Rétrospective projet & dossier déploiement

Master MIAGE 2 – UPJV

Théo BONTEMPS
Dorian DESCAMPS
Maël RHUIN

Sommaire

I. Contexte

Le département MIAGE de l'UPJV a sollicité la mise en place d'une plateforme DevOps destinée à centraliser et fiabiliser la gestion des projets des étudiants. Notre équipe, composée d'étudiants de Master MIAGE 2, l'a conçue et réalisée en étroite collaboration avec la cliente. Madame Lapujade, cliente et future utilisatrice, nous avait exposé ses besoins et présenté l'existant lors de plusieurs réunions.

Avant notre intervention, les projets des étudiants étaient gérés de manière semi-manuelle :

1. On importait un fichier Excel, rigoureusement structuré, pour générer équipes et dépôts ; toute anomalie dans sa mise en forme provoquait des erreurs de création.
2. Des scripts, lancés ensuite, créaient les dépôts GitHub en s'appuyant sur ces données ; il fallait surveiller en permanence les logs et corriger manuellement toute anomalie.
3. Lorsqu'un étudiant passait en Master 2, son dépôt ne pouvait pas être instancié automatiquement et nécessitait une configuration manuelle.

Ces procédures imposaient une charge de travail considérable, reposaient sur des scripts peu modulaires et ne proposaient ni solution d'archivage intégrée, ni interface de supervision centralisée. L'évaluation des projets se heurtait en outre à l'absence de suivi des contributions, en particulier pour les enseignants non familiarisés à GitHub.

Ce rapport présente une rétrospective sur la réalisation de ce projet, aussi bien d'un point de vue gestion que sur les mesures techniques mises en place pour être au plus proche du besoin.

II. Gestion de projet

A. Méthodologie

Nous avons choisi une méthodologie sous forme de cycle en cascade tout au long du projet.

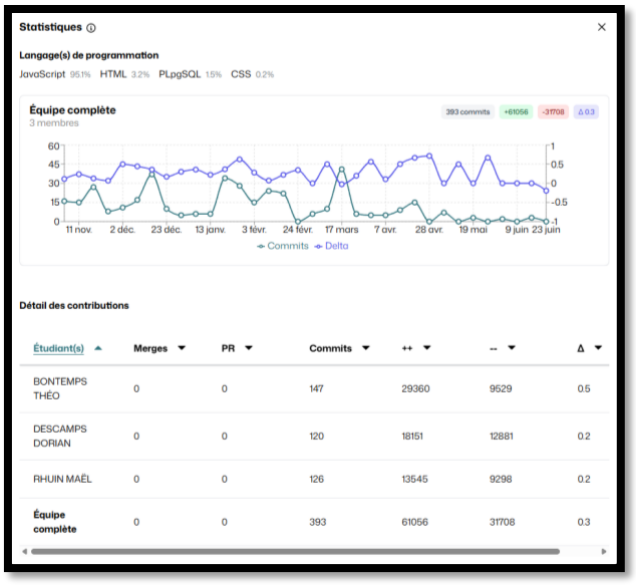
En effet, si une méthodologie agile avait été préférée par tous les membres de l'équipe, les emplois du temps de chacun n'auraient pas permis de réaliser des réunions assez fréquemment avec toutes les parties. Il était donc primordial que les tâches soient clairement définies et réparties avant le début de la phase de développement pour que chacun puisse avancer de son côté sur son temps libre.

D'autre part, cette méthodologie permettait de répondre précisément à la date limite imposée tout en gardant un pilotage simple.

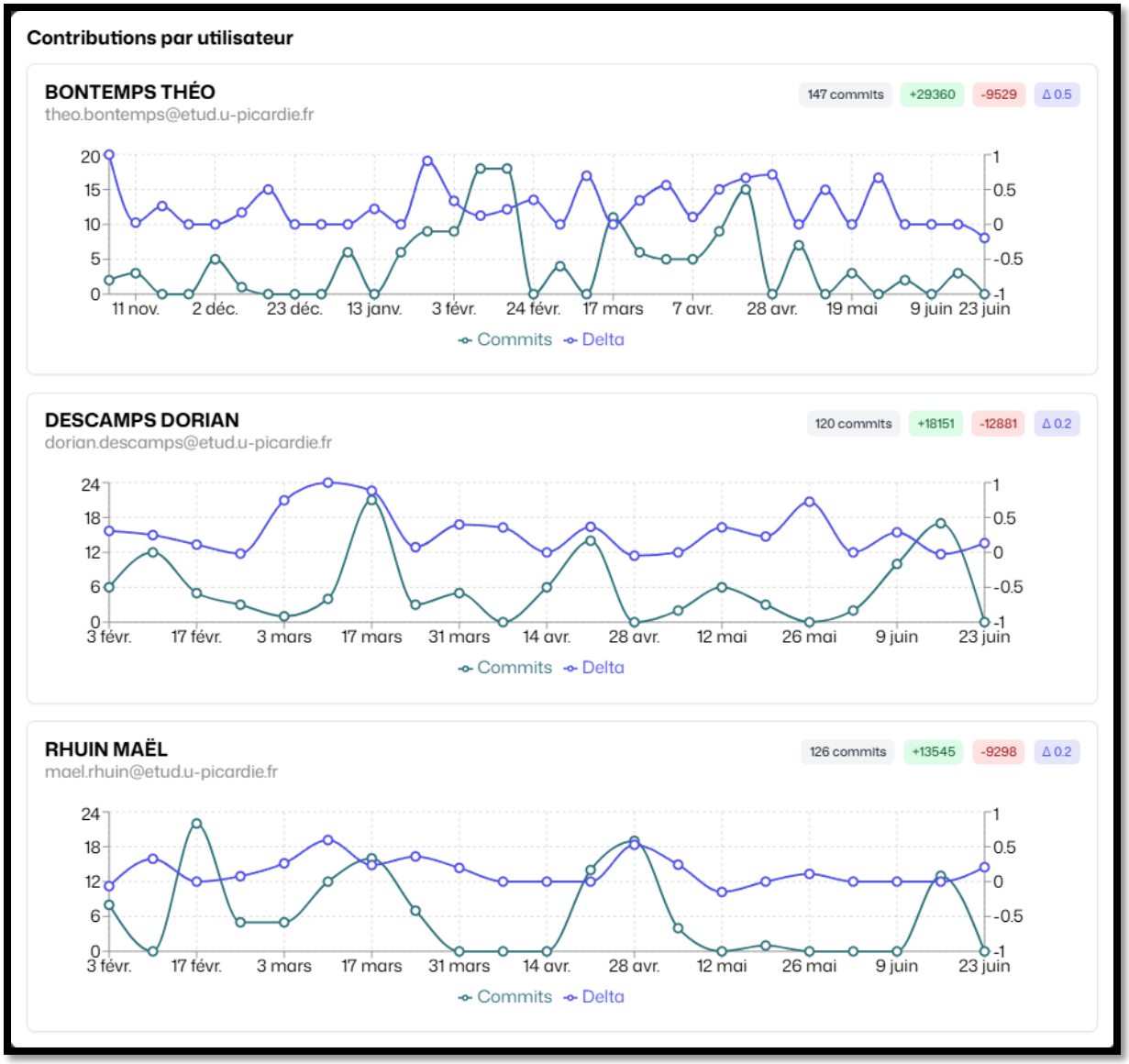
B. Répartition

Pour simplifier à nouveau le pilotage, nous avons fait le choix de séparer complètement le développement du back-end et du front-end sur les ressources. Ainsi, Théo BONTEMPS a travaillé sur le backend et Maël RHUIN et Dorian DESCAMPS ont collaboré sur le frontend. Cette répartition a permis à chacun de réduire le temps perdu dans les synchronisations.

Chacun a travaillé sur le projet régulièrement comme le montrent ces statistiques extraites de notre projet GitTown lui-même. Où la courbe des commits indique les codes validés envoyés et les deltas montrant les ratios de codes ajoutés sur retirés normalisés. On remarque pour nous un développement plutôt stable sur l'année avec quelques pics de développement plus intensifs.



Étudiant(s)	Merges	PR	Commits	++	--	Δ
BONTEMPS THÉO	0	0	147	29360	9529	0.5
DESCAMPS DORIAN	0	0	120	18151	12881	0.2
RHUIN MAËL	0	0	126	13545	9298	0.2
Équipe complète	0	0	393	61056	31708	0.3



C. Difficultés

Le découpage des tâches et une initialisation trop rapide se sont révélés être nos plus gros points noirs.

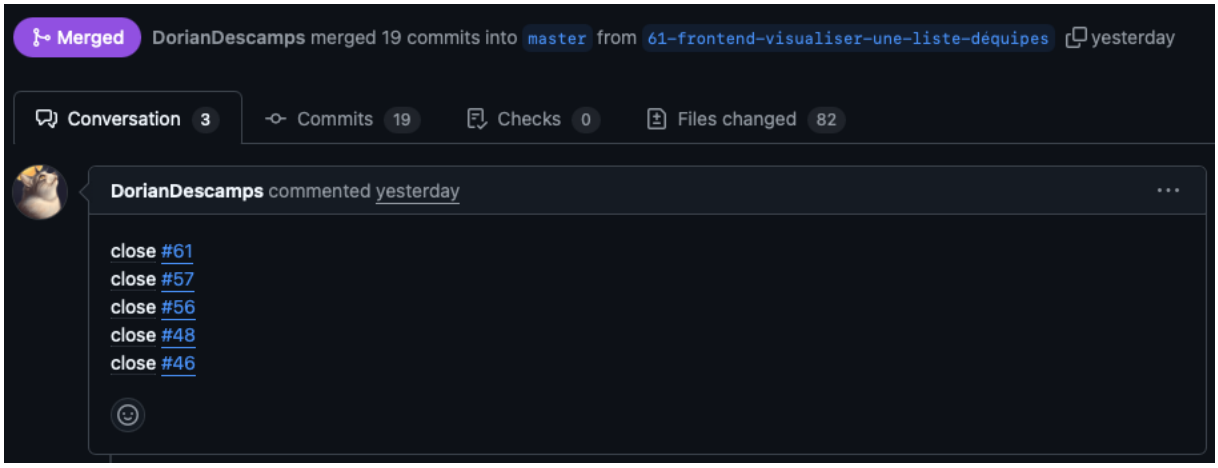
En effet, nous avions défini des tâches à une échelle micro mais des tâches manquaient :

- Création des composants
- Fonctions réutilisées fréquemment
- Configurations
- ...

Nous nous sommes donc retrouvés à utiliser des issues dédiées à certaines fonctionnalités pour en réaliser d'autres parce qu'elles dépendaient de ces composants ou fonctions.

Cette manière de fonctionner a non seulement eu un impact négatif sur la lisibilité des pull requests, mais aussi surtout sur l'avancement de chaque tâche et sur le pilotage global. Les estimations en temps étaient forcément erronées.

Pour faire face, nous avons réagi en regroupant certaines fonctionnalités entre elles comme sur cette pull request et en repriorisant certaines fonctionnalités.



D. Gantt actualisé

L'explication précédente et de nombreux TD annexes ont impliqué un retard imprévu sur plusieurs tâches.

Heureusement, nous avions fait le choix de nous laisser une marge de manœuvre importante sur la fin du projet (plus ou moins un mois). Si cette marge de manœuvre a complètement été consommée, elle a été plus qu'utile puisque le projet est à ce jour terminé.

Sur les captures en annexes, on trouve le Gantt prévisionnel et celui avec les tâches modifiées pour représenter fidèlement la chronologie du projet.

E. Réunions

Nous avons réalisé plusieurs entretiens avec notre client, avec notre tuteur et avec l'intégrateur.

Ces diverses réunions nous ont permis de partager les avancées du projet et ainsi collecter les retours au fil de l'eau.

Une des réunions les plus marquantes est celle avec le client du 13 mai 2025 durant laquelle nous avons collecté pas moins de 17 remarques ; preuve que ces réunions ont une importance majeure afin de produire une application qui soit la plus utile pour le client.

La réunion qui nous a permis de réaliser une première mise en intégration s'est déroulée le 27 du même mois. Celle-ci s'est déroulée sans problème. En effet, nous avions préalablement préparé un serveur de recette qui simulait exactement la même configuration que celle utilisée sur le serveur du Master MIAGE. Nous avions donc pu corriger tous les problèmes qui survenaient, à l'avance.

III. Application

L'application produite est le fruit de plusieurs centaines d'heures de développement. À ce jour, elle répond entièrement au cahier des charges précédemment validé par le client.

Pour chacun d'entre nous, il ne s'agit ni plus ni moins que de notre plus gros projet. Le seul temps consommé mesuré est celui qui correspond à la réalisation du back-end : 175 heures de développement soit précisément 5 semaines à temps plein pour un salarié au temps légal (toujours rien que pour le back-end).

La récompense, c'est une application dont nous sommes fiers et qui correspond aussi bien entièrement aux attentes du client, qu'à nos attentes personnelles.

A. Back-end

1. Défis techniques

a. Sécurité

Premier point : la sécurité. En effet, lorsqu'on construit une API exposée sur internet, c'est sûrement le point le plus important. Il ne faudrait pas que les noms, adresses e-mail ou mots de passe de tous les étudiants et enseignants se retrouvent accessibles à n'importe qui.

En pratique, nous avons mis en place toutes ces mesures (liste non exhaustive) :

- La base est complètement isolée. En effet, tous les morceaux de l'applicatif sont conteneurisés et seuls les conteneurs front-end et back-end sont accessibles via le serveur web exposé.
- Non pas un hachage simple des mots de passe, mais un hachage salé à clé aléatoire. Cela consiste à utiliser une nouvelle clé générée aléatoirement pour construire chaque hachage de mot de passe. Si le risque 0 n'existe pas, cela rend pratiquement irréaliste le déchiffrage d'un mot de passe et rend inutile toute table de correspondances.

- Toutes les routes sont soumises à des limites de taux adaptées afin que le système ne puisse être forcé par attaque brute force.
- Les utilisateurs possèdent tous des rôles (étudiants, enseignants et administrateurs) qui limitent leurs accès de manière hiérarchique.
- Les permissions des applications installées sur l'organisation sont restreintes au maximum de sorte qu'aucune action non prévue par l'application ne puisse être réalisée d'une quelconque manière.

b. Cache, vitesse et nombre d'utilisateurs

Une automatisation qui fonctionne correctement c'est bien, mais si on passe plus de temps qu'en le faisant à la main, c'est surtout inutile. Celle-ci vise potentiellement une centaine d'utilisateurs en parallèle, il faut tenir la charge et répondre à tous les utilisateurs en même temps sans ralentissement.

Ici, le défi réside dans la connexion de l'application à GitHub qui lui-même impose des limites de taux et ajoute une latence.

Pour ce cas, 2 solutions ont été mises en place :

- Nous répercutons les limites de taux de GitHub sur nos utilisateurs et le tout de manière hiérarchique. En effet, il n'y a pas de magie, si 50 utilisateurs demandent chacun 100 ressources différentes, on ne peut pas les fournir puisque GitHub refusera de nous communiquer autant de ressources en parallèle. De même, si un utilisateur demande 5000 ressources différentes, nous ne pourrons plus servir les autres utilisateurs. Nous avons donc défini des limites de taux pour chaque ressource en fonction des limites imposées par GitHub. Cependant, là où nous refuserons à un étudiant de rafraîchir 100 fois des statistiques dans la même heure, nous autoriserons un enseignant à la faire puisque celui-ci pourrait accéder à de multiples dépôts différents.
- Après de très nombreuses tentatives infructueuses via un proxy conteneurisé, nous avons trouvé une solution consistant à monter un cache local directement dans l'API. C'est un plugin que seules 3 personnes ont aimé sur GitHub (c'est dire à quel point il est niche et peu référencé). Celui-ci utilise les en-têtes retournés par l'API GitHub pour choisir de servir le conteneur depuis la RAM ou directement depuis l'API externe. Ce plugin nous permet de diminuer drastiquement la latence en évitant tous les appels inutiles et nous laisse également la possibilité de profiter de nos limites de taux GitHub pour récupérer des ressources plus diverses.

c. Cohérence avec GitHub

Maintenant que nous avons parlé de cache, nous sommes obligés de parler de cohérence. Si ces avantages dépassent leurs inconvénients, ils ne sont pas pour autant parfaits … Quand vous faites appel au cache, vous n'êtes pas absolument sûrs que certaines données n'ont pas été modifiées depuis l'enregistrement. Le même problème s'impose avec les données persistantes : c'est même bien pire. Il serait très compliqué et lent de revalider chacune de ces données.

Le problème ? Imaginez que vous effectuiez une modification directement sur GitHub. Vous essayez de réaliser la même opération sur la plateforme. Puisque pour elle cette action n'existe pas encore, elle va bien tenter de la réaliser. À cet instant GitHub va répondre négativement et provoquer une erreur complètement inattendue dans le processus. En effet, la plateforme n'a pas connaissance que l'action a déjà été réalisée.

Il y a bien une solution : GitHub propose des « webhooks ». Il s'agit de simples requêtes HTTP qui sont envoyées à l'applicatif pour le tenir au courant des actions directement réalisées par la plateforme. Si nous avons bien utilisé cette méthode pour détecter lorsqu'un utilisateur rejoint l'organisation, et ce en temps réel, il était ingérable de tous les implémenter sans dédoubler la charge de travail.

Cependant, nous avons quand même travaillé à l'atténuation de ces effets de bord en réalisant des tests sur certaines opérations qui peuvent être réalisées directement par les étudiants comme la modification des jalons directement sur le dépôt. Ainsi, les requêtes ne devraient pas tomber en erreur. Pour ce qui est des administrateurs, il est fortement recommandé de ne pas réaliser d'actions sur les dépôts en dehors de l'applicatif sous peine de créer des conflits durables inutilement.

d. Maintenabilité & bonnes pratiques

Chaque nœud de la chaîne a été scrupuleusement réfléchi pour délivrer une maintenabilité optimale.

Concernant l'infrastructure, les images utilisées pour héberger les composants back-end font l'objet de mises à jour fréquentes. De plus, leur légèreté, traduite par un nombre réduit de lignes de code source, facilite la maintenabilité et diminue la surface d'attaque potentielle.

Concernant l'architecture, nous avons mené une étude approfondie pour sélectionner l'ensemble des modules constituant le projet (avec une attention particulière portée au module hôte). Chacun de ces modules bénéficie, là encore, de mises à jour régulières et actives.

Enfin, pour ce qui concerne le code, nous avons réparti le code du back-end dans de très nombreuses classes, fichiers et dossiers : il y a 96 fichiers, par exemple. Nous avons mis en place les conventions de développement suivant les standards Airbnb (les plus utilisés au monde) et contrôlés par un linter. L'intégralité des retours remontés par le linter ont été corrigés. Enfin, nous avons autant que possible évité la duplication de code.

e. Vrai projet

Ce projet, ce n'est pas juste un projet de cours qui ne sera utilisé qu'une fois par un enseignant : nous avons donc travaillé sur un vrai projet terminé.

Pour nous, une application finie, ce n'est pas simplement une application qui respecte toutes les conditions précédentes, c'est une application où chaque processus, ou chaque logique est réfléchi et contrôlé pour être le plus optimal.

D'abord, cela commence évidemment par une cohérence : toutes les routes implémentent des fonctions identiques, réalisent les actions dans le même ordre et surtout où chaque cas est réfléchi. Toutes les éventuelles erreurs sont monitorées afin de pouvoir être corrigées.

Ensuite, il y a les petits plus qui ne sont pas forcément nécessaires mais qui rendent l'application plus qualitative. Nous pensons notamment à l'exemple du mail pour créer son accès sur la plateforme. Ici, on garde notre charte graphique et on garde la confiance de l'utilisateur.

2. Tests

Nous avons testé le back-end de façon manuelle, en réalisant pour chaque route un appel REST et en vérifiant tous les cas possibles. Cela nous a permis de confirmer que chacune des contraintes fonctionnelles est bien respectée. Ces vérifications ont été effectuées systématiquement avant de clôturer chaque fonctionnalité, pour ne rien laisser passer.

Plutôt que de mettre en place un test automatisé pour chaque contrainte, ce qui aurait été disproportionné ici, nous avons choisi de nous concentrer sur la diversité des scénarios à couvrir, plutôt que sur leur implémentation technique.

3. Documentation

Le back-end est exhaustivement documenté via un Swagger et peut être utilisé sans passer par l'IHM pour connecter notre applicatif à d'autres plateformes. Cette documentation est disponible ici.

B. Front-end

1. Défis techniques

a. Planification et modularité des composants UI

Dès le début du projet, l'absence d'un planning précis concernant le développement des composants réutilisables s'est rapidement fait sentir. Nous avons donc décidé de poser les bases d'une architecture modulaire, en créant des composants UI génériques tels que les tableaux, boutons ou cartes. Cette approche nous a permis, au fil de l'avancement des fonctionnalités prévues dans notre GANTT, de décliner ces briques de base en éléments plus spécifiques (combobox, notifications, notifications de confirmation, etc.).

Cette standardisation et modularisation ont eu un double bénéfice :

- Assurer la cohérence visuelle et fonctionnelle de l'interface
- Faciliter l'évolution du code, chaque nouveau besoin pouvant s'appuyer sur des composants déjà existants et éprouvés

b. Uniformisation de la structure et des conventions de code

Travailler à deux sur le frontend a naturellement fait émerger des divergences sur plusieurs aspects :

- L'organisation des fichiers
- Le nommage des composants
- Les conventions de styles

Après quelques difficultés initiales (conflits de fusion, incompréhensions sur la répartition des responsabilités des composants), nous avons pris le temps de formaliser : une arborescence de projet standardisée et des conventions de nommage partagées. Cela a considérablement réduit les conflits et facilité les revues de code.

c. Accessibilité et gestion multi-profils utilisateurs

L'une des principales contraintes du projet était de rendre accessible et compréhensible une grande quantité d'informations sur un espace scolaire : à savoir, répondre aux besoins spécifiques de chaque profil utilisateur (administrateur, enseignant, étudiant).

La modularité de nos composants, combinée à une maquette validée en amont avec le client (issue de notre cahier des charges), nous a permis d'adapter dynamiquement : l'affichage des données et les parcours utilisateurs. L'objectif étant de maintenir une interface épurée et intuitive,

tout en offrant à chaque type d'utilisateur les informations et fonctionnalités pertinentes à son rôle.

d. Synchronisation d'avancement et coordination entre développeurs

Partager le même codebase a nécessité une communication continue pour éviter doublons et conflits. Nous sommes alors passés par des stand-ups réguliers, des revues de code fréquentes, des démonstrations internes à la fin de fonctionnalité clé. Cela nous a permis de mieux anticiper les dépendances entre tâches, de débloquer rapidement les points bloquants et d'assurer une cohérence globale dans la réalisation du projet.

e. Intégration et standardisation des échanges avec le backend

Le backend, très robuste et complet, exposait de nombreux endpoints. Chaque nouvelle fonctionnalité frontend impliquait donc : une compréhension précise des schémas de données et une gestion rigoureuse des erreurs côté client. Notre réponse a été la création de services front centralisés pour tous les appels API, un système global et modulable de gestion des erreurs. Mais aussi des fonctions et utilitaires de transformation et de formatage pour adapter les données de l'API aux besoins spécifiques de nos composants UI.

f. Responsive design et gestion de la performance en réseau

Garantir une expérience utilisateur fluide sur desktop, tablette et mobile a représenté un défi supplémentaire. Nous avons dû adapter nos composants à différents formats et résolutions d'écran, effectuer des tests répétés sur plusieurs appareils. Mais aussi optimiser les comportements en cas de faiblesse de connexion ou de latence réseau. Nous avons veillé à toujours notifier l'utilisateur de l'état de l'application : avant une action (bouton désactivé, message de confirmation), pendant le traitement (indicateurs de chargement, skeleton screens), après l'action (messages de succès ou d'échec). L'objectif étant d'éviter toute zone d'incertitude pour l'utilisateur et de minimiser la frustration liée aux délais de réponse.

On peut citer un exemple important par rapport à cela qu'est l'affichage des statistiques pour le travail d'un groupe d'étudiants où nous devions parfois faire face à des données incomplètes ou partielles (limitation, latence GitHub indépendamment de notre application). Nous avons dû réfléchir à un système d'analyse des données pour détecter les informations manquantes. Ce qui a donné lieu à un autre mécanisme, de rafraîchissement automatique et progressif des données.

2. Tests

Pour garantir la qualité et la robustesse de la partie front-end, nous avons adopté une méthodologie de tests en plusieurs étapes, appliquée principalement en phase de clôture du projet mais aussi de façon continue tout au long du développement.

a. Méthodologie générale des tests fonctionnels

Identification des scénarios de test : Nous avons listé pour chaque fonctionnalité les scénarios principaux et secondaires, en nous basant sur :

- Les parcours utilisateur définis (ex. : première connexion, navigation entre les pages, consultation des statistiques GitHub, etc.)
- Les contraintes spécifiques du cahier des charges
- Les cas de gestion des erreurs (ex : erreurs de saisie, absence de données, comportements réseau)

Définition des résultats attendus : pour chaque scénario, nous avons précisé :

- Les éléments attendus à l'écran
- Les valeurs chiffrées
- Les messages d'erreur ou de succès attendus

Exécution des tests manuels avec checklist : nous avons élaboré une checklist de tests, puis parcouru l'ensemble de l'application en suivant rigoureusement chaque scénario.

- Chaque écart par rapport aux critères d'acceptation a été consigné
- Les problèmes détectés étaient pour la plupart mineurs et visuels
- La majorité des correctifs ont été réalisés dans la journée

Vérification post-correction et détection des régressions : après chaque série de corrections, nous avons rejoué l'ensemble des tests pour vérifier que :

- Les problèmes détectés avaient bien été corrigés
- Aucune régression n'avait été introduite ailleurs dans l'application

Tests multi-environnements avec le client : en phase finale, nous avons fait tester l'application sur deux environnements différents par notre client. Ces tests externes ont permis de valider :

- L'absence d'anomalies côté back-end

- La robustesse et la stabilité de l'interface utilisateur, quels que soient les profils utilisateurs

b. Tests d'intégration front-back et robustesse des échanges API

La richesse et la complexité du backend nous ont conduit à tester chaque intégration API de manière approfondie. Pour cela :

- Chaque appel API a été soigneusement intégré, en tenant compte des cas d'usage réels et des spécificités des endpoints.
- Nous avons vérifié systématiquement :
    - Les cas d'usage courants
    - Les réponses inattendues ou erreurs retournés par le serveur (ex : statuts 401, 403, 404, 500…)
    - Les états de l'interface associés à chaque réponse (chargement, succès, échec)
- Comme dit dans la partie précédente, la gestion des erreurs a été centralisée, permettant à chaque composant de réagir de manière cohérente face aux éventuels problèmes côté serveur.

De plus, pour pallier l'indisponibilité ou la difficulté de simulation de certains endpoints en cours, nous avons :

- Créé des mocks API "maison", simulant des réponses réalistes pour nos tests
- Développé des composants temporaires dits "laboratoire", permettant de tester l'affichage et le comportement des vues sans dépendance directe au backend

Cette stratégie nous a permis de garder un rythme de développement soutenu tout en anticipant l'intégration avec le backend.

La double validation (interne puis client) nous a apporté la certitude que le produit était robuste, fiable et prêt à être mis en production.

IV. Déploiement

La procédure ci-contre vise à permettre le déploiement sur n'importe quel nouvel environnement.

A. Prérequis

Docker doit être installé sur le serveur hébergeur du projet (procédure d'installation).

B. Téléchargement des sources

1. Télécharger les sources compressées via cette URL.
2. Se connecter au serveur hébergeur.
3. Téléverser les sources compressées sur le serveur.
4. Décompresser les sources dans un dossier adapté.

C. Complétion des environnements

Pour chaque service :

1. Renommer le fichier .env.example en .env.
2. Le compléter selon les exigences.
    - Les informations concernant GitHub seront complétées ci-après.

D. Droits des membres dans l'organisation

Cette section vise à restreindre au maximum les droits par défaut des nouveaux invités et à permettre à l'applicatif de fonctionner avec toutes ses fonctionnalités.

1. Se rendre dans les paramètres GitHub.
2. Cliquer sur « Switch settings context ».
3. Sélectionner l'organisation.
4. Cliquer sur « Member privileges ».
5. Appliquer les paramètres suivants (donnés dans l'ordre). Penser à enregistrer à chaque catégorie.

Catégorie	Paramètre		Valeur
Member privileges	Base permissions		No permission
	Repository creation	Public	Non
		Private	Non
	Repository forking		Oui

	Repository discussions		Non
	Projects base permissions		Write
	Pages creation	Public	Non
		Private	Non
	Integration access requests		Non
Admin repository permissions	Allow members to change repository visibilities for this organization		Non
	Allow members to delete or transfer repositories for this organization		Non
	Allow repository administrators to delete issues for this organization		Oui
Member team permissions	Allow members to create teams		Non

E. Équipe pédagogique

Une équipe pédagogique spécifique à l'application doit être créée. Ses membres seront gérés par l'application elle-même. Le nom ne fait pas foi. Aucun membre ne doit être ajouté ou supprimé manuellement. L'équipe ne doit pas être supprimée.

1. Se rendre dans les organisations GitHub.
2. Sélectionner l'organisation.
3. Cliquer sur « Teams ».
4. Cliquer sur « New team ».
5. Appliquer les paramètres suivants (donnés dans l'ordre).

Paramètre	Valeur
Team name	Équipe pédagogique
Description	***Laisser vide***
Team visibility	Secret
Team notifications	Disabled

F. Webhooks

1. Se rendre sur ce générateur de nombre hexadécimal aléatoire.
2. Cliquer sur « Copy ».
3. Ajouter la valeur dans le fichier d'environnement Fastify.

4. Se rendre dans les paramètres GitHub.
5. Cliquer sur « Switch settings context ».
6. Sélectionner l'organisation.
7. Cliquer sur « Webhooks ».
8. Cliquer sur « Add webhook ».
9. Appliquer les paramètres suivants (donnés dans l'ordre).

Paramètre	Valeur
Payload URL	https://git-town-api.miage-amiens.fr/webhooks/github/organizations
Content type	application/json
Secret	***Coller la valeur générée aléatoirement précédemment.***
SSL verification	Enable SSL verification
Which events would you like to trigger this webhook?	Let me select individual events.
-	Organizations
Active	Oui

G. Création des applications GitHub

3. OAuth

1. Se rendre dans les paramètres GitHub.
2. Cliquer sur « Switch settings context ».
3. Sélectionner l'organisation.
4. Cliquer sur « Developer settings ».
5. Cliquer sur « OAuth Apps ».
6. Cliquer sur « New OAuth App ».
7. Appliquer les paramètres suivants (donnés dans l'ordre).

Paramètre	Valeur
Application name	GitTown-OAuth
Homepage URL	https://git-town.miage-amiens.fr
Application description	***Laisser vide***

Authorization callback URL	https://git-town.miage-amiens.fr/login/authorize
Enable Device Flow	Non

8. Cliquer sur « Save ».
9. Repérer « Client ID ».
10. Copier la valeur.
11. Ajouter la valeur dans le fichier d'environnement Fastify.
12. Cliquer sur « Generate a new client secret ».
13. Copier la valeur.
14. Ajouter la valeur dans le fichier d'environnement Fastify.
15. Cliquer sur « Upload new logo ».
16. Téléverser le fichier intitulé « SANS NOM - SANS CONTOUR.png » fourni avec cette documentation.
17. Cliquer sur « Set new application logo ».

4. App

1. Se rendre dans les paramètres GitHub.
2. Cliquer sur « Switch settings context ».
3. Sélectionner l'organisation.
4. Cliquer sur « Developer settings ».
5. Cliquer sur « GitHub Apps ».
6. Cliquer sur « New GitHub App ».
7. Appliquer les paramètres suivants (donnés dans l'ordre).

Catégorie	Paramètre	Valeur
Register new GitHub App	GitHub App name	GitTown-Manager
	-	***Laisser vide***
	Homepage URL	https://git-town.miage-amiens.fr
Identifying and authorizing users	Callback URL	https://git-town.miage-amiens.fr
	Expire user authorization tokens	Non
	Request user authorization (OAuth) during installation	Non
	Enable Device Flow	Non

Post installation	Setup URL (optional)		https://git-town.miage-amiens.fr
	Redirect on update		Oui
Webhook	Active		Non
	Webhook URL		***Laisser vide***
	Secret		***Laisser vide***
Permissions	Repository permissions	***Tous***	No access
		Administration	Read and Write
		Contents	Read and Write
		Issues	Read and Write
		Metadata	Read-only
		Pull requests	Read-only
	Organization permissions	***Tous***	No access
		Members	Read and Write
	Account permissions	***Tous***	No access
Where can this GitHub App be installed?	-		Only on this account

8. Cliquer sur « Save ».
9. Repérer « App ID ».
10. Copier la valeur.
11. Ajouter la valeur dans le fichier d'environnement Fastify.
12. Repérer « Client ID ».
13. Copier la valeur.
14. Ajouter la valeur dans le fichier d'environnement Fastify.
15. Cliquer sur « Generate a new client secret ».
16. Copier la valeur.
17. Ajouter la valeur dans le fichier d'environnement Fastify.
18. Cliquer sur « Upload a logo... ».
19. Téléverser le fichier intitulé « SANS NOM - SANS CONTOUR.png » fourni avec cette documentation.
20. Cliquer sur « Set new avatar ».
21. Cliquer sur « Generate a private key ».
22. Ouvrir le fichier téléchargé avec un éditeur de texte.
23. Remplacer tous les sauts de ligne par « \n » de sorte à avoir une clé sur une seule ligne.

24. Ajouter la valeur dans le fichier d'environnement Fastify.
25. Cliquer sur « Install App ».
26. Cliquer sur « Install » sur l'organisation.
27. Cliquer sur « Install ».
28. Se rendre dans les paramètres GitHub.
29. Cliquer sur « Switch settings context ».
30. Sélectionner l'organisation.
31. Cliquer sur « GitHub Apps ».
32. Cliquer sur « Configure » sur « GitTown-Manager ».
33. Repérer l'identifiant d'installation dans l'URL.
34. Copier la valeur.
35. Ajouter la valeur dans le fichier d'environnement Fastify.

H. Récupération de l'ID de l'organisation et de l'ID de l'équipe pédagogique

1. Générer un jeton d'accès à l'installation via la méthode Fastify getInstallationAccessToken.
2. Effectuer les 2 appels REST.
   - Get an organization
   - List teams
3. Copier les valeurs.
4. Ajouter les valeurs dans le fichier d'environnement Fastify.

I. Lancement de l'application

1. Se connecter au serveur hébergeur.
2. Se rendre dans le dossier contenant les sources.
3. Exécuter la commande « docker compose up -d ».

J. Redirection des noms de domaine

1. Proxy

Le proxy interne du serveur doit servir les conteneurs Docker sur le serveur web publiquement exposé en combinant les configurations déjà en place et celles nécessaires pour l'exposition du back-end et du front-end

2. DNS

Les sous-domaines choisis pour le back-end et le front-end doivent pointer vers le serveur web publiquement exposé.

V. Conclusion et prise de recul

A. Résultats

Il semble important de commencer par un état des lieux sur le projet. Très clairement, nous sommes fiers de ce qui a été réalisé. Nous proposons à ce jour un système qui fait preuve de robustesse en restant flexible. Les tests finaux, n'ont plus relevé la moindre anomalie qui n'ait pas été corrigée.

Le back-end répond toujours en assurant sécurité, vitesse et maintenabilité, le front-end respecte la maquette prévue initialement. L'interface utilisateur a vraiment été travaillée de manière que toutes les fonctionnalités soient complètes mais aussi simples d'utilisation. Le parcours de l'utilisateur sur la plateforme est fluide.

Nous avons hâte que ce système soit mis à disposition de tous – administrateurs, enseignants et étudiants – dès la prochaine rentrée.

B. Axes d'amélioration

Même si ces résultats correspondent parfaitement à nos attentes, il faut bien dire que tout ne s'est pas déroulé exactement comme nous l'attendions.

1. Gestion du planning

Nous avions défini une marge très importante en fin de projet et nous nous sommes parfois trop reposés sur celle-ci lorsque du retard a été pris. En effet, les indisponibilités de chacun n'avaient pas forcément été anticipées, des fonctionnalités se sont avérées plus complexes que d'autres et parfois notre côté perfectionniste nous a rattrapé : nous avons voulu faire mieux que la fonctionnalité imaginée. La cause directe, c'est qu'à force, on finit par arriver en fin de projet avec une marge bien consommée et beaucoup de tâches qui restent. Si cela devait être à refaire, nous fixerions sans doute plus de jalons afin de répartir les pics de développement à différentes périodes plutôt qu'uniquement à la fin. Évidemment, les marges seraient alors réparties aux niveaux de ces jalons plutôt qu'uniquement à la fin.

2. Back-end

Même si le travail réalisé en back-end est en soi une petite prouesse lorsqu'on connait le temps normalement alloué à ce projet, il n'est en effet pas parfait. Le plus gros point noir nous semble être la cohérence lorsque des modifications sont appliquées directement sur GitHub. Certes, le

cahier des charges n'a pas été conçu en précisant que des modifications manuelles pourraient avoir lieu mais là où chaque cas est en permanence testé, ici ce n'est pas contrôlé. Cependant, comme nous le disions dans la partie intitulée « Cohérence avec GitHub », la mise en place d'un système de mise à jour en temps réel des données persistantes aurait demandé un temps conséquent. Cela n'était pas envisageable au regard de la date limite imposée.

3. Front-end

Si aucune concession n'a été faite sur l'interface utilisateur pour proposer un parcours agréable, simple et utile, notre autocritique se concentre sur la cohérence à l'intérieur du code. Si le code a été découpé au mieux pour rester lisible, il a manqué un élément essentiel pour le rendre davantage maintenable : des règles pour le linter. À l'origine, nous avions pris le parti de se dire que nous ne configurions celui-ci qu'à la fin du projet pour ne pas ralentir les développements et éviter les conflits lors des fusions, puisque cette partie a été réalisée par 2 développeurs différents. La réalité des choses, c'est que nous sommes arrivés à la fin du projet à quelques jours de la date limite imposée et que corriger toutes les erreurs soulevées par le linter aurait été absolument impossible dans le temps imparti. Si le code n'est donc pas forcément de mauvaise qualité en soi, il est vrai qu'il reste perfectible.

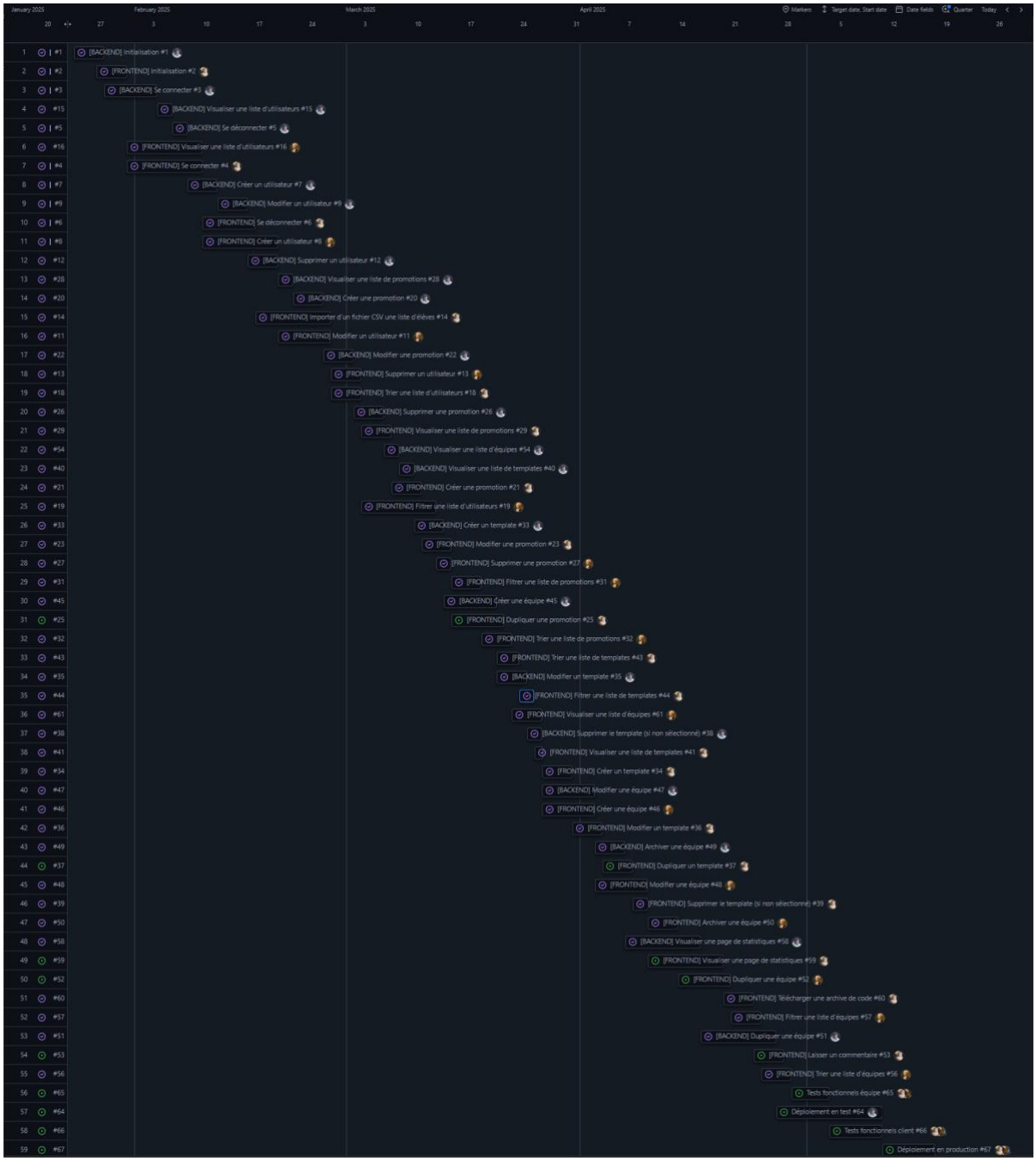
C. Perspectives

Ce rapport prend fin nous aurons cherchés à retracer le parcours de ce développement. En plus des points soulevés dans les parties précédentes, nous pouvons donner 2 perspectives d'améliorations qui vont avec l'évolution des technologies de nos jours :

- Ces dernières années, l'usage des mots de passe se perd. Il est possible qu'un jour un site vous ait proposé de vous inscrire, ou d'ajouter une méthode de connexion nommée « clé d'accès ». Ces clés uniques et aléatoires sont générées et stockées par votre système d'exploitation. Ainsi, vous n'aurez jamais à en créer, ni à vous en souvenir. Implémenter ce système comme méthode de connexion viserait à rendre les comptes utilisateurs plus sécurisés.
- Enfin, pour rester dans le domaine de la sécurité, nous avons expliqué précédemment que nous avions haché les mots de passe via des clés aléatoires. Nous avons utilisé un algorithme nommé SHA qui bien qu'encore très utilisé dans le domaine du WEB, est moins robuste que des algorithmes plus récents comme Argon2. Ces nouveaux algorithmes émergent pour parer les innovations technologiques qui décuplent le nombre d'actions réalisables à la seconde.

VI. Annexes

A. Gantt prévisionnel



B. Gantt final