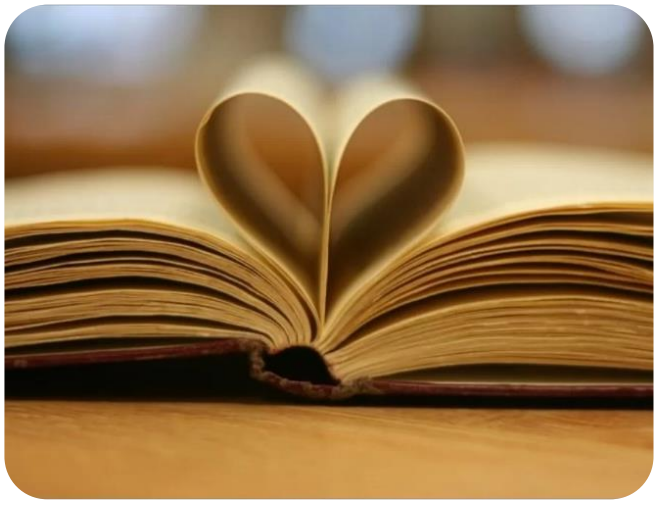
P_Web295 – Passion lecture



Fabre Antoine, Segalen Alban, Thode Mateo– MID2B
ETML - Vennes
32 périodes
Charmier Gregory

Table des matières

1 SPÉCIFICATIONS

1.1 Titre

Passion lecture

1.2 Description

Réaliser le backend d'une application permettant de partager sa passion pour la lecture.

1.3 Matériel et logiciels à disposition

- Microsoft Windows
- Vscode
- Serveur local
- Navigateurs Web
- Accès à Internet

1.4 Prérequis

Modules 106, 162, 164, 231, 293, 319, 320, 322 et ~~426~~

1.5 Cahier des charges

Le cahier des charges a été lu, compris et signé par tous les membres du groupe. Il est disponible ici.

1.6 Les points suivants seront évalués

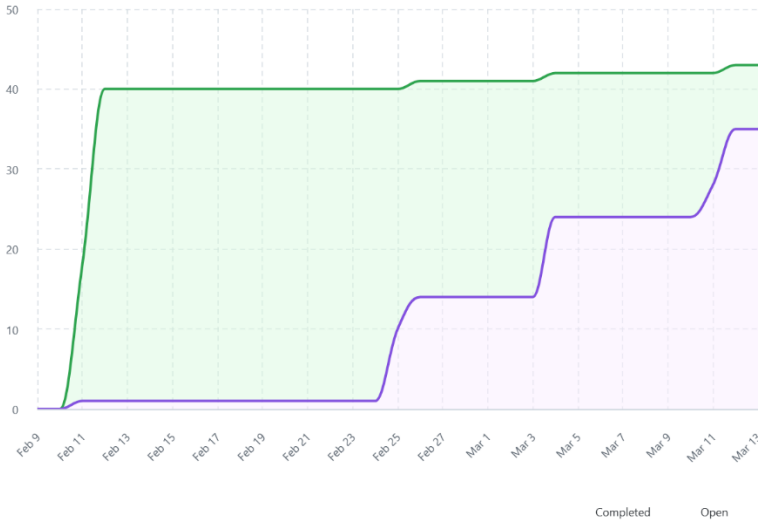
- Le rapport
- Le code et les commentaires
- Les documentations de mise en œuvre et d'utilisation
- L'utilisation de GitHub
- L'utilisation de GitHub Projects

1.7 Validation et conditions de réussite

- Compréhension du travail
- Possibilité de transmettre le travail à une personne extérieure pour le terminer, le corriger ou le compléter
- Etat de fonctionnement du produit livré

2 PLANIFICATION INITIALE

La planification a été faite avec GitHub project. Nous avons commencé par mettre toutes les tâches connues dans le Backlog (colonne de gauche) avant de commencer le développement. Il y a ensuite eu des tâches que l'on a ajoutées, mais celles-ci sont peu nombreuses.

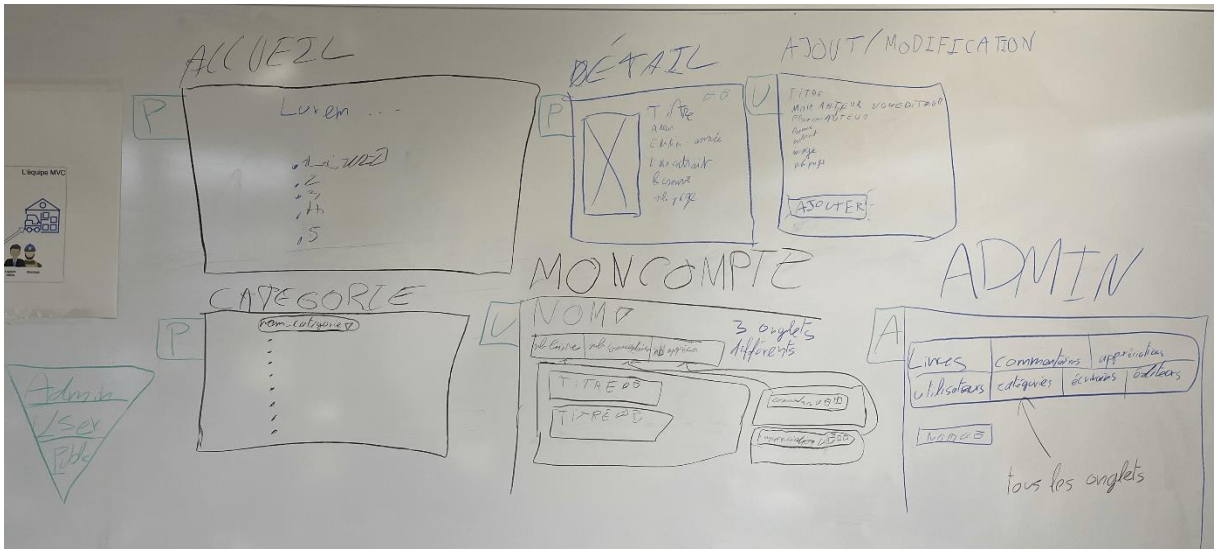


En vert les tâches planifiées et en violet celles qui sont terminées.

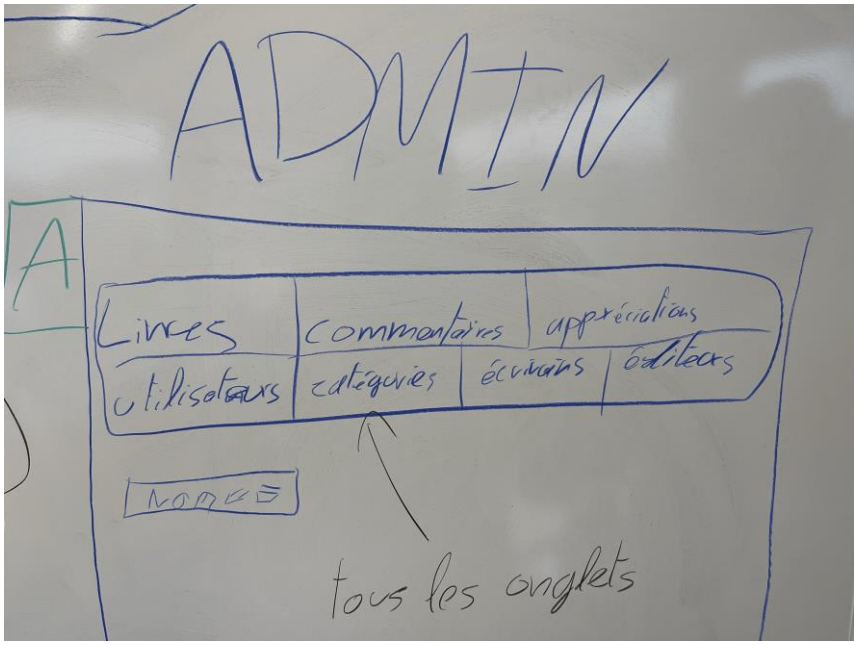
3 ANALYSE

3.1 Document d'analyse et conception

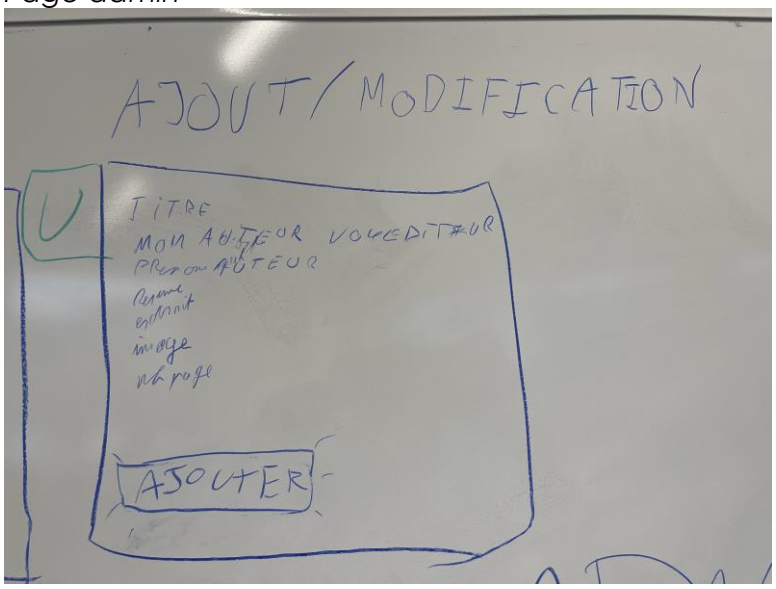
Au début du projet, nous avons fait la maquette des pages afin de savoir quelles informations devront y être affichées. Ceci afin de pouvoir faire le tableau des routes en conséquence. Nous voyons également quelles pages sont visibles par les admins, les utilisateurs connectés et celles qui sont publiques (schéma en bas à gauche de l'image de toute la maquette).
Voici les photos de la maquette (disponibles dans le dossier *images-maquette*) :



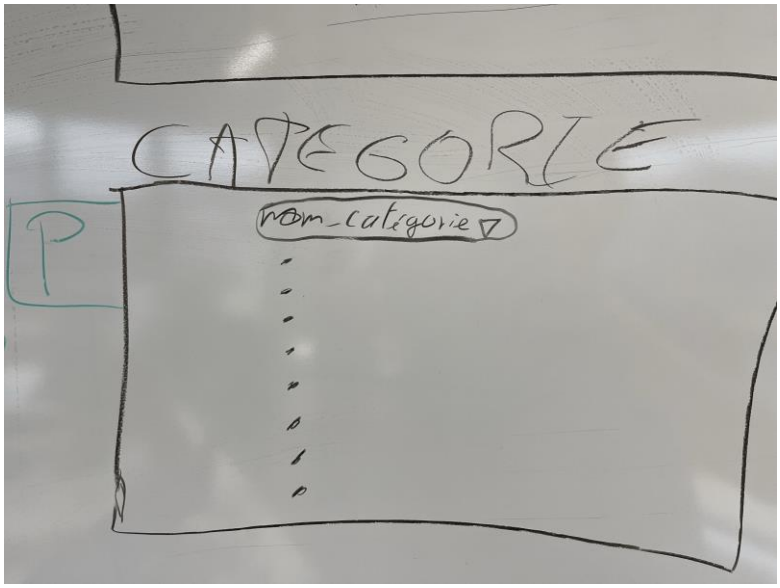
Toute la maquette



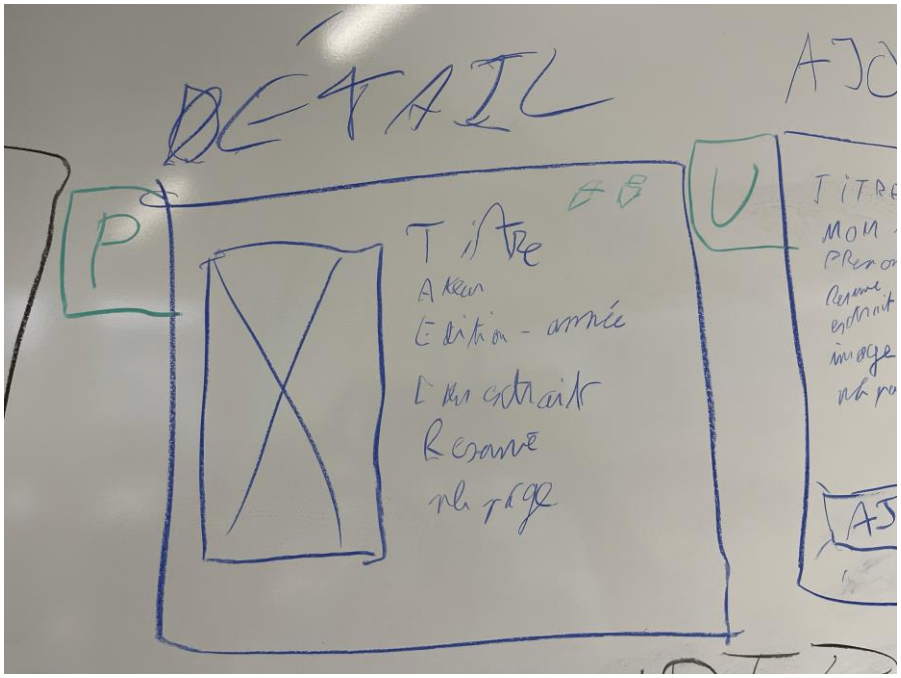
Page admin



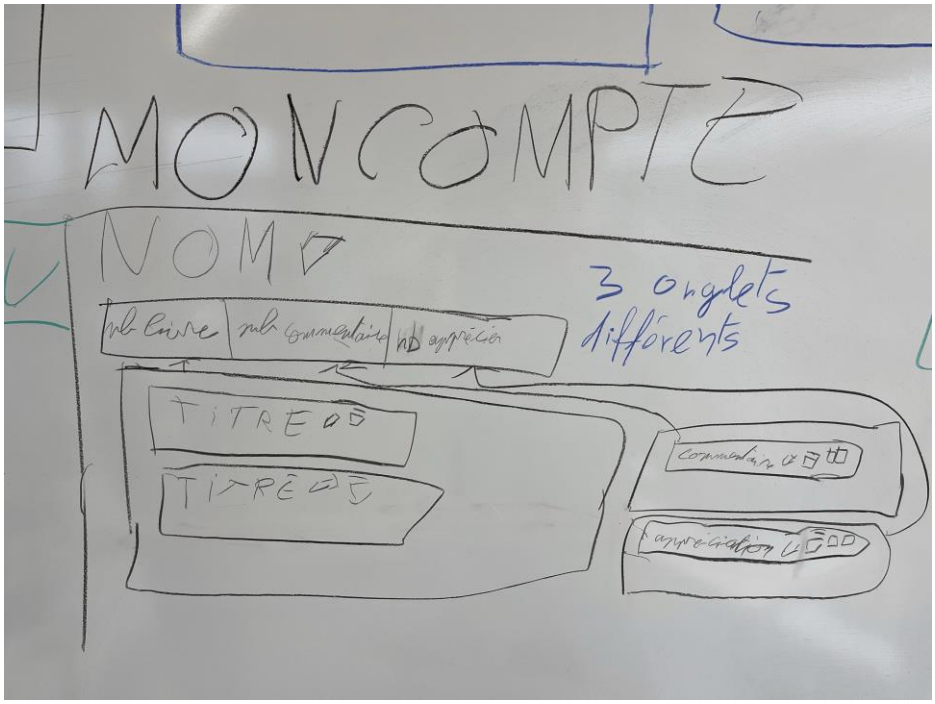
Page d'ajout des livres



Page de recherche par catégorie



Page de détail d'un livre



Page de présentation du compte de l'utilisateur

Il y a trois onglets sur cette page, une qui montre tous les livres que l'utilisateur a publiés, une pour tous ses commentaires et une pour toutes ses appréciations.

4 RÉALISATION

4.1 Mode de fonctionnement

Pour le développement de ce projet, nous avons utilisé la méthodologie agile ainsi que des éléments de la méthode « Scrum ». Nous prônons la communication plutôt que les processus en ayant beaucoup d'interactions pendant le travail. Nous n'hésitons pas à nous demander de l'aide et à discuter de nos doutes quant à la réalisation.
Chaque séquence de travail est débutée par un « meeting » de 5-10 minutes (daily scrum) dans lequel nous résumons ce qui s'est passé précédemment et nous répartissons le travail pour la session. Quand la séquence est longue, nous prenons un second moment pour une réunion du même type que la première.
Notre atmosphère de travail est décontractée et amicale, afin de rendre les moments de travail agréables et pour conserver notre motivation à avancer ensemble.

Nous utilisons GitHub pour la gestion des versions de l'application et GitHub Project pour la planification et répartition des tâches.

4.2 Conventions

Notre équipe de développement, utilisant les méthodologies agiles, est dynamique. Cela se traduit par les éléments mentionnés au point précédent et ceux qui suivent.

4.2.1 Commits GitHub

Afin de rendre la lecture et la compréhension des commits GitHub, nous nous sommes mis d'accord pour que le message de chaque commit contienne au moins un emoji. Ceci a également pour but de mettre en valeur notre dynamisme tout en restant sérieux et professionnel.

L'outil d'aide utilisé est le site gitmoji.dev

4.3 Architecture du répertoire

Pour mener à bien ce projet notre équipe de développement a dû penser à une architecture de notre répertoire GitHub.
Donc voici la liste de nos dossiers :

- Application

Dans le dossier application il y a le dossier « node_modules » qui contient toutes les dépendances utilisées dans le code.
Ensuite il y a le dossier « src » qui contient tout le code de notre projet avec les dossiers auth, db, models, routes et tests.

- Dev

Dans le dossier Dev il y a tous les fichiers qui servent au bien du développement avec des fichiers batch pour lancer le serveur plus rapidement ou alors le fichier JSON qui contient les tests insomnia.

- Docker-DB

Dans ce dossier comme le nom le dit, il y a les fichiers de configuration de docker pour la base de données, il y a aussi le script SQL pour créer la base de données.

- Images-maquette

Ce dossier répertorie toutes les images de la maquette faites au début du projet.

- Liens

Dans ce dossier comme le nom l'indique il y a tous les liens utiles pour le projet,
Comme le lien du repo GitHub et du projet GitHub.

- Livres

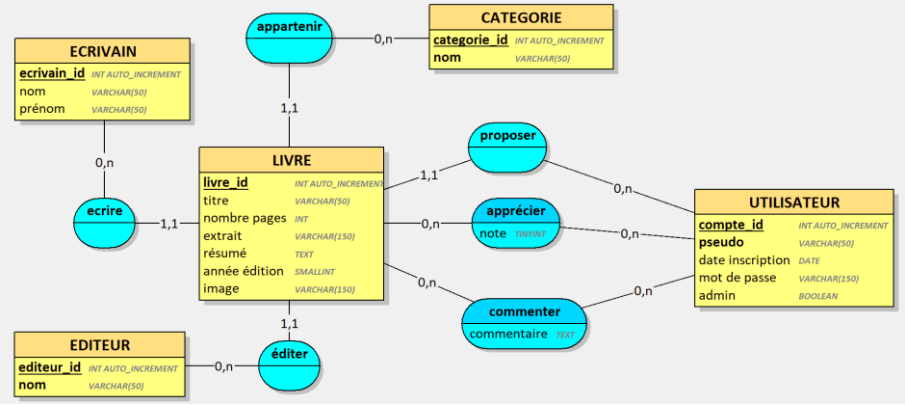
Dans le dossier livres il y a le cahier des charges et le tableau Excel de la feuille d'évaluation.

- Rapport

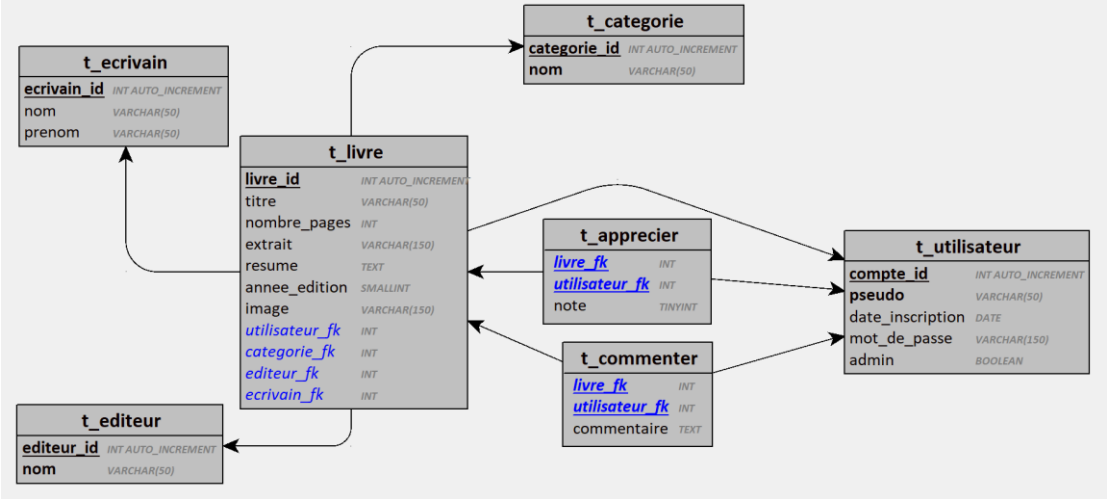
Et pour finir dans le dossier rapport il y a le document Word qui contient le rapport.

4.4 Base de données

Nous avons créé le schéma de la base de données avec le logiciel Looping.
Voici le MCD :



Et le MLD :



Nous avons utilisé le script SQL de création de la base de données au début de la phase de développement mais nous avons arrêté de l'utiliser quand nous avons créé les modèles. Car Sequelize crée les tables automatiquement.

4.5 Authentification et utilisateurs

4.5.1 Authentification

Dans une API REST il est nécessaire d'utiliser un moyen d'authentification pour éviter que tout le monde n'y ait accès. Donc pour gérer les accès à notre API REST, un système de jeton JWT est utilisé. Pour rentrer dans chaque route (mise à part la route de connexion).

```
livreRouter.get("/", auth, async (req, res) => {
```

Figure 1 middleware "auth"

Un middleware est utilisé pour vérifier que le jeton reçu est bien valide. Donc en premier le middleware récupère le jeton.

```
const auth = (req, res, next) => {
  const authorizationHeader = req.headers.authorization;
```

Si aucun jeton n'a été envoyé alors, un message d'erreur est renvoyé à l'utilisateur.

```
if (!authorizationHeader) {
  const message = `vous n'avez pas fourni de jeton d'authentification`;
  return res.status(401).json({ message });
}
```

Donc si un jeton est intercepté par le middleware alors grâce à la librairie « jsonwebtoken » on peut vérifier que le jeton est valide.

```
else {
  const token = authorizationHeader.split(" ")[1];
  const decodedToken = jwt.verify(
    token,
    privateKey,
```

Si le jeton n'est pas valide alors, un message qui dit que l'utilisateur n'est pas autorisé à accéder à cette ressource.

```
(error, decodedToken) => {
  if (error) {
    const message = `l'utilisateur n'est pas autorisé a cette ressource`;
    return res.status(401).json({ message, data: error });
  }
```

Et si l'identifiant de l'utilisateur est absent ou alors qu'il ne correspond pas à l'identifiant préciser dans le payload du jeton JWT, alors un message d'erreur est envoyé à l'utilisateur.

```
const userId = decodedToken.userId;
if (req.body.utilisateurId && req.body.utilisateurId !== userId) {
  const message = `l'identifiant de l'utilisateur est invalide`;
  res.status(401).json({ message });
}
```

Et si le jeton est valide alors le middleware autorise l'accès à la route.

```
else {
  next();
}
```

4.5.2 Gestion des rôles

Pour la gestion des rôles, il y a deux types de rôle : les non-admins ainsi que les admins.

utilisateur_id	pseudo	mot_de_passe	admin	created
1	AgileMaster	$2b$10$jO.uLyrvonS0JKPbHgDjsO8hz0jbq.eoa5MR4bvnfERVS5lJv49Oq	0	2025-03-12 10:11:57
2	ScrumSensei	$2b$10$UGFioZgNqqS1KDWxKnql5uCEDZFLheGgecmuuuKkVjbiyIpBjlpMa	1	2025-03-12 10:11:57
3	ScrumLover22	$2b$10$4hBzqaEwmAof37h/QmEVLON5b5WovOMul68F3Ocoj0mTzFl9db4hy	0	2025-03-12 10:11:57
4	DevOpsWizard	$2b$10$2M/wfW5PuuLKUgHNM3Kgaeh4kr5WSyPRoaqdn7wnMkqYwjO.HOz6i	1	2025-03-12 10:11:57
5	BugHunter99	$2b$10$J.vBl.QL7CIuOaZcRxj.AeoOrGjN.yUBYaP1ZrbcGG4ovIK/NrTYe	1	2025-03-12 10:11:57
6	UXGuru	$2b$10$jbGGLkQnkOzt4y9gv57dkexyNNhkGxNS4V7U2WZr8uGZc.JjBtkdu	0	2025-03-12 10:11:57
7	TechExplorer	$2b$10$IEXfbuInJDk.bkppwjSSHesU5Bi8GrFRXIOe94I62yZ56olRCRcY6	0	2025-03-12 10:11:57
8	CyberKnight	$2b$10$r4PNGAHfbT42jXZT88DICeQ9UgmAlxdweioyXGEZ0WNYX0ePDnLjm	0	2025-03-12 10:11:58
9	CodeNinja	$2b$10$lyV2yLVxQumJx54/fUPvruLRN4iCtgKk1qvcGRH9qiQf2qaza.V02	0	2025-03-12 10:11:57
10	AIEnthusiast	$2b$10$lDaL8vnfsDj29ntb4652y.F5YZ39FVhwGiihluuPkIIqW29unJNzW	1	2025-03-12 10:11:58
11	CloudNomad	$2b$10$TBSUz88qOxkaFUVM29AcnuN1S0OaPucU2Mp0FxXCyTAFEhMpPGhpC	0	2025-03-12 10:11:58
12	QuantumCoder	$2b$10$MGScxie8K/KYvWfUHmufjeAD5l5vNEZHw3Ghh1BcHeREkCPDKGhDa	0	2025-03-12 10:11:58

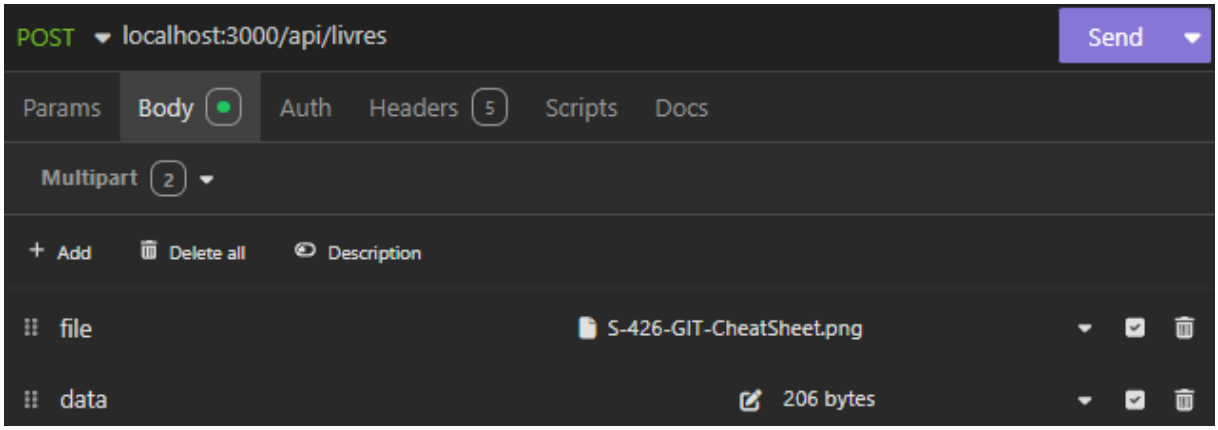
Donc pour plus de sécurité, un utilisateur peut devenir admin seulement si un admin change le champ dans la base de données.

4.6 Gestion des images

L'outil utilisé pour gérer les images est Multer.

```
//Stockage des images
import multer from "multer";
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "uploads/");
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + "-" + file.originalname);
  },
});
const upload = multer({ storage });
```

Ce code crée l'emplacement de stockage des fichiers envoyés à notre API. Les fichiers sont sauvegardés dans /uploads et sont nommés [timestamp]-[nom_original].



Comme on envoie une image, le corps de la requête est de type formulaire multipart. Il faut passer l'image dans un champ de type fichier avec comme nom « file » et le json du livre dans un champ de texte multiligne avec comme nom « data ».

```
//Ajout d'un livre
livreRouter.post("/", auth, upload.single("file"), (req, res) => {
  let object;
  try {
    //Récupération du json de la requête
    object = JSON.parse(req.body.data);
  } catch {
    const message =

"Le livre n'a pas pu être créé. Merci de réessayer dans quelques instants. Il faut
envoyer les données dans un formulaire multipart, et le json dans un champ data"
;
    return res.status(400).json({ message, data: error });
  }
  //Création de l'objet livre avec l'image
  const livre = { ...object, image: req.file.filename };
  Livre.create(livre)
```

Voici à quoi ressemble une route qui reçoit une image. Le middleware upload.single("file") permet de sauvegarder le fichier et d'accéder au fichier depuis la route grâce à req.file.
Pour récupérer le json du livre, on transforme le contenu de « data » en json avec JSON.parse. On crée ensuite un livre avec le contenu de « data » et le nom de l'image.

```
//Pour récupérer les images
app.use("/uploads", express.static("uploads"));
```

Cette ligne fait en sorte qu'il soit possible d'accéder au fichier de manière statique depuis une url par exemple : http://localhost:3000/uploads/1741765583809-S-426-GIT-CheatSheet.png.
Pour afficher l'image d'un livre, il suffira d'utiliser une balise <img> avec le lien de l'image.

4.7 Liste des routes

Ces JSON sont utilisés dans le tableau afin de faciliter la lecture.

JSON *livre* :

```
{"livre_id", "titre", "nombre_pages", "extrait", "annee_edition", "image",
"ecrivain_nom", "ecrivain_prenom", "editeur_nom", "categorie_nom",
"moyenne_appreciations", "commentaires":[{"commentaire"}]}
```

JSON *livre_preview* :

```
{"livre_id", "titre", "annee_edition", "ecrivain_nom", "ecrivain_prenom",
"editeur_nom", "categorie_nom", "moyenne_appreciations"}
```

JSON *user* :

```
{"pseudo", "date_inscription", "admin", "nombre_commentaires",
"commentaires":[{"commentaire"}], "nombre_appreciations",
"appréciations":[{"appréciation"}], "nombre_livres",
"livres":[livre_preview]}
```

Verbe http	URI	JSON entrée	JSON sortie	Description
/api/…				
GET	/api/livres	NON	[*livre_preview*, *livre_preview*, …]	Récupère tous les livres qui correspondent à la recherche (paramètres en GET) (informations non exhaustives)
GET	/api/livres?search=&cat =	NON	JSON livres	Récupère tous les livres qui correspondent au terme de la recherche et qui correspondent à la catégorie précisée dans l'URI
POST	/api/livres	JSON *livre*	NON	Ajoute un livre
PUT	/api/livres/:id	JSON *livre*	NON	Modifie un livre
DELETE	/api/livres/:id	NON	NON	Supprime un livre
GET	/api/livres/:id	NON	JSON *livre*	Détails d'un livre (informations exhaustives)
POST	/api/inscription	`{"pseudo", "mot_de_passe", "confirmation_mot_de_passe" }`	Token	Création du compte et connexion au compte créé
POST	/api/connexion	`{"pseudo", "mot_de_passe"}`	Token	Connexion au compte
PUT	/api/utilisateurs/:id	`{"pseudo", "mot_de_passe", "admin"}`	NON	Modification d'un utilisateur (⚠modification du champ "admin")
DELETE	/api/utilisateurs/:id	NON	NON	Suppression d'un utilisateur
GET	/api/utilisateurs/:id	NON	JSON *user*	Récupère les informations du profil d'un utilisateur
POST	/api/categories	`{"nom"}`	NON	Ajoute une catégorie

PUT	/api/categories/:id	{"nom"}	NON	Modifie le nom d'une catégorie
GET	/api/categories	NON	[{"id","nom"}]	Récupère les catégories avec leurs informations
DELETE	/api/categories/:id	NON	NON	Supprime une catégorie
POST	/api/ecrivains	{"nom", "prenom"}	NON	Ajoute un écrivain
PUT	/api/ecrivains/:id	{"nom", "prenom"}	NON	Modifie un écrivain
GET	/api/ecrivains	NON	{"id", "nom", "prenom"}	Récupère tous les écrivains avec leurs informations
DELETE	/api/ecrivains/:id	NON	NON	Supprime un écrivain
POST	/api/editeur	{"nom"}	NON	Ajoute un éditeur
PUT	/api/editeur/:id	{"nom"}	NON	Modifie un éditeur
GET	/api/editeur	NON	{"id", "nom"}	Récupère tous les éditeurs avec leurs informations
DELETE	/api/editeur/:id	NON	NON	Supprime un editeur
GET	/api/commentaires	NON	{"livre_fk", "utilisateur_fk", "commentaire"}	Récupère tous les commentaires
POST	/api/commentaires	{"livre_fk", "utilisateur_fk", "commentaire"}	NON	Publie un commentaire
PUT	*/api/commentaires/*	*{"livre_fk", "utilisateur_fk", "commentaire"}*	*NON*	*Modifie un commentaire*
DELETE	*/api/commentaires/*	*{"livre_fk", "utilisateur_fk"}*	*NON*	*Supprime un commentaire*
GET	*/api/appreciations*	*NON*	*{"livre_fk", "utilisateur_fk", "note"}*	*Récupère toutes les appréciations*
DELETE	*/api/appreciations/:id*	*NON*	*NON*	*Supprime une appréciation*

POST	/api/appreciations	{"livre_fk", "utilisateur_fk", "note"}	NON	Publie une appréciation
PUT	/api/appreciations	{"livre_fk", "utilisateur_fk", "note"}	NON	Supprime une appréciation

En plus clair sont les routes que nous ne devions pas développer.

4.8 Documentation de l'API

Nous avons mis en place la documentation Swagger et fonctionnelle pour la route GET /livres.
La documentation de l'API est accessible à cette adresse (quand le serveur est lancé) : http://localhost:3000/api-docs

4.9 Eco-conception Web

Dans notre application, nous avons, dès le début de la conception, fait attention à ne pas utiliser plus de ressources que nécessaire. Dans la mesure de nos connaissances, le code est propre et sans boucle inutile par exemple.

Un point d'éco-conception dans notre application est que dans la base de données, étant donné que nous n'avons pas d'arborescence dans les images « uploadées » (nous ne stockons pas les photos de profils des utilisateurs par exemple) et stockons uniquement les images des livres, au lieu d'enregistrer le chemin complet de l'image dans la DB, nous enregistrons uniquement son nom.
De cette manière, nous ne stockons pas d'informations inutiles car sinon, tous les enregistrements de cette colonne commenceraient pareil.

Nous n'avons pas de chargement d'images ou d'informations qui ne sont pas utilisés pour le front-end (grâce à notre maquette).
Nous ne faisons pas de rafraîchissement de la page inutile non plus.

4.10 Sécurité

Dans notre application, nous avons veillé à la sécurité à plusieurs niveaux afin de garantir protection des données des utilisateurs ainsi que du serveur.

L'authentification repose sur l'utilisation de jetons JWT (JSON Web Token), qui permet de sécuriser l'accès à certaines routes. Chaque requête nécessitant une authentification est filtrée par un middleware qui vérifie la validité du token et l'autorisation de l'utilisateur. Un jeton expiré ou invalide causera un refus d'accès (erreur 403).
Les détails sont au point authentification.

Nous avons également mis en place une gestion des permissions en différenciant les rôles des différents comptes, s'ils sont admins ou pas. Les actions critiques, comme la modification des données utilisateur ou la

suppression de ressources, sont restreintes aux administrateurs. Ceci empêche un utilisateur standard d'effectuer des opérations non autorisées.

La validation des entrées utilisateur a été intégrée afin de prévenir les injections SQL. L'ORM Sequelize permet d'échapper automatiquement les valeurs insérées dans la base de données. De plus, les mots de passe sont hachés avec bcrypt avant d'être stockés, garantissant qu'ils ne puissent pas être (trop) compromis en cas de fuite.
Bien que notre API REST ne gère pas d'interface utilisateur, nous avons restreint les origines autorisées et nous nous assurons que seules les requêtes légitimes peuvent accéder aux données.

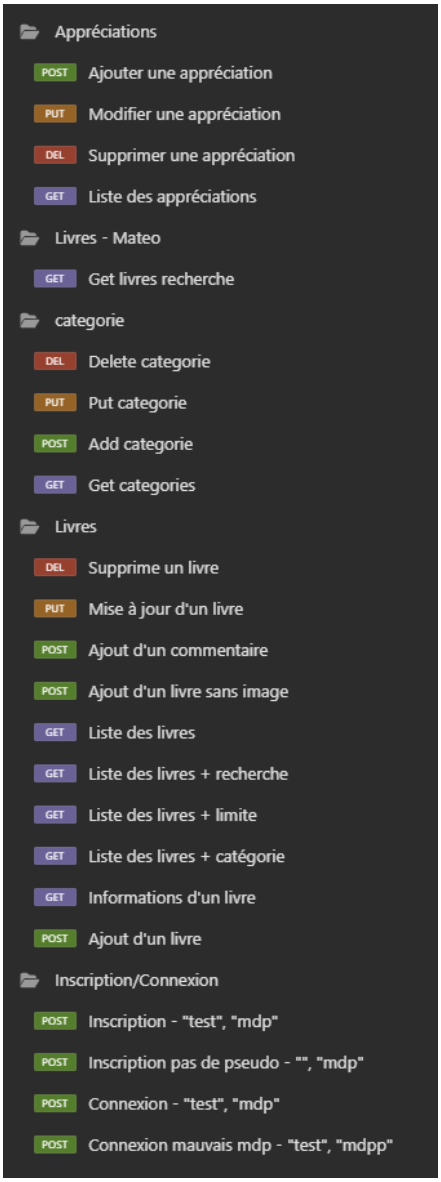
Nous estimons donc que la sécurité implémentée dans notre application est suffisante et adéquate.

5 TESTS

Les tests de l'API ont été réalisés avec Insomnia. Quelques tests unitaires ont été réalisés avec Vitest.

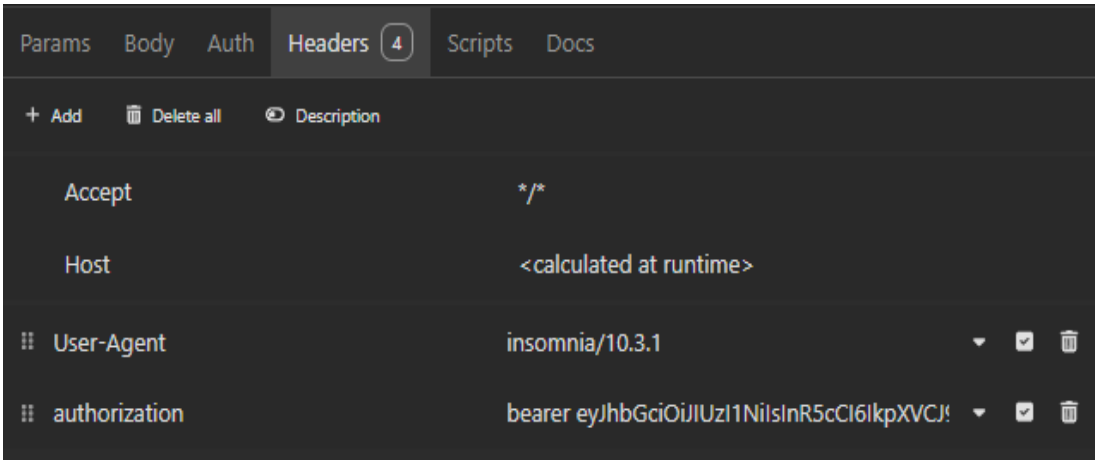
5.1.1 Insomnia

Un export est disponible dans le dossier dev. Les exports précédents sont disponibles dans le dossier dev/archives-insomnia.



La liste des tests effectués

Il faut fournir un token de l'en-tête de la requête. Si le jeton fourni avec l'export n'est plus valide, il faut régénérer un token avec la route de connexion (il se peut qu'il y ait besoin de s'inscrire).



5.1.2 Vitest

Les tests unitaires avec Vitest ont été utilisés pour tester la méthode sucess(). Ces tests sont complémentaires à ceux d'Insomnia.

```
//Tests de helper.mjs
import { success } from "../routes/helper.mjs";
import { expect, test } from "vitest";

test("sucess simple", () => {
  expect(success("Test", { testData: "Data" })).toMatchObject({
    message: "Test",
    data: { testData: "Data" },
  });
});
```

Ce test appelle la méthode `sucess()` avec un message "Test" et un objet { testData: "Data" }. Le test est valide si le résultat est un objet

```
{ message: "Test",
   data: { testData: "Data" }}
```

Pour lancer les tests, il faut exécuter 'npm run test'.

6 CONCLUSION

6.1 Gestion du code avec GitHub

Nous avons utilisé la méthodologie Trunk-Based dans ce projet, ce qui consiste à centraliser l'essentiel du développement sur une branche principale (« main »). Chaque nouvelle fonctionnalité est développée dans une branche de courte durée et intégrée régulièrement sur la branche principale.

6.2 Critique de la planification

La planification nous a semblé adaptée. La majorité des tâches avaient une granularité correcte. Certaines tâches auraient pu être séparées en diverses tâches plus petites, comme par exemple les Seeders.
Il n'y a eu que très peu de tâches qui ont été créées en cours du projet, voir le graphique au point planification initiale.

6.3 Conclusion générale

L'application demandée répond à tous les points demandés dans le cahier des charges (certains points ont été retirés à l'oral).

- Nous validons les données fournies par l'utilisateur
- Nous gérons les statuts http ainsi que les erreurs
- Nous pouvons effectuer les recherches demandées
- Il y a un système d'authentification avec les jetons JWT
- La documentation Swagger est mise en place et fonctionnelle pour la route GET /livres
- Notre API est testée avec Insomnia

Voici les points que nous n'avons pas développés car ils ont été rendus optionnels :

- Des tests automatisés avec Vitest
- Une intégration continue avec GitHub actions

- La « dockerisation » du backend

6.4 Bilan personnel

6.4.1 Mateo

Ce projet m'a beaucoup appris sur la conception et le développement d'un site web dynamique. J'ai aimé travailler dessus en équipe en mettant en pratique les méthodologies agiles. Je suis quand même déçu de ne pas avoir eu le temps de mettre en place les outils d'intégration continue avec GitHub actions.

6.4.2 Alban

J'ai bien aimé ce projet car j'ai beaucoup apprécié le module. J'ai bien aimé le format en petits groupes et je me réjouis de la réalisation du frontend

6.4.3 Antoine

Ce projet m'a beaucoup servi pour comprendre comment fonctionne le travail en équipe avec GitHub (branche, …).

Et la partie backend d'un site web m'a beaucoup parlé vu que je fais du backend chez moi donc j'ai pu approfondir mes connaissances.

7 DIVERS

7.1 Journal de travail

Il n'y a pas de journal de travail en tant que tel dans ce projet mais tout ce qui a été réalisé peut être suivi et compris à l'aide des commentaires des commits GitHub, ici.

7.2 Webographie

- Emojis pour les commits Git : https://gitmoji.dev/
- Blog de questions/réponses pour le code : https://stackoverflow.com/
- La documentation officielle de Sequelize
- ChatGPT pour la relecture, du rapport et des routes dans l'application

8 ANNEXES

Tous les documents utiles à la correction du projet sont dans l'arborescence (GitHub) ainsi qu'éventuellement dans le GitHub Project.