

Journal de bord TPI – Tanguy CAVAGNA

J1 : lundi 25 mai 2020

Objectifs

L'objectif de cette journée est de lire l'énoncé dans son intégralité afin de prendre connaissance du cahier des charges, d'en extraire les *user stories* pour pouvoir correctement rédiger mon *product backlog* et enfin de rédiger les scénarios de tests fonctionnels, indispensables pour le bon fonctionnement de mon projet.

Déroulement

Je commence ma journée à 8h00. Je lis avec attention l'énoncé que M. Terrond m'a fait parvenir la veille. Ainsi, j'aborde la première étape de la **méthodologie en 6 étapes : S'informer**. Je vais utiliser cette méthodologie durant tout le déroulement de ce TPI.

Je réalise que quelques points de l'énoncé ne sont pas clairs dont un particulièrement embêtant. Je décide de poser mes questions à mon référent durant la matinée.

Je commence à **Planifier**, seconde étape de la méthodologie utilisée. Pour cela, je sépare ma journée en tranches de 4 heures (des demi-journées), et je remplis les différentes tranches horaires avec les *user stories* extraites de mon cahier des charges.

8h15 : je décide d'utiliser des alias afin de nommer les jours de travail mis à disposition pour le TPI : **J1** à **J11**. Voici les alias :

- J1 : lundi 25 mai 2020
- J2 : mardi 26 mai 2020
- J3 : mercredi 27 mai 2020
- J4 : jeudi 28 mai 2020
- J5 : vendredi 29 mai 2020
- J6 : samedi 30 mai 2020
- J7 : dimanche 31 mai 2020
- J8 : lundi 1er juin 2020
- J9 : mardi 2 juin 2020
- J10 : mercredi 3 juin 2020
- J11 : jeudi 4 juin 2020

- J10 : lundi 8 juin 2020
- J11 : mardi 9 juin 2020

8h25 : Lors de la création des *user stories* je remarque qu'il me faut décider d'une manière de prioriser les tâches. Je choisis la méthode **MoSCoW**. Cependant, les niveaux de priorité ne correspondent pas entièrement pour mon TPI. Je décide alors de modifier les intitulés :

- **Must** devient ☹ **Bloquant**
- **Should** devient ⚡ **Critique**
- **Could** devient ↓ **Important**
- **Won't** devient ? **Secondaire**

Je décide aussi d'utiliser la syntaxe suivante pour de présenter mes *user stories* :

Nom	S<n° de la story > : <Nom de la user story >
Description (user story)	<Description de la story pour connaître avec précision le but à atteindre>
Critère d'acceptation	<n° des tests à passé pour valider cette story >
Priorité	<Priorité de la story >

9h : Je fais un script bash me permettant un rassembler tous mes fichiers Markdown de ma documentation dans un seul et même fichier. Ceci est nécessaire car je prévois de publier ma documentation en ligne, à l'aide du site **readthedocs.org**.

10h : En plus de la documentation publique, il faut une version PDF. Pour ce faire j'utilise le logiciel **Typora** qui me permet d'exporter mon fichier réunissant toute ma documentation en PDF. Une fois cela fait, j'utilise un autre script bash permettant de fusionner plusieurs fichiers PDF en un seul. Ce dernier se nomme : **Rapport du TPI et documentation technique**. Il contient le rapport, les annexes, le résumé, l'énoncé, le journal de bord, et le code source.

10h30 : Je décris mes outils de bureautique. J'utilise **Typora** (un éditeur Markdown compatible sous tout OS) pour rédiger l'entièreté de ma documentation. La création des fichiers PDF est faite grâce à l'export vers PDF de Typora ainsi qu'à un script écrit par moi-même.

Concernant le style appliqué à ma documentation, j'utilise la couleur ● **#006EDB** comme base. La police s'appelle Poppins. J'utilise cette police également dans le projet en lui-même.

10h50 : J'ai un rendez-vous GMeet avec mon formateur pour vérifier que tout va bien. Je pose la question suivante :

Est-ce que le planning que vous m'avez donné est celui qu'il faudra utiliser ?

☞ Mon formateur me répond que le planning donné est un modèle permettant de suivre de façon basique l'avancée du projet.

Je comprends que si j'ai un planning plus précis, je peux l'utiliser et comparer le mien avec celui qui m'a été donné.

11h25 : Je termine la rédaction de mon *product backlog* temporaire. Des modifications peuvent encore être apportées si j'en trouve le besoin.

11h45 : Je compile une version de test de ma documentation pour vérifier qu'il n'y ait pas d'erreur. Je prends ma pause de midi.

12h50 : Reprise de la journée. Je m'attaque maintenant au diagramme de Gantt. Je choisis de le réaliser avec un tableau HTML car je ne suis pas à l'aise avec les outils spécialisé comme Gantter.

14h15 : Je remarque un souci lors de la fusion des fichiers Markdown. Une partie d'un fichier se dédouble mais je ne sais pas encore pourquoi.

15h50 : Mon souci de duplication est résolu. Ce problème venait du fait que j'ajoutais ma table des matières sans supprimer le contenu précédent. Désormais, je supprime le contenu du fichier avant de le remplacer par le contenu mis-à-jour avec la table des matières. Je peux désormais commencer l'écriture des scénarios de tests fonctionnels.

16h45 : J'écris une partie des scénarios de tests. Il m'en reste encore quelques-uns que j'ajouterai demain matin.

Bilan

La journée c'est plutôt bien passée. J'ai cependant pris un peu de retard sur la rédaction des scénarios de tests à cause de mon problème de duplication lors de la compilation de la documentation. Cependant, ceci reste un écart très minime sur mon planning. Je suis tout de même satisfait de cette première journée.

J2 : mardi 26 mai 2020

Objectifs

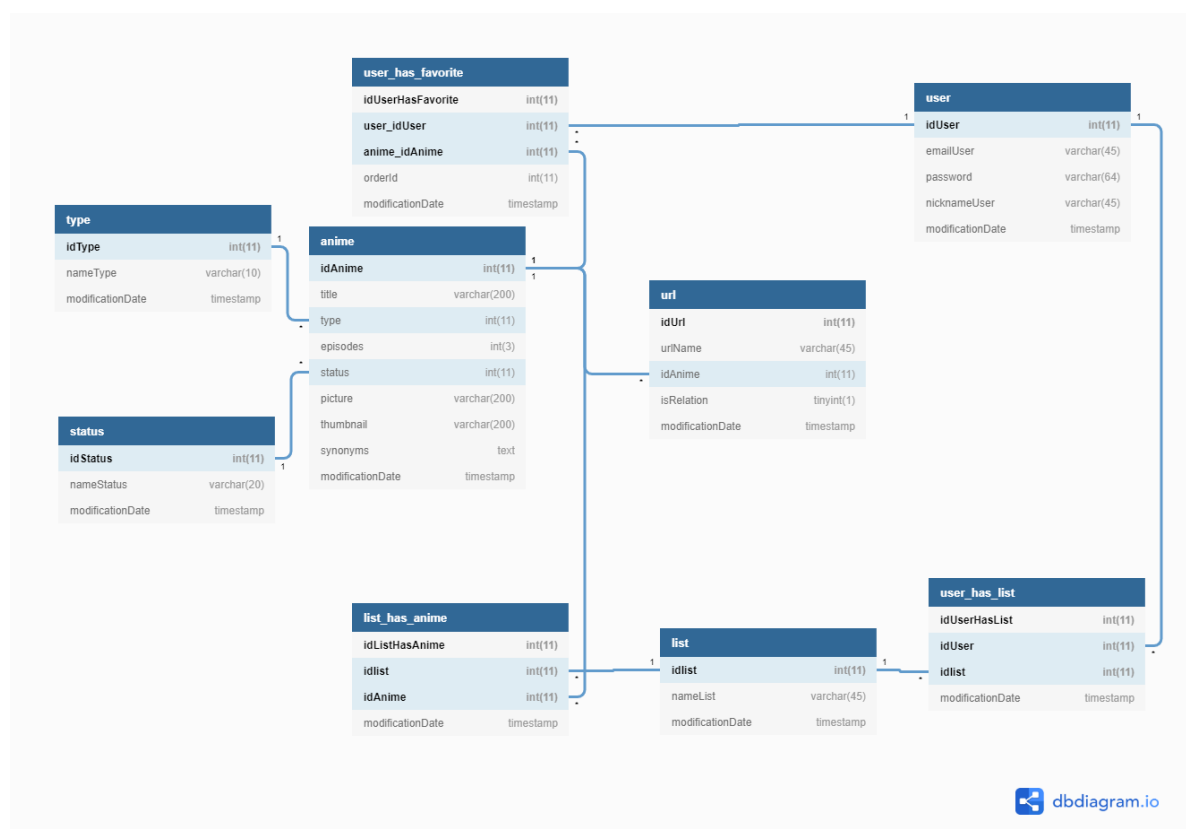
L'objectif de cette journée est premièrement de rattraper le petit retard que j'ai pris hier sur les scénarios de tests. Ensuite, je prévois de faire le modèle de base de données, de configurer l'application Flask, et de faire le code permettant d'importer les données.

Déroulement

8h : Je commence ma journée. Je finis les scénarios de tests que je n'avais pas pu terminer hier. Ceci ne prend pas beaucoup de temps.

8h30 : Je termine les scénarios. Je passe à la conception de la base de données. Grâce à l'énoncé, j'extrais les différentes tables du projet : `anime`, `status`, `type`, `url`, `list`, `list_has_anime`, `user`, `user_has_list`, `user_has_favorite`.

8h55 : Je réalise le modèle de base de données que voici :



9h : Je fais la partie *Base de donnée* du chapitre *Implémentation* de la documentation étant donné que j'ai toutes les informations nécessaires.

9h25 : Je termine de documenter la partie *Base de données*. J'y ai mis le modèle ci-dessus ainsi que le dictionnaire de données.

9h30 : Je configure l'application Flask pour pouvoir avoir un environnement de développement fonctionnel et pouvoir ainsi faire la suite du projet.

J'inscris une `secret_key` à l'application Flask. Cette clef est utilisée dans les systèmes d'encryptions. Flask lui-même n'a pas besoin de cette clef mais d'autres librairies externes, tel que `flask-login`, que j'utiliserai afin de pouvoir connecter un utilisateur, doit avoir cette clef. La valeur de cette clef est `Super` en Sha256.

9h55 : J'ai une application Flask basique fonctionnelle. Je peux rendre des vues depuis une route sans problèmes.

10h : Je mets en place le système de documentation automatique d'API : **Swagger**.

Pour que ce système puisse être mis en place, il faut une librairie externe nommée `flask_swagger`. De plus, certains fichiers sont indispensables au bon fonctionnement de Swagger. Le plus primordiale est la page HTML. Je décide de la nommer `endpoints.html` et de la placer dans le dossier `templates`. Cette page est disponible [ici](#). Cependant, je l'ai adapté pour qu'elle soit correctement implémentée dans l'application Flask.

En plus de la page HTML, je rajoute 2 fichiers `javascript` qui vont dans le dossier `static/js`, 2 images qui vont dans le `static/img` et enfin 1 fichier `css` qui va dans le `static/css`. Voici le lien pour télécharger les fichiers :

- **swagger-ui-bundle.js**
- **swagger-ui-standalone-preset.js**
- **favicon-16x16.png**
- **favicon-32x32.png**
- **swagger-ui.css**

La structure du dossier `static` ressemble désormais à ceci :

```
1  Animanga
2  └─ static
3      └─ css
4          └─ swagger-ui.css
5      └─ js
6          └─ swagger-ui-bundle.js
7          └─ swagger-ui-standalone-preset.js
8      └─ img
9          └─ favicon-16x16.png
10         └─ favicon-32x32.png
```

10h25 : J'installe la police Poppins en local. De ce fait, je n'aurai pas de soucis si le site perd la connexion à internet et que les polices sont chargées depuis Google Fonts.

10h30 : Je commence l'importation des données en base.

11h40 : Je termine l'importation des types et statuts des animes depuis le fichier JSON. Il me reste à faire l'importation des animes eux-même. Je prends ma pause de midi.

13h : Reprise de la journée et continuation de l'importation des données dans la base de données.

14h : Je termine l'import des données. Tout semble parfaitement bien s'ajouter en base. Étant donné que j'ai un peu d'avance, je mets ma documentation en ligne sur **readthedocs.org**. Le site hébergeant la documentation utilise **Mkdocs** pour convertir la documentation de Markdown à HTML. C'est pourquoi j'installe mkdocs dans **WSL 2** sur ma machine pour vérifier si ma documentation compile correctement.

14h15 : La documentation est en ligne à l'adresse : **<https://animanga.readthedocs.io/fr/latest/>**.

Pour le moment, étant donné que je n'ai pas encore mis en place la connexion, je ne peux effectuer mes tests que manuellement. Cependant, dès lors que la connexion sera mise en place, j'utiliserai **Katalon Recorder** pour automatiser mes tests.

14h25 : J'effectue le test *3.1* pour vérifier que mes données soient correctement importées. Comme j'ai de l'avance, je fais l'affichage de la *landing page* .

15h20 : Je termine l'implémentation de la *landing page* . Pour le moment, je ne peux tester le basculement de l'état connecté à l'état déconnecté que via la variable `is_authenticated` de que je change dans le fichier `routes.py` . Demain, je pourrai changer puisque je mettrai en place la connexion et l'inscription. Je n'aurai donc plus besoin de cette variable. Le test *11.1* passe concernant l'affichage de la *landing page* .

Je configure deux vérificateurs de syntaxe différents pour mon projet. Le premier est **pylint** pour Python, et le second est **eslint** pour le JavaScript. Pour eslint, j'utilise un preset de vérification : `airbnb`. Cela me permet d'écrire du code JavaScript dans un cadre syntaxique strict et donc de ne pas sortir des conventions actuelles.

15h30 : Eslint est installé et je lance la commande `npm run lint static/js` pour vérifier mes fichiers JavaScript.

15h35 : Je corrige les erreurs que eslint m'a montrées et je m'attelle maintenant à pylint.

16h : La vérification de syntaxe pylint fonctionne correctement. Je lance la commande `python3 -m pylint --output-format=colored packages` pour vérifier la syntaxe des fichiers se trouvant dans le dossier `packages`.

16h10 : Je corrige les erreurs relevées par pylint. Je mets à jour mon planning et mes scénarios de tests afin d'ajouter la vérification syntaxique.

16h30 : J'ajoute la vérification syntaxique dans le planning comme *user story* et je crée un test pour le Python et un pour le JavaScript afin de valider la *user story*.

Bilan

Je suis très content de l'avancement d'aujourd'hui. J'ai eu un peu de retard hier, mais je l'ai rapidement rattrapé ce matin. Je réussis à prendre un peu d'avance dans le projet et donc je suis très confiant pour la suite. Cela me permet de rajouter la vérification syntaxique pour Python et pour JavaScript.

J3 : mercredi 27 mai 2020

Objectifs

L'objectif de cette journée est de faire le système de connexion et d'inscription. Comme j'ai déjà fait l'affichage de la landing page hier, je pense avancer sur une autre tâche.

Déroulement

8h : Je commence à faire la partie inscription. Je crée un contrôleur pour les utilisateurs pour mieux pouvoir gérer ces derniers.

8h30 : Je fais le formulaire HTML et je commence à faire la partie prise en charges de ce dernier.

9h : J'ai la visite de M. Terrond. M. Bouille étant pris à la Protection Civile, il ne peut pas être présent. Le but de cette visite est de voir si tout se passe bien et de répondre à d'éventuelles questions. Étant donné que tout se passe bien pour moi, la visite se révèle être très brève !

9h40 : J'ai des soucis avec mon système d'export de documentation vers PDF. L'export ne s'effectue pas et je n'ai aucun moyen de savoir pourquoi. Je cherche activement ce qui pose problème.

10h20 : Je parviens à régler le problème d'export pour le moment mais rien ne me dit que cela n'arrivera pas de nouveau. Le cas échéant, j'ai un moyen auxiliaire d'exporter ma documentation. Je me remets à travailler sur la partie inscription.

11h45 : Je termine l'inscription. L'insère de nouvel utilisateur semble fonctionner. Avec les quelques minutes qu'il me reste avant la pause de midi, je mets en place des automatisations **Katalon Recorder** pour les futurs tests.

12h : Je termine l'automation du test fonctionnel d'inscription avec valeurs correctes. Je ferai les autres tests dès que j'aurai du temps durant la journée. Je prends ma pause de midi.

13h : Je me remets au travail en commençant la partie connexion. Étant donné le retard pris à cause du problème d'exportation vers PDF, je ne commence la connexion que maintenant. Ceci ne va causer d'autres retards car j'ai pris de l'avance hier concernant la *landing page* que j'aurais dû réaliser aujourd'hui.

13h40 : La *connexion* est terminée et fonctionne à merveille. Je crée maintenant des tests d'automations pour éviter de retaper tout le temps la même chose lors des futurs tests. J'utilise Katalon Recorder pour cela.

14h20 : J'optimise un peu le code pour l'inscription et la connexion étant donné que je n'ai pas pu le faire ce matin à cause de mon souci d'erreur d'export de la documentation.

15h20 : J'ai un rendez-vous avec mon référent TPI pour faire un point de la situation. Comme tout ce passe bien, le rendez-vous ne dure que très peu de temps et je retourne tout de suite travailler.

17h : Je termine l'optimisation de la validation des champs. Je me suis basé sur la librairie **wtforms**. Cette librairie est utilisée pour générer et valider automatiquement des formulaires. Comme cette librairie est trop imposante pour un TPI, je décide de m'inspirer de cette dernière uniquement pour la partie validation de formulaire. Je demanderai demain matin à mon référent s'il est d'accord que je garde cette manière de valider mes champs de formulaire ou si c'est toujours trop imposant.

En plus de cela, je configure Katalon Recorder pour prendre aussi en compte la connexion. J'ai maintenant un dossier à la racine de mon projet nommé **tests** qui contient le fichier HTML contenant tous les *tests cases* pour Katalon Recorder.

Bilan

Je suis plutôt content de ma journée. J'ai pu correctement faire le code de l'inscription ainsi que la connexion. De plus, comme j'avais du temps restant avant la fin de la journée, j'ai décidé de mettre en place Katalon Recorder afin d'automatiser mes tests fonctionnels. J'ai aussi décidé d'optimiser le code de validation des champs de mes formulaires. Je n'ai cependant toujours pas fait valider cette idée d'optimisation donc je le ferai demain matin.

J4 : jeudi 28 mai 2020

Objectifs

Le but de cette journée est de réaliser la partie recherche et affichage de la carte d'un anime. C'est un des aspects principaux de l'application.

Déroulement

8h : Après réflexion, je réalise que ce que j'ai entrepris la hier sur l'optimisation de la validation des champs était beaucoup trop imposant. Je décide de le refaire de manière bien plus légère.

9h20 : Je termine ma nouvelle version de l'optimisation des champs. Je suis très content du résultat, c'est bien plus simple à lire et à maintenir ! Je commence la partie recherche d'anime. C'est une des parties les plus importantes du projet.

Pour les recherches en base, j'ai une table virtuelle Sqlite3 pour pouvoir faire de la recherche **Fulltext**. Sqlite3 ne supporte pas le Fulltexte sur une table standard, il faut créer une table virtuelle avec un template supportant Fulltext. Tout le procédé est expliqué [ici](#).

10h10 : La recherche par chaine de caractères est terminée. Il me reste à faire l'affichage. Ce n'est pas long. Je décide cependant de le faire cet après-midi préférant m'occuper de la récupération d'une anime aléatoire.

10h20 : J'ai un rendez-vous GMeet avec mon référent pour voir mon avancement et pour lui poser une question concernant la validation des champs. Je demande si ce que j'avais fait convient et si je peux le garder. Sa réponse est positive. Je continue mon projet.


10h40 : La récupération d'un anime aléatoirement est terminée. Je remarque que j'ai des **try/except** dans pratiquement toutes les méthodes de classes. Le **except** est toujours le même mais rien n'est centralisé. Je décide de faire une fonction centralisant tous les logs.

11h35 : La fonction de log est terminée. Je place dans tous les **except** un appel à cette méthode.

Comme j'ai encore un peu de temps avant la pause de midi, je commence à afficher les résultats de la recherche.

12h : Je termine l'affichage des résultats de la recherche et je prends ma pause de midi.

13h : Je reprends le travail en faisant la prise en charge de la barre de recherche.

13h20 : Je termine la prise en charge de la barre de recherche et je commence à faire l'affichage de la carte pour les animes. Pour la barre de recherche. J'opte pour deux manières différentes de l'afficher. La première, classique, est de cliquer sur la . Une modale s'affiche alors avec un champ de type de texte pour y entrer la recherche, ainsi qu'une croix sur la droite pour effacer le contenu du champ. La seconde manière est par le raccourci `Ctrl` + `S`.

14h30 : Jr termine d'afficher la carte (modale) de l'anime. Je commence la mise à jour de l'anime pour l'utilisateur connecté.

17h : Je n'ai pas terminé la mise à jour de l'anime Comme je dois, en principe, le faire demain, ce n'est pas un soucis.

Bilan

Cette journée était très sympa. Je n'ai pas eu de problème et j'ai pris un peu d'avance. Le journal n'est pas très rempli pour cet après-midi. Cependant, tout ce qu'il y a à savoir sur cette journée y est. Il ne c'est rien de particulier. Je suis très content d'avoir mis en place les automatisations Katalon car dès que je change quelque chose, je peux lancer les tests pour voir si rien n'a régressé.

J5 : vendredi 29 mai 2020

Objectif

Le but de cette journée est de faire la mise à jour de l'anime pour un utilisateur, d'afficher sa page de profil, et d'afficher ses favoris. Je prévois de faire l'affichage des favoris en même temps que la mise à jour de l'anime pour pouvoir avoir un retour visuel sur le bon fonctionnement ou pas du code.

Déroulement

8h : Je commence ma journée par la mise à jour de l'anime. Ne pensant pas que cela va me prendre toute la matinée comme prévu, je vais faire la partie affichage des favoris en même temps.

9h30 : Je termine la partie mise à jour de l'état de favoris pour les animes. Je les affiche pour pouvoir avoir un retour visuel et aussi pour avoir un élément à tester en case de réussite lors des tests d'automatisation.

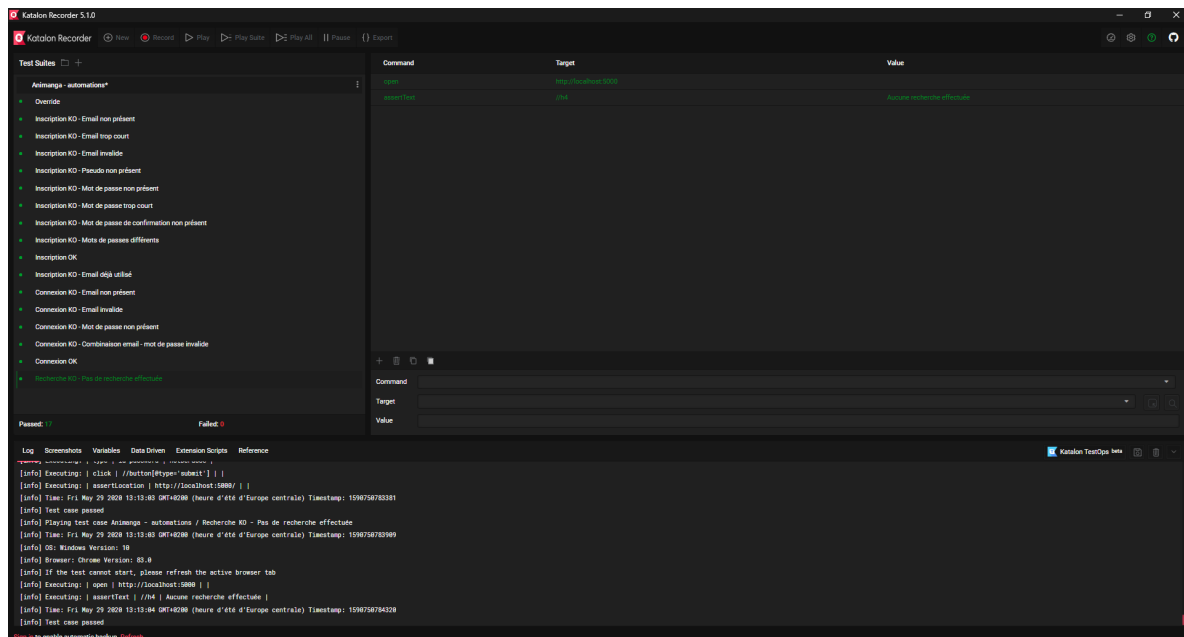
9h55 : Je termine l'affichage des favoris sur la page d'accueil. L'affichage spécifique des favoris est sur la page de profil donc pour le moment il n'y a que la page d'accueil pour voir ses propres favoris. La gestion de ces derniers ne viendra que plus tard. Je commence la mise à jour du statut de visionnement de l'anime.

10h15 : J'ai un rendez-vous GMeet avec mon référent pour vérifier mon avancement. Je lui montre ce que j'ai fait les jours précédents et ce que je suis en train de faire. La seule remarque concerne le planning. Je dois réécrire les dates car les numéros des jours sont faux et je dois mettre les cases en ● **#F34C56** si elles n'étaient pas prévues dans le planning prévisionnel.

11h40 : Je termine la mise à jour du statut de visionnement des animes pour l'utilisateur connecté. Je dois encore afficher quel statut l'anime a quand on clique sur son nom pour afficher sa carte. Je décide de le faire maintenant. Ainsi, je ne prend pas de retard.

12h : Je termine l'affichage du statut correct dans le combo box de la carte de l'anime. Il me reste seulement à afficher les listes personnelles à l'utilisateur. Je ferai cela au moment où la tâche apparaîtra dans le diagramme de Gantt. Je prend ma pause de midi.

13h : Suite au rendez-vous GMeet, je me souviens que mon référent m'avait dit de mettre des captures d'écrans de Katalon dans mon journal de bord. Voici donc une capture d'écran montrant tous les tests créés depuis le début et leur état :



Tout les tests passent. Je n'ai pas encore mis les tests concernant la mise à jour d'un anime mais cela ne saurait tarder. Pour continuer mon travail, je m'occupe de l'affichage de la page de profil. Les images utilisées comme bannière de fond ainsi que l'image de profil sont arbitraires. Il serait préférable de faire un système d'upload pour chacune des images lors d'une amélioration du projet.

13h40 : Je termine l'affichage de la page de profil de l'utilisateur. En plus de seulement pouvoir afficher la page de profile de l'utilisateur en allant sur l'url `/profile/<pseudo>`, je fais en sort de rediriger l'utilisateur sur `/profile/<pseudo utilisateur connecté>` s'il va sur `/profile` ou `/profile/`. Je mets les liens pour les futures pages d'affichage des listes ainsi que pour la gestion des favoris. Pour le moment, les liens redirigent vers une 404. J'améliorerai cela sous peu. De plus, le lien `favoris` ne s'affiche que si l'on est sur sa propre page de profil. J'ajoute le contenu de la page profile, à savoir les statistiques de l'utilisateurs ainsi que ses favoris.

14h : J'ajoute les favoris sur la page de profil. Ils s'affichent de manière différente que sur la page d'accueil. En effet, c'est bien plus ergonomique horizontalement que verticalement sur une page telle que la page de profil. J'affiche les statistiques de l'utilisateur.

16h45 : Je termine d'afficher les statistiques de l'utilisateur. Ces dernières fonctionnent de cette manière : elles ne sont basées que sur les animes marqués comme `Complétés`. Je compte chaque type et je les affiche. J'ai alors quelque chose comme ceci :

```
1  {
2    'TV': 2,
3    'Movie': 1,
4    'Ona': 0,
5    'Special': 1
6  }
```

Je termine ma journée.

Bilan

Cette journée c'est très bien passée. J'ai pu faire tout ce qu'il fallait pour la journée et aucun bug ne s'est présenté pour le moment.

J6 : mardi 2 juin 2020

Objectif

Cette journée est relativement remplie. Premièrement, je dois afficher le contenu des listes d'un utilisateur. Ensuite, je dois m'occuper de la gestion des listes. Cela comprend l'ajout de nouvelles listes, la suppression des listes existantes et le renommage des listes existantes. Enfin, je prévois de faire la gestion de l'ordre des favoris.

Déroulement

8h : Je commence par l'affichage du contenu des listes d'un utilisateur. Une route doit être faite pour récupérer les animes d'une liste.

9h : Je remarque que le javascript ordonne automatiquement les listes lorsqu'il les reçoit via un fetch. Je mets donc l'id de la liste devant le nom pour que l'ordre soit fait directement sur l'id et non pas sur le nom de la liste. Cela permet de garder l'ordre de création des listes.

9h30 : Je termine l'affichage des listes. Dès que l'on clique sur le nom d'une liste, le site montre les animes contenus dans la liste cliquée. Par défaut, c'est **Tous** qui est sélectionné, ce qui permet de voir tous les animes de toutes les listes de l'utilisateur.

Je commence la gestion des listes.

10h : Je termine l'ajout de nouvelles listes. Cependant, je dois changer la manière de récupération des listes car l'ordre d'affichage est faux.

10h30 : J'ai un rendez-vous GMeet avec mon référent pour faire le point. Je lui explique sur quoi je travaille. Je lui dis que je n'ai pas de souci. Le rendez-vous se termine rapidement.

11h : Je résous le souci d'ordre d'affichage des listes. Ma route `/get/animés` renvoie le json suivant:

```
1  {
2      'Complétés': [
3          animés ...
4      ],
5      'En cours': [
6          animés ...
7      ],
8      listes ... : [
9          animés ...
10     ]
11 }
```

Les listes présentes dans le json ne sont que les listes contenant des animés.

Je commence la suppression des listes et de leur contenu.

11h40 : La suppression des listes et de leur contenu est terminée. Il ne me reste plus qu'à faire le renommage dans lequel je me lance.

12h : Je n'ai pas encore terminé la mise à jour du nom de la liste mais il ne me reste que la partie base de données à faire. Toute la partie logique est faite. Je poursuivrai cet après-midi.

13h : Je recommence la journée en travaillant sur la mise à jour du nom d'une liste.

13h10 : Je termine la mise à jour du nom d'une liste et je passe à la mise à jour de l'ordre des favoris.

14h : La mise à jour de l'ordre des favoris est terminée. Je m'occupe de l'affichage des activités de l'utilisateur connecté. L'ordre des favoris est modifié grâce à la librairie **jQueryUI**, qui permet rendre une `div` capable de prendre en charge du drag&drop. J'utilise le drag&drop pour déplacer les animés dans l'ordre que l'utilisateur veut sur la page des favoris.

Cette tâche n'est pas présente dans le planning prévisionnel car je l'avais oubliée lors de la rédaction du planning. Je l'ai cependant rajoutée dans le *product backlog*. C'est pourquoi il n'y a pas de case orange dans le planning prévisionnel mais uniquement une case rouge.

15h30 : Les activités sont implémentées. J'affiche toutes les activités des 24 dernières heures : les anime mis en favoris et les anime mis dans des listes. Ces activités sont présentes sur la page d'accueil si l'utilisateur n'a fait aucune recherche.

16h : Au début de la session de TPI, j'ai mis en place une méthode permettant de récupérer un anime aléatoire. J'ai maintenant terminé d'afficher cette méthode sur le site via un bouton placé dans la barre de navigation.

16h45 : Je termine l'affichage de l'anime tiré aléatoirement et clos ma journée.

Bilan

Comme prévu, cette journée était plutôt bien remplie. J'ai réussi à faire plusieurs tâches indispensables et toutes fonctionnent. Il ne me reste plus que la partie synchronisation à faire au niveau technique de l'application. Il me restera ensuite les parties de documentation, d'aide, et de page à propos à terminer. Je suis content du travail effectué aujourd'hui.

J7 : mercredi 03 juin 2020

Objectif

Le but de cette journée est de commencer une des parties principales de l'application : la synchronisation entre SQLite3 et MySQL. En effet, cette partie est de très loin la plus complexe à mettre en place car aucune librairie ne permet de le faire automatiquement. C'est pour cela que je dois moi-même penser à un algorithme de synchronisation et le mettre en place. D'où les 4 jours de planification.

Déroulement

8h30 : J'ai un rendez-vous avec mes experts (M. Terrond et M. Bouille) afin de faire le point sur mon avancement dans le travail. Je leur explique le cas de la tâche manquante dans le planning prévisionnel, et je leur montre le planning dans son ensemble pour qu'ils aient une idée globale de l'avancement. Ils souhaitent que je leur fasse une démonstration. C'est alors que je décèle un bug apparu après avoir corrigé ma syntaxe hier vers 16h. Je n'avais alors pas retesté le bon fonctionnement de mon application et le bug s'est manifesté ce matin.

M. Bouille me suggère de faire une slide spécifique dans ma présentation pour toutes les fonctionnalités qui pourraient être rajoutées ou améliorées.

9h10 : Le rendez-vous est terminé. Je corrige les bugs rencontrés lors de la démonstration.

9h30 : Les bugs sont corrigés. Rien de grave, simplement une variable mal initialisée ainsi qu'un id HTML pas exclu lors d'un test javascript. Je commence à réfléchir à l'algorithme que je pourrai utiliser lors de la synchronisation.

10h30 : Je fais une première version d'algorithme. Elle n'est de loin pas optimisée. Je continue donc à chercher. Voici la première version :

```
1  foreach table {
2      stocker le nom de la table courante
3      récupérer les enregistrements et les stocker
4
5      découper les données en parties de 5k enregistrements
6      // [
7      //   [
8      //     {données}
9      //     ... x5k
10     //   ]
11     //   ... autant de tableau de 5k données pour avoir toutes les
données
12     // ]
13
14     foreach paquet in tout les paquets découpé {
15         insérer les données dans la table courante MySQL via un
insert multiple
16         // INSERT INTO table( ... colonnes) VALUES( ... values),
( ... values), ...
17     }
18 }
```

J'essaye d'améliorer mon algorithme pour le rendre moins coûteux en ressources et plus rapide.

11h40 : Je travaille encore sur la seconde version de mon algorithme. Je continuerai cet après-midi.

13h : Je reprends ma journée et je continue l'élaboration de mon algorithme de synchronisation.

13h10 : J'ai une seconde version de mon algorithme. Le voici :

```
1  foreach table {
2      stocker le nom de la table courante
3
4      récupérer le nombre d'enregistrements de Sqlite3 et le stocker
5      récupérer le nombre d'enregistrements de MySQL et le stocker
6  }
```



```

7      if nombre enregistrements Sqlite3 égale nombre enregistrements
      MySQL {
8          passé à la table suivante
9      } else {
10         vider la table MySQL
11
12         découper les données en parties de 5k enregistrements
13
14         foreach paquet in tout les paquets découpé {
15             insérer les données dans la table courante MySQL via un
            insert multiple
16         }
17     }
18 }

```

Je suis persuadé que je peux faire autrement que de découper mes données en paquets de 5000 enregistrements. Je continue à chercher une manière plus efficace.

16h : Je pense avoir une version presque finale de l'algorithme. Je décide de travailler avec les timestamps que j'ai en base. Je ne sais pas pourquoi je n'y avais pas pensé plus tôt mais avec les timestamps, je peux savoir quand est-ce que les données ont été altérées pour la dernière fois et donc trier plus facilement les enregistrements à modifier, \$ supprimer, ou à ajouter.

Il se peut qu'il y ait des modifications à venir sur cet algorithme mais voici la version actuelle avec laquelle je pense continuer l'application :

```

1  foreach table {
2      stocker le nom de la table courante
3
4      récupérer le nombre d'enregistrements de Sqlite3 et le stocker
5      récupérer le nombre d'enregistrements de MySQL et le stocker
6
7      if nombre enregistrements Sqlite3 différent de nombre
      enregistrements MySQL {
8          récupérer ids et timestamp Sqlite3 pour stocker dans tableau
      id: timestamp
9          récupérer ids et timestamp MySQL pour stocker dans tableau
      id: timestamp
10
11         trier les tableaux par id
12
13         if tableau Sqlite3 > tableau MySQL {
14             stocker ids Sqlite3 non présent dans tableau MySQL
15             retirer les ids ci-dessus du tableau Sqlite3
16
17             récupérer les enregistrements Sqlite3 correspondant aux
            ids non présent MySQL

```

```

18
19         foreach enregistrement Sqlite3 {
20             insérer dans MySQL
21         }
22     } else {
23         stocker ids MySQL non présent dans Sqlite3
24
25         foreach id MySQL {
26             supprimer l'entrée MySQL
27         }
28     }
29 }
30
31     déclarer tableau pour ids Sqlite3 dont timestamp diffère du
MySQL
32     foreach enregistrements Sqlite3 {
33         if timestamp courant différent du timestamp MySQL
correspondant {
34             ajouter l'id dans le tablea créer à cet effet
35         }
36     }
37
38     récupérer les enregistrements Sqlite3 correspondant aux ids
39     foreach enregistrement {
40         mettre à jour l'enregistrement correspondant dans MySQL
41     }
42 }

```

J'implémente cet algorithme dans l'application.

17h : Je n'ai pas terminé l'implémentation. Je continuerai demain. Je clos ma journée.

Bilan

Cette journée n'a pas été très variée. Je n'ai fait que développé mon algorithme et j'ai à peine commencé à l'implémenter dans l'application. Je terminerai l'implémentation demain je pense.

J8 : jeudi 04 juin 2020

Objectif

Aujourd'hui, je prévois de terminer l'implémentation de l'algorithme de synchronisation. Je suis entièrement dans les temps et très content de la vitesse d'avancement du projet.

Déroulement

8h10 : Je commence ma journée. J'implémenter la synchronisation entre les bases de données.

9h30 : J'ai un rendez-vous avec mon référent pour faire le point. Comme tout ce passe bien, le rendez-vous ne dure que très peu de temps. Mon référent va cependant prendre un moment pour regarder le code et nous allons faire une conférence pour corriger les parties qui ont besoin de modifier.

10h30 : Un élève (Vincent Steinmann) me demande de l'aide pour son projet. Comme j'ai de l'avance, je prend l'initiative de l'aider un moment.

11h30 : J'ai terminé d'aider Vincent. Cette aide ne m'a pas fait prendre de retard sur mon planning donc tout va bien. Je continue à implémenter la synchronisation.

12h : Je prends ma pause de midi.

13h : Je reprends l'implémentation de l'algorithme.

14h : Je remarque, lors de la mise à jour des données dans MySQL, que j'ai stocké les timestamp avec les millisecondes dans Sqlite3. Cela pose problème pour MySQL car lorsque je mets les nouveaux enregistrements, les timestamps sont arrondis.

Exemple:

J'ai un enregistrement dans Sqlite3 dont le timestamp est de 2020-05-04 14:28:30.67293

Dans MySQL, il deviendra : 2020-05-04 14:28:31

Je modifie par conséquent le code de l'ajout des timestamp dans Sqlite3 pour retirer les millisecondes à la source directement. Je remplace tous les `dt.now()` par `dt.now().strftime('%Y-%m-%d %H:%M:%S')`.

16h : La synchronisation entre les bases est terminée. Je gagne 1 jour d'avances grâce à cela. J'en profite pour corriger tous les potentiels bugs qui pourraient y avoir et faire de la documentation. De plus, ça me permettra de vérifier que je n'ai rien oublié des points mentionnés dans le cahier des charges.

Je mets à jour la documentation.

17h : Je termine ma journée.

Bilan

Cette journée a été très fructueuse. J'ai pu finir la synchronisation et j'ai 1 jour d'avance sur mon planning. Cela m'a permis de terminer les fonctionnalités de mon application. Maintenant, je vais vérifier que je n'ai rien oublié et corriger les potentiels bugs qui pourraient se présenter.

J9 : vendredi 05 juin

Objectif

Le but de cette journée est de faire de la documentation. Je vais vérifier si toutes les fonctionnalités sont correctement implémentées.

Déroulement

8h : Je remarque que je me suis trompé dans le numéro des jours sur mon planning. Ceci m'a rendu confus lors de la rédaction du journal de bord. Les jours indiqués dans le planning étaient : J1, J2, J3, J4, J5, J7, J8, J9, J10, J11, J12.

Je n'ai pas remarqué cela lors de mes mises à jour du planning et donc les plannings donnés précédemment aux experts étaient erronés.


Je corrige cette erreur.

8h30 : Le planning est corrigé. J'ajoute la partie *libraires et outils externes* .

10h : Les librairies sont documentées et j'ajoute la structure du projet dans la partie *implémentation* .

10h30 : Je remarque que je pourrais ajouter dans les scénarios de tests, toutes les possibilités d'erreurs pour l'inscription et la connexion. Cela ajoute ~10 tests supplémentaires. Étant donné que je suis relativement à jour dans la documentation, je vérifie si toutes les fonctionnalités sont bien implémentées.

10h35 : Il manque une fonctionnalité : la possibilité de supprimer un anime des favoris depuis la page d'organisation des favoris. Je décide de l'implémenter.

11h50 : La possibilité de supprimer un anime des favoris depuis la page d'organisation des favoris est implémentée. Pour pouvoir faire ceci, j'ai mis une icône, , pour pouvoir retirer les animes. Cette icône s'affiche lorsque l'utilisateur clique sur le bouton **Réorganiser les favoris** .

Je prends ma pause de midi.

13h : Je reprends ma journée et je vérifie tout le reste du projet pour m'assurer qu'il n'y a aucun bug.

14h30 : Toute l'application est propre. Je n'ai détecté qu'un seul bug et je l'ai corrigé très rapidement. Un simple problème de type. Je m'attelle à la documentation. Cela comprend corrections orthographiques et ajout de parties manquantes si le cas se présente.