Fichiers	Fonction	Description
Quizz.js	`cacheTexte(ClassName)`	Fonction pour cacher notamment la page d'accueil de Quizz permettant d'afficher les questions
	`afficheTexte(ClassName)`	Fonction qui permet d'afficher notamment la page d'accueil de Quizz
	`afficherQuizz(event)`	Fonction permettant d'aller chercher les différentes question et réponses dans la Base de données puis d'afficher le bon quizz
	`afficherQuestionsSuivantes()`	Fonction permettant d'afficher la question suivante. S'il n'y a plus de question suivantes affiche qu'il n'y a plus de question
	`retourAuxThemes()`	Fonction permettant de cacher les questions et de réafficher la page de base des quizz par thème
Theme.js	`afficheChapitre(event)`	Fonction permettant d'aller chercher les différents chapitres dans la base de données pour accéder à son contenue
	`goBack()`	Fonction permettant de retourner à l'affichage des titres des thèmes et des chapitres
urgenceQuestionReponses.js	`findNoeudRacine()`	Fonction permettant d'aller chercher le premier nœud
	`doRequestResponse()`	Fonction permettant de faire une requête ajax sur l'api questions en fonction du noeudID
	`showResultResponse(resultJson)`	Fonction qui traite le résultat de la fonction doRequestResponse() et qui permet de créer un template avec Mustache
	`showErrorResponse (xhr, status, message)`	Fonction qui traite les erreurs de la requête doRequestResponse()
	`showResultNoeud(resultJson)`	Fonction qui traite le résultat de la requête de doRequestNoeud()

	`showErrorNoeud (xhr, status, message)`	Fonction qui traite les erreurs de la requête doRequestNoeud()
	`doRequestNoeud()`	Fonction qui envoie la requête ajax en de noeudId
	`doRequestNoeudFils()`	Fonction qui fait une requête pour trouver le nœud fils d'une réponse en fonction de reponseId
	`changeIdNoeud()`	Fonction pour changer de nœud sur la page en fonction de la réponse de l'utilisateur
	`showResultIllustrations()`	Fonction qui permet le trier le résultat de la requête doRequestIllustration(), selon la valeur de positionDessin
	`getIdPositionIllustrations(resultJson)`	Fonction permettant de récupérer les id et positions de la fiche
	`showErrorgetIdPositionIllustrations(xhr, status, message)`	Fonction qui traite les erreurs de la requête doRequestIllustration()
	`doRequestIllustrations()`	Fonction pour faire l'appel AJAX des informations sur une fiche
	`afficheIllustrations()`	Fonction qui permet d'afficher les illustrations puis déclenche showResultIllustration() pour avoir l'affichage dans l'ordre
	`showResultACIllustrations()`	Fonction qui permet le trier le résultat de la requête doRequestACIllustration(aideID), selon la valeur de positionDessin
	`getIdPositionACIllustrations(resultJson)`	Fonction permettant de trouver l'id et l'illustration de la fiche d'aide
	`showErrorgetIdPositionACIllustrations(xhr, status, message)`	Fonction qui traite les erreurs de la requête doRequestACIllustration()
	`afficheACIllustrations()`	Fonction qui permet d'afficher les illustrations puis déclenche showResultACIllustration() pour avoir l'affichage dans l'ordre
	`doRequestACIllustrations(aideId)`	Fonction pour faire l'appel AJAX des informations de la fiche

	`showResultAideComprehen-sion(resultJson)`	Fonction qui traite les résultats de la requête doRequestAideComprehension() et affiche le titre de la fiche
	`showErrorAideComprehen-sion(xhr, status, message)`	Fonction permettant de traiter les erreurs de la fonction doRequestAideComprehension()
	`doRequestAideComprehension(re-sultJson)`	Fonction pour faire l'appel ajax des informations de la fiche
	`gestionAbsenceAideComprehen-sion(xhr, status, message)`	Fonction qui gère la fonction 404 pour éviter de faire crash le site quand on demande une Aide Comprehension inexistante
	`findAideComprehension()`	Fonction permettant d'aller chercher un Aide Comprehension dans une fiche

. A. Modèle de données

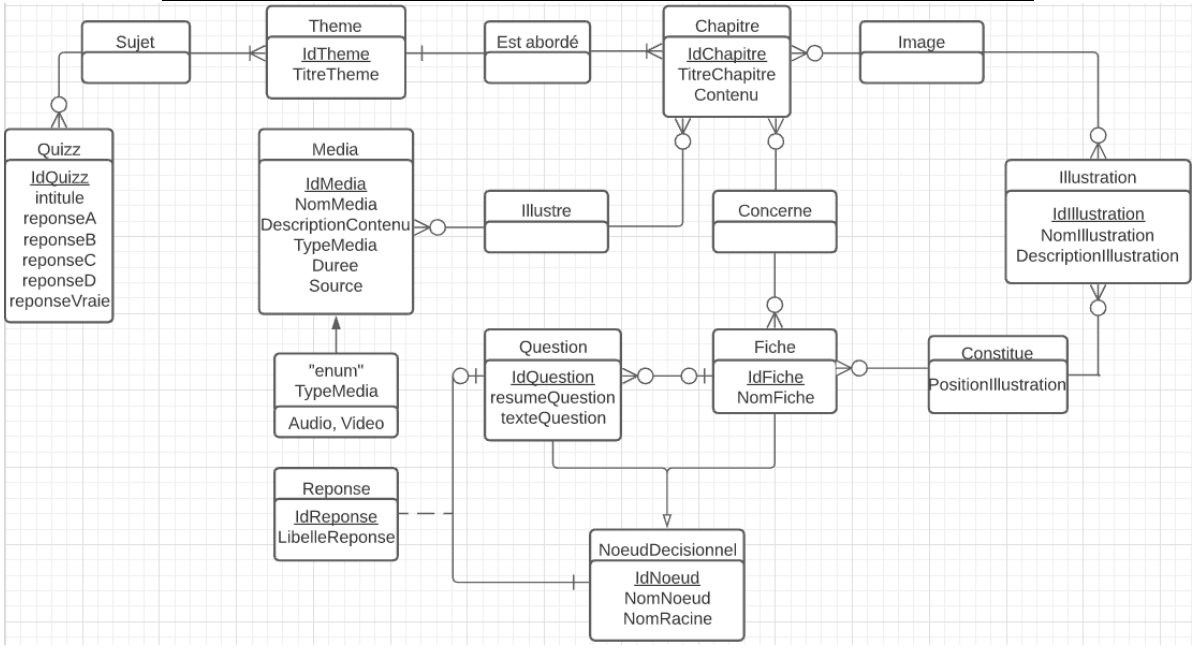
Une base de données est nécessaire pour notre projet, afin de regrouper, organiser et rendre accessibles toutes les données de nature diverses que comptera notre projet.

D'après les objectifs que nous avons fixés dans la partie précédente, nous pouvons dores et déjà définir l'ensemble des entités et des associations qui constitueront notre base de données. En effet, après avoir analysé notre projet, nous obtenons les observations suivantes :

- Le site contiendra des fiches, point central de notre site, constituées le plus souvent par une ou plusieurs illustrations. Ils sont la marche à suivre dans une situation d'urgence précise.
- Chaque fiche concerne un ou plusieurs chapitres, eux-mêmes appartenant à un thème. Chaque chapitre possède un titre ainsi qu'un contenu. Les chapitres seront disponibles dans la partie "Me former".
- Chaque thème peut être abordé par un ou plusieurs chapitres et possède un titre. Chaque thème possède peut également posséder un ou plusieurs quizz associées.
- Chaque quizz, étant sujet d'un thème, est vu comme une question. Elle possède donc un intitulé, plusieurs réponses possibles ainsi qu'une réponse vraie. Ces quizz seront visibles dans la partie "M'entraîner".
- Chaque chapitre pourra aussi être illustré grâce à des médias de type vidéo ou audio permettant de compléter le cours.
- Chaque chapitre pourra également être accompagné d'images, faisant office d'illustrations à la situation concernée par ce chapitre.
- Dans la partie "Urgence", on aura donc affaire à un noeud décisionnel qui, en analysant les réponses données aux questions posées, proposera une fiche d'urgence, expliquant de manière simple et concise les bons gestes à suivre.

Nous pouvons donc proposer le modèle conceptuel de données suivant pour ce projet :

Figure 1 : Modèle Conceptuel de Données du Secouriste de Poche



Ce qui correspond au modèle relationnel suivant :

QUIZZ (IdQuizz, intitule, reponseA, reponseB, reponseC, reponseD, reponseVraie)
SUJET (*IdQuizz, IdTheme*)
THEME (IdTheme, TitreTheme)
CHAPITRE (IdChapitre, TitreChapitre, Contenu, *IdTheme*)
IMAGE (*IdChapitre, IdIllustration*)
ILLUSTRATION (IdIllustration, NomIllustration, DescriptionIllustration)
ILLUSTRE (*IdChapitre, IdMedia*)
MEDIA (IdMedia, NomMedia, DescriptionContenu, TypeMedia, Duree, Source)
CONCERNE (*IdChapitre, IdFiche*)
CONSTITUE (IdIllustration, IdNoeudDecisionnel, PositionIllustration)
NOEUDDECISIONNEL (IdNoeud, NomNoeud, NomRacine, resumeQuestion, texteQuestion, NomFiche, *IdFiche*, Dtype)
REPONSE (IdReponse, LibelleReponse, *IdNoeud, IdQuestion*)

Figure 2 : Modèle Relationnel du secouriste de poche

B. Structure de l'application

On peut décomposer la structure de notre application en différents niveaux.

1. Éléments de base

Avant de passer à des éléments de structure un peu plus complexes, attardons nous un instant sur les éléments de base.

Tout d'abord, notre projet comporte ce qu'on appelle des entités. Ces entités JPA permettent de modéliser le modèle de données vu plus tôt. En effet, les différentes relations ainsi que la création des colonnes peuvent être réalisées de manière assez instinctive. Notre projet comporte donc 10 entités différentes (correspondant bien aux tables du MCV).

Ensuite, notre projet comporte également des "repository" pour chacune des entités JPA. Ces "repository" permettent d'insérer des données dans chacune des tables relationnelles.

Ci-dessous, un exemple d'une entité et de son repository associé.

Figure 3 : Entity Theme.java

```java
package secouristedepoche.entity;
import java.util.LinkedList;
import java.util.List;
import javax.persistence.*;
import lombok.*;


// Un exemple d'entité
// On utilise Lombok pour auto-générer getter / setter / toString...
// cf. https://examples.javacodegeeks.com/spring-boot-with-lombok/
@Getter @Setter @NoArgsConstructor @RequiredArgsConstructor @ToString
@Entity // Une entité JPA
public class Theme {
    @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(unique=true)
    @NonNull
    public String titre;

    @OneToMany(mappedBy = "theme")
    private List<Chapitre> chapitres = new LinkedList<>();

    @ManyToMany (mappedBy = "sujets")
    private List<Quizz> interrogations = new LinkedList<>();

}
```

Figure 4 : Theme.Repository

```java
package secouristedepoche.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import secouristedepoche.entity.Theme;

// This will be AUTO IMPLEMENTED by Spring

public interface ThemeRepository extends JpaRepository<Theme, Integer> {

}
```

2. Contrôleurs

Les contrôleurs sont des outils indispensables pour le développement de notre site web. Ils sont chargés de traiter les requêtes entrantes et de générer les réponses sortantes. Pour notre projet, nous en avons créé trois: EntrainementController, ThemeController et UrgenceController.

Ces contrôleurs nous permettent, grâce aux “entity” et aux “repository” définis plus tôt, de pouvoir changer de page quand nous sommes dans notre site web ainsi que d’afficher les données des tables associées.

Tout ceci est possible grâce aux actions dont sont constitués les contrôleurs. Prenons par exemple l’exemple du contrôleur *ThemeController*:

```java
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import secouristedepoche.dao.ThemeRepository;
import secouristedepoche.entity.Theme;

import lombok.extern.slf4j.Slf4j;

/**
 * Edition des catégories, sans gestion des erreurs
 */
@Controller
@Slf4j
@RequestMapping(path = "/SecouristeDePoche")
public class ThemeController {

    @Autowired
    private ThemeRepository dao;

    @GetMapping(path = "themes")
    public String afficheThemes(Model model) {
        model.addAttribute("themes", dao.findAll());
        return "afficheTheme";
    }
```

Figure 5 : ThemeController

Ici, on voit que *ThemeController* a bien besoin du *ThemeRepository* et de l’entité *Theme* pour que son action *AfficheThemes* puisse bien fonctionner.

3. Templates

Dans le cadre de notre projet, nous avons utilisé les templates afin de créer des pages HTML permettant la présentation de nos données. Nous avons créé trois templates, en plus de notre page HTML d’accueil *index.html* : *afficheEntrainement.html*, *afficheTheme.html* et *afficheUrgence.html.* Ces quatres pages HTML permettent de voir respectivement la page “Accueil”, la page “M’entraîner” , la page “Me former” et la page “Urgence”.

Ces pages sont codées en HTML et les données importées le sont grâce à ThymeLeaf.

4. Aspect Client

Notre projet comporte également un fichier CSS ainsi que plusieurs fichiers JavaScript. Le fichier CSS permet une lisibilité et un esthétisme plus important pour le client tandis qu'un fichier JavaScript permet, à l'aide de fonctions, un changement dynamique de la page html qui lui est associée. Notre projet comporte quatre fichiers JavaScript: Quizz.js, Theme.js, main.js, urgenceQuestionsReponses.js reliant respectivement les pages HTML *afficheTheme.html, index.html afficheTheme.html* et *afficheUrgence.html.*

C. Bilan du projet

1. "ToDo", fonctionnalitées restant à développer

Globalement, notre projet possède toutes les fonctionnalités prévues initialement : une page d'accueil fonctionnelle permettant d'appeler les secours en appuyant sur une photo, une page urgence affichant en fonction des réponses données les fiches correspondantes, une page de formation regroupant par thème des cours sur les gestes de premiers secours et une page de quizz permettant de tester nos connaissances sur les cours reçus précédemment.
Néanmoins, quelques améliorations permettraient un meilleur fonctionnement de notre projet. Tout d'abord, nous pensons que coder la majorité des fonctions à l'aide de "mustache" permettrait d'avoir moins de code sans que cela soit trop compliqué. De plus, on pourrait rajouter des vidéos, images ou fiches dans la partie "Me former", afin que la formation soit plus complète. Toujours dans cette partie, nous pensions également à un système permettant de filtrer les chapitres que l'on souhaite consulter, en placer certains en favoris ou encore y ajouter des commentaires pour poser des questions, proposer des modifications ou apporter des précisions pour les autres utilisateurs. Ensuite, dans la partie Quizz, nous pensons qu'il serait possible de le rendre un peu plus complexe, en créant un système de compteur permettant de compter nos points par exemple. De plus, dans la page urgence, un bouton permettant de revenir à la question précédente pourrait être envisageable au cas où l'utilisateur se serait trompé ou en cas de changement de la situation sur place. Enfin, nous avons essayé de coder en responsive afin que le site soit lisible et facile d'utilisation sur mobile mais nous n'avons pas réussi à mener ce projet à bout. Sachant que ce site a de forte chance d'être plus utilisé sur mobile, c'est une piste à aborder qui nous semble essentielle.

2. Problèmes rencontrés

La création des nœuds s'est révélée très compliquée. En effet, il a fallu intégrer un moteur de template "mustache" afin que cela soit réalisable. Il a donc été nécessaire de s'approprier le fonctionnement de ce moteur de template afin de s'assurer du bon déroulement de cette partie centrale de notre projet.

De plus, l'affichage des chapitres sous les thèmes à l'aide de ThymeLeaf dans la partie "Me former" nous a initialement posé problème.

Ensuite, il faut mentionner la création de l'entité Noeud_Decisionnel_Dessin qui a modifié la structure de l'API.Il a donc fallu faire une modification profonde et difficile de urgenceQuestionsReponses.js.

Quelques autres problèmes liés au GitHub ont également perturbé notre projet ainsi que notre méconnaissance de l'existence et de notre capacité à pouvoir consulter des API auto-générées et de la console H2. Lorsque nous avons su cela, le travail a donc pu être fait beaucoup plus facilement et rapidement

3. Problèmes non résolus

Malgré une bonne avancée de notre projet, quelques problèmes n'ont pas pu être résolus. Tout d'abord, malgré tous nos efforts, nous n'avons pas réussi à mettre en place un menu burger seulement en mode mobile. De plus, la fonction permettant d'afficher les quizz connait quelques petits bugs. Il serait donc intéressant de trouver comment les résoudre. Enfin, nous avons réussi à coder en responsive sur deux des quatre pages HTML mais pour une raison que l'on ignore, ce code ne fonctionne pas sur les deux autres pages.

4. Avis et Conclusion

Nous sommes plutôt satisfait de notre projet compte tenu du fait qu'il possède toutes les caractéristiques que nous voulions. Globalement, beaucoup des "ToDo" sont des fonctionnalités faisables mais auxquelles nous n'avons pensé que trop tard, ne nous permettant pas de les finir dans les temps.