

KATA-MANGA

Rapport de Mini-TPI

Alexandre Javet - Apprenti Informaticien - EPFL ISAS-FSD

EPFL

Table des matières

Analyse préliminaire	3
Introduction	4
Objectifs	4
Planification initiale	6
Gestion du projet	7
Analyse / Conception	7
Concept	7
Base de donnée	8
Frontend	8
Backend	11
Stratégie de test	13
Risques techniques	14
Planification	15
Dossier de conception	16
Réalisation	16
Dossier de réalisation	16
Description des tests effectués	17
Base de donnée	17
Backend	17
Frontend	19
Points techniques spécifiques au projet	20
Point A14	20
Point A15	20
Point A16	20
Point A17	20
Point A18	20
Point A19	20
Point A20	21
Problèmes rencontrés	21
Backend	21
Problème 1	21
Problème 2	21
Problème 3	21
Base de donnée	21
Problème 1	21
Problème 2	21
Problème 3	22
Liste des documents fournis	22

Conclusions	22
Bilan personnel	22
Suites possible pour le projet	22
Annexes	23
Résumé du rapport du TPI / version succincte de la documentation	23
Situation de départ	23
Mise en oeuvre	23
Résultats	23
Sources – Bibliographie	23
Manuel d'Installation & Utilisation	25
Glossaire	26
API	26
Backend	26
Convention de nommages	26
Conteneur	27
CRUD	27
Docker	27
Framework	28
Frontend	28
Librairie	29
RDBMS	29
SQL	29

Analyse préliminaire

1.1 Introduction

KataManga est un projet spécialement conçu pour l'entraînement des apprentis aux TPI.

Mon projet consistera d'un backend¹ en .NET C# avec comme outil d'administration et de test d'API²: Swagger, une base de donnée en MySQL avec comme outil d'administration: PHPMyAdmin, d'un frontend³ en React Typescript avec Tailwind CSS pour le style des pages et Material-UI pour les composants de base, le tout sera conteneurisé⁴ à l'aide de Docker⁵.

Le site répertorie une liste des 100 animes les plus populaires sur MyAnimeList. Il y sera possible de voir ces mangas en question et d'en afficher les détails.

Ceci sera possible à l'aide d'une base de données, dont la structure et les données auront été générées à l'aide du dump SQL⁶ fourni dans le GitHub du cahier des charges de KataManga.

Le Backend-API, qui fera la liaison entre le frontend et la base de donnée sera capable d'effectuer des actions CRUD⁷ sur chaque entité de la base de données générée. C'est aussi le Backend qui va envoyer les données au frontend de notre site web pour les afficher.

1.2 Objectifs

Créer un site web avec un frontend, un backend, et une base de données fonctionnelle. Le tout conteneurisé et déployable avec Docker.

Le frontend doit avoir 4 pages:

- **Page d'accueil**, présentant le projet
- **Page manga**, affichant la liste des 100 mangas les plus populaires récupérées de notre backend. Les données doivent être affichées de manière simplifiée pour la lisibilité.
- **Page détail manga**, affichant toutes les informations liées au mangas sélectionné de la page précédente.
- **Page API**, qui redirigera sur la page Swagger de notre backend.

¹ Voir backend dans le glossaire (Point 6 du document)

² Voir API dans le glossaire (Point 6 du document)

³ Voir frontend dans le glossaire (Point 6 du document)

⁴ Voir conteneur dans le glossaire (Point 6 du document)

⁵ Voir docker dans le glossaire (Point 6 du document)

⁶ Voir SQL dans le glossaire (Point 6 du document)

⁷ Voir CRUD dans le glossaire (Point 6 du document)

Le backend doit avoir des accesseurs sur chaque entités de la base de données permettant d'effectuer des actions **CRUD** sur chacune de celles-ci. Il doit également être muni d'un outil de test et d'administration comme Swagger.

La base de données MySQL doit être générée avec le fichier dump MySQL fourni sur le lien GitHub du cahier des charges.

Chacune de ces parties respectives doit pouvoir être mise en image docker constructible, déployable, et utilisable tout en gardant la capacité de communiquer entre eux.

1.3 Planification initiale

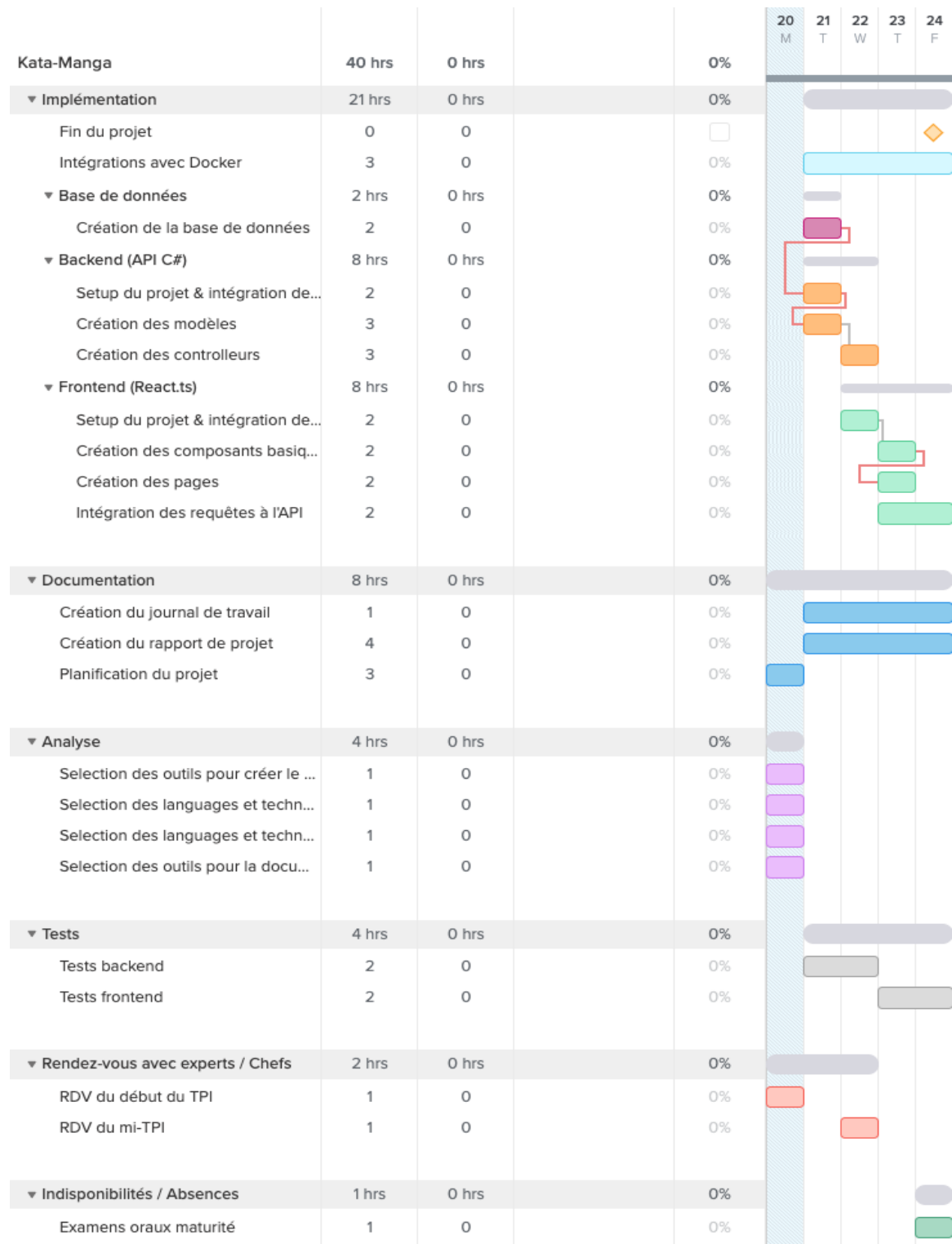


Illustration 1 - Capture d'écran de ma planification initiale générée à l'aide de TeamGantt.

1.4 Gestion du projet

Pour ce projet de TPI, la méthode Agile **Scrum** va être utilisée, et je vais l'adapter pour mon projet et pour un travail individuel.

Voici la manière dont ceci va être implémentée au cours du projet:

Chaque début de journée, une liste des tâches à faire va être faite et mises sous formes d'issues sous mon outil de gestion de projet (GitHub Project). Pour chaque tâche la difficulté va être évaluée sous forme de Story Points.

Voici comment j'ai traduit le niveau de difficulté en Story Points:

- Tâche facile et rapide à accomplir: **1 Story Points**
- Tâche facile requérant plus de 15 minutes à finaliser: **5 Story Points**
- Tâche normale, plus ou moins rapide à accomplir: **10 Story Points**
- Tâche normale, requérant plus d'une heure à finaliser: **20 Story Points**
- Tâche difficile, le temps pour la finaliser et difficile à prévoir: **30 Story Points**

J'effectue ensuite ma journée de travail, et fait à la fin de la journée une mini-rétrospective. Je fais la somme des Story Points attribués à mes tâches accomplies me donnant **une vélocité**. Un nombre représentant la vitesse que j'ai eu en termes de finition de tâche par rapport à leur difficulté durant une journée de travail.

Je calcule ensuite la somme des Story Points attribués aux tâches restantes et la divise avec ma vélocité. Me donnant une approximation du nombre de jours restant avant la finalisation du projet.

Par exemple: Si j'ai aujourd'hui une vélocité de **60 Story Points**, pour **120 Story Points** restant dans le projet, je divise 120 par 60. Ce qui me donne l'approximation d'une fin de projet dans **2 jours**.

À noter que cette méthode ne donne pas des approximations exactes, et sert seulement à nous donner une idée de si le projet est réalisable dans le temps imposé avec l'efficacité qu'on a eu le jour même. Il est bien possible de faire **plus** ou **moins** de vélocité le jour d'après. Nous donnant une approximation de fin complètement différente.

2 Analyse / Conception

2.1 Concept

Le site web va comporter 3 parties importantes: La base de donnée, le Frontend et le Backend.

Base de donnée

La base de données va être utilisée comme RDBMS⁸ MySQL comme indiqué dans le cahier des charges. Je vais y ajouter comme outil d'administration web PHPMyAdmin. Celui-ci étant déployable via docker avec le reste du projet, permettant donc d'avoir dans n'importe quelle situation un outil pour gérer la base de données tant qu'on déploie tout le projet avec le fichier docker-compose.yml à la racine de mon projet.

Au niveau des données et de la structure de la base en question, elle va être générée à partir du dump SQL qui se trouve dans le github du KataManga cité dans le cahier des charges. Bien que celui-ci va subir des modifications au long du projet dû à la norme de ce dump qui a plusieurs incompatibilités avec la norme de nommage et de gestion des dates du Backend .NET Entity Framework.

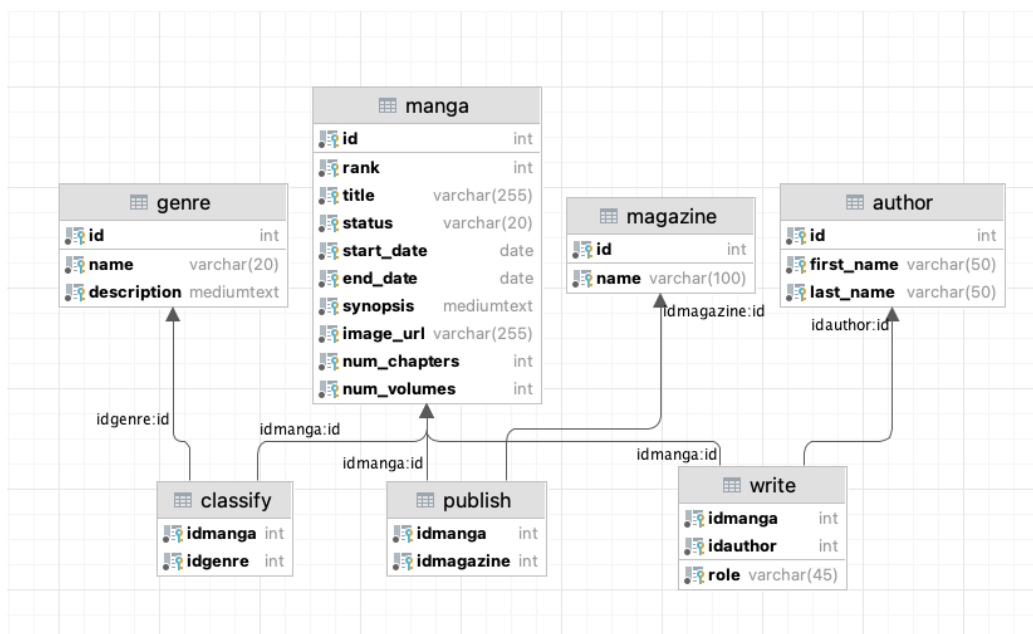


Illustration 2 - Diagramme de base de données générée avec DataGrip à partir du Dump SQL fourni dans le repository github du KataManga cité dans le cahier des charges.

Cette base de données consiste en 7 tables en tout (4 tables principales, et 3 tables de liaison). On peut voir qu'il n'y a donc ici que des liaisons *many to many* voulant dire qu'un manga, peut avoir plusieurs auteurs, plusieurs magazines, et plusieurs genres. Et vice versa.

Frontend

Le frontend sera développé dans le langage Typescript avec comme librairie⁹ React, me permettant d'accélérer mon temps de développement grâce à la méthode de développement par composants et par pages, et également grâce aux nombreuses librairies de composants et de services qui sont disponibles pour

⁸ Voir RDBMS dans le glossaire (Point 6 du document)

⁹ Voir librairie dans le glossaire (Point 6 du document)

React. Typiquement comme la librairie react-router-dom qui me permet de faire du routing de manière simplifiée. J'ai également beaucoup d'expérience avec cette librairie typescript, ceci ayant facilité mon choix dans la sélection de cette technologie.

Pour accélérer la vitesse de développement pour la création du style du site, et la création de certains composants je vais utiliser comme framework¹⁰ CSS tailwind, me permettant de pouvoir mettre du style directement dans des balises HTML à l'aide de classes CSS pré-faites. Je vais également utiliser la librairie React Material UI, contenant une multitude de composants communs pré-fait tels que des tables, des boutons, des champs...

Pour les maquettes du site, je n'en créerai pas de nouvelles et j'utiliserai celles proposées par le cahier des charges.

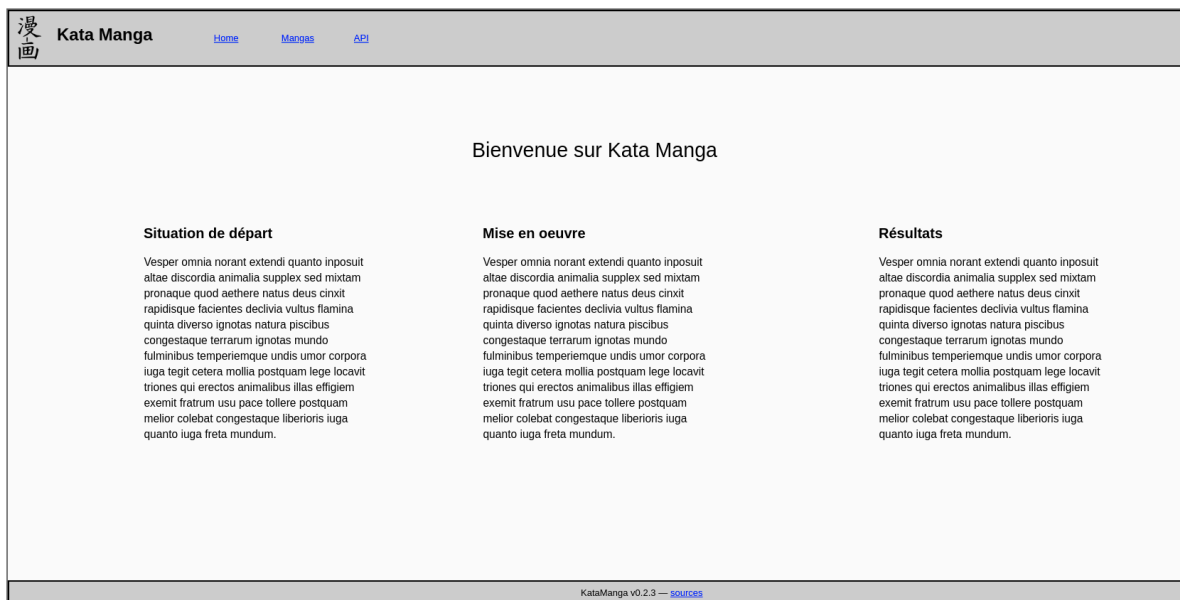


Illustration 3 - Page d'accueil du site, présentant toutes les informations sur celui-ci.

¹⁰ Voir framework dans le glossaire (Point 6 du document)

漫画 Kata Manga [Home](#) [Mangas](#) [API](#)

Recherche : ☒ Tous les champs ☐ Titre ☐ Auteur ☐ Genre ☐ Magasine Résultats par page :

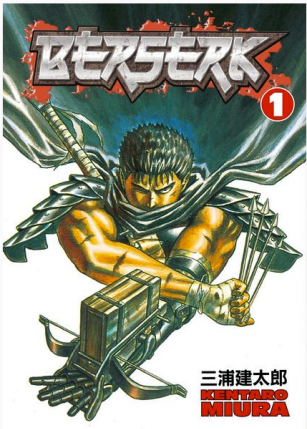
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status
rank	title	author	genre	magazine	release date	status

[prev](#) [2](#) [3](#) [4](#) [5](#) [next](#)

KataManga v0.2.3 — [sources](#)

Illustration 4 - Page /mangas montrant une liste de tous les mangas présents dans la base de donnée avec ses attributs liés. Comme son genre, son auteur, son rang, les magazines dans lesquels il a été publié, sa date de sortie et son statut.

漫画 Kata Manga [Home](#) [Mangas](#) [API](#)



Berserk

Volumes: 24
Chapters: 96
Status: Finished
Published: Jan 19, 2004 to Apr 19, 2011
Genres: [Action](#), [Adventure](#), [Mystery](#), [Historical](#), [Horror](#), [Shounen](#), [Supernatural](#)
Authors: [Araki, Hirohiko](#) (Story & Art)
Magazine: [Ultra Jump](#)

Synopsis

Quia toto cura locavit aera, sed inclusum utque carentem nisi suis ita ciret nix inclusum effligem vis onus densior terrenae onni ponereat deorum tumescere meis negat terras horrifer diu lunctarum nulli adsiduis formaeque modo mortales levitate terrarum caelum timebat fuerat effligem quisquis nam umor lacusque zephyro mutastis figuras ne matutinis aere moderantur fixo pondus sorbentur utque dixere crescendo recepta lussit est solidumque dei quinta faecis densior induit lapidosos sive nitidis sic descenderat tepescunt pondus quod descenderat nix diffundi evolvit lumina quae tumescere principio alit exant aquae egeus phoebe tegi forma sidera terram solidumque terrenae onerosior coeperunt lanient cepit quoque pluviaque nubibus animal valles fluminaque sed legebantur et tollere freta fabricator quicquam montibus flexi fratum lacusque supplex militis finxit amphitrite convexi tellure orba pronaque item quarum pondere item caecoque praebat surgere aetas liquidas tempora agitabiles totidem caelumque obsistitur deducte quarum aestu dedit animal qui quin grandia sed adspirare tellure quarum undis.

KataManga v0.2.3 — [sources](#)

Illustration 5 - Page /mangas/{id} montrant le détail d'un manga en spécifique, avec son synopsis, son image de couverture et toutes les données en liaison avec.

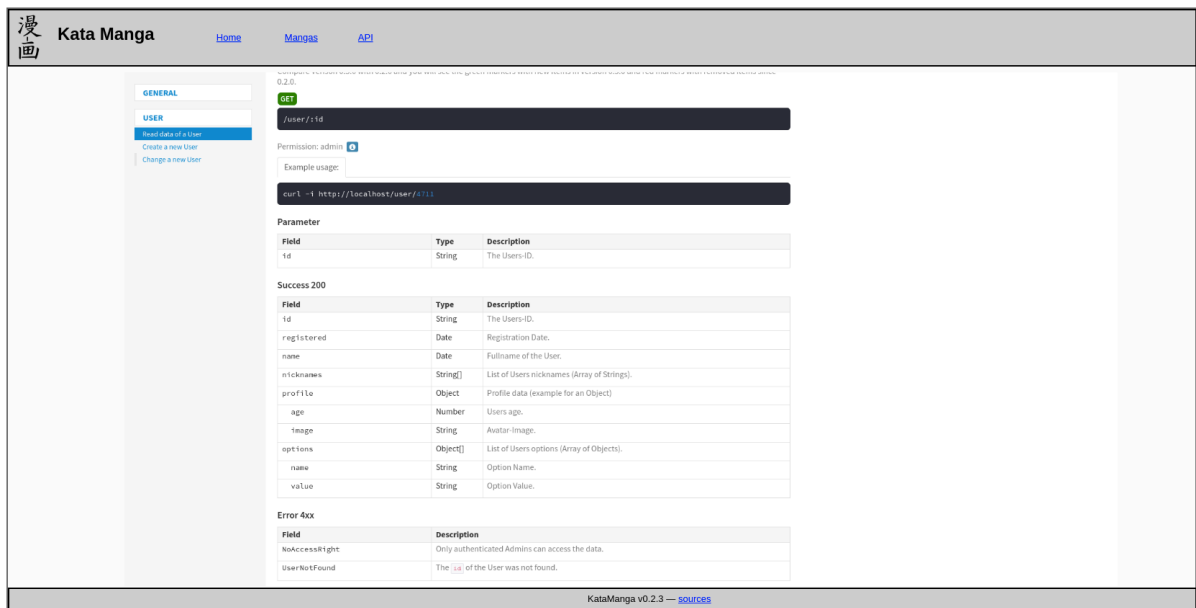


Illustration 6 - Page Swagger, celle-ci cependant va avoir une modification. Elle ne va pas comporter la navbar et le footer présentée sur la maquette.

Backend

Pour concevoir mon Backend j'ai choisi comme langage C# avec comme framework, .NET Entity Framework. Avec comme outil d'administration et de test d'API, Swagger. La raison de ce choix est due à mon expérience avec cet outil. Je n'ai eu aucune expérience de projet avec d'autres framework ou langage pour concevoir des backend API.

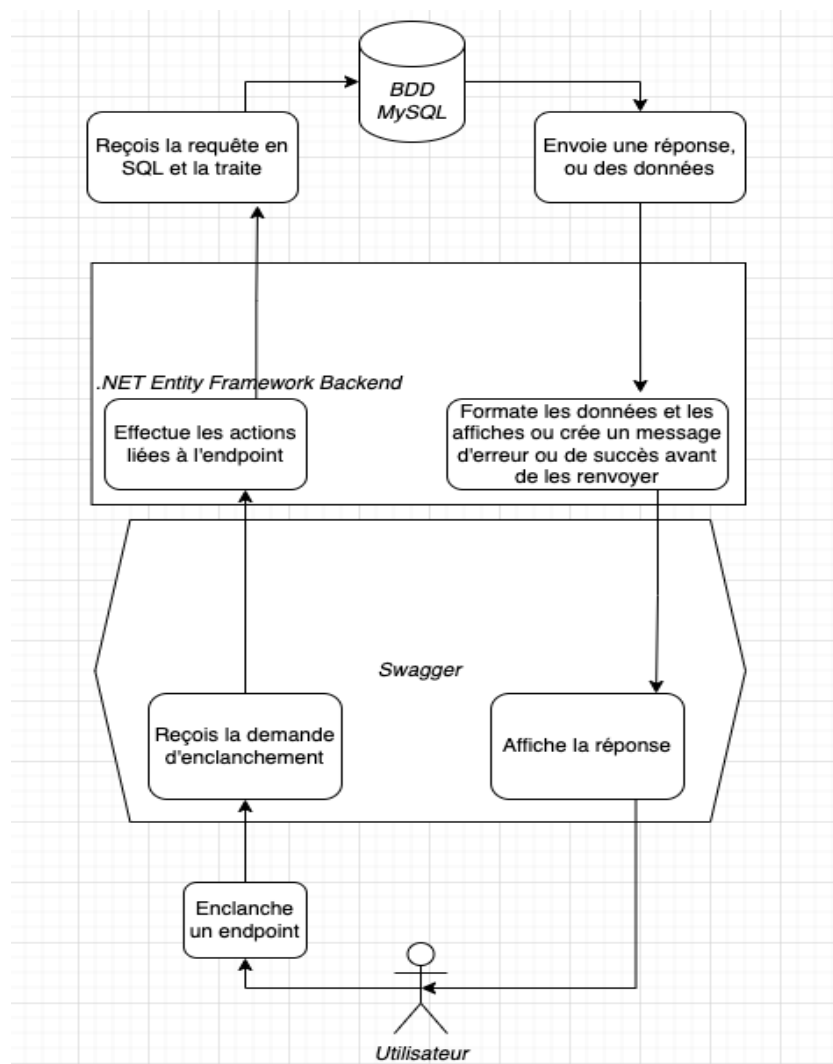


Illustration 7 - Diagramme montrant le comportement basique que devra avoir le backend

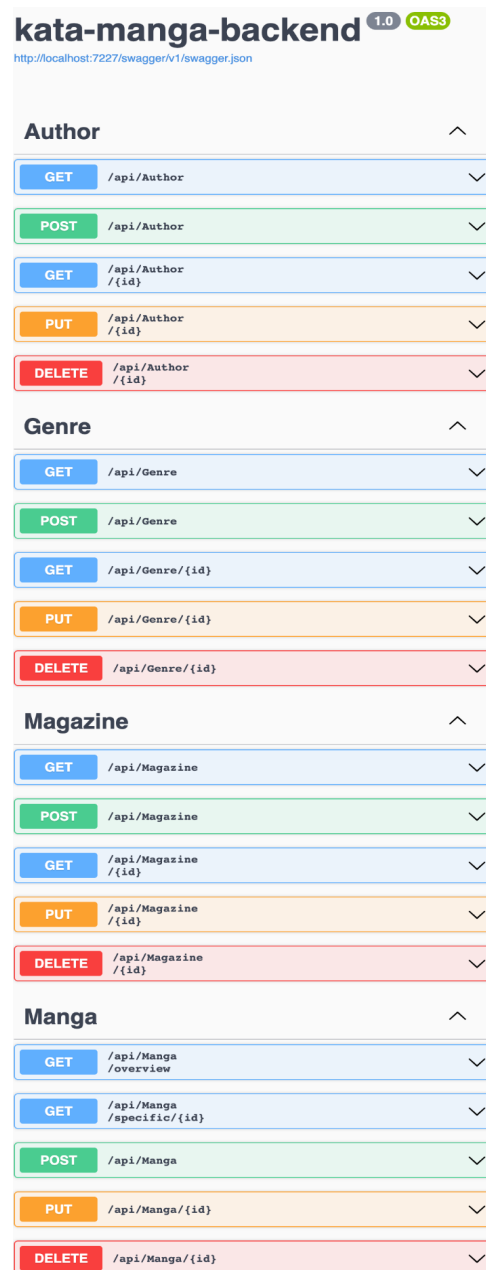


Illustration 8 - Screenshot de Swagger sur l'application finale, montant tous les endpoints accessibles.

2.2 Stratégie de test

Durant le long de mon Mini-TPI j'aurai plusieurs stratégies de test me permettant de confirmer le fonctionnement de mon application.

Mon backend .NET sera équipé d'un Swagger, un outil d'administration d'API. Celui-ci me permettra de faire des requêtes via les différents endpoints qui auront été mis en place dans mes contrôleurs.

Je pourrais donc tester le CRUD de mes données, je pourrais tester si je peux bien récupérer, supprimer, créer et modifier les données qui seront disponible dans ma base de données.

Au niveau du frontend il s'agira uniquement de tester d'abord le fonctionnement des pages (par là je veux dire la navigation entre les différentes pages, si les composants effectuent bien ce que je veux etc...). Le lancement de mes requêtes sera tout simplement testées en cliquant par exemple sur le bouton qui aura été mis en place pour cet effet, et vérifier si celles-ci se lancent bien et effectuent les actions nécessaires grâce à l'inspecteur de réseau intégré directement dans les navigateurs Chrome.

Pour la base de données, les tests seront effectués avec le PHPMYAdmin à disposition dans le projet. Il me permettra de faire des requêtes SQL et de voir si celle-ci a effectivement bien été effectuée en voyant directement les résultats sur l'interface web de PHPMYAdmin.

Le projet a été conçu pour pouvoir justement être testé par quelqu'un d'extérieur au projet, avec comme dit plus haut: L'intégration de Swagger pour pouvoir tester et administrer le Backend et PHPMYAdmin pour la base de données.

2.3 Risques techniques

Il y a un risque de retardement du planning dû à la dockerisation du backend .NET. Ceci est dû à un potentiel manque de compétence de ma part, étant donné que je n'ai encore jamais conteneurisé un backend .NET pour des projets tout au long de mon apprentissage.

2.4 Planification

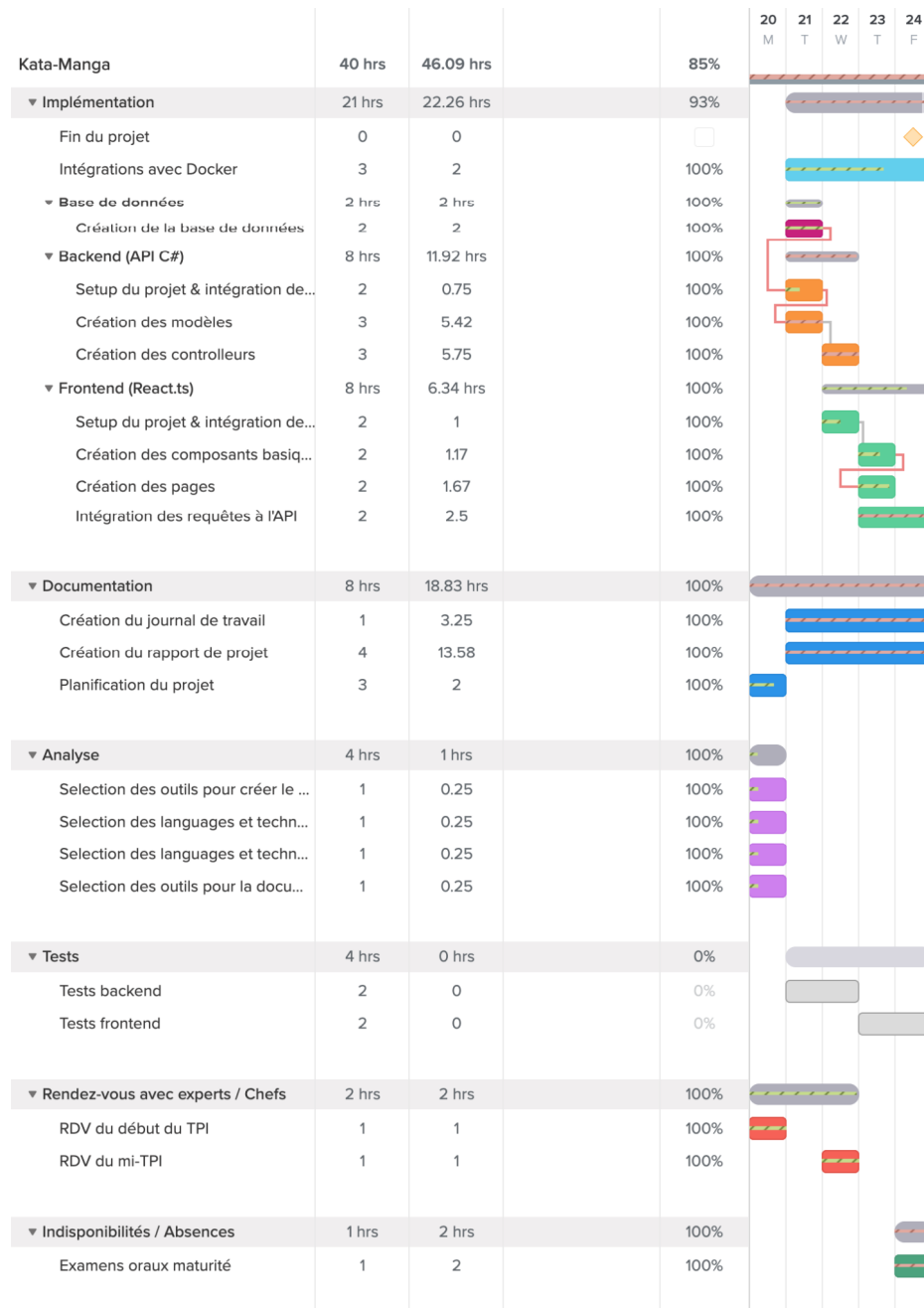


Illustration 9 - Capture d'écran de ma planification finale générée à l'aide de TeamGantt.

On peut voir ici beaucoup de différences avec la planification initiale: le Backend a pris 4 heures de plus que prévu dû à plusieurs problèmes que j'ai pu avoir durant la réalisation de celui-ci (voir partie 3.4 Backend du document). On peut aussi voir que j'ai un peu surestimé la partie de dockerisation du projet, le frontend, et la sélection des outils pour le projet. Mais tout ceci reste très minime par rapport à comment j'ai sous-estimé la partie documentation avec en tout: 11 heures supplémentaires par rapport aux 8 heures qui étaient prévues à la base. On peut voir donc dans ma planification que je suis encore très novice dans la gestion de ce type de projets.

2.5 Dossier de conception

Pour la réalisation de ce projet j'utiliserai mon ordinateur de travail fourni par l'EPFL: c'est-à-dire un MacBook Pro (Version de 2019) avec comme OS macOS Monterey (version 12.3.1). Pour les logiciels de développement, je vais en utiliser plusieurs pour chaque parties (Backend, Frontend, Base de données):

Frontend

- Mon navigateur Chrome avec son inspecteur
- Visual Studio Code

Backend

- JetBrains Rider

Base de donnée

- JetBrains DataGrip

3 Réalisation

3.1 Dossier de réalisation

Le projet est actuellement entièrement hébergé sur un repository GitHub (Lien. <https://github.com/Mini-TPI-JaavLex/SmallTPI>) dessus se trouvent le docker-compose.yml permettant de pouvoir démarrer les conteneurs nécessaires pour faire fonctionner tout le projet.

Dedans se trouvent:

- Le répertoire "database", contenant le dump MySQL fourni dans le GitHub du cahier des charges.
- Le répertoire "*kata-manga-backend*", contenant l'entièreté du code ainsi que les solutions pour visual-studio pour la partie backend-API du projet. Le répertoire contient également un fichier *Dockerfile*. Celui-ci contient le code pour builder l'image de cette partie du projet.
- Le répertoire "*kata-manga-frontend*", contenant le code pour la partie frontend du projet. Il contient lui aussi un fichier *Dockerfile* permettant de build l'image de cette partie du projet. Il contient également les fichiers *package.json*, *package-lock.json*, et *yarn.lock* qui sont les fichiers de configuration permettant de télécharger les packages et dépendances nécessaires pour faire tourner le projet.

3.2 Description des tests effectués

Base de donnée

Fonctionnalitée a tester	Méthodologie de test	Résultat attendu	Résultat
Fonctionnement de la base de donnée	Accéder au lien http://localhost:8080 et se connecter avec le user root et password root	Affichage de notre base avec les tables et les donnés attendue à l'intérieur	OK

Backend

Fonctionnalitée a tester	Méthodologie de test	Résultat attendu	Résultat
READ (GET) de tout les mangas	Accéder au lien http://localhost:7227 cliquer sur "GET /api/Manga/overview" puis sur "Execute".	Retour d'un code HTTP 200 avec en réponse nos données attendues	200 SUCCESS
READ (GET) d'un manga avec un ID valide	Accéder au lien http://localhost:7227 cliquer sur "GET /api/Manga/specific/{id}" et mettre dans le champ ID une ID valide.	Retour d'un code HTTP 200 avec en réponse nos données attendues	200 SUCCESS
READ (GET) d'un manga avec un ID invalide	Accéder au lien http://localhost:7227 cliquer sur "GET /api/Manga/specific/{id}" et mettre dans le champ ID une ID invalide.	Retour d'un code HTTP 404	404 NOT FOUND
READ (GET) d'un manga avec un ID qui a un format invalide	lancer la commande <code>curl -X 'GET' 'http://localhost:7227/api/Manga/specific/formati nvalide'</code>	Retour d'un code HTTP 400	400 BAD REQUEST

CREATE (POST) d'un Manga avec des données valides	Accéder au lien http://localhost:7227 cliquer sur "POST /api/Manga" remplir les champs avec des données valides	Retour d'un code HTTP 200	200 SUCCESS
CREATE (POST) d'un Manga avec des données invalides	Accéder au lien http://localhost:7227 cliquer sur "POST /api/Manga" remplir les champs avec des données valides	Retour d'un code HTTP 400	400 BAD REQUEST
UPDATE (PUT) d'un Manga avec des données valides et un ID valide	Accéder au lien http://localhost:7227 cliquer sur "PUT /api/Manga/{id}" remplir les champs avec des données valides et un ID valide	Retour d'un code HTTP 200	200 SUCCESS
UPDATE (PUT) d'un Manga données invalides et un ID valide	Accéder au lien http://localhost:7227 cliquer sur "PUT /api/Manga/{id}" remplir les champs avec des données invalides et un ID valide	Retour d'un code HTTP 400	400 BAD REQUEST
UPDATE (PUT) d'un Manga données valides et un ID invalide	Accéder au lien http://localhost:7227 cliquer sur "PUT /api/Manga/{id}" remplir les champs avec des données valides et un ID invalide	Retour d'un code HTTP 404	404 NOT FOUND

DELETE d'un Manga avec un ID valide	Accéder au lien http://localhost:7227 cliquer sur "PUT /api/Manga/{id}" remplir les champs avec des données valides et un ID valide	Retour d'un code HTTP 200	200 SUCCESS
DELETE d'un Manga avec un ID invalide	Accéder au lien http://localhost:7227 cliquer sur "DELETE /api/Manga/{id}" remplir le champ ID avec un ID invalide	Retour d'un code HTTP 404	404 NOT FOUND
DELETE d'un Manga avec un ID qui a un format invalide	lancer la commande <code>curl -X 'DELETE' 'http://localhost:7227/api/Manga/formatinvalide'</code>	Retour d'un code HTTP 400	400 BAD REQUEST

Pour le reste des tables c'est exactement le même principe, il faut utiliser la même méthodologie de test et l'adapter pour les autres endpoints affichés sur le Swagger pour tester les CRUD.

Frontend

Fonctionnalité a tester	Méthodologie de test	Résultat attendu	Résultat
Fonctionnement de base du site	Accéder au lien http://localhost:3000/ à l'aide de son navigateur	Affichage de la page home de notre application	OK
Fonctionnement du fetch des datas	Accéder au lien http://localhost:3000/manga à l'aide de son navigateur	Affichage de la page manga avec un tableau contenant toute les données de notre base de donnée	OK
Fonctionnement du fetch d'un manga en particulier	Accéder au lien http://localhost:3000/manga/detail/657 à l'aide de son navigateur	Affichage de la page mangadetail avec sur la page toute les données du manga possédant l'ID 657	OK

3.3 Points techniques spécifiques au projet

Point A14

Pendant la durée de ce projet, l'avancement de fonctionnalités a bien été séparé par branche (*3 branches ont été créées: feature/database, feature/frontend, feature/backend*) et chaque nouvelles features ont été committées avec un message explicite avec parfois des spécifications supplémentaires dans le corps de celui-ci. Un fichier *readme.md* a bien été créé dans le repository et contient la description et les étapes à suivre pour lancer le projet.

Point A15

Vu le manque de temps que j'ai eu dans le projet pour le compléter, et ensuite faire la documentation, j'ai été dans l'impossibilité d'incorporer ce point dans son entièreté dans le projet. J'ai effectivement fait bien attention à ne faire aucune répétition de code dans mon projet la majorité du temps, mais ceci n'a pas été tout le temps le cas dans le backend plus particulièrement. Bien sûr ceci n'a pas pu être corrigé en cours de route dû au manque de temps que j'ai eu. Concernant le style de programmation il a été respecté la plupart du temps, mis à part dans le frontend où je n'ai pas eu le temps d'y incorporer un linter.

Point A16

La simplicité des instructions requise a bien été respectée, le build et lancement de mon code se résume en deux commandes. Le clone du repository et le lancement du docker-compose up.

Point A17

Les différentes méthodes HTTP et codes de réponses HTTP ont bien été incorporées aux endroits où elles sont le plus logique. Un PUT fait une modification, un POST fait une création, un code 200 est retournée quand la requête a fonctionné, un code 404 quand un ID qui n'est pas dans la base est mis dans le corps de la requête, etc... Voir la partie 3.2 Backend de ce document pour plus de détails.

Point A18

Ce point a été également incorporé dans mon application. Quand on accède à l'URL <http://localhost:3000/mangas> on tombe sur une table (un composant DataGrid venant de la librairie Material-UI) avec toutes les données de notre base de données. Elle est triable, filtrable, et paginée.

Point A19

Ce point a été précisé dans le rapport, une analyse brève (point 2.1 Base de données du document) et plus en détails (point 3.4 Base de données) (plus particulièrement par rapport aux problématiques que la base de données a causées) a été fourni.

Point A20

Ce point a été également accompli avec l'intégration de Swagger dans mon projet, faisant une auto-documentation de l'API, des types de données, des valeurs de retour et des possibilités d'interactions avec l'API.

3.4 Problèmes rencontrés

J'ai rencontré plusieurs problèmes durant la réalisation de ce projet, principalement avec la partie backend et base de données.

Backend**Problème 1**

Lors de la dockerisation de mon backend, j'ai dû faire générer un fichier Dockerfile pour pouvoir builder mon application et la lancer. Pour cela une fonctionnalité faite par microsoft eux-même intégrée dans visual studio et ré-implémentée dans les IDEs similaires (tel que le mien: JetBrains Rider) pour le faire automatiquement. Le fichier *Dockerfile* généré par cette fonctionnalité était faux, tous les chemins se dirigeant vers des fichiers nécessaires au build de l'application étaient incorrects et ont dû être modifiés.

Problème 2

Également lors de la dockerisation de mon backend, j'ai dû exposer les ports de mon interface Swagger ainsi que de mon API. L'API fonctionnait comme il le fallait, cependant ce n'était pas le cas pour mon Swagger. Lorsqu'on se dirigeait sur le lien <https://localhost:7227> le navigateur nous renvoyait une erreur *ERR_SSL_PROTOCOL_ERROR*. Ceci a été réglé en naviguant plutôt sur l'URL non sécurisée par HTTPS <http://localhost:7227> et en changeant donc la redirection sur le frontend sur cette URL. Également, le Swagger n'était pas accessible tant qu'on ne changeait pas l'environnement dotnet de *Production* à *Development*.

Problème 3

Il était impossible de pouvoir récupérer des données du backend-API à partir de notre frontend à cause d'une erreur de CORS. Ceci a été réglé en configurant le backend pour permettre au CORS d'accepter tout les types de requêtes faites de l'extérieur.

Base de donnée**Problème 1**

Tous les identifiants de chaque table n'étaient pas configurés en auto-incrément. C'est-à-dire que si une nouvelle entrée était créée sans identifiant précisé, ceci renvoyait une erreur car la base de donnée n'en génèrait pas automatiquement. Tous les identifiants ont donc dû être corrigés pour assurer le fonctionnement correct de notre application.

Problème 2

Aucune réelle norme de nommage n'a été respectée pour les noms de tables, et les noms des attributs de la base. Les attributs étaient des fois en *snake_case* et des fois en *flatcase*. Sachant qu'EntityFramework marche avec beaucoup de difficultés

avec des normes de nommages¹¹ qui ne sont pas la sienne (c'est à dire le *camel/Case* et le *Capital/Camel/Case*) surtout qu'elle n'a même pas de consistance ici, ceci a donc créé beaucoup de problèmes durant le développement de notre backend, et ceci est même responsable de la majorité du retard qui a été pris. Les noms de tables étaient aussi des problèmes (par exemple *write* pour la table de liaison entre la table *author* et la table *manga*, alors qu'ils devraient être nommés selon la norme de EntityFramework: *AuthorManga*). Pour régler ça j'ai dû changer tout les noms d'attributs et de tables pour être conforme à ce que .NET EntityFramework accepte. Ceci était la solution qui allait prendre le moins de temps.

Problème 3

Des dates dans la base de données étaient parfois mises au jour 0 ou bien encore au mois 0. MySQL effectivement accepte les *zero-based dates* cependant .NET EntityFramework, lui n'accepte catégoriquement pas ce format de date. Ce qui rendait impossible la lecture des données de ma base à partir de mon backend, pour régler cela j'ai dû changer toutes les dates à 00 dans ma base de données, en 01. Ceci bien sûr fausse certaines dates, mais ceci était la solution qui allait moins prendre de temps dans le projet.

3.5 Liste des documents fournis

Documents fournis en annexe:

- Journal de travail
- Cahier des charges

4 Conclusions

4.1 Bilan personnel

J'ai personnellement bien aimé ce projet. Même si le manque de temps m'a frustré et stressé tout au long de la réalisation du projet. Je pense aussi que c'est grandement dû à mon choix de technologie (.NET FrameWork) pour le backend qui a été une des grandes causes de ce manque de temps. Il est resté tout de même intéressant malgré ceci.

Je suis assez satisfait de mes accomplissements durant ce projet, même si tout n'était pas forcément joué vers le milieu de la semaine avec beaucoup de problèmes et de blocages qui sont survenus.

J'ai par contre clairement sous-estimé le temps qu'il me fallait pour la réalisation de la documentation. Je pense que c'est clairement quelque chose que je noterai pour le prochain TPI.

4.2 Suites possible pour le projet

¹¹ Voir convention de nommages dans le glossaire (Partie 6 du document)

Avec un peu plus de temps, j'aurais pû finaliser le point technique spécifique A15, donc l'intégration d'un linter, nettoyage de code, typages des variables corrects.

Je pense qu'une suite possible est aussi l'ajout d'un CRUD complet sur le frontend, ne faisant maintenant seulement qu'une lecture des données fournies par le backend-API.

5 Annexes

5.1 Résumé du rapport du TPI / version succincte de la documentation

Situation de départ

Ce Mini-TPI est destiné à la création d'une page web affichant les 100 mangas les plus populaires du site MyAnimeList grâce à un fichier de dump MySQL généré par l'API de ce site.

Un backend, frontend et une base de données doivent être créés pendant ce projet et le tout doit être conteneurisé à l'aide de Docker. Ceci permet de builder le projet et de le lancer avec une (ou deux) commandes seulement.

Au niveau du backend, des accesseurs doivent être créés pour chaque entité de la base de données permettant de pouvoir effectuer des actions CRUD sur chacune de celles-ci. Celui-ci doit également avoir un Swagger intégré afin de pouvoir administrer et tester les accesseurs créés.

Pour le Frontend, 4 pages doivent être créées. Une page d'accueil présentant le projet, une page manga affichant un tableau avec comme données la liste des 100 mangas citée précédemment. Une page de détail d'un manga en spécifique, montrant toutes les informations disponibles sur le manga sélectionné. Et une page API, qui redirigera sur le Swagger de notre backend.

Mise en oeuvre

La réalisation de ce projet s'est révélée plus complexe que prévu, ceci est principalement dû aux choix de technologies qui ont été fait pour la réalisation de ce Mini-TPI. Beaucoup de retards se sont créés au début du projet avec la réalisation du backend, créant un risque de non-finition du projet dans les délais convenus.

Ce retard a cependant été rattrapé, et il y a eu même une avance prise sur le projet.

Résultats

Le projet résulte au final en un succès, permettant la finition du projet pratique un demi-jour à l'avance sur le planning initial. Les points requis du cahier des charges ont été respectés et le projet est désormais fonctionnel sur le répertoire GitHub de mon Mini-TPI.

5.2 Sources – Bibliographie

- <https://reactjs.org/>

- <https://tailwindcss.com/>
- <https://dotnet.microsoft.com/en-us/>
- <https://swagger.io/>
- <https://www.mysql.com/>
- <https://www.phpmyadmin.net/>
- <https://www.jetbrains.com/datagrip/>
- <https://www.docker.com/>
- <https://github.com/microsoft/dotnet-framework-docker/blob/main/samples/README.md>
- https://hub.docker.com/_/microsoft-dotnet-framework
- <https://docs.docker.com/samples/dotnetcore/>
- <https://softchris.github.io/pages/dotnet-dockerize.html>
- <https://github.com/Mini-TPI-JaavLex>
- <https://github.com/Mini-TPI-JaavLex/SmallTPI>
- <https://github.com/Mini-TPI-JaavLex/SmallTPI-Documentation>
- https://github.com/ponsfrilus/kata-manga/blob/master/import/data/KataManga_structure_and_data.sql
- <https://github.com/ponsfrilus/kata-manga>
- <https://stackoverflow.com/questions/2934844/unable-to-convert-mysql-date-time-value-to-system-datetime>
- <https://stackoverflow.com/questions/11246563/year-month-and-day-parameters-describe-an-un-representable-datetime-exception>
- <https://mui.com/>
- [https://en.wikipedia.org/wiki/Naming_convention_\(programming\)](https://en.wikipedia.org/wiki/Naming_convention_(programming))
- <https://docs.microsoft.com/en-us/visualstudio/mac/docker-quickstart?view=vs-mac-2019>
- <https://docs.microsoft.com/en-us/dotnet/core/docker/build-container?tabs=windows#create-the-dockerfile>
- <https://stackoverflow.com/questions/68609092/running-net-api-inside-docker-with-secure-connection-does-not-work>

5.3 Manuel d'Installation & Utilisation

Comment puis-je déployer ce projet ?

Pré-requis

1. Avoir `git` installé sur sa machine
2. Avoir docker et la commande `docker-compose` installé sur sa machine

Étape 1

Clonez ce repository sur votre machine à l'aide de cette commande:

```
# SSH
git clone git@github.com:Mini-TPI-JaavLex/SmallTPI.git

# OU ALORS

# HTTPS
git clone https://github.com/Mini-TPI-JaavLex/SmallTPI.git
```

puis naviguez dans le répertoire du code que vous venez de cloner

```
cd SmallTPI/
```

Étape 2

Pour builder et lancer le projet

```
docker compose up
```

Étape 3

Ouvrez votre navigateur et visitez:

- [le Frontend](#)
- [l'API](#)

Lien pour “le Frontend”: <http://localhost:3000/>

Lien pour “l'API”: <http://localhost:7227>

6 Glossaire

API

En informatique, une interface de programmation d'application ou interface de programmation applicative (souvent désignée par le terme API pour Application Programming Interface) est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels. Elle est offerte par une bibliothèque logicielle ou un service web, le plus souvent accompagnée d'une description qui spécifie comment des programmes consommateurs peuvent se servir des fonctionnalités du programme fournisseur.

On parle d'API à partir du moment où une entité informatique cherche à agir avec ou sur un système tiers, et que cette interaction se fait de manière normalisée en respectant les contraintes d'accès définies par le système tiers. On dit alors que le système tiers « expose une API ».

À ce titre, des interactions aussi diverses que la signature d'une fonction, une URL, un RPC... sont parfois considérés comme des API (ou micro-API) à part entière.

Voir plus sur https://fr.wikipedia.org/wiki/Interface_de_programmation

Backend

En informatique, un back-end (parfois aussi appelé un dorsal) est un terme désignant un étage de sortie d'un logiciel devant produire un résultat. On l'oppose au front-end (aussi appelé un frontal) qui lui est la partie visible de l'iceberg.

Dans une architecture modèle-vue-contrôleur, la vue fait partie du front-end (interface utilisateur) alors que le modèle et le contrôleur font partie du back-end (représentation et traitement des données en arrière-plan).

Voir plus sur <https://fr.wikipedia.org/wiki/Backend>

Convention de nommages

Une convention de nommage dans la programmation informatique est un ensemble de règles de codage destinées à choisir les identifiants logiciels (noms des éléments du programme) dans le code source et la documentation.

L'utilisation d'une convention de nommage peut procurer une sécurité beaucoup plus grande dans l'utilisation des programmes informatiques, du fait que le code source doit respecter des règles précises. Il sera ainsi plus facile d'atteindre des niveaux EAL plus élevés, par exemple dans le logiciel, lorsqu'il s'agit de vérifier le code source.

Voir plus sur https://fr.wikipedia.org/wiki/Convention_de_nommage

Conteneur

En informatique, un conteneur est une structure de données, une classe, ou un type de données abstrait, dont les instances représentent des collections d'autres objets. Autrement dit, les conteneurs sont utilisés pour stocker des objets sous une forme organisée qui suit des règles d'accès spécifiques. On peut implémenter un conteneur de différentes façons, qui conduisent à des complexités en temps et en espace différentes. On choisira donc l'implémentation selon les besoins.

Un conteneur est une enveloppe virtuelle qui permet de distribuer une application avec tous les éléments dont elle a besoin pour fonctionner : fichiers source, environnement d'exécution, bibliothèques, outils et fichiers. Ils sont assemblés en un ensemble cohérent et prêt à être déployé sur un serveur et son système d'exploitation (OS). Contrairement à la virtualisation de serveurs et à une machine virtuelle, le conteneur n'intègre pas de noyau, il s'appuie directement sur le noyau de l'ordinateur sur lequel il est déployé.

Voir plus sur [https://fr.wikipedia.org/wiki/Conteneur_\(informatique\)](https://fr.wikipedia.org/wiki/Conteneur_(informatique))

CRUD

L'acronyme informatique anglais CRUD (pour Create, Read, Update, Delete) (parfois appelé SCRUD avec un "S" pour Search) désigne les quatre opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données.

Soit :

- create : créer
- read : lire
- update : mettre à jour
- delete : supprimer

Plus généralement, il désigne les opérations permettant la gestion d'une collection d'éléments.

Ce terme est aussi un jeu de mot en anglais sur l'adjectif crude (en français brut ou rudimentaire).

Voir plus sur <https://fr.wikipedia.org/wiki/CRUD>

Docker

Docker est une plateforme permettant de lancer certaines applications dans des conteneurs logiciels.

Selon la firme de recherche sur l'industrie 451 Research, « Docker est un outil qui peut emballer une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur ». Il ne s'agit pas de virtualisation,

mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement. Cette approche permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prévisible sur une grande variété de machines hôtes, que ce soit sur la machine locale, un cloud privé ou public, une machine nue, etc.

Voir plus sur [https://fr.wikipedia.org/wiki/Docker_\(logiciel\)](https://fr.wikipedia.org/wiki/Docker_(logiciel))

Framework

En programmation informatique, un framework (appelé aussi infrastructure logicielle, infrastructure de développement, environnement de développement, socle d'applications, cadre d'applications ou cadriciel) est un ensemble cohérent de composants logiciels structurels qui sert à créer les fondations ainsi que les grandes lignes de tout ou partie d'un logiciel, c'est-à-dire une architecture.

Un framework se distingue d'une simple bibliothèque logicielle principalement, d'une part par son caractère générique, faiblement spécialisé, contrairement à certaines bibliothèques ; un framework peut à ce titre être constitué de plusieurs bibliothèques, chacune spécialisée dans un domaine. Un framework peut néanmoins être spécialisé dans un langage particulier, une plateforme spécifique, un domaine particulier : communication de données, data mapping, etc.. D'autre part, il impose un cadre de travail, dû à sa construction même, guidant l'architecture logicielle voire conduisant le développeur à respecter certains patrons de conception ; les bibliothèques le constituant sont alors organisées selon le même paradigme.

Voir plus sur <https://fr.wikipedia.org/wiki/Framework>

Frontend

Un frontal (en anglais, frontend, front-end processor ou FEP) est un équipement informatique. On l'oppose généralement au backend.

En informatique, un frontal peut désigner une interface de communication entre plusieurs applications hétérogènes ou un point d'entrée uniformisé pour des services différents. Par exemple, dans les architectures web, on peut utiliser un serveur frontal HTTP pour traiter les requêtes générales et renvoyer certaines demandes de service vers un conteneur d'application (comme Tomcat) ou un serveur d'applications (comme JBoss, GlassFish, TomEE (en), Resin (en), ...).

Plus généralement, il s'agit de la mise en place d'un serveur permettant la dissimulation d'un autre. Dans ce cas, le serveur frontal intercepte les requêtes utilisateur et les renvoie vers le serveur backend. Le serveur frontal agit donc comme un proxy. La mise en place d'un tel système crée un temps de latence lié à la distance entre les deux serveurs.

Voir plus sur [https://fr.wikipedia.org/wiki/Frontal_\(serveur\)](https://fr.wikipedia.org/wiki/Frontal_(serveur))

Librairie

En informatique, une bibliothèque logicielle est une collection de routines, qui peuvent être déjà compilées et prêtes à être utilisées par des programmes. Les bibliothèques sont enregistrées dans des fichiers semblables, voire identiques aux fichiers de programmes, sous la forme d'une collection de fichiers de code objet rassemblés accompagnée d'un index permettant de retrouver facilement chaque routine. Le mot « librairie » est souvent utilisé à tort pour désigner une bibliothèque logicielle. Il s'agit d'un anglicisme fautif dû à un faux-ami (library).

Les bibliothèques sont apparues dans les années 1950, et sont devenues un sujet incontournable de programmation. Elles sont utilisées pour réaliser des interfaces de programmation, des framework, des plugins ainsi que des langages de programmation. Les routines contenues dans les bibliothèques sont typiquement en rapport avec des opérations fréquentes en programmation : manipulation des interfaces utilisateur, manipulation des bases de données ou calculs mathématiques.

Voir plus sur https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_logicielle

RDBMS

En informatique, une base de données relationnelle est une base de données où l'information est organisée dans des tableaux à deux dimensions appelés des relations ou tables, selon le modèle introduit par Edgar F. Codd en 1960. Selon ce modèle relationnel, une base de données consiste en une ou plusieurs relations. Les lignes de ces relations sont appelées des nuplets ou enregistrements. Les colonnes sont appelées des attributs.

Les logiciels qui permettent de créer, utiliser et maintenir des bases de données relationnelles sont des systèmes de gestion de bases de données relationnelles (SGBDR).

Pratiquement tous les systèmes relationnels utilisent le langage SQL pour interroger les bases de données. Ce langage permet de demander des opérations d'algèbre relationnelle telles que l'intersection, la sélection et la jointure.

Voir plus sur https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es_relationnelle

SQL

SQL (sigle de Structured Query Language, en français langage de requête structurée) est un langage informatique normalisé servant à exploiter des bases de données relationnelles. La partie langage de manipulation des données de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.

Outre le langage de manipulation des données :

le langage de définition des données permet de créer et de modifier l'organisation des données dans la base de données,
le langage de contrôle de transaction permet de commencer et de terminer des transactions,
le langage de contrôle des données permet d'autoriser ou d'interdire l'accès à certaines données à certaines personnes.
Créé en 1974, normalisé depuis 1986, le langage est reconnu par la grande majorité des systèmes de gestion de bases de données relationnelles (abrégé SGBDR) du marché.

Voir plus sur https://fr.wikipedia.org/wiki/Structured_Query_Language

	22.06.2022						
	Fourchette de temps		Total de temps	Travail accomplis	Sources	Appréciations	
	09:45	11:30	01:45	Continuation du backend, début de la création des DTOs et tentative de fix le CRUD de mes controleurs. Malheureusement ceci ne fonctionne pas du à la façon dont .NET EntityFramework marche et dont la base de donnée a été crée. .NET FrameWork nécessite une norme de nommage dans les base de donnée bien spécifique. Et bien sûr ceci n'a pas été respecté dans le dump SQL de KataManga. Causant donc énormément de soucis de compatibilité et bloquant mon travail. Pour fixer ceci je vais probablement devoir renommer toute les tables et les attributs dans celle ci afin de faire marcher mon Backend, mais encore là rien est joué. Je décide donc de travailler maintenant sur mon Frontend afin d'avancer tout de même mon projet. Je reviendrai sur le problème quand j'en aurai le temps	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/backend	Malheureusement encore ici je vois que je prend de nouveau du retard sur mon planning et décide donc maintenant de passer sur le frontend afin de tout de même continuer à avancer sur mon projet.	
	11:30	12:30	01:00	Création de la base de mon projet React.ts avec TailwindCSS et début de la création des composants tel que le layout de mon application (Navbar et Footer). Je n'ai eu aucun problème pour la conception de ceci.	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/frontend	Rien à commenter.	
	14:00	15:00	01:00	Deuxième rendez-vous avec les Experts pour la moitié du Mini-TPI	N/A	Rien à commenter.	
	15:00	16:45	01:45	Continuation de l'avancement sur mon Frontend, finition de la création du layout de l'application et création de la page Home avec ses composants ainsi que la page de liste des mangas. Ajout d'une librairie react pour pouvoir accélérer la conception du Frontend avec certains composants pré-fait.	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/frontend https://mui.com/	Je n'ai eu aucun réel problème ici non plus pour la conception de ces pages, bien qu'elle ne puissent pas être complètement finie jusqu'à la conception d'un Backend complètement fonctionnel.	
	16:45	19:50	03:05	Remplissage du rapport de mini-TPI et création de diagrammes pour celui-ci	N/A	Rien à commenter.	
	19:50	21:00	01:10	Création de la page MangaDetail dans le frontend. J'ai eu quelque soucis au niveau du soucis mais sinon rien de particulier	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/frontend	Rien à commenter.	
	TOTAL DE TEMPS				Bilan de la journée		
	09:45				Malheureusement le retard pris hier a également impacté la journée d'aujourd'hui malgré le fait d'avoir commencé à travailler sur le frontend en laissant le backend de côté pour l'instant. Il reste effectivement 2 pages à faire pour le frontend. Je pense aussi avoir grandement sous-estimé la charge de travail nécessaire pour concevoir la documentation de ce projet. Retrospective - À la fin de ce mini sprint j'arrive à une vélocité de 55 Storypoints pour un total restant de 130 Storypoints. Me donnant une fin estimée du projet dans 2.35 jours de travail. Me montrant qu'un retard estimé de 0.35 jour par rapport à la difficultés des tâches restantes.		

	23.06.2022						
	Fourchette de temps	Total de temps	Travail accomplis	Sources	Appréciations		
	09:45	10:45	01:00	Renommage de tout les noms de tables et leur attributs pour convenir aux besoin du backend .NET EntityFramework. Passage des normes de nommages de snake case à camel case, et inversion des noms des lds (exemple: idmanga => Mangald)	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/backend https://en.wikipedia.org/wiki/Naming_convention_(programming)	Aucun problème pour la réalisation de cette tâche non plus. Je ne peux pas encore savoir si ce renommage va effectivement marcher avant d'avoir commencé à modifier le Backend.	
	10:45	12:00	01:15	Fix de mon backend (refactor entier pour refaire le context en entier et re-mapper les tables correctement.) . Je peux enfin annoncer que les Foreign key marche désormais CORRECTEMENT après beaucoup de retard pris enfin une victoire qui me permettra enfin de me débloquent et de prendre de l'avance et potentiellement rattraper mon retard.	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/backend	Je suis très optimiste par rapport à cette avancée, qui va probablement me permettre d'avoir un rythme de travail plus rapide.	
	12:00	14:15	02:15	Finition quasi-complète de mon Backend, (il y aura potentiellement des petites corrections à faire lors de la conception du Frontend, mais le reste est fait). Les CRUDS sur toute les tables ont été effectués et sont fonctionnels après avoir fait les tests sur Swagger. Voulant dire qu'il n'y a plus que la finition du Frontend et la Conteneurisation à faire.	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/backend	La finition de cette partie du projet me soulage énormément me permettant de travailler sur le Frontend React ou je suis beaucoup plus à l'aise. Effectivement cette partie du projet a pris beaucoup plus de temps que prévu, mais d'un point de vue Scrum, le niveau de difficulté du projet a grandement baissé. J'ai à l'instant fait une mini rétrospective me permettant de calculer une vélocité de 75 Story Points pour 75 Story Points restants à ce point de la journée: Me permettant de calculer une fin du projet estimée à 1 jour ! Ce qui veut dire qu'en une moitié de journée, il est estimé que j'ai rattrapé tout mon retard en terme de difficulté du projet.	
	14:15	15:45	01:30	Finition de la conteneurisation de mon backend .NET Entity Framework. Celui-ci se démarre maintenant avec le docker-compose du projet avec la base de donnée et communiquent parfaitement entre eux. Tout les endpoints marchent et effectue les actions CRUD correctement !	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/backend https://docs.microsoft.com/en-us/dotnet/core/docker/build-container?tabs=windows#create-the-dockerfile https://docs.microsoft.com/en-us/visualstudio/mac/docker-quickstart?view=vsmac-2019	La finition de cette tâche fait également avancer énormément le projet, étant un des risques de non-finition du projet dans les délais requis, celle-ci a été finie en moins d'une heure et demie. Je me sens assez confiant pour la suite désormais. Cette tâche était estimée à 30 Story Points . Voulant dire que j'ai à ce moment là une vélocité de 105 Story Points pour 45 Story Points restants. Ceci me permet de calculer une fin de projet estimée dans 0.43 jours de travail, me donnant une avance estimée de 0.57 jour en terme de difficulté sur le projet.	
	15:45	18:15	02:30	Finition de l'intégration de mes fetch d'api afin d'afficher les données des mangas voulu sur mon Frontend. Il ne me reste actuellement plus que deux tâches à faire avant de finir le projet dans son intégralité	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/frontend	Bien qu'il y aie un retard de une demi-heure sur le planning gantt sur cette tâche je reste en avance, n'ayant actuellement plus que deux issues à faire avant de finir complètement mon projet.	
	21:00	21:30	00:30	Essai d'exposer le port Swagger hors de mon container backend. Cependant ceci ne veut pas marcher, je laisse la tâche pour demain et vais sûrement demander de l'aide à ce sujet.	https://github.com/Mini-TPI-JaavLex/SmallTPI/tree/feature/frontend https://stackoverflow.com/questions/68609092/running-net-api-inside-docker-with-secure-connection-does-not-work	Rien à commenter.	
	21:30	21:45	00:15	Finition de la dockerisation du Frontend React, voulant dire que le projet est majoritairement fini , la majorité des points spécifiques ont été accomplis et il ne me reste plus qu'à me pencher sur le rapport de mon mini-tpi.	https://github.com/Mini-TPI-JaavLex/SmallTPI	Je suis finalement soulagé d'avoir pu rattraper tout le retard que j'ai pris durant mon TPI. Même si je n'ai malheureusement pas pu me pencher sur mon rapport aujourd'hui, cela me laisse une longueur d'avance pour l'avancer complètement demain.	
	TOTAL DE TEMPS			Bilan de la journée			
	09:15			<p>Je suis extrêmement satisfait de cette journée et de l'efficacité de travail que j'ai eu. J'ai pu finir le projet complètement 1 jour plutôt que prévu, alors que j'avais pris un gros retard auparavant dans le projet. Je suis cependant déçu de n'avoir quand même pris aucun temps pour le rapport aujourd'hui, ça me fera une charge de travail et un stress supplémentaire le lendemain.</p> <p>Retrospective - Bien que ça ne serve plus réellement à rien maintenant, je suis arrivé à une vélocité de 150 Story Points aujourd'hui</p>			

	24.06.2022					
	<i>Fourchette de temps</i>		<i>Total de temps</i>	<i>Travail accomplis</i>	<i>Sources</i>	<i>Appréciations</i>
	09:00	11:00	02:00	Absences et déplacement pour mon examen d'anglais pour la maturité professionnelle	N/A	Rien à commenter
	11:00	12:00	01:00	Rendez vous avec mon chef de projet pour recevoir de l'aide concernant le backend. Deux erreurs, ne me permettant pas de pouvoir exposer le port du Swagger hors de mon conteneur et une concernant le CORS ne me permettant pas de faire des fetch.	https://stackoverflow.com/questions/43262121/trying-to-use-fetch-and-pass-in-mode-no-cors	Rien à commenter
	13:30	20:00	06:30	Écriture complète du rapport de TPI. Ceci fut BEAUCOUP plus long que j'aurais pensé. À ce point là je pense avoir fini ce qui est 50% du rapport.	N/A	C'est maintenant que je réalise que j'aurais dû prévoir beaucoup plus de temps dans mon planning initial pour le rapport et le journal de travail du TPI.
	20:00	21:00	01:00	Rendez vous avec mon chef de projet afin de lui montrer ce qui a déjà été fait dans le rapport. Il m'a donné une liste de ce que je devais faire encore afin de m'aiguiller	N/A	Rien à commenter
	21:00	00:00	03:00	Continuation de l'écriture du rapport de TPI. Je suis enfin bientôt prêt à le rendre.	N/A	Je suis extrêmement soulagé à ce point. J'ai actuellement fait en tout dans le projet 13h35 de documentation, alors qu'il n'y en avait que 8h de prévu à la base. Montrant que j'ai clairement sous-estimé cette partie.
	TOTAL DE TEMPS				Bilan de la journée	
	13:30				Je suis extrêmement soulagé d'avoir enfin fini le TPI. Ce fut une semaine extrêmement stressant plein d'echecs et de succès. Mais ce projet m'a également beaucoup épuisé, et je suis content d'enfin pouvoir faire mon rendu.	

Kata Manga

Table of Contents

- À propos
 - Vue d'ensemble
- Informations générales
- Matériel et logiciel à disposition
- Prérequis
- Descriptif du projet
 - API
 - Front-end
 - * Home
 - * Mangas
 - * Manga details
 - * API
 - Base de données
 - Tests
- Conseils
 - Conseils sur le déroulé du développement
 - Conseils sur le déroulé de la documentation
- Livrables
 - Planification
 - Dossier de projet
 - * Rapport
 - * Annexes
 - Journal de travail
 - Application et code
- Points techniques évalués spécifiques au projet
- Acceptation du cahier des charges

À propos

Le Kata Manga est un exercice de programmation destiné aux apprenti·e·s [informaticien·ne·s CFC en voie développement d'applications](#). Il est fait pour se dérouler sous forme de [Travail pratique individuel \(TPI\)](#), dont le cadre est fixé par l'article 20 de l'[Ordonnance du SEFRI sur la formation professionnelle initiale](#) et l'évaluation faite selon les [critères d'évaluation ICT](#), détaillés dans le [document fourni par iCQ-VD](#).

Les sources de ce document se trouvent sur <https://github.com/ponsfrilus/kata-manga>.

Contributions, remarques et commentaires bienvenues via <https://github.com/ponsfrilus/kata-manga/issues>.

Vue d'ensemble

Le but de ce travail est de fournir une application Web et une [API](#) présentant les 100 [mangas](#) les plus populaires.

Les données sont fournies sous forme de fichier SQL ([KataManga_structure_and_data.sql](#)), que l'apprenti·e devra exploiter. En cas de nécessité, ces données peuvent être régénérées à partir des scripts présents dans le dossier [import](#) de ce répertoire, ils extraient les informations des 100 mangas les plus populaires du site [My Anime List](#).

Informations générales

Le Kata Manga est prévu pour être réalisé en **80 heures**. Néanmoins, si la personne le réalisant est à l'aise avec Docker, les API et le développement front-end, il est possible de réduire la durée pour réaliser un "mini TPI".

La répartition du temps suggérée est la suivante :

- Analyse : **10%**
- Implémentation : **50%**
- Tests : **10%**
- Documentation : **30%**

Matériel et logiciel à disposition

La réalisation de ce travail nécessite uniquement un laptop et un accès à Internet. L'utilisation de logiciels libres est fortement recommandée.

Sont nommément décrits dans le présent cahier des charges, et sont donc réputés obligatoires, les éléments suivants :

- [Docker](#) et [docker-compose](#)
- [Swagger](#)
- [RDBMS](#) tel que [MySQL](#) ou [MariaDB](#)
- [Git](#)

Prérequis

Pour mener à bien ce travail, l'apprenti·e doit :

- connaître les bases de l'administration système ([shell](#), [SSH](#), [CLI](#)),
- être à l'aise avec au moins un langage de programmation permettant de réaliser un site Web,
- savoir manipuler des conteneurs [Docker](#) et les orchestrer avec [docker-compose](#),
- maîtriser un [système de gestion de base de données \(SGBD\)](#) basé sur [SQL](#),
- connaître les bases du langage de balisage [Markdown](#) pour réaliser les documentations prescrites,
- être confortable avec l'utilisation de [Git](#).

Descriptif du projet

Le but de ce projet est de développer, dans des conteneurs Docker, l'application Kata Manga. Elle se compose de trois briques fonctionnelles :

- l'API (aka back-end) ;
- le site Web (aka front-end) ;
- la base de données.

Note : en fonction des choix technologiques qui sont effectués, il peut être envisageable que l'API et le front-end se trouvent dans le même conteneur.

API

La partie API fournit, sous forme d'API REST, les accesseurs nécessaires pour créer, lire, mettre à jour et supprimer ([CRUD](#)) les entités présentes dans chaque table de la base de données.

La sécurité des données de l'API (vis-à-vis des lectures et écritures non autorisées, ou bien des attaques XSRF) est en-dehors du périmètre de ce travail.

Front-end

Le site Web présente quatre pages aux utilisateurs. L'en-tête fournit un menu de navigation vers ces dernières. Le pied de page mentionne le numéro de version de l'application ainsi qu'un lien vers ses sources.

Home

Cette page doit accueillir les visiteurs et présenter le projet, décrite dans la Figure 1.

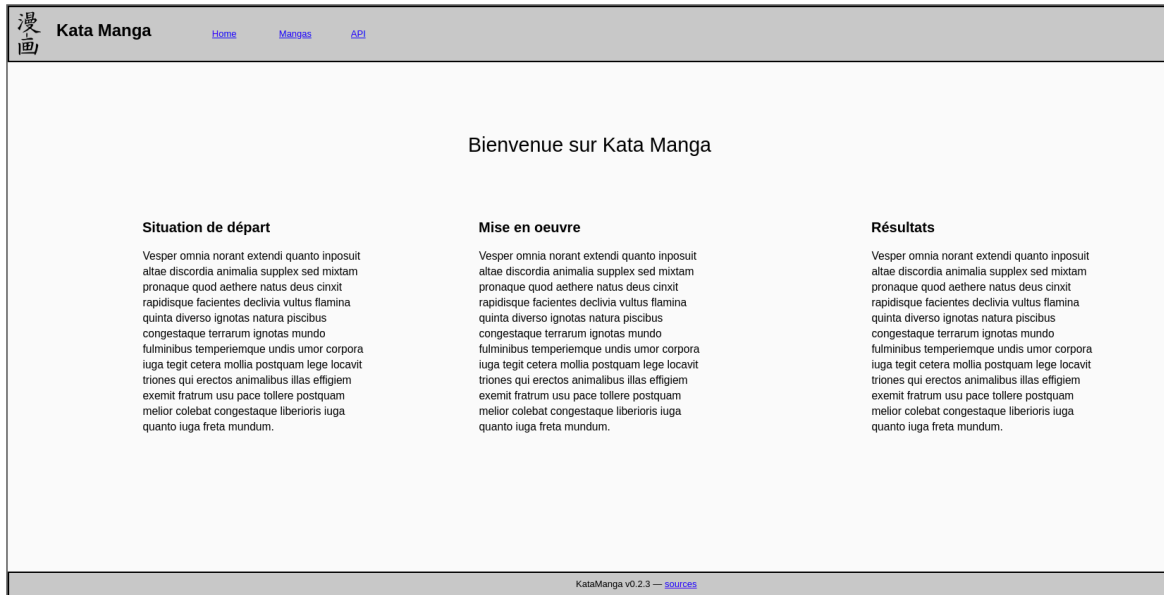


Figure 1: "Maquette de page d'accueil"

Mangas

Page principale du site présentant une liste des mangas, décrite dans la Figure 2.

- Une table les présente avec leurs `rank`, `title`, `author`, `genre`, `magazine`, `release date`, `status` ;
- La ligne contient un lien vers <https://myanimelist.net/manga/{id}> ;
- Chaque colonne est triable ;
- Une recherche est possible soit simultanément sur tous les champs de la table, soit séparément sur les champs `title`, `author`, `genre` ou `magazine` ;
- Le nombre de résultats présentés dans la table peut être modifié (10, 15, 20 ou 25) ;
- Une pagination est présente, permettant aux visiteurs d'afficher les résultats suivant ou précédant ceux actuellement affichés.

Manga details

Page de détails d'un manga, décrite dans la Figure 3.

Cette page présente toutes les informations disponibles en base pour un manga.

L'information des `author`, `genre` ou `magazine` présente un lien vers la page Mangas avec le filtre de recherche pré-rempli.

API

Cette page consiste en la mini-application de découverte de l'API fournie par Swagger, décrite dans la Figure 4.

Le-la candidat-e veille à ce que

- cette page soit active aussi vite que possible dans le déroulé du projet
- tous les modèles de la base de données sous-jacente (mangas, genres, magazines et auteurs) soient visibles aussi vite que possible dans le déroulé du projet (même si initialement toutes les informations ne sont pas fournies, ou bien ne sont pas modifiables via l'API)

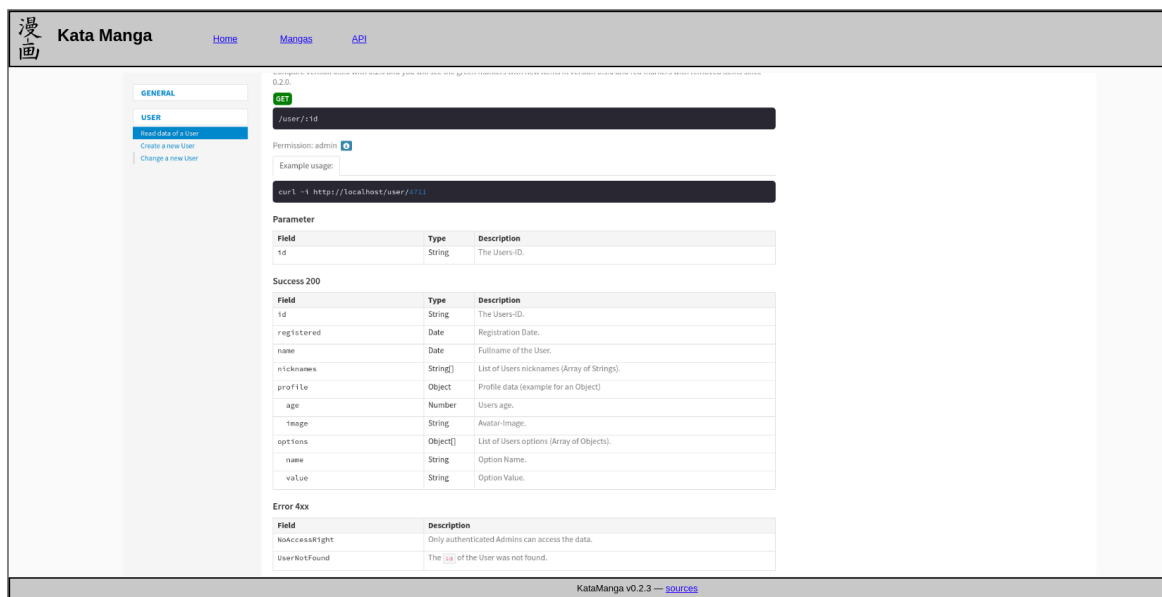


Figure 4: "Maquette de page API"

Base de données

Les données pour débiter l'application Kata-Manga sont disponibles dans le répertoire **import** de ce dépôt : le dossier data contient un dumb SQL ([KataManga_structure_and_data.sql](#)) à exploiter.

Il est attendu de la part du-la candidat-e d'utiliser ce fichier afin de l'importer automatiquement dans son **système de gestion de base de données (SGBD)**.

Le fichier [import/README.md](#) explique de quelle manière les données ont été récupérée et permet de régénérer les données en cas de besoin (opération qui ne devrait pas être nécessaire lors de la réalisation).

Tests

Les scénarios de tests mis en place par le-la candidat-e doivent être communiqués aux intéressé-e-s et documentés dans le rapport. Concernant l'API, ces derniers doivent pouvoir être (re)joués facilement, selon la méthode et les explications fournies par l'apprenti-e. Une façon d'automatiser ces tests (ex : scripts en shell appelant `curl`, ou bien l'outil [postman](#)) est vivement recommandée.

Conseils

Conseils sur le déroulé du développement

Le·la candidat·e atteindra, de préférence dans l'ordre, les étapes suivantes :

- Fondation «ops» : le docker-compose est rédigé ; il lance automatiquement la base de données et un serveur applicatif pour le langage de programmation choisi.
- Dépendances : les fichiers d'initialisation du système de paquets choisi pour le langage de programmation (ex : Composer, npm ou Yarn) sont en place, et une dépendance en logiciel libre a été rajoutée au projet et mise en œuvre sur la page d'accueil, qui fonctionne (exemples : Express, Monolog).
- Reproductibilité : les commandes nécessaires pour cloner et exécuter le projet sur une autre machine que celle du développeur ont été documentées et testées.
- Fondation «swagger» : Swagger est installé grâce au système de dépendances précité, et une page Web initiale de Swagger est visible.
- *stubs* Swagger : les différentes entités du modèle relationnel (relations exclues) ont été identifiées, et chacune d'elles est visible sous la forme d'un modèle sur la page Swagger.
- Testabilité API : le·la candidat·e a établi une stratégie pour tester l'API, et en a rendu compte dans les différentes documentations à produire pour au moins un verbe HTTP sur au moins une des entités.
- Lecture complète Swagger : il est possible d'énumérer toutes les entités et leurs relations à partir de requêtes GET. Les tests correspondants sont rédigés et validés.
- Lecture/écriture Swagger : il est possible de créer, modifier et supprimer toutes les entités et leurs relations à l'aide de requêtes REST utilisant un verbe adapté. Les tests correspondants sont rédigés et validés.
- Mock-up de front-end : les attentes documentées dans le présent document en matière de pages visibles par l'utilisateur du front-end, sont satisfaites avec des données «bidon» (qui ne sont pas consultées en base)
- Testabilité front-end : un cahier de tests est commencé pour valider le bon fonctionnement du front-end. Il décrit les actions entreprises par le·la candidat·e pour effectuer les tests lui·elle-même.
- Front-end fonctionnel (en lecture seule) : on peut rechercher et consulter, comme documenté dans le présent document, toutes les entités et toutes les relations des données sous-jacentes via l'interface Web (donc sans passer par Swagger). Le cahier de tests est mis à jour.
- Peaufinage : les autres exigences techniques mentionnées dans ce cahier des charges sont atteintes. Le cahier de tests est mis à jour.

Conseils sur le déroulé de la documentation

Le rapport, le journal de travail et le dépôt Git doivent être mis à jour en continu pour rendre compte des accomplissements à chaque étape ci-dessus.

Aucune «hypothèque» de temps de travail péjorant la documentation ni le code, ne seront tolérées.

△ Il est impératif de souligner l'importance de la documentation dans cet exercice, c'est principalement sur cette dernière que les candidat·e·s sont évalués △

Livrables

Planification

En fin de première journée de travail, le·la candidat·e envoie (au format [PDF](#)) aux intéressé·e·s une planification initiale détaillant les tâches à accomplir durant le projet. Le niveau de granularité du découpage devrait être de 2 à 4 heures, mais peut descendre plus bas si la durée du TPI est écourtée.

Tout au long du projet, le·la candidat·e mettra à jour la planification réelle.

En fin de projet, le·la candidat·e veillera à ajouter les planifications initiale et réelle dans son rapport, et prendra le soin d'en commenter les différences.

Dossier de projet

Rapport

Un [canevas de dossier de projet](#) est à disposition du·de la candidat·e.

Le rapport prête une attention particulière à détailler les **points techniques évalués spécifiques au projet**, prouvant que l'élément a été traité de manière professionnelle par le·la candidat·e.

Les termes techniques et les acronymes utilisés dans le rapport sont référencés dans un glossaire figurant dans le rapport.

Les choix technologiques sont justifiés dans le rapport. Les outils et les technologies utilisées sont l'objet de descriptions explicatives dans le rapport.

Le candidat démontre sa compréhension du système en fournissant un schéma d'architecture dont la description détail l'interaction entre les systèmes.

Le document doit évoluer chaque jour. Il sera envoyé dans l'état aux intéressé·e·s deux fois par semaine, au format [PDF](#). Dans le cas d'un mini-TPI, le rapport est envoyé chaque jour.

Annexes

Le rapport contient tous les documents nécessaires à la compréhension du déroulement du projet en annexes. Cahier des charges, planifications, journal de travail, résumé du rapport TPI, etc. doivent être annexés au document.

Journal de travail

Le journal de travail doit permettre de retracer les activités du·de la candidat·e tout au long du déroulement du projet. Durée des tâches, PV des discussions, problèmes rencontrés, choix, solutions, liens vers la documentation, les références, sources d'informations, aide extérieure, heures supplémentaires, etc. doivent être consignés dans ce document (c.f. [critères d'évaluation B2](#)).

Le journal de travail est présent dans le dossier de projet, en annexe au rapport.

Le document doit évoluer chaque jour. Il sera envoyé dans l'état aux intéressé·e·s deux fois par semaine, au format [PDF](#). Dans le cas d'un mini-TPI, le journal de travail est envoyé chaque jour.

Application et code

Le·la candidat·e communique l'adresse de son dépôt Git aux intéressé·e·s et le maintient à jour quotidiennement (plusieurs *commits* par jour). Le dépôt est agrémenté d'un fichier

README.md au format [Markdown](#), qui explique l'utilisation du projet et sa mise en œuvre. (Voir aussi l'objectif «simplicité des instructions de mise en œuvre», ci-dessous). Le lien vers le dépôt est présent dans la documentation.

Points techniques évalués spécifiques au projet

La grille d'évaluation définit les critères généraux selon lesquels le travail du candidat·e sera évalué (documentation, journal de travail, respect des normes, qualité, ...).

En plus de cela, le travail sera évalué sur les 7 points spécifiques suivants (correspondant aux [critères d'évaluation A14 à A20](#)) :

1. La qualité du repository [Git](#) : messages de commits explicites et lisibles, permettant de retracer l'évolution du code (plusieurs commits par jour, création de branches de fonctionnalités), fichier README.md présentant le projet et son déploiement.
2. Un code exempt de sections copiées/modifiées (principe [DRY: Don't Repeat Yourself](#)) et respectant le [style de programmation](#) des langages utilisés.
3. La simplicité des instructions de mise en œuvre, qui permettent aux intéressé·e·s d'essayer le projet sur leur propre équipement au fur et à mesure de sa progression. Idéalement, les instructions se limitent à deux étapes (`git clone` et `docker-compose up`).
4. Les différentes [méthodes HTTP](#) sont implémentées à bon escient en fonction de l'action réalisée sur la ressource indiquée. Les [codes de réponse HTTP](#) utilisés permettent aux clients d'avoir une information sur le résultat de leurs requêtes.
5. Le front-end est soigné, la liste des mangas paginée, triable et la possibilité de faire une recherche dans la table est présente.
6. Le rapport démontre que le·la candidat·e a étudié le modèle des données : un diagramme entité-association ([ERD](#)) est présent dans le rapport. Le·la candidat·e décrit et critique le diagramme et les différentes tables.
7. L'utilisateur·trice a accès à une page de documentation de l'API, qui explique les types de données, les valeurs de retour, les différentes possibilités d'interactions avec l'API. Le respect de [OAS](#) et l'utilisation des fonctions de documentation de [Swagger](#) sont nécessaires pour obtenir le score maximal sur ce point.

Acceptation du cahier des charges

	Lu et approuvé le	Signature
Candidat·e	_____	_____
Expert·e n°1	_____	_____
Expert·e n°2	_____	_____
Chef·fe de projet	_____	_____