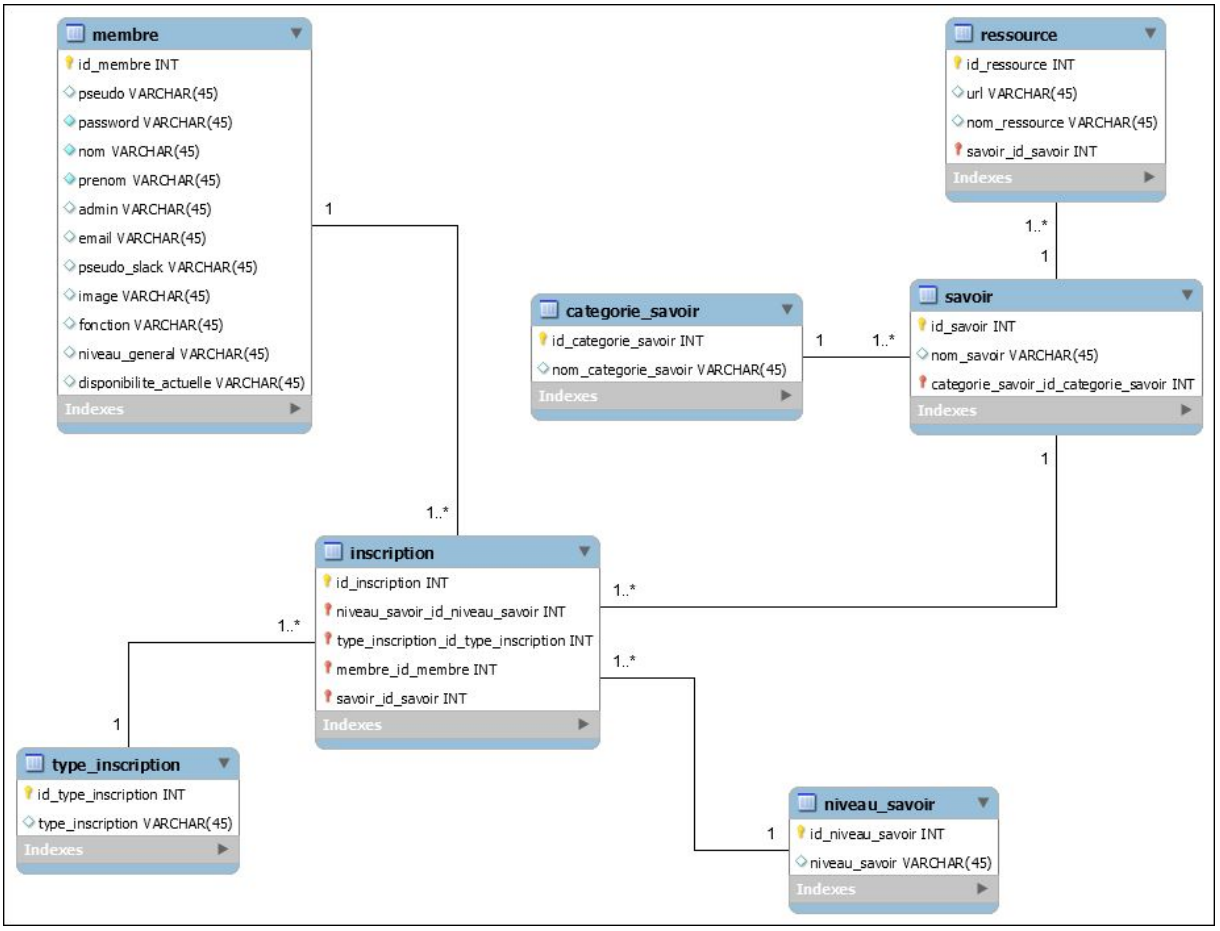
c) Flux et protocoles

Flux	Protocole
Consultation, création, édition et suppression de données par l'utilisateur	**HTTP**
Lecture/écriture des données MySQL	**SQL**

3. MISE EN PLACE DE LA BASE DE DONNÉES

a) Schéma

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.

b) Structure

La base de données a été réalisé dans un premier temps “manuellement” (via MySQL Workbench), en s’appuyant sur le modèle ER (*Entity-Relationship*).

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id_membre	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
admin	BIT(1)	☐	☑	☐	☐	☐	☐	☐	☐	
disponibilite_actuelle	BIT(1)	☐	☑	☐	☐	☐	☐	☐	☐	
disponibilite_habituelle	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
email	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
fonction	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

id_inscription	BIGINT(20)	☑	☑	☐	☐	☐	☐	☑	☐	
nom_inscription	VARCHAR(255)	☐	☐	☑	☐	☐	☐	☐	☐	NULL
membre_id_membre	BIGINT(20)	☐	☐	☐	☐	☐	☐	☐	☐	NULL

Nous remarquons :

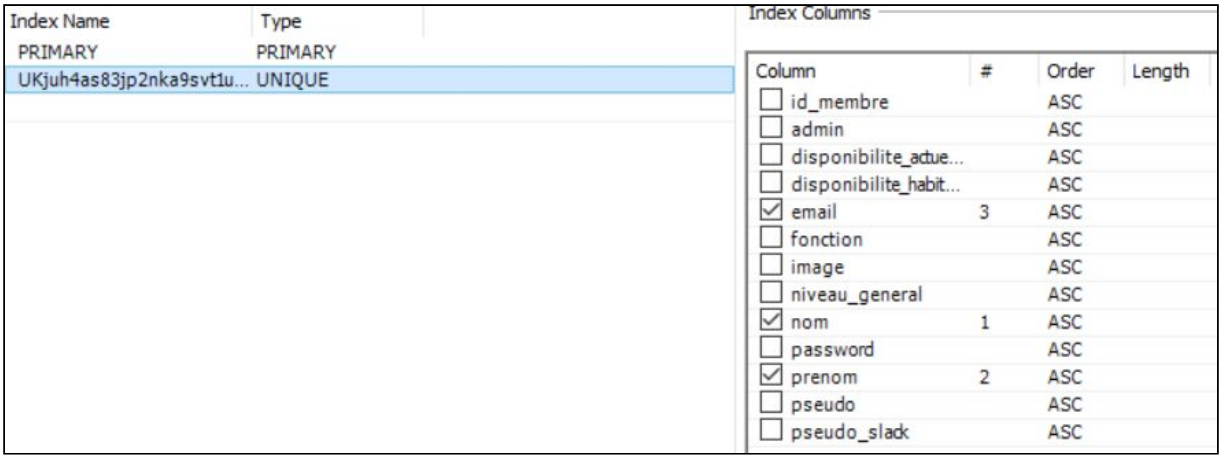
- La clé primaire auto-incrémentée (PK, AI)
- L'impossibilité de certains champs à être définis comme étant "null" (NN)
- La précision d'une valeur par défaut pour d'autres (NULL)
- L'impossibilité d'avoir des doublons sur certains champs (UQ)
- Les clés étrangères liant les tables entre elles (membre_id_membre)

Cependant, pour plus de praticité, d'efficacité et afin d'utiliser notamment la puissance de SpringBoot, la base de données s'est vue développée par la suite directement dans le Back-End, avec une intégration d'un fichier "data.sql" à des fins d'exemples et de tests multiples :

```
23 @Table(name = "membre",
24 uniqueConstraints=
25 @UniqueConstraint(columnNames={"nom", "prenom", "email"}))
26
27 public class Membre {
28
29     @Id
30     @GeneratedValue(strategy = GenerationType.AUTO, generator="native")
31     @GenericGenerator(name = "native", strategy = "native")
32
33     private Long id_membre;
34     private String pseudo;
35     private String password;
36     private String nom;
37     private String prenom;
38     private boolean admin = false;
39     private String email;
40     private String pseudo_slack;
41     private String image;
42     private String fonction;
43     private String niveau_general;
44     private String disponibilite_habituelle;
45     private boolean disponibilite_actuelle = false;
46
47     @OneToMany(cascade = CascadeType.ALL/*, fetch = FetchType.LAZY*/, mappedBy = "membre")
48     @JsonIgnoreProperties(value = {"membre"}, allowSetters = true) // Evite la 'lecture' infinie
```

Aussi, nous remarquons l'annotation *@OneToMany* gérant les liens existants entre les différentes tables (clés étrangères).
Ou encore, afin de gérer les contraintes d'unicité (éviter des doublons selon de multiples critères), l'utilisation de l'annotation *@UniqueConstraint* exécutant la commande SQL :

***ALTER TABLE** `membre` **ADD UNIQUE** `unique_index` (`nom`, `prenom`, `email`)*



c) Données

Ci-dessous, un exemple de données insérées afin de pouvoir tester efficacement les fonctionnalités du Frontend :

id_ressource	nom_ressource	url	savoir_id_savoir
1	OpenClassRoom	https://www.ressource1@openclassroom.com	1
2	Codecademy	https://www.ressource2@codecademy.com	2
3	Wikipedia	https://www.ressource3@wikipedia.com	6

4. DÉVELOPPEMENT DU BACK-END

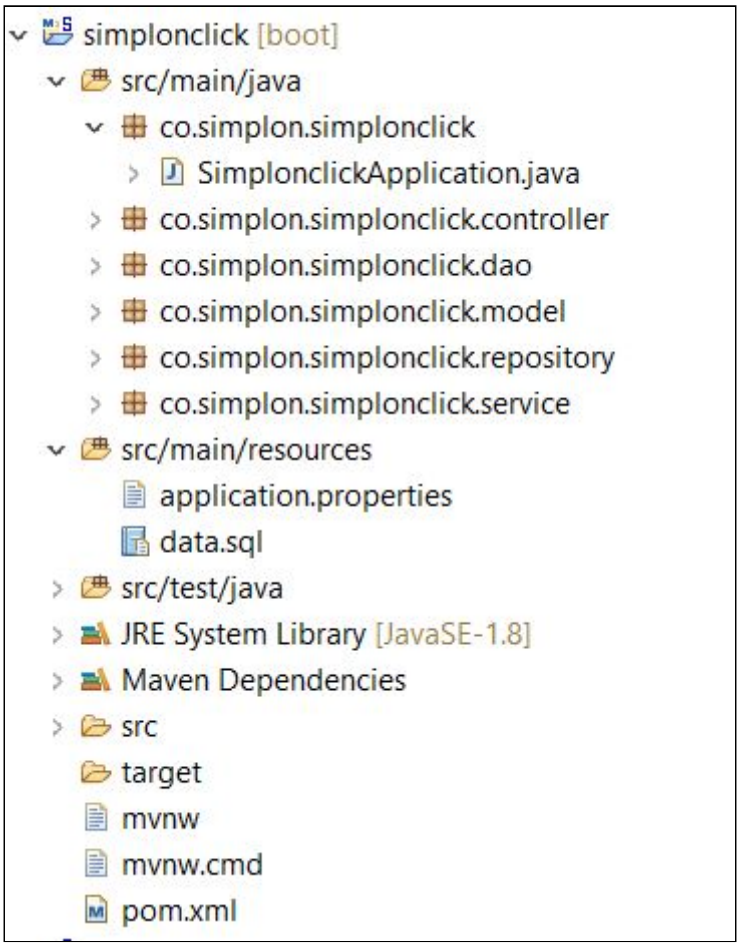
Le choix personnel du langage de programmation a été porté sur **Java**. Ce langage (étudié durant le temps de formation) n'étant pas utilisé par mon entreprise, il m'a semblé intéressant de l'utiliser afin de conserver et d'approfondir mes connaissances et compétences dans celui-ci lorsque des modifications 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 (model, repository, DAO, controller 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.

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.

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.

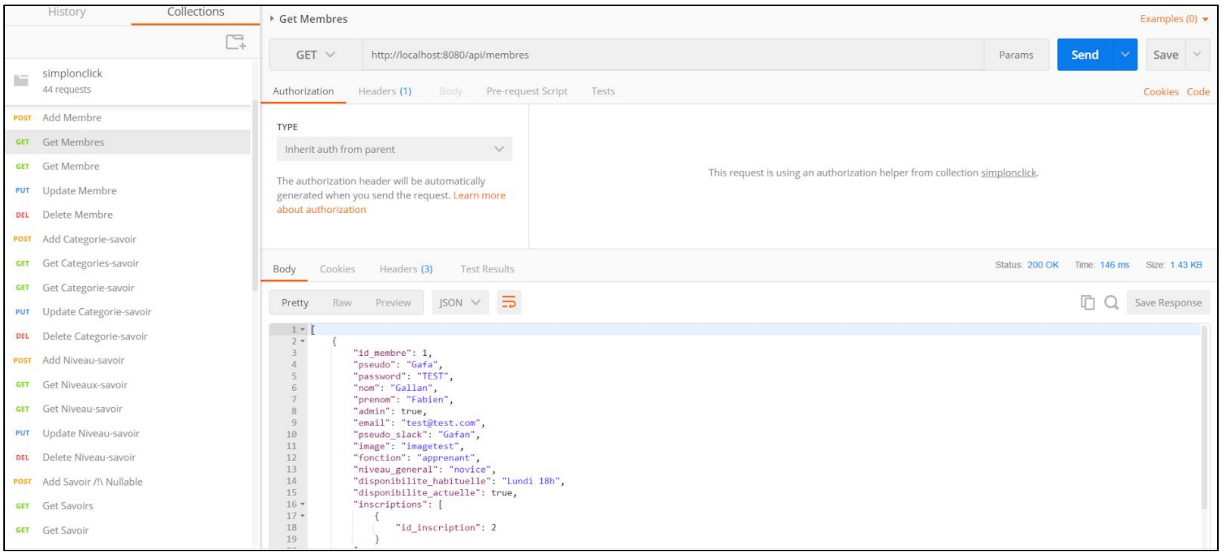
```
package co.simplon.simplonclick.model;

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) Tests

Tests manuels

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

Les tests unitaires permettent de vérifier que :

- L'application respecte le cahier des charges
- Les évolutions ultérieures apportées à l'application n'entraînent pas de régression

C'est **JUnit,** le framework de test unitaire pour le langage de programmation Java qui est utilisé ici.

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class SimplonclickApplicationTests {

    static Savoir savoir;
    static Savoir newSav;
    static ResponseEntity<?> newSavoir;
    static SavoirService savoirService;

    @Autowired
    SavoirDAO savoirDAO;

    @BeforeClass
    public static void initSavoir() throws Exception{
        savoirService = new SavoirService();
        savoir = new Savoir();
    }

    @Autowired
    private SavoirRepository savoirRepository;

    @Transactional
    @Rollback(true)
    @Test
    public void testUpdateSavoir() {

        newSavoir = null;
        savoir = createMock("Angular");

        try {
            newSav = savoirRepository.save(savoir);
        } catch (Exception e) {
            e.printStackTrace();
        }
        assertTrue(newSavoir != null);
        assertEquals("Angular V", newSav.getNom_savoir());
    }
```

- L'annotation *@SprinbBootTest* renseigne la classe qui lance l'application
- Chaque test est identifié par l'annotation *@Test*
- L'annotation *@Rollback(true)*, combinée à *@Transactional* (du framework Spring), permet d'annuler les modifications effectuées dans la base de données
- Les méthodes *assert* sont fournies par JUnit

5. DÉVELOPPEMENT DU FRONT-END

Ma décision a été de développer une Single Page Application (SPA), légère et ergonomique, destinée aussi bien aux mobiles qu'aux pc et ce, en utilisant la plate-forme de développement Angular.

a) Angular

Angular est une plate-forme d'application Web open-source basée sur TypeScript.

NodeJs

Node.js est un environnement bas niveau permettant l'exécution de JavaScript côté serveur.

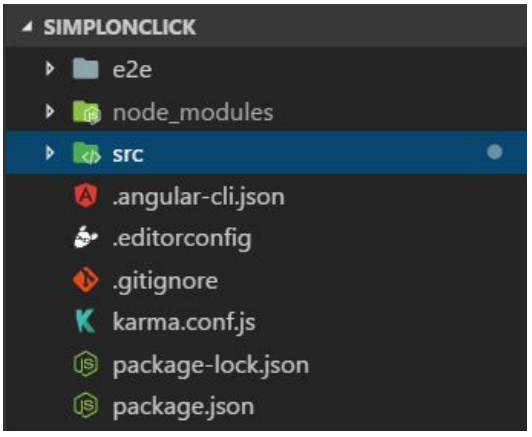
NPM *(npm install -g npm@latest)*

NPM est un package manager qui permet l'installation d'outils et de librairies nécessaires pour tout type de développement.

Angular/CLI *(npm install -g @angular/cli)*

Donne accès à la commande "ng" depuis la ligne de commandes et ce, à partir n'importe quel dossier de l'ordinateur.

New Project



Les dossiers de l'application de base contiennent :

- *node_modules* : Les dépendances qui seront ajoutées manuellement
- *src* : les sources (nos modules/composants)
- *package.json* : la configuration de l'application

Ensuite, il a fallu ajouter les dépendances dont j'avais besoin *(npm install --save @angular/material)*.

On les retrouve alors dans le fichier package.json :

```json
{
  "name": "simplonclick",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build --prod",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^5.2.9",
    "@angular/cdk": "^5.2.4",
```

Angular Material

Angular Material est un framework CSS moderne et responsive, qui propose tous les outils correspondant à mes besoins : menus déroulants, formulaires, tableaux etc.

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

Composants

Angular découpe la structure du Frontend en composants, en les combinant pour former des modules indépendants et réutilisables.

Ainsi, chaque bouton que l'on retrouve dans la page d'accueil d'un membre (Gérer mes Objectifs) est un composant

Membre.css

```css
img {
    position: absolute;
    left: 60px;
    top: 5px;
    width: 70px;
    height: auto;
}

.background {
    background-color: #e3e2d5;
}
```

Représente la partie CSS (Style) du composant Membre

Membre.html

Représente la partie HTML (Template) du composant Membre

```html
<div>
  <mat-card class="background">
      <div mat-card-avatar class="example-header-image">
          <img mat-card-image src="{{memb.image}}" alt="Avatar">
      </div>
      <mat-card-header>
          <mat-card-title>{{ memb.nom }} {{ memb.prenom }}</mat-card-title>
          <mat-card-subtitle>{{ memb.email }}</mat-card-subtitle>
      </mat-card-header>
      <mat-card-content>
        <h3>Bonjour {{ memb.prenom }}</h3>
```

Ici, nous trouvons la syntaxe pour l'interpolation : les doubles accolades {{ }} . Ce qui se trouve entre ces doubles accolades correspond à l'expression TypeScript que nous voulons afficher, l'expression la plus simple étant une variable.

Rappelons que l'intérêt d'utiliser Angular est de pouvoir gérer le DOM (Document Object Model : les éléments HTML affichés par le navigateur) de manière dynamique, et pour cela, il faut utiliser la liaison de données, ou "databinding".

Le databinding est la communication entre le code TypeScript et le template HTML qui est montré à l'utilisateur. Cette communication est divisée en deux directions :

- Les informations venant du code qui doivent être affichées dans le navigateur, comme par exemple des informations que le code a calculé ou récupéré sur un serveur. Les deux principales méthodes pour cela sont le "string interpolation" (cf. image ci-dessus) et le "property binding" (ci-dessous) ;

```html
<button *ngIf="!edition" mat-raised-button color="primary" type="submit" [disabled]="!inscriptionForm.valid">
```

La liaison par propriété ou "property binding" est une autre façon de créer de la communication dynamique entre le code TypeScript et le template : plutôt qu'afficher simplement le contenu d'une variable, il est possible de modifier dynamiquement les propriétés d'un élément du DOM en fonction de données dans le code TypeScript.

La propriété *disabled* (ci-dessus) permet de désactiver le bouton. Afin de lier cette propriété au TypeScript, il faut la mettre entre crochets [] et l'associer à la variable concernée.

Notons aussi la présence de la directive structurelle *ngIf (simple condition, comme son nom l'indique).

- Les informations venant du template qui doivent être gérées par le code : l'utilisateur a rempli un formulaire ou cliqué sur un bouton, et il faut réagir et gérer ces événements. On parlera alors de "event binding".

Membre.ts

Représente la partie TypeScript du composant Membre

```typescript
import { Component, OnInit, ViewChild, OnDestroy } from '@angular/core';
import {
  MatTableDataSource,
  MatDialog,
  MatDialogConfig,
  MatSort,
  MatSnackBar
} from '@angular/material';

import { Subscription } from 'rxjs/Subscription';
import { Imembre } from '../imembre';
import { MembreService } from '../membre.service';
import { InscriptionService } from './../inscription.service';
import { ActivatedRoute, ParamMap } from '@angular/router';
import { Iinscription } from '../iinscription';
import { InscriptionAuMembreComponent } from '../inscription-au-membre/inscription-au-membre.component';
import { ItypeInscription } from '../itype-inscription';
import { TypeInscriptionaInscriptionComponent } from '../type-inscriptiona-inscription/type-inscriptiona-inscription.component';
import { NiveauSavoiraInscriptionComponent } from '../niveau-savoira-inscription/niveau-savoira-inscription.component';

@Component({
  selector: 'app-membre',
  templateUrl: './membre.component.html',
  styleUrls: ['./membre.component.css']
})
export class MembreComponent implements OnInit {
  id: number;
  memb: Imembre;
  errText: string;
  selectedRowIndex = -1;
  edition = false;
  edit = false;
  objectif = false;
  inscriptionBoolean = false;
  insc: Iinscription;
  selectedInscription = false;
  typeInsc: ItypeInscription;

```

```
constructor(
  private snackBar: MatSnackBar,
  private route: ActivatedRoute,
  private membreService: MembreService,
  private inscriptionService: InscriptionService,
  public dialog: MatDialog
) {}

displayedColumns = ['type_inscription', 'nom_savoir', 'niveau_savoir', 'nom_inscription'];
dataSourceInscription = new MatTableDataSource();

@ViewChild(MatSort) sort: MatSort;

applyFilter(filterValue: string) {
  filterValue = filterValue.trim(); // Remove whitespace
  filterValue = filterValue.toLowerCase(); // MatTableDataSource defaults to lowercase matches
  this.dataSourceInscription.filter = filterValue;
}

ngOnInit() {
  this.memb = {
        id_membre: null,
        pseudo: '',
        password: '',
        nom: '',
        prenom: '',
        admin: false,
        email: '',
        pseudo_slack: '',
        image: '',
        fonction: '',
        niveau_general: '',
        disponibilite_habituelle: '',
        disponibilite_actuelle: false
      };
  this.route.paramMap.subscribe((params: ParamMap) => {
    console.log( params.get('id'));
    this.id = +this.route.snapshot.paramMap.get('id');
```

Plusieurs points importants :

- Les imports : *Component, Oninit, composants Angular Material etc.*
- *@Component :* Spécifiant le nom des chemins des composants (style, template etc.) du module Membre
- La classe Membre hérite de la classe (interface) Oninit de Angular
- La fonction “ngOnInit()” qui permet d’initialiser les données utilisées par le module

Ci-dessous, le code de la fonction “deleteMembre()” : *cf. event binding*

```
deleteMembre() {
  this.edition = false;
  this.membreService
    .deleteMembre(this.memb.id_membre)
    .subscribe(
      result => {this.afficherMessage('Votre compte a bien été supprimé', ''); }
    );
  this.clearInput();
}
```

On remarque que la fonction fait elle-même appel à une fonction présente dans un autre “composant” (Service).

Un service permet de centraliser des parties du code et des données qui sont utilisées par plusieurs parties de l’application ou de manière globale par l'application entière. Les services permettent donc :

- de ne pas avoir le même code doublé ou triplé à différents niveaux de l'application - cela facilite donc la maintenance, la lisibilité et la stabilité du code.

- de ne pas copier inutilement des données - si tout est centralisé, chaque partie de l'application aura accès aux mêmes informations, évitant beaucoup d'erreurs potentielles.

api.service.ts

```ts
import { Imembre } from './imembre';
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { map } from 'rxjs/operators';
import { IcategorieSavoir } from './icategorie-savoir';
import { Isavoir } from './isavoir';
import { Iressource } from './iressource';
import { Iinscription } from './iinscription';
import { ItypeInscription } from './itype-inscription';
import { IniveauSavoir } from './iniveau-savoir';

@Injectable()
export class ApiService {

URL = 'http://localhost:8080/api';


 constructor(private http: HttpClient) {}

 getMembres() {
   return this.http.get<Imembre[]>(`${this.URL}/membres`);
 }

 getMembre(id) {
   return this.http.get<Imembre>(`${this.URL}/membre/${id}`);
 }
```

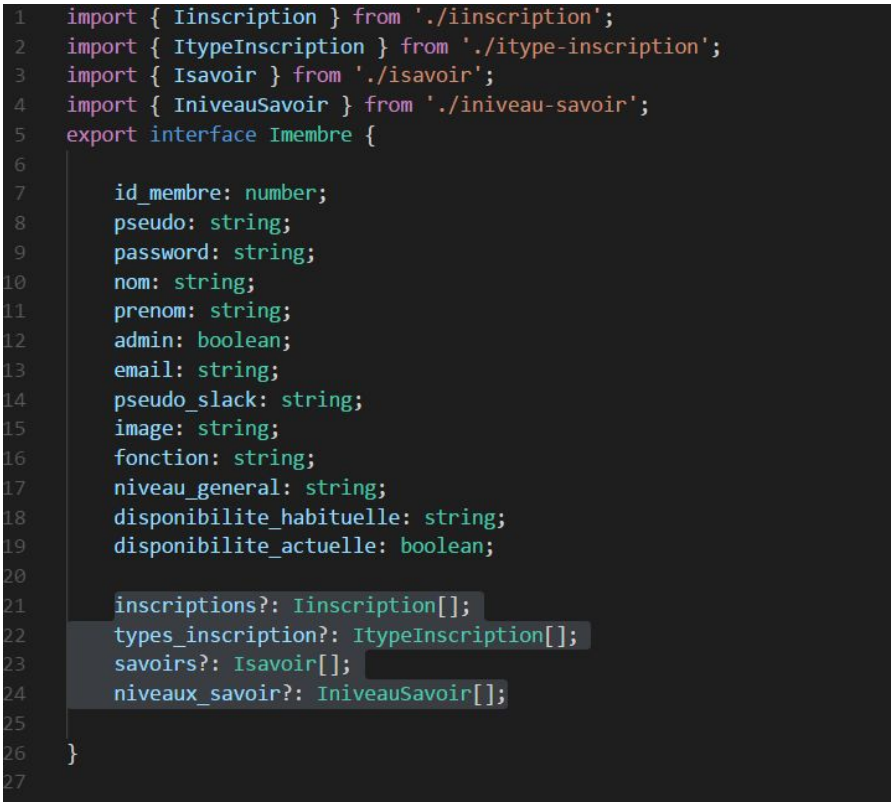
Le Service “Api” est utilisé ici en guise de Contrôleur.
Il précise les adresses URL sur lesquelles l’application sera exécuté ainsi que toutes les méthodes (CRUD) définis dans le Back-End (API REST).

membre.service.ts

```ts
import { Injectable } from '@angular/core';
import { ApiService } from './api.service';
import { Observable } from 'rxjs/Observable';
import { Subject } from 'rxjs/Subject';
import { tap } from 'rxjs/operators';
import { Imembre } from './imembre';

@Injectable()
export class MembreService {
  constructor(private api: ApiService) {}

  update$: Subject<any> = new Subject<any>();

  selectedAffaire: Imembre;

  getMembres(): Observable<Imembre[]> {
    return this.api.getMembres() as Observable<Imembre[]>;
  }

  getMembre(id): Observable<Imembre> {
    return this.api.getMembre(id) as Observable<Imembre>;
  }
```

Les Services peuvent être toutefois aussi dissociés selon les modules afin de rendre le code plus lisible.
Ici, nous constatons l'utilisation d' "Observable" pour faire appel au Service API.
C'est ce qui permet la transmission de messages. Un observable est une technique efficace de gestion d'événements d'une ou plusieurs valeurs.

imembre.ts



```typescript
import { Iinscription } from './iinscription';
import { ItypeInscription } from './itype-inscription';
import { Isavoir } from './isavoir';
import { IniveauSavoir } from './iniveau-savoir';
export interface Imembre {

    id_membre: number;
    pseudo: string;
    password: string;
    nom: string;
    prenom: string;
    admin: boolean;
    email: string;
    pseudo_slack: string;
    image: string;
    fonction: string;
    niveau_general: string;
    disponibilite_habituelle: string;
    disponibilite_actuelle: boolean;

    inscriptions?: Iinscription[];
    types_inscription?: ItypeInscription[];
    savoirs?: Isavoir[];
    niveaux_savoir?: IniveauSavoir[];

}

```

Autre composant, l'*interface* qui modélise l'objet (ici, le membre). A noter que les variables suivis d'un "?" les rendent non obligatoires.

LIVRAISON DE L'APPLICATION

a) Ateliers utilisateurs

Des "ateliers" utilisateurs ont été effectués auprès de personnes volontaires. Même si l'application n'était pas terminée au moment des ateliers, leur retour a été très positif :

- application très utile
- interface agréable et intuitive
- simplicité de navigation (discussions sur les versions ultérieures)

b) Build du livrable

Lorsque que le développement de l'application sera terminé, un build sera créé.
Ci-après un **mode opératoire** prévisionnel :

Prérequis (Back)

- Java 1,8
- Maven (+ commandes exécuter/déployer)
- SGBDR MySQL
- schema.sql (création de la base et des tables)
- data.sql (obtention des données)

Modifications nécessaires (Java : /simplonclick/src/main/resources/application.properties)

- spring.datasource.url=jdbc:mysql :
- spring.datasource.username
- spring.datasource.password
- Si besoin de changer le port du serveur Tomcat, ajouter la ligne : server.port: XXXX

Prérequis (Front)

- NodeJS

Lancement

Back

Via GitHub :

1. Cloner le repo
2. Ajouter le projet à votre IDE préférée
3. Faire un MVN Clean Install
4. Faire un Maven Update Project
5. Lancer l'application via : Run as Spring Boot App

Via le fichier jar: /!\ Avec le Jar déjà compilé, aucune modification de la section 2 n'est possible /!\

1. Ouvrir une invite de commande dans le dossier contenant le fichier jar
2. Entrer : java -jar monFichier.jar

Front

1. Cloner le repo
2. cd Projet_Pro_Front/Simplonclick
3. npm install
4. npm install -g @angular/cli
5. ng serve -o
6. ng build (créer le dist/ avec l'index html)

CONCLUSION

La réalisation de ce projet et ce, de la **conception à la livraison finale** m'a permis d'acquérir une **expérience très enrichissante**.

C'est aussi grâce à cette expérience que ma **confiance**, au niveau personnel, s'est vue **améliorée.**

J'ai alors pu **mettre en application** des **notions abordées en cours** ou encore, **me débloquer de certaines situations** au moyen de **conseils reçus** durant les périodes de **stage**.

Également, c'est en rédigeant le **cahier des charges**, en particulier le détail des fonctionnalités, que je me suis rendu compte de son importance. En effet, celui-ci **structure notre travail** de développeur et permet d'éviter tout omission.

Dans une **version 2**, 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 pour le "savoir" (sujet) intéressé
- Une **restriction,** par mesure de sécurité, des **habilitations (**modifications ou ajouts liés au contenu de l'application) et les transmettre à l'administrateur