Rapport

Technologie Objet & Gestion de Projet
Navigateur Web

Groupe 5

Adrien CAUBEL
Guillaume CHAUDON
Simon JANDA
Simon MOULIN
Victor NANCHE

1ère année par apprentissage
Promotion 2023

INP TOULOUSE ENSEEIHT

Table des matières

1 Introduction

1.1 Le projet

Ce projet s'inscrit dans le cadre des unités d'enseignement Technologie Objet et Gestion de Projet dispensées durant le second semestre du cycle ingénieur. L'objectif principal de se projet est de mettre en application les principes agiles vus en Gestion de Projet. Tandis que l'implémentation de l'application est un objectif secondaire car les notions de programmation ont été travaillé en TDs et TPs. Il était donc conseillé d'avoir un projet ambitieux qui serait difficile à réaliser dans son intégralité dans le temps imparti.

Pour ce projet, nous avons donc décidé de créer un navigateur web.

1.2 Liens utiles

Vous pouvez suivre l'avancement du projet avec le :

- Github https://github.com/simjnd/navigateur/wiki
- Trello https://trello.com/b/HyinoLEE/navigateur
- SVN http://cregut.svn.enseeiht.fr/2020/1air/cpo/pl/E5

2 Orginisation de l'équipe

2.1 Itération initiale

Cette première itération a été consacré à l'organisation du travail. Ainsi, nous nous sommes mis d'accord sur les outils à utiliser. Nous avons donc créé un tableau Trello pour maintenir et mettre à jour le backlog produit, les sprints et identifier le travail à réaliser, le travail en cours, les tâches terminées (Kanban).
Nous avons également déterminé les outils de travail, à savoir le framework JavaFX, l'outil de gestion Maven, les IDE IntelliJ/Eclipse et l'outil de contrôle des sources Git.
Enfin, sur un Wiki associé au depôt GitHub nous avons défini un workflow Git (gestion et nommage des branches, nombre de revues de validation des pull requests).

2.2 Organisation générale

Après avoir identifié les différentes épiques et user stories, nous avons procédé à un planning poker pour déterminer les points d'efforts et valeurs métier de ces stories.
Nous avons ensuite fournit le backlog du premier sprint, au début duquel nous avons chacun choisi les stories sur lesquelles nous souhaitions travailler. Au début des périodes de travail nous partagions des croissants pour du teambuilding et nous faisions un "daily" au cours duquel nous faisions rapport sur notre progrès, puis au cours des périodes de travail nous communiquions pour être sûr qu'aucun de nous ne soit bloqué.

2.3 Organisation à chaque sprint

Pour chaque sprint, nous répétons les étapes suivantes :
- Se donner un objectif pour le sprint.
- Réalisation avec l'équipe des stories qui seront inluses dans le backlog du sprint.
- Raffinage des points d'efforts et des valeurs métier pour chaque users stories.
- Choix des users stories par le développeur.
- *Réalisation du backlog du sprint*
- Sprint review : création d'une branche release, tag du commit, et démonstration.
- Retrospective du sprint (à partir du deuxième) sur ce qu'on a apprécié et moins.

3 Découpage des fonctionnalités

Avant de débuter la conception du projet, nous avons du établir les principales fonctionnalités et définir leur point d'effort et leur valeur métier. Une fois ces deux métriques définies nous avons définie les fonctionnalités à faire dans chaque sprint.

3.1 Liste des fonctionnalités

Epic	Fonctionnalité	Point d'effort	Valeur métier
Fenetre	Avoir l'affichage	M	10
	Bouton rafraichir	S	8
	Bouton précédent	S	8
	Bouton suivant	S	8
	Barre de recherche	S	10
	Bouton Menu	S	6
Historique	Enregistrer historique	XL	4
	Afficher historique	L	4
	Supprimer historique	S	4
	Rechercher historique	S	3
	Cliquer sur lien dans historique	M	4
Onglet	Ajouter onglet	XL	8
	Naviguer entre onglet	L	8
	Supprimer onglet	M	8
	Interchanger onglet	XL	3
	Grouper onglet	XL	3
Barre recherche	Ecrire dans la barre	M	5
	Loader (barre de chargement)	S	3
	Sauvegarder les cookies	L	6
	Suggestion basée sur historique	L	6
	Choisir son moteur de recherche	L	6
Favoris	Ajouter la page actuelle favoris	M	7
	Modifier nom d'un favori	L	5
	Modifier adresse principale favori	L	5
	Créer un dossier	L	7
	Cliquer sur un favori	S	7
	Modifier nom dossier	L	5
Mot de passe	Proposer génération lors d'une inscription	L	4
	Saisie automatique du mot de passe	XL	4
	Afficher liste des mots de passes	M	4
	Détection des mots de passe pwned	M	2
Téléchargement	Choix du dossier destination	M	4
	Etat du téléchargement	XL	5
	Liste des téléchargement	M	4
Données navigations	Sauvegarder cookies	L	6
	Sauvegarde localstorage	L	7
Navigation privée	Navigation privée	M	6

Vous pouvez retrouver ces mêmes users story sur le Trello

3.2 Fonctionnalités sprint n°1

Ce premier sprint a été consacré à la mise en place de l'infrastructure nécessaire pour l'utilisation d'un navigateur web. Cependant, étant donné que nous ne connaissions pas nos capacités et lors du *sanity check* nous avons du ajouter des tâches. On retrouve ainsi les fonctionnalités suivantes.

Epic	Fonctionnalité	Point d'effort	Valeur métier	Affectée à
Fenetre	Avoir l'affichage	M	10	Victor
	Bouton rafraichir	S	8	Guillaume
	Bouton précédent	S	8	Guillaume
	Bouton suivant	S	8	Guillaume
	Barre de recherche	S	10	Victor
	Bouton Menu	S	6	Simon J
Historique	Enregistrer historique	XL	4	Simon M, Adrien
	Afficher historique	L	4	Simon M, Adrien
Barre recherche	Loader (barre de chargement)	M	5	Simon J

Ainsi, nous avons pu calculter la vélocité de l'équipe. Etant donné l'utilisation de taille de T-shirt, nous avons défini une conversion arbitraire $S = 3, M = 5, L = 8, XL = 13$. Ainsi, la vélocité du premier sprint est de 46.

3.2.1 Conclusion du sprint

Toutes les tâches du sprint ont été réalisées.

3.3 Fonctionnalités sprint n°2

Epic	Fonctionnalité	Point d'effort	Valeur métier	Affecté à
Onglet	Ajouter un onglet	XL	8	Adrien, Guillaume
	Naviguer entre onglet	L	8	Adrien, Guillaume
	Supprimer un onglet	L	8	Adrien, Guillaume
Historique	Rechercher historique	S	3	Simon M, Victor
	Supprimer historique	S	4	Simon M, Victor
Données navigation	Sauvegarder cookies à la fermeture	L	6	Simon J

La vélovieté théorique du sprint est de 48. Mais comme expliqué dans la conclusion du sprint ci-dessous, nous n'avons pas réussir à finir toutes les fonctionnalités.

3.3.1 Conclusion du sprint

Nous avons mal évalué la difficulté pour la mise en place des onglets de navigation. En effet, la tâche s'est avérée plus longue que prévu et de nombreux problèmes auxquels nous n'avions pas pensés nous ont obligé à modifier notre conception et une partie du code réalisé au premier sprint.
Par conséquent, l'épic *onglet* sera continuée sur le troisième sprint.

3.4 Fonctionnalités sprint n°3

Epic	Fonctionnalité	Point d'effort	Valeur métier	Affecté à
Onglet	Ajouter un onglet	XL	8	Guillaume, Adrien
	Naviguer entre onglet	L	8	Guillaume, Simon J
	Supprimer un onglet	L	8	Guillaume, Simon J
Favoris	Créer un dossier	L	7	Simon M
	Ajouter un favori	M	7	Simon M
	Cliquer sur un favori	S	7	Simon M
Barre de recherche	Choisir son navigateur (fichier de configuration)	L	6	Victor
Navigation privée	Navigation privée	M	6	Guillaume
Données de navigation	Sauvegarde localstorage	L	7	Simon J
Autre	Revue de code BDD			Adrien
	Revue de code fichier de conf.			Adrien

La fonctionnalité *Choisir son navigateur* a également été sous-évaluée. Malgré qu'elle soit finie, elle nous a prise plus de temps que prévue.

La vélocité de ce sprint est de 69, ce qui est largement au-dessus des deux dernières vélocités. Il faut également considéré que l'épic onglet c'est réalisé sur deux sprints. Mais, on peut quand même se demander si certain point d'effort n'ont pas été sur-estimés.

3.4.1 Conclusion du sprint

Ce troisème et dernier sprint nous a permi d'avoir une application réellement fonctionnelle.

3.5 Fonctionnalités manquantes

Ce dernier tableau résume les fonctionnalités qui n'ont pas pu être développé par manque de temps. Cependant, ces fonctionnalités sont les moins valorisantes pour l'application, là, où, nous avons voulu nous concentrer sur les fonctionnalités essentielles

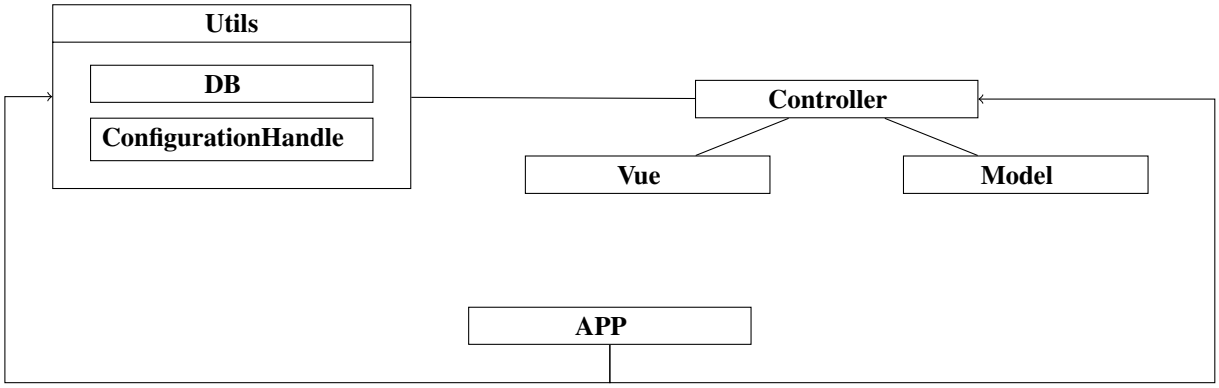
Epic	Fonctionnalité	Point d'effort	Valeur métier
Favoris	Modifier nom d'un favori	L	5
	Modifier adresse principal du favori	L	5
	Modifier nom d'un dossier	L	5
Mot de passe	Proposer génération lors d'une inscription	L	4
	Saisie automatique du mot de passe	XL	4
	Afficher liste des mots de passes	M	4
	Détection des mots de passe pwned	M	2
Téléchargement	Choix du dossier destination	M	4
	Etat du téléchargement	XL	5
	Liste des téléchargement	M	4

4 Architecture de l'application

L'architecture de l'application est très importante afin de s'assurer d'une bonne évolution et moyennabilité. Ainsi, dans cette section nous présentons l'architecture principale de l'application à travers un diagramme de paquetage, un diagramme de classe et un diagramme de séquence.
D'autres choix de conception serons présentés dans la section *Choix de conception*

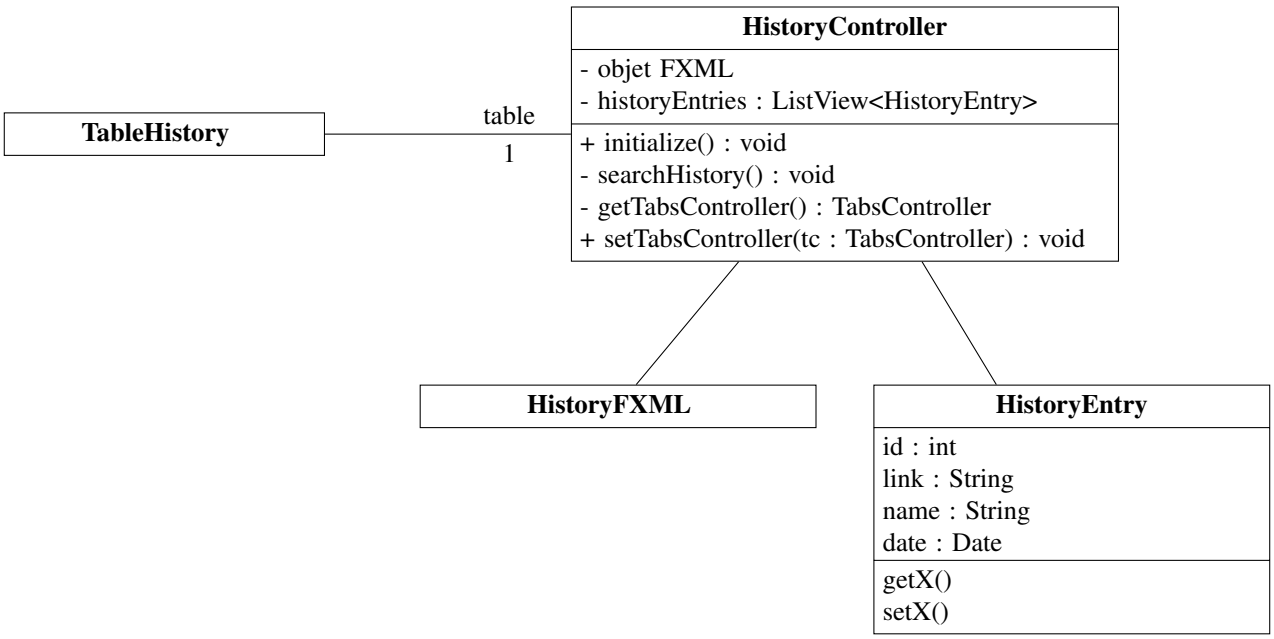
4.1 Organisation des paquetages

Les classes et interfaces de notre application ont été structurées dans nos différents paquetages en respectant le modèle MVC.



4.2 Exemple d'un diagramme de classe

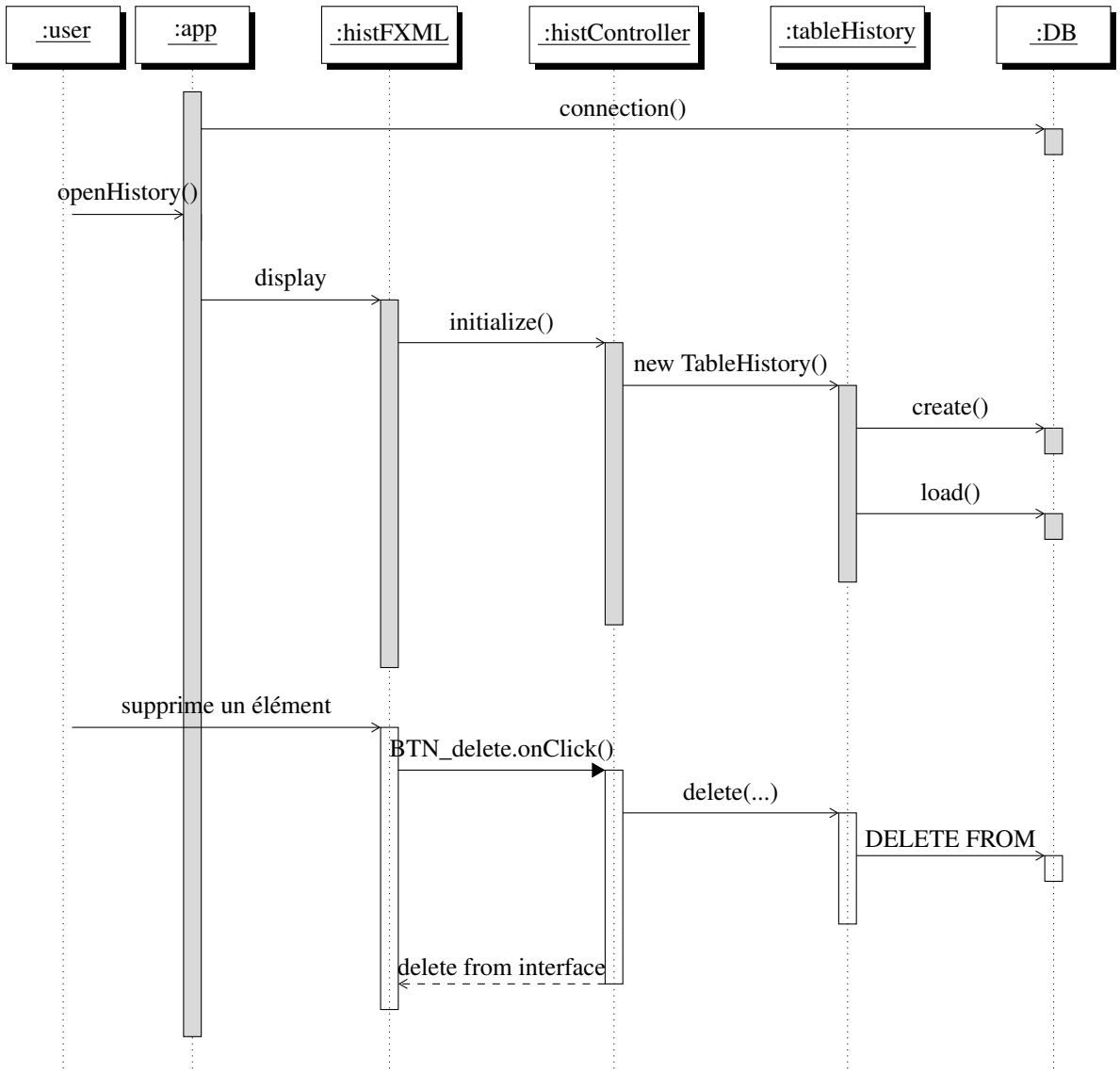
Les deux sections suivantes illustrent un diagramme de classe et un diagramme de séquence à travers le service *Historique*. Etant donné que tous les services de l'application (favoris, onglets, cookies, paramètres) suivent la même architecture nous avons trouvé plus pertinent de ne représenter en détail qu'un seul service. Ainsi, ci-dessous vous retrouvez le diagramme de classe du service *Historique*.



L'implémentation des bases de données pour l'application est présentée plus en détail dans la section 5.1.1.

4.3 Exemple d'un diagramme de séquence

Au lieu de réaliser un diagramme de classe décrivant les dépendances entre les classes des packages `Model`, `Vue`, `Controller` nous avons préféré décrire l'enchainement des instructions d'une user storie. Ainsi, ci-dessous vous retrouvez un diagramme de séquence pour la suppression d'un élément de l'historique.


:user
:app
:histFXML
:histController
:tableHistory
:DB
connection()
openHistory()
display
initialize()
new TableHistory()
create()
load()
supprime un élément
BTN_delete.onClick()
delete(...)
DELETE FROM
delete from interface

5 Choix de conception

5.1 Stockage des données

Lors de la conception de l'application, nous nous sommes demandé s'il serait interessant d'utiliser une base de données. En effet, nous avons besoin de stocker des informations persistantes telles que l'hsistorique, les favoris, les cookies, etc ... Différentes solutions s'offrent à nous pour réaliser le stockage. Nous pouvions soit utiliser un fichier texte soit utiliser une base de données.

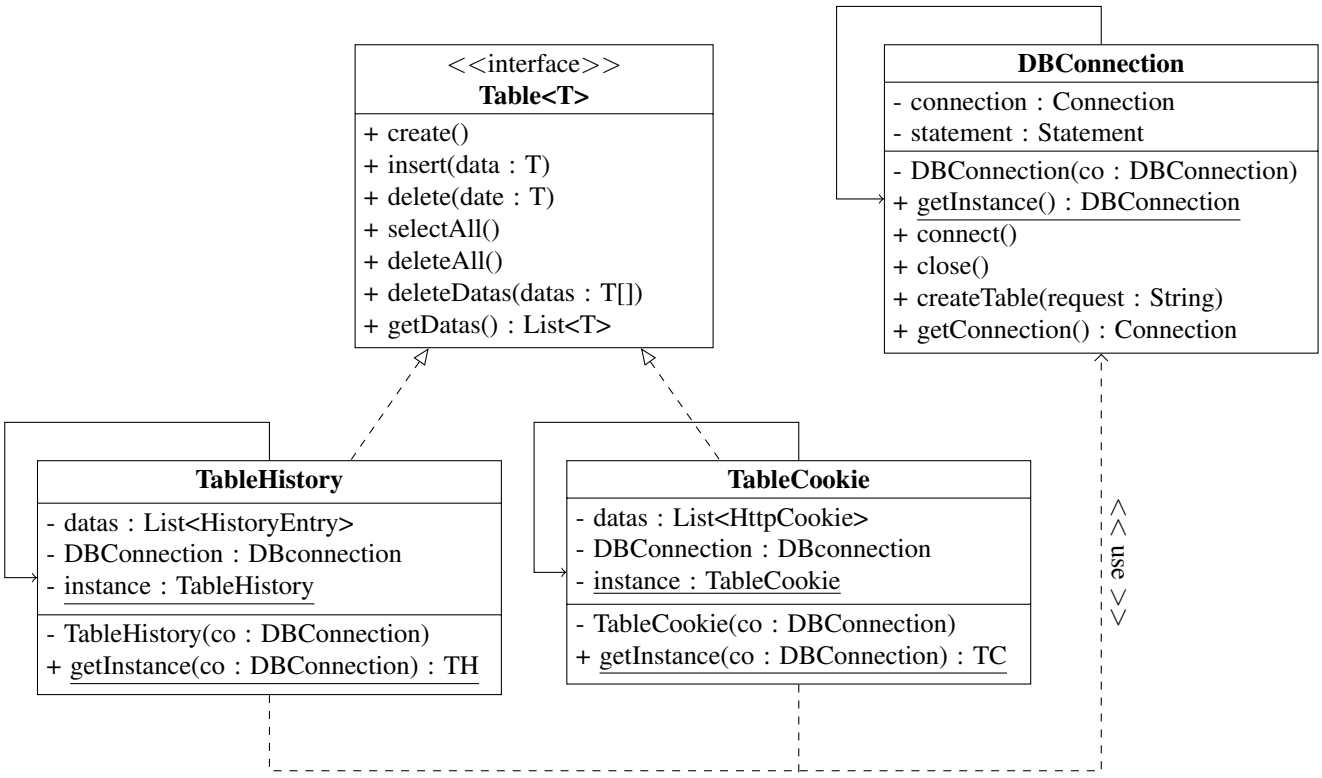
Les fichiers texte
Ils ont l'avantage d'être simple à créer et à utiliser. De plus si nous utilisons une structure comme JSON ou XML il existe des librairies Java qui permettent de créer et récupérer les informations d'un fichier facilement.

Les bases de données
Est un outils conçu spécialement pour le stockage d'un grand nombre de données. Cependant, il est plus difficile de manipuler les bases de données en Java que les fichiers. De plus, nous devons également stocker les informations enregistrer en base de données sans que l'utilisateur n'est besoin d'installer lui même une base de données.
Cette solution bien que plus compliquée à mettre en place a été retenue car elle présente de nombreux avantages une fois la technologie maitrisée.

Ainsi, nous avons choisi une base de données SQLite qui permet d'avoir les informations stockées dans la base dans un fichier `.db`. Ensuite, nous devions faire le lien entre nos classes Java (POJOs) et la table SQL. Une solution serait d'utiliser JavaEE et les annotations mais étant données que nous ne souhaitons pas charger le projet nous avons implémenter différents services qui permettent d'intéragir avec la base de données (`insert()`, `selectAll()` ...).

5.1.1 Diagramme de classe

Ci-dessous nous représentons l'organisation des classes pour la gestions des tables de la base de données. La classe `DBConnection` permet de gérer l'accès à la base de données et les implémentations de `Table` fait le lien entre les objets Java et les tables de la base de données.



Un objet de type `DBConnection` est créé au lancement de l'application. Ensuite, pour pouvoir manipuler les tables, nous devons récupérer cet objet pour pouvoir effectuer des traitements dessus, c'est pour cela que nous le passons en paramètre de nos constructeurs `TableHistory` et `TableCookie`.

```
class HistoryController {

   private TableHistory table;

   @FXML
   intinialize() {
      table = TableHistory.getInstance(DBConnection.getInstance());
      ...
   }
}

class TableHistory implements Table<HistoryEntry> {

   private List<HistoryEntry> historyList;
   private DBConnection DBconnection;

   private static TableHistory instance;

   private TableHistory(DBConnection DBconnection) {
      this.DBconnection = DBconnection;
      this.historyList = new ArrayList<HistoryEntry>();

      this.create(); this.selectAll();
   }

   public static TableHistory getInstance(DBConnection DBconnection) {
      if(instance == null) {
         instance = new TableHistory(DBconnection);
      }
      return instance;
   }
```

5.2 Stockage des configurations

Nous avons souhaité offrir à l'utilisateur la possibilité de configurer certaines options de l'application.

5.2.1 Configurer le navigateur

Premièrement, l'utilisateur peut enregistrer des configurations pour l'intégralité du navigateur web. Par exemple, le moteur de recherche par défaut, s'il souhaite être en mode sombre, etc ...
De même que dans la section précédente, nous devons nous poser la question s'il faut utiliser une base de données ou un fichier de configuration. Pour ce cas, étant donné que les configurations possibles ne représentent pas un grand nombreux de données à stocker, il est plus simple de manier un fichier formaté comme JSON.

```
config.json
{
    engine: google // le moteur de recherche par défaut est google
}
```

L'utilisateur à la possibilité de modifier la valeur du moteur de recherche par défaut depuis le menu *paramètre* de l'application. Ce qui aura pour conséquence de modifie la valeur dans le fichier `config.json`.

5.2.2 Configurer les moteurs recherches

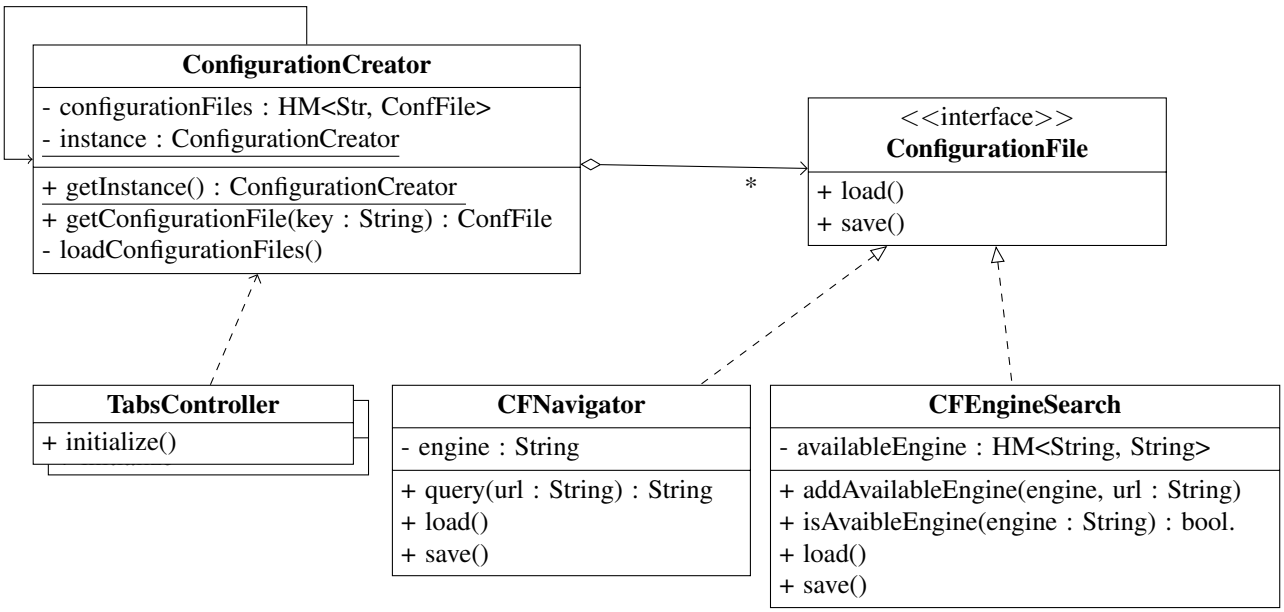
Ensuite, nous souhaitons que l'utilisateur puissent ajouter des moteurs de recherche ou des raccourcis de recherche au navigateur. Par exemple au lieu d'aller sur Youtube et d'effectuer une rechercher, l'utilisateur peut directement écrire `@youtube nom_video` dans la barre de recherche.
Pour ce faire, de même que la configuration du navigateur nous utilisons un fichier JSON en associant un nom et un url.

```
webengine.config
{
    youtube: https://youtube.com/result?search_query=
    google: https://google.com/search?q=
}
```

L'utilisateur a également la possibilité d'ajouter un raccourci `@` depuis les paramètres du navigateur.

5.2.3 Diagramme de classe

Nous présentons ci-dessous, un diagramme de classe sur la configuration de l'application



C'est dans la méthode `TabsController#initialize()` que nous instancions pour la premièrement fois un objet de type `ConfigurationCreator`. Lors de la création de cet objet, nous exécutons la méthode`loadConfigurationFile` qui va se charger de créer un nouvel objet pour chaque fichier de configuration avec une clé. Ainsi, par la suite nous pourrons récupérer cet objet grâce à la méthode `getConfigurationFile(key)`

```
class TabController {
   @FXML
   void initialize() {
      config = ConfigurationCreator.getInstance();
   }
}

class ConfigurationCreator {

   private ConfigurationCreator() {
      configurationsFiles = new HashMap<>();
      loadConfigurationFiles();
   }

   public static ConfigurationCreator getInstance() {
      /* Singleton */
   }

   public void loadConfigurationFiles() {
         configurationsFiles.put("configurationFileNavigator",
                                 new ConfigurationFileNavigator());
         configurationsFiles.put("configurationFileEngineSearch",
                                 new ConfigurationFileEngineSearch());
   }
}
```

Par la suite, les autres contrôleurs pourront récupérer les objets associés aux fichiers de configuration. C'est le cas dans `ParametersController` qui a besoin des fichiers de configuration du navigateur et des moteurs de recherche.

```
class ParametersController {
   @FXML
   private void initialize(){
      // get config from file
      ConfigurationCreator config = ConfigurationCreator.getInstance();

      ConfigurationFileNavigator configNavigator =
            (ConfigurationFileNavigator)
               config.getConfigurationFile("configurationFileNavigator");

      ConfigurationFileEngineSearch configEngineSearch =
            (ConfigurationFileEngineSearch)
               config.getConfigurationFile("configurationFileEngineSearch");

      /* manipulation des fichiers àtravers les deux objets */
}
```

5.3 Complémentarité des controlleurs

L'une des principales difficultés du projet était de pouvoir communiquer entre controlleurs. En effet, nous avons définie un WebView qui permet d'avoir l'affichage des pages web. Mais lors de l'épic onglet, il a fallu faire en sorte que chaque onglet et une WebView différente. De plus, les boutons de la barre de navigation (suivant, précédent, etc) doivent communiquer seulement avec l'onglet et la WebView courante.

Pour résoudre ce problème, nous avons du imbriquer les différents controleurs afin de pouvoir manipuler plusieurs controleurs sur la même vue. Ainsi, la vue définie dans le `main.fxml` avec la fenêtre de navigation et la barre de navigation contient également une imbriquation pour pouvoir utiliser la classe `TabsController`.
`<fx:include fx:id="tab" source="tabs.fxml" VBox.vgrow="ALWAYS" />`

5.3.1 Communication du fils vers le père

Et dans le controleur du `main.fxml` nous avons

```
@FXML // vaut toujours null si manquant
private TabsController tabController;

private void initialize() {
   tabController.setControlsController(this);
   ...
}
```

La méthode `TabsController.setControlsController` va permettre d'avoir connaissance de la barre de recherche dans la classe `TabsController`. Ainsi, lorsqu'on va changer d'onglet on va pouvoir également changer le nom dans la barre de recherche.

```
/**
 * Assign the window controller instance.
 * @param controlsController the window controller.
 */
public void setControlsController(WebViewController controlsController) {
   this.controlsController = controlsController;
   if (this.controlsController != null) {
      this.addressBar = this.controlsController.getAddressBar();
      this.progressBar = this.controlsController.getProgressBar();
   }
}
```

5.3.2 Communication du père vers le fils

Nous venons de donner connaissance du controleur parent au controleur fils. Maintenant, nous pouvons utiliser les méthodes définies dans `TabsController`. Par exemple en cliquant sur le bouton *ajouter un onglet* qui est défini dans `main.fxml` on va utiliser la méthode `TabsController#addNewTab()`.

```
newTabButton.setOnAction(new EventHandler<ActionEvent>() {
   @Override
   public void handle(ActionEvent event) {
      tabController.addNewTab(false);
      tabController.getCurrentTab().getTab().setText("Loading...");
      NavigationUtils.search(addressBar.getText(),
         tabController.getCurrentTab().getWebView().getEngine());
   }
});
```

6 Conclusion

6.1 Pistes d'amélioration

Avant de conclure sur ce projet, nous pouvons lister quelques pistes d'amélioration :

- Pour finaliser le projet, en se basant sur la vélocité, on estime qu'il nous faudrait deux semaines supplémentaires.
- De même, certain point d'effort ont été mal évalué. Mais de meilleures estimations nécessite une plus grande expérience.
- Une meilleure conception dès le début du projet aurait pu nous éviter d'avoir à réaliser des refactors.
- Suivre jusqu'à la fin du projet le workflow détaillé sur le GitHub car à la fin du projet nous ne nous relisons plus (prise de confiance)
- Enfin, il manque des tests unitaires car l'architecture est très simple et les tests se baseraient sur l'interface graphique.

6.2 Conclusion sur l'agilité

L'obligation d'utiliser l'agilité pour ce projet nous a prouvé que ce concept fonctionne bien. En effet, au fur et à mesure des itérations on arrivait à prioriser les tâches et à apporter des modifications sur des points d'effort et des valeurs métier. De plus, le fait d'avoir un rendu de l'application chaque semaine est très gratifiant et nous pousse à continuer.
Ensuite, la répartition des tâches était équitable. Chacun pouvait travailler sur son service indépendamment puis ensuite le mettre en commun via git.
Enfin, nous finissions chaque itération par un sprint review pour se remettre en question et continuer à s'améliorer

6.3 Conclution du projet