

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, extraire les *user stories* de ce dernier pour pouvoir correctement rédiger mon *product backlog* et enfin rédiger les scénarios de tests fonctionnels, indispensable pour le bon fonctionnement de mon projet.

Déroulement

Je commence ma journée à 8h00. M. Terrond m'a fait parvenir mon énoncé la veille, que j'ai lu avec attention ce dernier. Par ce biais, j'ai complété avec succès la première étape de la **méthodologie en 6 étapes**, méthodologie que je vais utiliser durant tout le déroulement de ce TPI : **S'informer**.

J'ai quelques points incertains concernant mon énoncé dont un quelque peu embêtant. Je poserai mes questions à mon formateur durant la matinée. Je vais maintenant commencer à **Planifier**, seconde étape de la méthodologie utilisée. Je séparerai ma journée en tranches de 4 heures, soit par demi-journée, et remplirai des différentes tranches horaires avec les *user stories* extraites de mon cahier des charges.

8h15 : J'ai décidé d'utiliser des alias afin de nommer les jours de travail mis à disposition pour le TPI. Les jours seront nommer de **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 1 juin 2020
- J9 : mardi 2 juin 2020
- J10 : mercredi 3 juin 2020
- J11 : jeudi 4 juin 2020

8h25 : Lors de la création des *user stories* j'ai remarqué qu'il me fallait décider d'une manière de prioriser les tâches. J'ai opté pour me baser sur la méthode **MoSCoW**. Cependant les niveaux de priorité ne correspondaient pas entièrement pour un TPI. J'ai alors décidé de modifier les intitulés :

- **Must** devient ☹ **Bloquant**
- **Should** devient ⚡ **Critique**
- **Could** devient 📌 **Important**
- **Won't** devient 🤔 **Secondaire**


J'ai aussi décidé d'utiliser la syntaxe suivante afin de présenter mes *user stories* :

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

9h : J'ai fait un script bash me permettant un rassembler tout 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** pour exporter mon fichier réunissant toute ma documentation en PDF. Une fois cela fait, j'utilise un autre script bash que j'ai réalisé 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 : Descriptif de 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'ai utilisé la couleur  #006EDB comme principale. La police est Poppins, aussi utilisée dans le projet en lui même.

10h50 : J'ai eu un rendez-vous GMeet avec mon formateur pour vérifier que tout allait bien. J'ai posé la question suivante et voici la réponse donnée :

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

 Le planning que j'ai donné est un modèle permettant de suivre de façon basique l'avancée du projet. Si vous avez un planning plus précis, vous pouvez sans autre l'utiliser et comparer ensuite le vôtre avec celui que j'ai donné.

11h25 : J'ai terminé la rédaction de mon *product backlog* temporaire. Des modifications peuvent encore être apportés si j'en trouve le besoin.

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

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

14h15 : J'ai remarqué un soucis lors de la fusion des fichiers Markdown. Une partie d'un fichier se dédouble mais je ne sais pas encore pourquoi.

15h50 : Mon soucis 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 vais pouvoir commencer l'écriture des scénarios de tests fonctionnels.

16h45 : J'ai écrit une partie des scénarios de tests. Il m'en reste encore quelques un que j'ajouterais 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 que très minime sur mon planning. Je sort tout de mais satisfait de cette première journée.

J2 : mardi 26 mai 2020

Objectifs

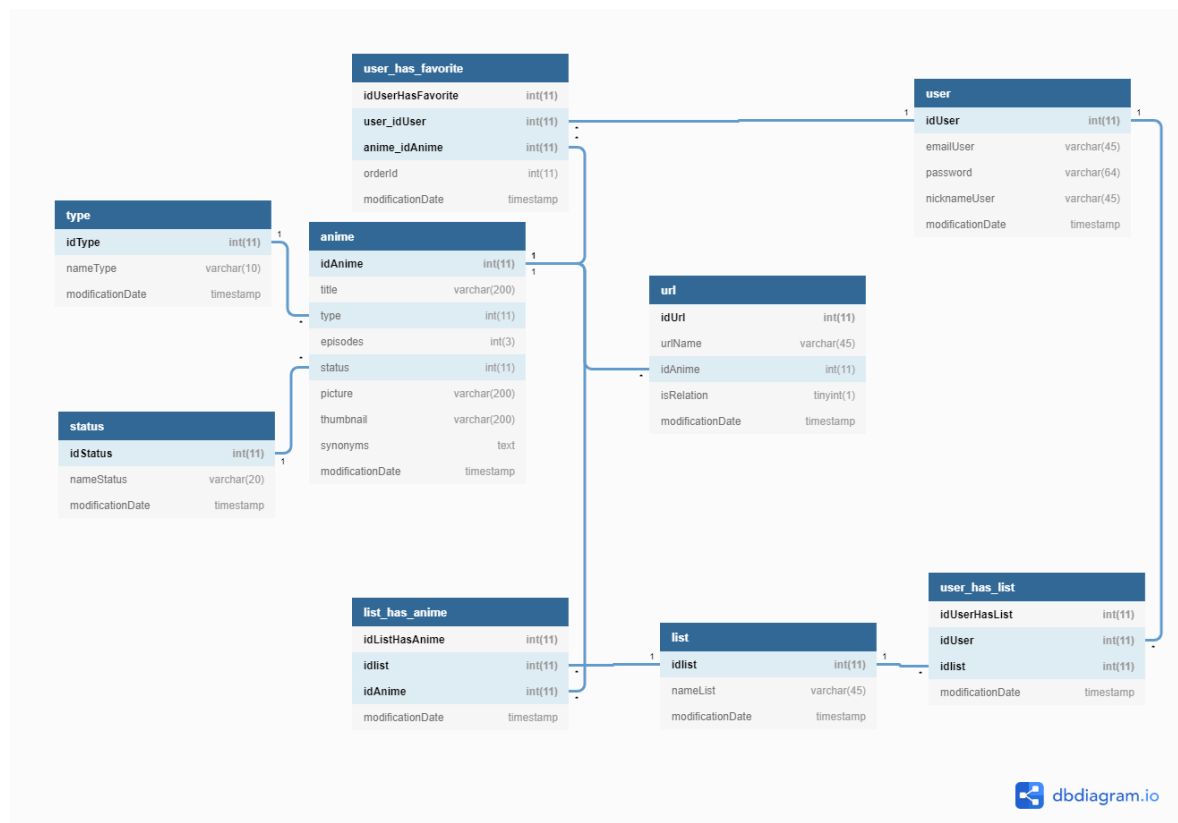
L'objectif de cette journée est premièrement de rattraper le peu de retard que j'ai eu hier sur les scénarios de tests. Ensuite, je ferai le modèle de base de données, je configurerai l'application Flask, et je ferai le code permettant d'importer les données.

Déroulement

8h : Je commence ma journée. Je dois finir les scénarios de tests que je n'avais pas pu terminer hier. Ceci ne devrait pas me prendre beaucoup de temps.

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

8h55 : J'ai réalisé le modèle de base de données que voici :



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

9h25 : J'ai terminé 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 ainsi pouvoir faire la suite du projet.

J'ai inscrit une `secret_key` à l'application Flask. Cette clef est utilisé dans les systèmes d'encryptions. Flask lui-même n'a pas besoin de cette clef mais d'autre 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 : J'ai mis en place le système de documentation automatique d'API : **Swagger**.

Pour que ce système puisse être mis en place, il faut une librairie externe nommé `flask_swagger` . De plus, certains fichiers sont indispensable au bon fonctionnement de Swagger. Le plus primordiale est la page HTML. J'ai décidé de la nommée `endpoints.html` et de la placé 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, il nous faut rajouter 2 fichiers `javascript` qui iront dans le dossier `static/js` , 2 images qui iront dans le `static/img` et enfin 1 fichier `css` qui ira 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 :



10h25 : J'ai installer 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 maintenant l'importation des données en base.

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

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

14h : J'ai terminé l'import des données et tout semble parfaitement bien s'ajouter en base. Étant donné que j'ai un peu d'avance, je vais mettre 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'ai installé mkdocs dans **WSL 2** sur ma machine pour vérifier si ma documentation compilait 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'ai effectué le test *3.1* pour vérifier que mes données soient correctement importées. Comme j'ai de l'avance de vais faire l'affichage de la *landing page*.

15h20 : J'ai terminé 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 changé puisque je met 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 vais configurer deux vérificateurs de syntaxe différent pour mon projet. Le premier est **pylint** pour Python, et le second est **eslint** pour le JavaScript. Pour eslint, je vais utilisé un preset de vérification : `airbnb`. Cela me permet d'écrire mes script 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 : J'ai corrigé les erreurs que eslint m'avait montré et je m'attaque maintenant à pylint.

Bilan