BORDEAUX INP Ensc

Gestionnaire de tâches

Rendu final - P2I

GOY Clément

2023/2024

Sommaire

Introduction

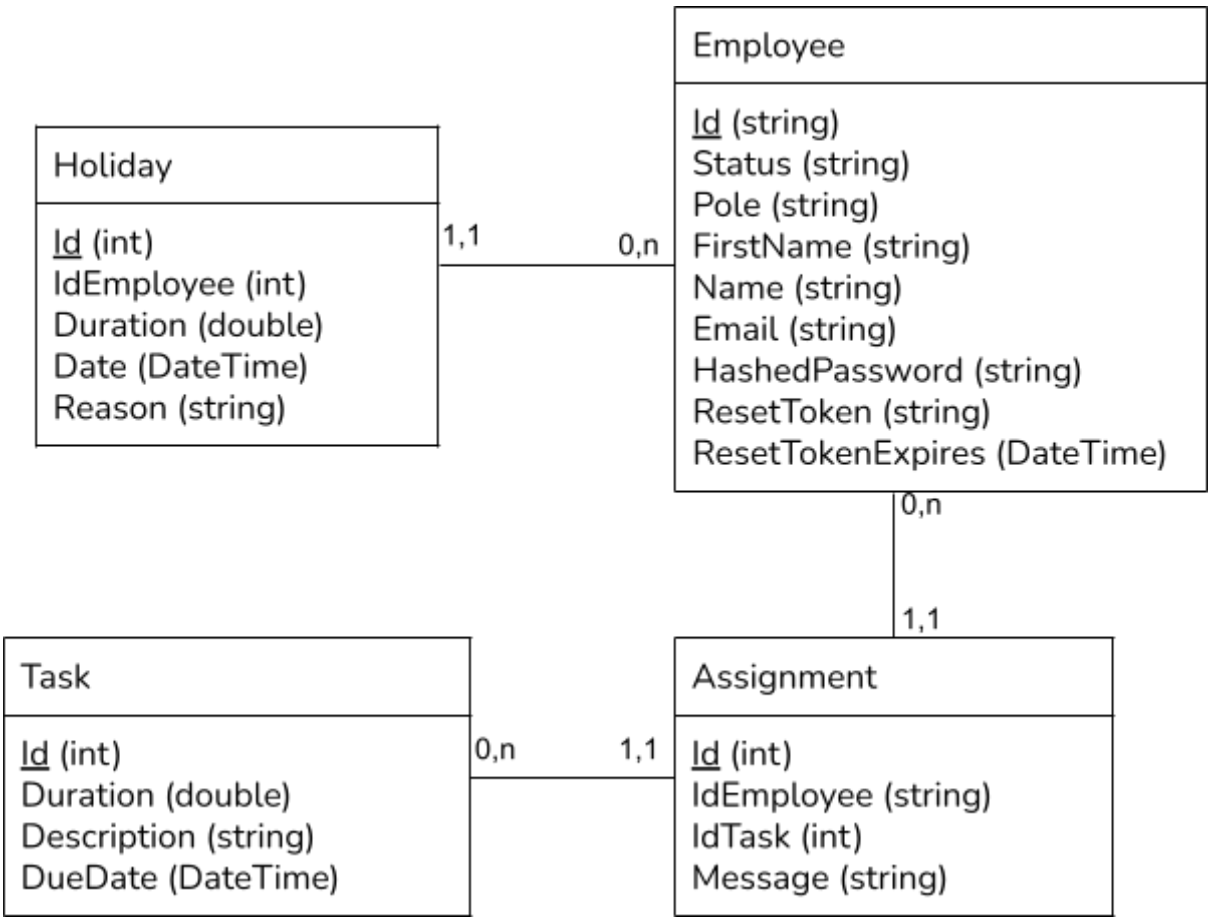
Ce projet individuel d'informatique intervient dans le cadre du second semestre de deuxième année à l'École Nationale Supérieure de Cognitique (ENSC) de Bordeaux. L'objectif de ce module est d'encourager les étudiants à développer leurs compétences en informatique et en gestion de projet. Ce projet doit être réalisé seul, une soutenance aura lieu le 28 mars et un livrable doit être rendu le 5 avril.

Le sujet qui a été retenu pour ce projet est la programmation d'une application de gestion de tâches et d'organisation du travail à réaliser en entreprise. Il sera constitué d'un projet d'API C# pour le back-end et d'un projet Vue.js pour le front-end.

La société fictive cliente du projet est composée d'employés répartis dans plusieurs pôles (logistique, développement...) qui ont des statuts différents (managers ou membres d'équipe). Les employés ont chacun des tâches qui leur sont assignés et des congés qui leur sont attribués. Cet outil de gestion de tâche doit permettre aux managers de réaliser plusieurs actions, certaines accessibles pour les membres d'équipe et d'autres réservées aux managers.

L'objectif principal de ce projet était de se concentrer sur le back-end et notamment sur la sécurité de ce dernier et des données qu'il gère.

Afin d'avoir une vue d'ensemble, voici l'organisation des classes de l'API :



Choix techniques

Outils utilisés

Visual Studio Code a été utilisé comme support pour la programmation, github a permis la sauvegarde et le partage de code tandis que la suite google a été employé pour réaliser les différents plannings et rendus.

Le back-end a été codé en C# avec .NET 8.0 et le swagger a contribué à visualiser la progression du projet d'API. Ce choix s'explique par la proximité avec ce qui a été vu en cours et donc les compétences déjà acquises avec ces outils.

Le front-end en javascript et html s'est fait à partir du framework Vue.js, il n'est pas terminé et ne comporte aucun style. Le choix du framework a été effectué car c'est l'un des plus utilisés et je ne l'avais jamais utilisé, je souhaitais donc me familiariser avec.

Langue des variables

Les variables sont toutes en anglais dans le projet d'API, cela s'explique par la nécessité d'avoir toutes les variables dans la même langue par volonté de cohérence. Les espaces de noms et les bibliothèques utilisés étant en anlais, choisir cette langue pour l'ensemble des variables permettait d'homogénéiser le code.

Description fonctionnelle

Fonctionnalités initiales

Les fonctionnalités prévues initialement ont été revues au fur et à mesure de l'avancée du projet et adaptées de manière plus cohérentes. Des changements plus ou moins importants ont donc été apportés.

Ci-dessous figurent les fonctionnalités initialement prévues ainsi que leur état d'avancement au commencement du projet. Elles sont répertoriées par ordre décroissant de priorité dans le tableau ci-dessous :

Numéro	Description	Etat
01	Choix de l'organisation des classes & création d'un projet d'API et d'une base de donnée en locale	Commencé
02	Possibilité de créer, supprimer et modifier des comptes utilisateurs avec différents statuts (employé ou manager)	Pas commencée

03	Possibilité d'assigner des tâches pour les managers	Pas commencée
04	Création d'un projet Javascript et liaison avec le back-end	Pas commencé
05	Possibilité de s'identifier à son compte	Pas commencé
06	Renforcement de la sécurité de la gestion des données dans l'application	Pas commencé
07	Amélioration de l'interface	Pas commencé

Fonctionnalités intermédiaires

En comparaison, voici les fonctionnalités retenues lors du rendu intermédiaire du projet ainsi que l'état d'avancement à ce même moment, il y a plusieurs changements. Elles répertoriées par ordre décroissant d'importance dans le tableau ci-dessous :

Numéro	Description	Etat
01	Choix de l'organisation des classes & création d'un projet d'API et d'une base de donnée en locale	Fait
02	Possibilité de créer, supprimer et modifier des comptes utilisateurs avec différents statuts (employé ou manager)	Fait
03	Possibilité d'assigner des tâches pour les managers	Fait
04	Création d'un projet Javascript et liaison avec le back-end	Fait, mais nécessite d'être complété au fur et à mesure de l'avancement du projet
05	Possibilité de s'identifier à son compte	En cours
06	Renforcement de la sécurité de la gestion des données dans l'application	En cours
07	Possibilité de récupérer son mot de passe	A peine commencé

	grâce à la réception d'un email avec un lien et une autorisation sous forme de token	
08	Faire une maquette de l'application	Pas commencé
09	Amélioration de l'interface avec du style en suivant la maquette	Pas commencé

L'ordre d'importance des fonctionnalités choisies pour le rendu intermédiaire suivait la logique expliquée ci-dessous.

Les fonctionnalités 01 à 05 sont indispensables et constituent le cœur du projet. Les fonctionnalités 06 et 07 sont pertinentes et seront complétées dans la mesure du temps qu'il restera après la réalisation des précédentes. Les fonctionnalités 08 et 09 seront davantage perçues comme une piste d'amélioration n'a que peu de chances d'être mise en application.

Ces choix de modifications ont été effectués car un des buts du projets était de progresser en développement back-end et d'apprendre à faire le lien avec un projet front-end. J'ai eu du mal à faire le lien entre mon projet Vue.js et mon API et j'ai rencontré des erreurs pour lesquels notre formation à l'école ne donne pas de clés de résolution (comme les erreurs de "Cross Origin Resource Sharing" ou celle liées à l'utilisation d'Axios). Lorsque j'ai fini par les résoudre et que le lien a été fait, j'ai préféré faire une API complète, avec un maximum de fonctionnalités et une bonne sécurisation des données et garder le design de l'interface en ultime apport au projet à adapter en fonction du temps restant.

Fonctionnalités finales

Voici les fonctionnalités intermédiaires avec l'état d'avancement de fin de projet correspondant à chacune d'entre elles :

Numéro	Description	Etat
01	Choix de l'organisation des classes & création d'un projet d'API et d'une base de donnée en locale	Fait

02	Possibilité de créer, supprimer et modifier des comptes utilisateurs avec différents statuts (employé ou manager)	Fait
03	Possibilité d'assigner des tâches pour les managers	Fait
04	Création d'un projet Javascript et liaison avec le back-end	Avancé, mais nécessite d'être complété
05	Possibilité de s'identifier à son compte	Fait
06	Renforcement de la sécurité de la gestion des données dans l'application	Fait
07	Possibilité de récupérer son mot de passe grâce à la réception d'un email avec un lien et une autorisation sous forme de token	Fait
08	Faire une maquette de l'application	Pas commencé
09	Amélioration de l'interface avec du style en suivant la maquette	Pas commencé

Détail des fonctionnalités du projet

1. Authentification

a. Fonctionnement général

L'authentification (si elle réussie) va générer un token Json Web Token (JWT), qui permet la transmission d'informations sur l'employé connecté. Ces jetons contiennent une clé chiffrée via l'algorithme HMAC avec SHA-256 : `Microsoft.IdentityModel.Tokens.SecurityAlgorithms.HmacSha256Signature` pour s'assurer de l'intégrité des données.

Dans ce projet, la Key, le Issuer et l'Audience du jeton sont configurés dans le fichier appsettings.json.

Ce jeton est généré dans le fichier AuthenticationController.cs avec une durée de vie de 9 heures, ce qui est suffisant pour que l'employé reste connecté à la

journée. Des informations sont contenues par le token, comme le rôle de l'employé et son email.

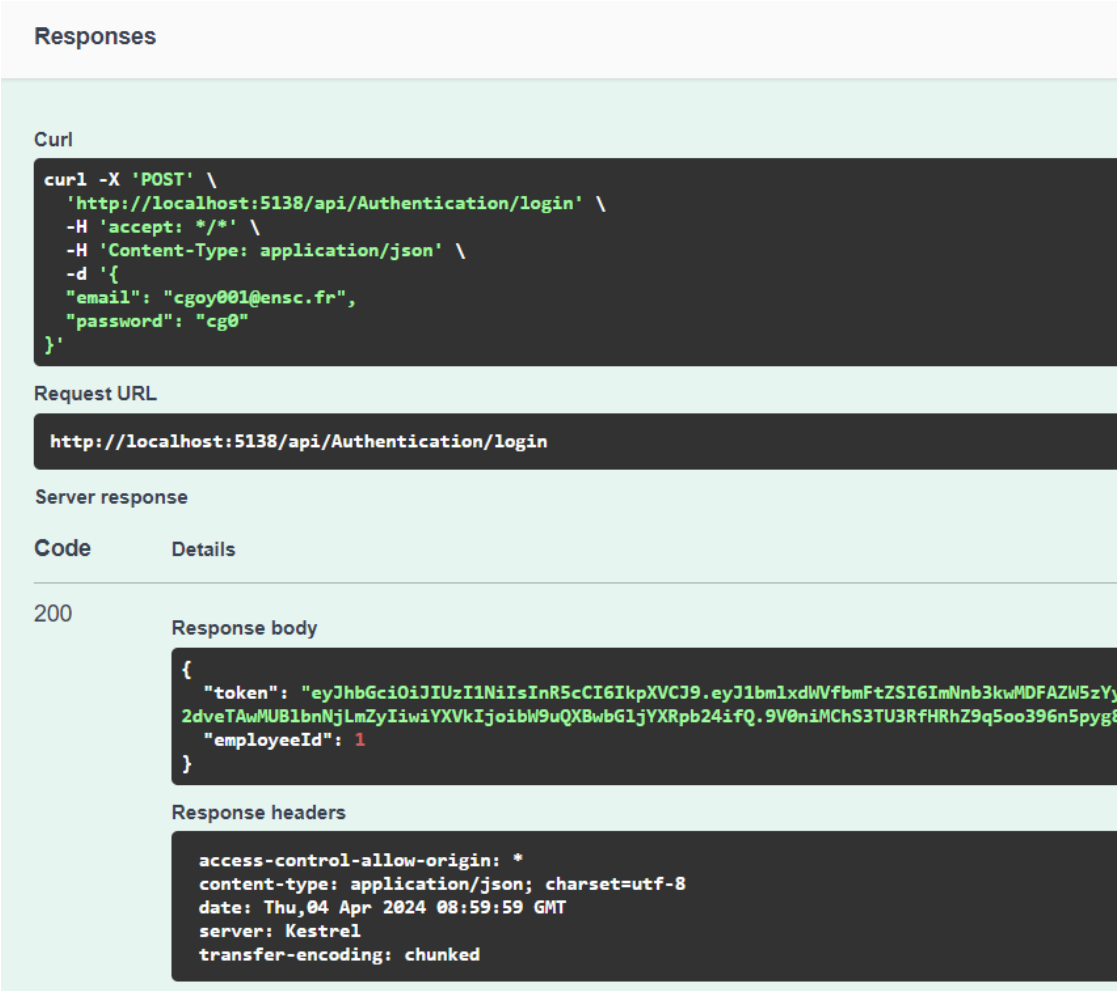
Ensuite, la vérification de la validité du jeton et l'extraction des informations qu'il contient s'effectue dans le fichier Program.cs. Sans cette validation, l'utilisateur ne peut effectuer aucune action que ce soit depuis le swagger ou depuis l'application.

Enfin, si l'authentification est réussie, il est redirigé depuis la page de connexion vers la page d'accueil. Mais cette dernière n'est pas la même en fonction de la composition du jeton, si l'utilisateur est un manager, la page d'accueil redirige vers des actions réservées aux managers.

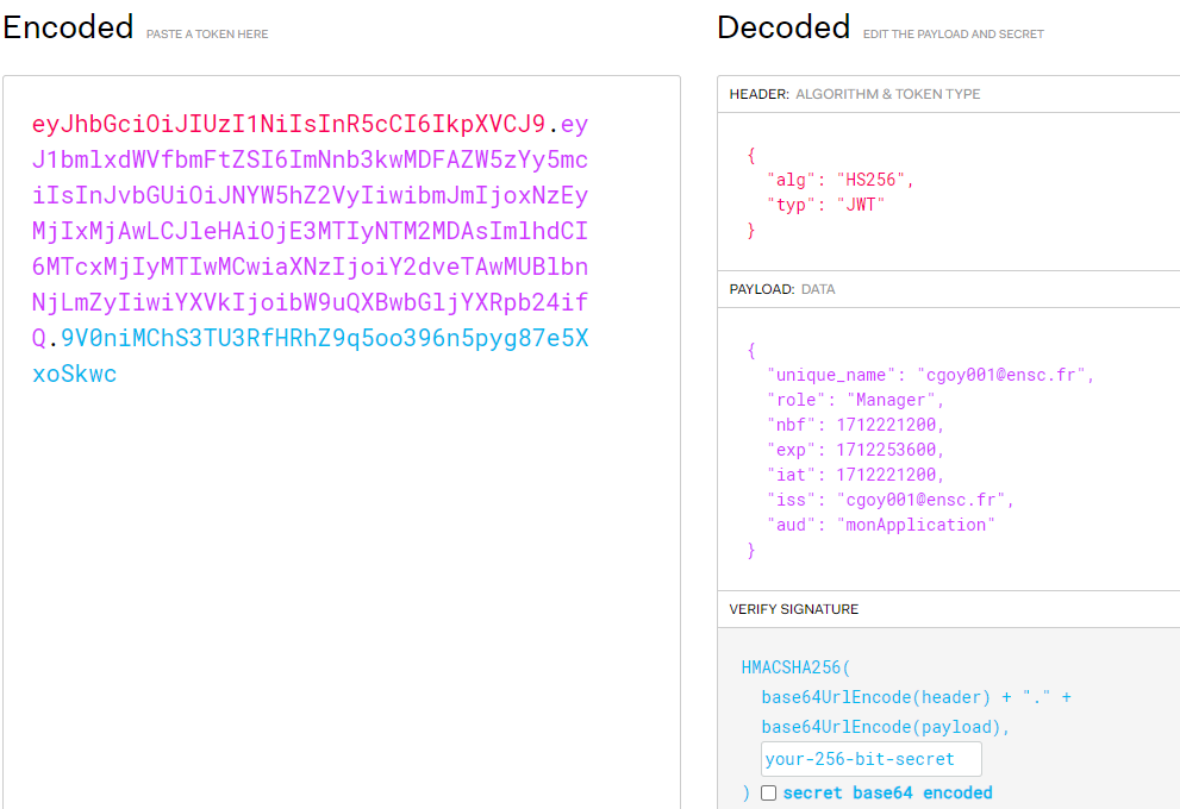
b. Tests

Plusieurs tests ont été effectués afin de constater le bon fonctionnement de cette étape. Les tests ci-dessous ont parfois été réalisés depuis le swagger et parfois depuis l'application en fonction de ce qui est le plus intéressant à montrer mais en soit c'est le même processus à l'œuvre.

Test de connexion dans le cas nominal : l'authentification réussie et renvoie bien un token lorsque l'on rentre l'email et le mot de passe correcte d'un employé présent dans la base de données. Dans le cas où l'on effectue cette action depuis l'application, on est directement redirigé vers la page d'accueil correspondant à notre statut.

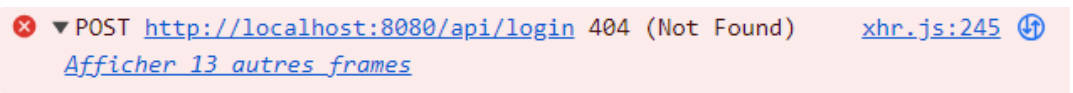


De plus, ce token contient bien les informations voulues, cela a été vérifié à l'aide du site jwt.io (cela a aussi été vérifié dans l'application) :



Test de connexion dans le cas d'erreur : plusieurs cas censés mener à un échec de la connexion ont été testés et ils sont tous concluants. Voici l'exemple de l'affichage du message d'erreur sur l'application :

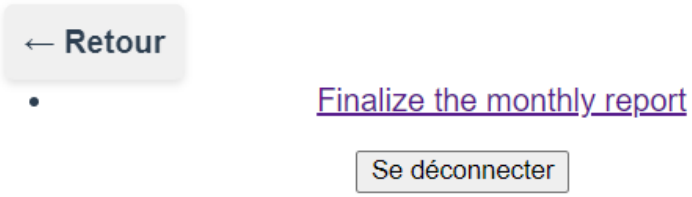
Et ici la réponse du serveur en cas d'erreur :



2. Fonctionnalités de l’application

La page d’accueil pour les membres d'équipe est la suivante :

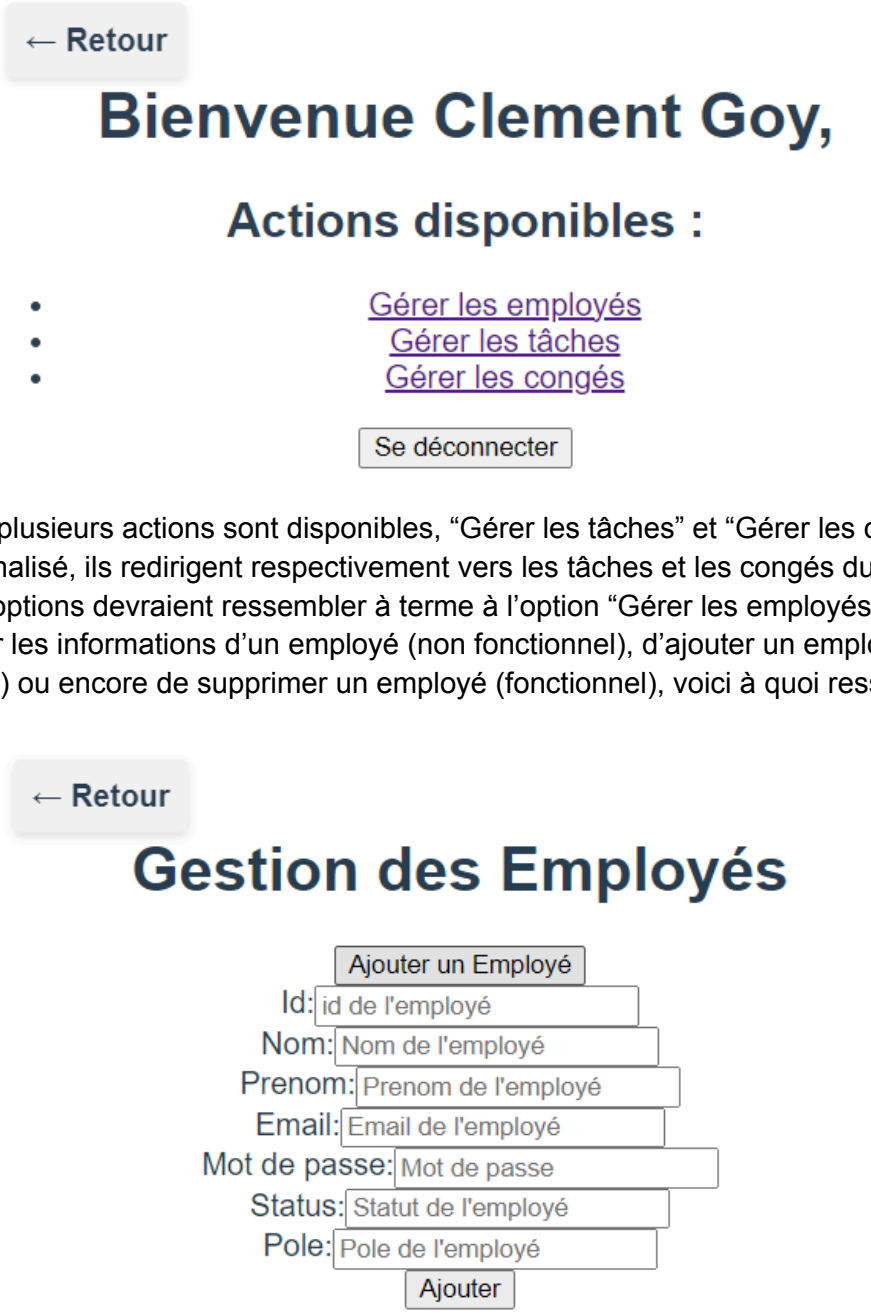
Ils ont alors accès aux fonctionnalités “Ma liste de tâches” ce qui redirige vers :



On peut alors accéder aux détails d’une tâche en cliquant dessus (le fonctionnementest analogue pour les congés) :

“Ma liste de congés” est aussi une fonctionnalité accessible, cependant je n’ai pas réussi à faire fonctionner l'outil “Contacter mon manager”.

La page d’accueil pour les membres d'équipe est la suivante :



Là encore plusieurs actions sont disponibles, “Gérer les tâches” et “Gérer les congés” ne sont pas finalisé, ils redirigent respectivement vers les tâches et les congés du manager. Ces deux options devraient ressembler à terme à l’option “Gérer les employés” qui propose de modifier les informations d’un employé (non fonctionnel), d’ajouter un employé (non fonctionnel) ou encore de supprimer un employé (fonctionnel), voici à quoi ressemble cette page :

3. Récupération du mot de passe

Lorsqu'un employé a oublié son mot de passe, il doit avoir une porte de sortie pour pouvoir accéder à son compte et réinitialiser son mot de passe, pour se faire, il doit entrer son email dans un formulaire. Un email lui a ensuite été envoyé avec un lien où il peut saisir son nouveau mot de passe avant de confirmer pour effectuer le remplacement dans la base de données.

En pratique, les actions qui permettent cette réinitialisation sont codées dans le fichier Controller nommé ForgetPassword.cs. La méthode statique GenerateToken va créer un tableau de 32 octets, ils prennent chacun une valeur entre 0 et 255 (en binaire) générée aléatoirement via un générateur sécurisé : `System.Security.Cryptography.RNGCryptoServiceProvider()`. Enfin, le tableau binaire est converti en chaîne de caractères pour former le token qui est renvoyé.

Ce token va être prendre la valeur de l'attribut ResetToken (de l'employé qui a oublié son mot de passe) dans la base de données. En parallèle, l'attribut ResetTokenExpires (qui est de type DateTime) prend la valeur correspondant au moment qu'il sera dans 1 heure.

Un email est envoyé à l'utilisateur via une adresse créée spécialement pour ce projet et configurée pour envoyer des emails via le Simple Mail Transfer Protocol (SMTP). Cet email contient un lien qui redirige vers une page du projet Vue.js contenant un formulaire permettant la réinitialisation du mot de passe. Le token généré précédemment est contenu dans le lien et une vérification est faite pour s'assurer que le token du lien est identique à celui présent dans la base de données pour permettre la réinitialisation. Une second test s'assure que le token est toujours valide, c'est à dire que ResetTokenExpires n'est pas dépassé. Le formulaire est comme cela :

4. Différentes actions des controllers

Différentes actions sont disponibles, elles ont été codées dans le dossier Controllers. Voici une liste de ces actions (hors l'authentification et la récupération de mot de passe) :

Pour la table Employee :

- GET : /api/employees
- POST : /api/employees
- GET : /api/employees/{id}
- PUT : /api/employees/{id}
- DELETE : /api/employees/{id}
- GET : /api/employees/byTask/{idTask}
- GET : /api/employees/byEmail/{email}

Pour la table Assignment :

- GET : /api/assignment/{id}
- PUT : /api/assignment/{id}
- DELETE : /api/assignment/{id}
- POST : /api/assignment

Pour la table Holiday :

- GET : /api/Holiday
- POST : /api/Holiday
- GET : /api/Holiday/Holiday/{id}
- GET : /api/Holiday/byEmployee/{idEmployee}
- DELETE : /api/Holiday/{id}
- PUT : /api/Holiday/{id}

Pour la table Task :

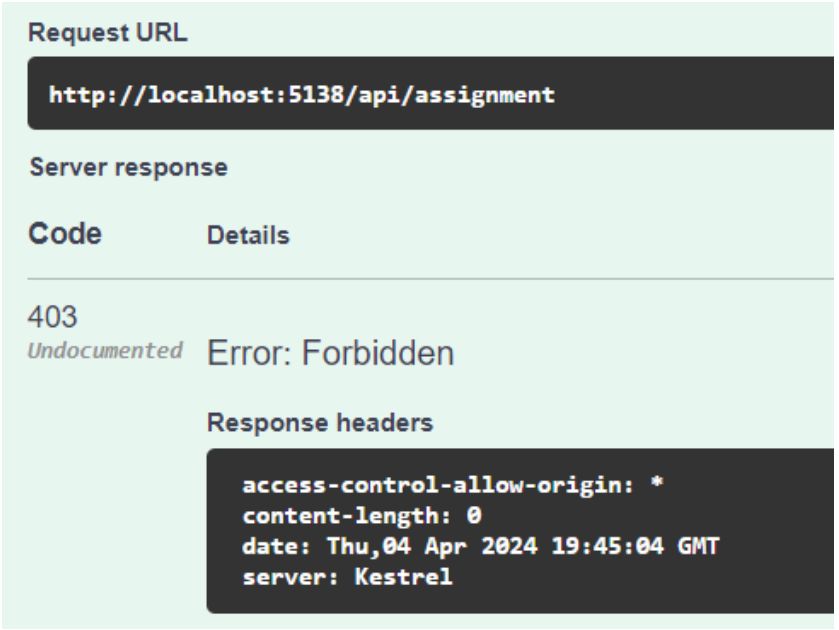
- GET : /api/Tasks
- POST : /api/Tasks
- GET : /api/Tasks/{id}
- PUT : /api/Tasks/{id}
- DELETE : /api/Tasks/{id}
- GET : /api/Tasks/byEmployee/{idEmployee}

Toutes ses méthodes ont été testées à la fois dans le cas nominal et dans le cas d'erreur. Les réponses ont été celles anticipés préalablement et ces tests s'avèrent donc concluants.e"

5. Restrictions de certaines actions aux managers

Certaines des actions précédemment énoncées sont restreintes aux employés dont le statut est "Manager". Cela se traduit par la présence de la ligne de code : `[Authorize(Roles = "Manager")]` au sommet des méthodes que l'on souhaite restreindre.

Ces restrictions ont été testées, les tests dans le cas nominal se sont avérés concluants : l'employé qui a le statut manager peut effectuer les actions réservées à ces derniers. Dans les cas d'erreurs, un employé obtient cette réponse signalant une erreur car l'action est inaccessible de par son statut. Depuis le swagger cela donne :

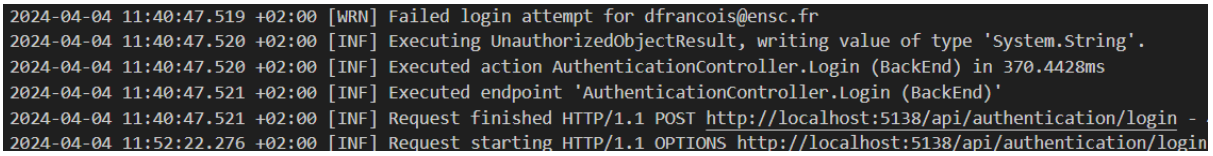


Comme précisé précédemment, les actions des managers ne sont pas accessibles par les employés et les cas d'erreurs ne sont donc pas censés se produire.

6. Journalisation de connexion

La journalisation de surveillance est gérée par la bibliothèque de .NET nommée Serilog. Elle permet d'enregistrer les événements de manière structurée et

lisible. La configuration est faite dans le fichier appsettings.json et permet d'avoir plusieurs niveaux de journalisation, le niveau par défaut ainsi que les détails sur le cycle de vie de l'application sont réglés sur "Information" tandis que les espaces de noms "Microsoft" et "Microsoft.AspNetCore" sont définis comme "Warning". Ce journal recense notamment les exécution de la méthode "/api/Authentication/login". Il est enregistré dans le dossier "logs" du projet, et voici à quoi ressemble un extrait du journal :



On voit bien les différents niveaux d'importance attribués en fonction des événements qui interviennent dans l'utilisation de l'API.

Cette pratique, en fournissant des détails sur le fonctionnement de l'application, facilite la détection précoce des activités suspectes et des problèmes de sécurité. Elle peut aussi permettre de garder une trace de chaque passage, des tentatives infructueuses et des actions menées ce qui peut être intéressant pour une entreprise souhaitant retracer l'origine d'un problème technique ou d'une fuite de données.

7. Rate Limiting

Le rate limiting permet de se prévenir des attaques par force brute ou par dénis de service distribué en limitant le nombre de tentatives de connexion à 5 par minute pour une même adresse IP. Encore une fois, c'est dans le fichier appsettings.json que cette mesure de sécurité est configurée, et cela est fait de manière à ce que le cette limitation ne concerne que la méthode "/api/Authentication/login". Cette méthode n'a pas été testée.

Gestion de projet

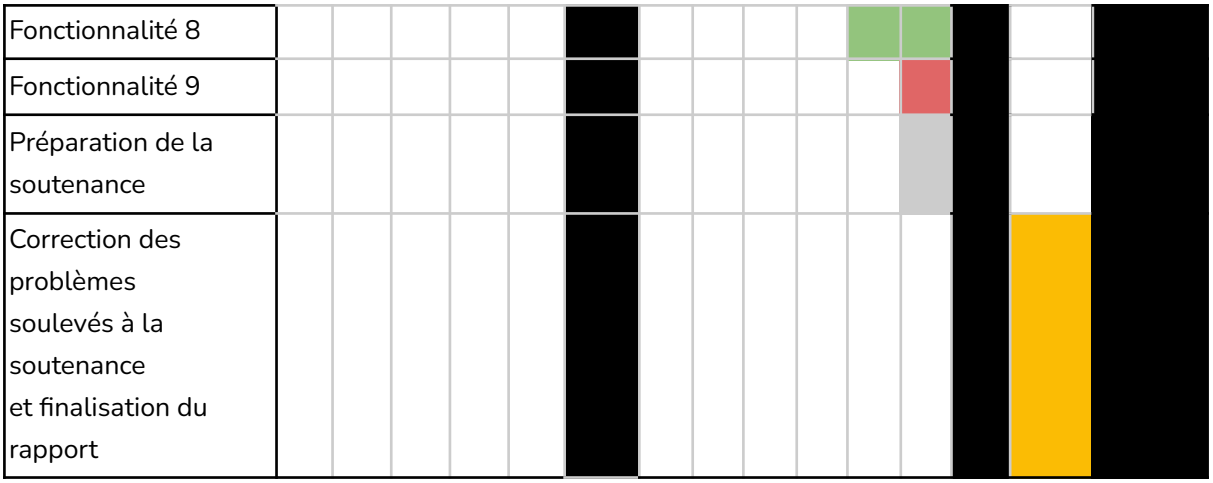
Planning prévisionnel

(Le détail des fonctionnalités est visible dans la partie Fonctionnalités finales).

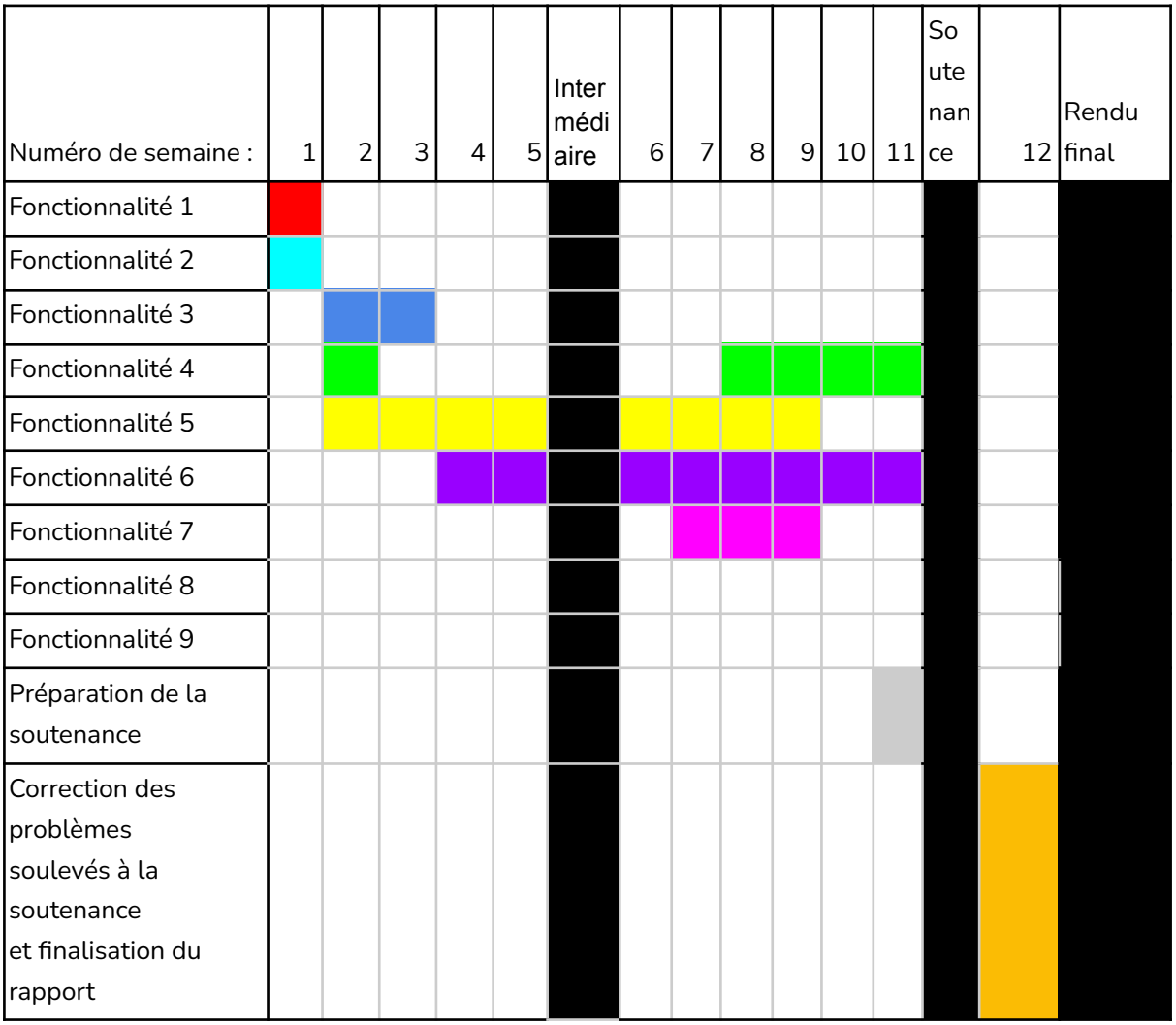
Numéro de semaine :	1	2	3	4	5	Intermédiaire	6	7	8	9	10	11	Soutenance	12	Final
Fonctionnalité 1															
Fonctionnalité 2															
Fonctionnalité 3															
Fonctionnalité 4															
Fonctionnalité 5															
Fonctionnalité 6															
Fonctionnalité 7															
Fonctionnalité 8															
Fonctionnalité 9															
Préparation de la soutenance															
Correction des problèmes soulevés à la soutenance et finalisation du rapport															

Planning intermédiaire

Numéro de semaine :	1	2	3	4	5	Intermédiaire	6	7	8	9	10	11	Soutenance	12	Rendu final
Fonctionnalité 1															
Fonctionnalité 2															
Fonctionnalité 3															
Fonctionnalité 4															
Fonctionnalité 5															
Fonctionnalité 6															
Fonctionnalité 7															



Planning respecté en pratique



Différences entre les plannings

Il y a des différences significatives entre le planning prévisionnel, celui établi lors du rendu intermédiaire et le planning respecté. On remarque déjà que la réinitialisation du mot de passe ainsi que le maquettage de l'application (les fonctionnalités 7 et 8) n'étaient pas présents dans le planning originel. En pratique on peut observer que seule la réinitialisation du mot de passe a été mise en place. Cela s'explique par une mauvaise estimation du temps nécessaire à la réalisation des tâches précédent l'amélioration du design. Mais aussi par les difficultés rencontrées qui sont détaillées dans la partie qui suit.

Difficultés rencontrées

Deux difficultés majeures ont été rencontrées durant la réalisation du projet : un problème de Cross-Origin Resource Sharing (CORS) policy et une difficulté dans la mise en place des tokens JWT créés lors de l'authentification.

Le premier problème est intervenu lorsque j'ai essayé d'exécuter mes premières actions depuis mon projet front-end. Cela m'a beaucoup retardé dans ma progression, rajoutant un temps de recherches sur le sujet.

Le second problème est intervenu un peu plus tard que le premier et a aussi demandé une importante phase de recherche avec de multiples tentatives infructueuses dont la création du fichier de code middleware censé valider le token. Finalement cette étape se fait dans le fichier Program.cs avec une configuration du jeton et de sa clé dans appsettings.json et une génération dans AuthenticationController.cs au moment de la connexion.

Gestion de projet générale

Le projet étant réalisé en autonomie, il était possible de négliger la gestion de projet. C'est ce que j'ai fait au début avant de me rattraper en notant chaque semaine mon avancée exacte ainsi qu'une liste de ce qui devait être fait (en accord avec mes plannings) pour la semaine suivante.

Deux réunions et des échanges d'emails ont été mis en place avec la tutrice de mon projet qui m'a apporté de précieux conseils.

J’ai surtout travaillé sur mon projet pendant les créneaux qui étaient accordés au P2I, même si j’ai aussi dû travailler activement dans mon temps libre.

Retour critique

Pistes d’amélioration

L’application n’est pas complètement fonctionnelle et la partie front-end n’est pas terminée. Finaliser cette étape serait la première chose à faire en cas de continuité du projet. Il manque certaines fonctionnalités qui ne sont pas fonctionnelles (voir les fonctionnalités de l’application) ainsi qu’une gestion des erreurs plus complète.

Progrès et enseignements tirés

Ce projet a représenté une belle opportunité pour moi d’approfondir un domaine que j’avais particulièrement apprécié qui est la programmation d’API et la programmation back-end de manière générale. Il a également été l’occasion de combler quelques lacunes dans le langage Javascript. Je suis très reconnaissant de la liberté qui nous a été laissée quant au choix du sujet et du domaine cadrant le projet. J’ai pris du plaisir à le réaliser, tout en apprenant beaucoup de choses tant sur mon projet qu'en m'intéressant à ceux des autres.

Guide d’installation

1. Commencez par extraire le contenu du fichier zip dans lequel se trouve ce rapport.
2. Ouvrez le dossier alors créé dans Visual Studio Code.
3. Ouvrez un terminal et exécutez la commande suivante : cd .\BackEnd\
4. Exécutez ensuite cette commande : dotnet build
5. Exécutez ensuite cette commande : dotnet run
6. Ouvrez un nouveau terminal et exécutez la commande suivante : cd..
7. Exécutez ensuite cette commande : cd .\gestion-taches-vue\
8. Exécutez ensuite cette commande : npm install
9. Exécutez ensuite cette commande : npm run serve
10. L’application est à l’adresse suivante : http://localhost:8080/login

11. Le swagger est à l’adresse suivante : http://localhost:5138/swagger/index.html

Si jamais vous n’y parvenez pas, le lien du github de mon projet est : https://github.com/clementgoy/GestionTaches.