



Haute Ecole Economique et Technique

AVENUE DU CISEAU, 15
1348 OTTIGNIES-LOUVAIN-LA-NEUVE

**Développement d'un progiciel
de gestion intégré
pour l'entreprise Master Services**

Travail de fin d'études présenté en vue de l'obtention du diplôme de bachelier en Informatique
et Systèmes orientation Technologie de l'Informatique

JAUJATE OULDKHALA Ikram

Table des matières

1	Introduction	3
1.1	Contexte général	3
1.1.1	Client	3
1.2	Objectifs	3
1.3	Cadre didactique de la réalisation	3
2	Méthode de travail	4
2.1	Méthodologie	4
2.1.1	Outils	4
2.1.2	Gitflow	4
2.2	Choix des technologies	5
2.2.1	Frontend	5
2.2.2	Web Server	6
2.2.3	Backend	6
2.2.4	Database	7
2.3	Autres	7
2.3.1	Documentation API	7
2.3.2	Linter	8
3	Analyse et réflexions	9
3.1	User Stories	9
3.1.0.1	Exemple	9
3.2	Liste des fonctionnalités développées	10
4	Déploiement	12
4.1	Études des différentes solutions	12
4.1.1	Heroku	12
4.1.2	Serveurs dédiés OVH	12
4.1.3	Justification du choix	12
4.2	Intégration Continue et Déploiement Continu	12
4.2.1	Docker	12
4.2.2	Scripts	12
4.2.3	Github workflow	12
5	Testing	13
5.1	Unitaires	13
5.2	Integrations	13
5.3	End-to-End	13
6	Optimisation	14
6.1	Cache	14
6.2	Caching Backend	14
6.2.1	Analyse des résultats	14
6.2.1.1	Sans Redis	14
6.2.1.2	Avec Redis	15
6.3	Caching Frontend	15
6.3.1	IndexedDB	15
6.3.2	Implémentation	16
6.3.3	Difficultés rencontrées	16

7	Sécurité	18
7.1	Frontend	18
7.1.1	Routage	18
7.2	Backend	18
7.2.1	Routage API	18
7.2.2	Base de données	18
7.2.3	En-têtes HTTPS	18
7.3	Web Server	18
7.3.1	Configuration de base	18
7.3.2	Firewall	18
7.3.3	Reverse-proxy	18
7.4	Autres	18
7.4.1	Libraries	18
8	Monitoring	19
9	Conclusion	20
10	Bibliographie	21

1 Introduction

1.1 Contexte général

1.1.1 Client

Le client, une entreprise active dans le secteur du bâtiment des travaux publics, situé à Sint-Pieters-Leeuw doit constamment gérer, dans le cadre de ses activités, le flux de matériaux, ses clients, ses employés, ses fournisseurs ainsi que toutes les facturations qui en découlent.

Il ne possède actuellement aucun moyen informatisé lui permettant de gérer l'ensemble de ses entités. Il m'a été demandé de concevoir une solution pour répondre à l'ensemble de ses besoins :

- Gestion clients
- Gestion des Projets
- Gestion Matériel
- Gestion Stock
- Gestion Main d'Oeuvre
- Gestion du personnel
- Gestion utilisateurs
- Gestion Factures
- Gestion Devis

1.2 Objectifs

Il a donc été décidé, en accord avec le client, que la solution sera déployée de telle sorte qu'elle puisse être adaptée au fil des années en fonction de l'évolution des besoins du client. De plus, le client étant souvent sur chantier, il est primordial que l'application puisse être facilement portable d'un système / ordinateur à un autre. Il a donc été décidé en commun accord avec le client que la solution sera déployée sous forme d'une application web ce qui apportera une grande flexibilité au niveau de l'utilisation de la solution.

Le volume d'information à gérer étant conséquent, la visualisation doit principalement être adaptée **aux écrans d'ordinateur.**

1.3 Cadre didactique de la réalisation

Je pense que ce projet rentre tout à fait dans le cadre d'un TFE étant donné qu'il requiert l'analyse, le développement, le déploiement et la maintenance d'une application web afin d'apporter une solution adéquate répondant aux spécificités d'un client. Le développement se fera à l'aide de technologies modernes qui me permettront d'offrir une interface au goût du jour et modulable. Afin d'améliorer la productivité, je ferais usage de multiples techniques DevOps telles que le déploiement continu et bien d'autres.

2 Méthode de travail

2.1 Méthodologie

Vu l'envergure du projet et des tâches techniques à accomplir, l'approche sélectionnée pour la réalisation de ce projet est une organisation en mode Agile. Plus spécifiquement, j'ai décidé de travailler avec le cadre de travail Scrum et son organisation en sprints.

Tout d'abord, j'ai pu définir avec le client les principales fonctionnalités à intégrer dans le projet. Ces fonctionnalités sont classées par ordre de priorité et par temps estimé à la réalisation de celles-ci.

En outre, les principales fonctionnalités contenues dans le projet sont détaillées. S'agissant d'un projet de grande envergure, chacune des principales fonctionnalités a été divisée en petites user stories qui me permettront d'avoir un produit livrable au client à la fin de chaque sprint. Les sprints auront **une durée d'environ 2 semaines**.

À la fin de chaque sprint, un livrable correspondant aux tâches / user stories effectuées lors du sprint devra être présenté au client. Ce dernier devra alors vérifier et valider les différentes tâches effectuées.

L'avantage de cette organisation est que, en cas d'erreurs et/ou non validation des tâches de la part du client, ces dernières pourront être revues et corrigées pour le prochain sprint.

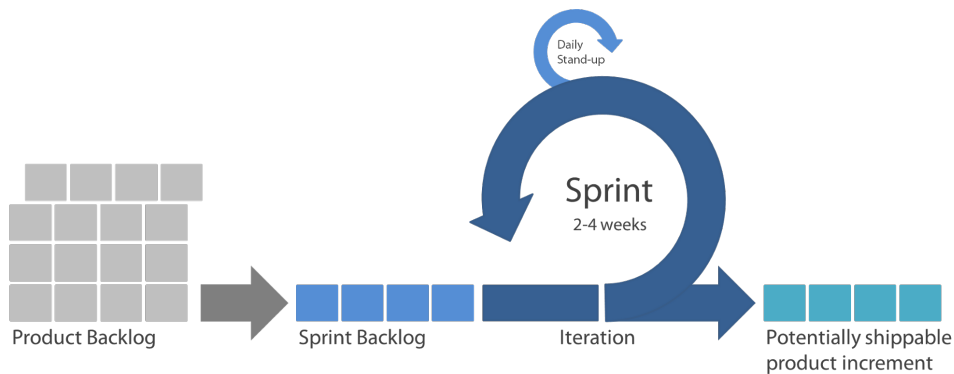


FIGURE 1 – *Cadre de travail Scrum* de Anna Pérez

2.1.1 Outils

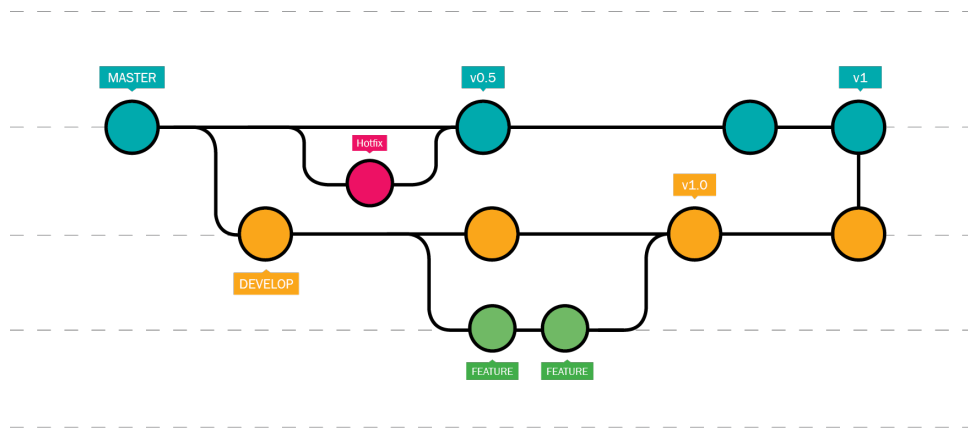
Afin d'optimiser les tâches à effectuer pour chaque sprint et d'améliorer ma performance de travail, plusieurs outils ont été utilisés dans l'objectif de faciliter la visualisation de l'avancement du projet.

- **Trello** : Tableau permettant d'organiser les différentes tâches et user stories de manière optimale.
- **Clockify** : Outil de suivi des heures travaillées sur le projet.
- **SqlDBM** : Outil permettant la création de mon schéma de base de données.

2.1.2 Gitflow

J'ai décidé de travailler avec le gitflow par branche, ce qui permet d'avoir une division au niveau des fonctionnalités qui seront implémentées lors du projet.

- Une branche 'develop' qui correspond à la branche 'master' du 'github-flow'
- Chaque fonctionnalité est développée sur une branche spécifique et une fois celle-ci est validée par le client, alors elle est fusionnée avec la branche develop.



2.2 Choix des technologies

En ce qui concerne la technologie frontend, React permet de créer des interfaces utilisateur ou des composants d'interface utilisateur rapides et interactifs pour les utilisateurs d'applications web et mobiles.

1. **DOM Virtuel** : Il permet de générer le DOM ("Document Object Model", structure des éléments qui sont générés dans le navigateur web lors du chargement d'une page) de manière dynamique, ce qui nous permet de visualiser les changements de données sans devoir recharger à nouveau la page entière, mais seulement le composant qui a été mis à jour.
2. **Grande communauté** : Il est soutenu par une large communauté, ce qui nous permet d'avoir un grand nombre de libraires disponibles.
3. **Composants réutilisables** : React est constitué de composants qui sont réutilisables, ce qui rend l'application plus portable et plus facile à maintenir.

2.2.2 Web Server

Au niveau du serveur web, mon choix se porte sur Caddy Server et ce pour plusieurs raisons



1. **Simple** : Possède une configuration très simple qui nous permettra de le mettre en place en quelques minutes.
2. **HTTPS par défaut** : Utilise Let's Encrypt pour mettre le site en HTTPS complet automatiquement, sans aucune configuration et le renouvellement des certificats SSL/TLS se réalise de manière automatique.
3. **Multiplateforme** : Il est multiplateforme et je serai en mesure d'exécuter Caddy directement par le biais de Docker, ce qui rendra sa mise en œuvre encore plus facile.

2.2.3 Backend

Comme pour le frontend, le choix du backend est également essentiel. Dans ce cas, j'ai décidé de travailler avec Node.js et ce pour plusieurs raisons.



1. **Très rapide** : Les tâches courantes comme la lecture ou l'écriture dans la base de données sont exécutées rapidement et il est capable de gérer des connexions simultanées à haut débit.
2. **Grande communauté** : Il est soutenu par une large communauté, ce qui nous permet d'avoir un grand nombre de librairies disponibles.
3. **MVC** : Permet de travailler en MVC, ce qui permet une structure correcte du code.
4. **Asynchrone** : Étant un système asynchrone, il permet d'accélérer les applications web. Il est capable d'envoyer gros volumes de données sans bloquer le serveur qui reste ainsi disponible pour traiter d'autres tâches.
5. **compatible** : Permet un développement multiplateforme qui est axé sur tous les types d'appareils et de plateformes d'OS (iOS, Android, desktop et web). Le code est réutilisable et entièrement compatible avec tous les principaux systèmes d'exploitation, notamment Linux, Windows, ainsi que macOS, ce qui va nous permettre de rendre notre Web Application accessible depuis toutes les plateformes.

2.2.4 Database

Mon choix pour la base de données est PostgreSQL pour plusieurs raisons :



1. **DB Relationnelle** : Puisque les données doivent être ordonnées et structurées et que des relations doivent exister entre les différentes données, il est essentiel d'utiliser une base de données SQL afin de garantir l'organisation de ces dernières.
2. **SQL** : PostgreSQL utilise le langage SQL, qui est le langage le plus utilisé pour les bases de données relationnelles.
3. **Compatible** : PostgreSQL est entièrement compatible ACID. ACID est un acronyme pour Atomicité, Cohérence, Isolation et Durabilité. Il garantit donc que les transactions n'interfèrent pas entre elles. Cela garantit les informations contenues dans les bases de données et la pérennité des données dans le système.
4. **Hot-Standby** : Il dispose de l'option Hot-Standby qui permet aux utilisateurs d'accéder aux tables en mode lecture pendant que les processus de sauvegarde ou de maintenance sont en cours.

PostgreSQL jouit d'une solide réputation en matière de fiabilité, de robustesse des fonctionnalités et des performances.

2.3 Autres

2.3.1 Documentation API

L'API (*Application Programming Interfaces*) est le backbone¹ d'une application. Elle permet l'interaction d'un utilisateur ou d'une application avec le système d'information sur base d'endpoints. Pour pouvoir utiliser l'API de manière efficace, il est nécessaire d'avoir une bonne documentation. Cette documentation doit pouvoir exprimer de manière claire l'action de chaque endpoint sur le système d'information. Dès lors on y retrouvera les méthodes telles que GET, CREATE, UPDATE, DELETE, les informations relatives aux données nécessaires pour le bon déroulement de la requête ainsi qu'un exemple et/ou le format du résultat attendu.

1. *Élément structurelle indispensable au bon fonctionnement de l'application. Ceci représente le coeur de l'application.*

Pour pouvoir réaliser cette documentation, il existe un grand nombre de technologies qui permettent de la créer et de la générer automatiquement. Dans mon cas, j'ai décidé d'utiliser **Swagger**. Celui-ci s'intègre parfaitement avec NodeJS / ExpressJS, il permet d'autogénérer en grande partie la documentation sur base de mes endpoints et il est open-source. La documentation générée par Swagger est facile à comprendre et permet de tester directement l'API sans devoir passer par un outil supplémentaire tel que Postman.

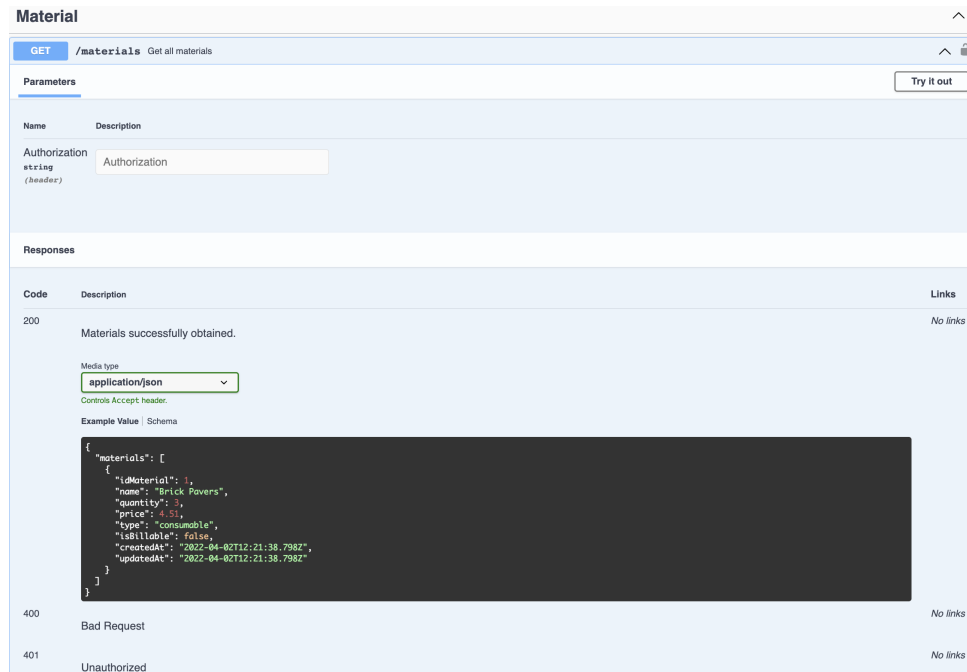


FIGURE 3 – Documentation API avec Swagger

2.3.2 Linter

Dans tout projet, il est inévitable d'avoir des erreurs de syntaxe, des styles de code et bien d'autres problèmes. De plus, avoir un code parfaitement cohérent et lisible est très compliqué. C'est pourquoi l'une des solutions est l'utilisation d'un Linter. Cet outil effectue un check du code source de l'application sans jamais l'exécuter.

Une fois la révision terminée, le Linter nous montre les erreurs de syntaxe, corrige automatiquement l'indentation si cette option a été configurée et fait également des suggestions visant à améliorer le code.

Considérant que c'est une partie essentielle de mon projet, j'ai décidé d'utiliser le Linter et plus précisément ESLint.

3 Analyse et réflexions

3.1 User Stories

3.1.0.1 Exemple

CLI-1: Consultation client

 ADMIN, SEC

En tant qu'utilisateur je souhaite pouvoir consulter la liste des clients actuels sous forme d'une tableau afin de pouvoir rapidement consulter les informations de ceux-ci.

Préconditions

- **Technical:**
 - La table `Person` doit exister dans la DB.
 - La table `Company` doit exister dans la DB.
 - La table `Address` doit exister dans la DB.
- **Logique:**
 - /

Détail

L'onglet Client redirige l'utilisateur vers cette page. Celle-ci reprend l'ensemble des clients récupérés depuis la base de données (voir requête API) sous forme d'un tableau. Ce tableau affichera les informations suivante :

- `DisplayName`
- `E-mail`
- `Phone`
- `TVA number`

Dans le tableau, l'utilisateur peut distinguer si le client est une personne physique ou une société. Le tableau doit contenir un système de pagination. L'utilisateur doit pouvoir afficher entre 10/20 éléments par page.

Critères de validation

- ☐ L'utilisateur peut consulter la table reprenant tous les clients
- ☐ Chaque ligne ne correspond qu'un un seul client
- ☐ Si aucun client n'existe, l'utilisateur verra un message indiquant qu'aucun client n'a été trouvé


API

Voici un exemple de l'API pour cette US :

```
method : GET
url : /api/client?show=0-10
```

Récupération de toutes les informations personnelles sur tous les clients disponibles dans la base de données

 En attendant les données, un spinner est affiché.

 Si la requête est correcte, alors les données seront affichés

 Si la requête est erronée, alors un **message d'erreur** sera affiché en haut à droite de l'écran.

S'il existe plus de 10 clients, des petites flèches en dessous du tableau permettent de charger les 10 clients suivants.

Points d'attention

Si aucun client ne se trouve dans la base de données, un message indique qu'aucun client n'a été trouvé.

 En attendant que les données chargent , un spinner est affiché.

FIGURE 4 – Exemple de description de User Story

3.2 Liste des fonctionnalités développées

Toutes les fonctionnalités prévues pour le projet ont été réalisées. Ci-dessous, vous pouvez voir une vue non-détaillée des fonctionnalités implémentées.

Utilisateur

- ✓ U01 - Connexion utilisateur
- ✓ U02 - Consulter les utilisateurs
- ✓ U03 - Ajouter un nouveau utilisateur
- ✓ U04 - Modifier utilisateur
- ✓ U05 - Supprimer utilisateur

Clients

- ✓ CLI-1 : Consultation la listes des clients
- ✓ CLI-2 : Consulter le détail d'un client
- ✓ CLI-3 : Ajouter un nouveau client
- ✓ CLI-4 : Modification d'un client
- ✓ CLI-5 : Supprimer un client
- ✓ CLI-6 : Rechercher un client
- ✓ CLI-7 : Trier la liste des clients

Projets

- ✓ PRJ-1 : Consulter la listes des projets
- ✓ PRJ-2 : Consulter le détail d'un projet
- ✓ PRJ-3 : Rechercher un projet
- ✓ PRJ-4 : Filtrer un projet
- ✓ PRJ-5 : Ajouter d'un projet
- ✓ PRJ-6 : Modifier le contenu d'un projet
- ✓ PRJ-7 : Supprimer un projet

Matériel

- ✓ MA-1 : Consulter la liste de matériel disponible
- ✓ MA-2 : Consulter le détail d'un matériel
- ✓ MA-3 : Ajouter d'un matériel à un projet
- ✓ MA-4 : Rechercher d'un matériel
- ✓ MA-5 : Désaffecter un matériau d'un projet
- ✓ MA-6 : Modifier un matériel
- ✓ MA-7 : Supprimer du matériel

Devis

- ✓ DE-1 : Consultation devis
- ✓ DE-2 : Générer un devis
- ✓ DE-3 : Consulter le détail d'un devis

- ✓ DE-4 : Consulter les devis appartenant à un projet
- ✓ DE-5 : Envoyer devis par mail
- ✓ DE-6 : Recherche un devis
- ✓ DE-7 : Trier la liste des devis

Factures

- ✓ FA-1 : Consultation facture
- ✓ FA-2 : Générer une facture
- ✓ FA-3 : Consulter détail d'une facture
- ✓ FA-4 : Consulter les factures appartenant à un projet
- ✓ FA-5 : Envoyer une facture par mail
- ✓ FA-6 : Télécharger facture
- ✓ FA-7 : Recherche une facture
- ✓ FA-8 : Trier la liste des factures

4 Déploiement

4.1 Études des différentes solutions

4.1.1 Heroku

4.1.2 Serveurs dédiés OVH

4.1.3 Justification du choix

4.2 Intégration Continue et Déploiement Continu

4.2.1 Docker

Dans le but de rendre l'application portable, facilement configurable et cross-platform, j'ai décidé de dockeriser l'ensemble des composants constituant l'application, c'est à dire :

- **masterservices-caddy** : Reverse-proxy
- **masterservices-app** : Web Server utilisant le port **8081**
- **masterservices-postgres** : Base de données utilisant le port **66453**
- **masterservices-redis** : Serveur pour gérer l'optimisation de l'application. Utilise le port **6379**

Le fonctionnement de chaque conteneur docker dépend des autres. Dans le cadre du projet, l'application sera fonctionnelle si la base de données et le serveur Redis sont opérationnels. Si l'un de ces deux serveurs fait défaut, l'application ne sera pas en mesure de fonctionner. Afin d'orchestrer les différents conteneurs, l'utilisation d'un docker-compose a été essentielle pour pouvoir, en une seule commande, créer et démarrer tous les services.

Le seuls ports exposés à l'exterieur sont le port **80** et **443**, utilisé par le serveur caddy, ce qui est un premier pas de sécurisation de l'application.

4.2.2 Scripts

4.2.3 Github workflow

5 Testing

5.1 Unitaires

5.2 Integrations

5.3 End-to-End

6 Optimisation

6.1 Cache

Le cache est une zone de stockage dans laquelle une quantité d'information peut être sauvegardée afin de la rendre accessible et réutilisable très rapidement. En d'autres termes, elle nous permet de réutiliser efficacement les informations qui ont été préalablement stockées. Pour améliorer le temps de latence de l'application, deux type de caching ont été mis en place.

6.2 Caching Backend

Le caching backend a été implémenté avec l'aide de Redis. Avant toute chose, il est nécessaire de comprendre ce qu'est Redis.



Redis est une base de données NoSQL mais elle n'est pas très similaire aux autres bases de données NoSQL. Cela est dû au fait que Redis ne fonctionne pas vraiment avec des tables, mais toutes les données dans Redis sont stockées dans des paires clé-valeur. Il y a donc une clé ayant un nom et une valeur appelée Kyle. L'objectif n'est donc pas de stocker un ensemble de données structurées, mais simplement de stocker une paire clé-valeur individuelle à partir de laquelle nous pouvons accéder aux données. Il est important de noter que Redis fonctionne en utilisant la mémoire RAM de la machine sur laquelle il tourne, cela signifie qu'il est extrêmement rapide.

Dans le cas où nous devons accéder à des informations qui prennent beaucoup de temps à charger, par exemple une quantité de données assez importante provenant de la base de données, nous pouvons utiliser Redis pour stocker ces valeurs pendant une durée définie. Ainsi, lorsque nous devons accéder à cette même information, la réponse sera beaucoup plus rapide puisque l'information sera déjà chargée.

6.2.1 Analyse des résultats

Le temps de chargement des données sur certaines de mes endpoints était assez important et je devais trouver une solution pour le réduire. Afin d'améliorer le temps de chargement des données, j'ai décidé d'utiliser Redis. Pour tester Redis, j'ai décidé de récupérer tous les clients se trouvant dans ma base de données, ce qui correspond à, environ, 1400 clients.

6.2.1.1 Sans Redis

Dans la figure ci-dessous, nous pouvons voir que le temps de réponse de la requête réalisée sans Redis a tendance à diminuer un peu après la première requête. Ceci est dû à la gestion du cache de Sequelize et de Postgres. Bien que ce temps ait diminué, la moyenne de ces temps est encore élevée, elle correspond à 495 ms, une demi-seconde.

Name	Status	Type	Initiator	Size	Time
<input type="checkbox"/> persons	200	fetch		949 kB	952 ms
<input type="checkbox"/> persons	200	fetch		949 kB	374 ms
<input type="checkbox"/> persons	200	fetch		949 kB	417 ms
<input type="checkbox"/> persons	200	fetch		949 kB	301 ms
<input type="checkbox"/> persons	200	fetch	<u>users...</u>	949 kB	433 ms

FIGURE 5 – Temps de réponse sans Redis

6.2.1.2 Avec Redis

La même requête a été réalisée mais cette fois ci avec Redis. Dans la figure 5, la première requête, sans Redis, dure une seconde, ce qui est assez conséquent. Les requêtes suivantes utilisant Redis diminuent considérablement en obtenant une moyenne de 204ms.

Name	Status	Type	Initiator	Size	Time
<input type="checkbox"/> persons	200	fetch		949 kB	1.06 s
<input type="checkbox"/> persons	200	fetch		949 kB	211 ms
<input type="checkbox"/> persons	200	fetch		949 kB	185 ms
<input type="checkbox"/> persons	200	fetch		949 kB	178 ms
<input type="checkbox"/> persons	200	fetch		949 kB	232 ms
<input type="checkbox"/> persons	200	fetch	<u>users...</u>	949 kB	217 ms

FIGURE 6 – Temps de réponse avec Redis

La performance obtenue en utilisant redis a augmenté de 58,8% ce qui justifie son utilisation.

6.3 Caching Frontend

Il est bien connu que les navigateurs peuvent stockées des informations relatives à la consultation de sites webs. L'application web va utiliser cette fonctionnalité à son avantage. Plusieurs moyens de stockage existent, tels que le local storage, session storage, IndexedDB, et d'autres, tous avec leurs avantages et inconvénients. Après une analyse approfondi de ces différentes solutions, j'ai décidé d'opter l'IndexedDB.

6.3.1 IndexedDB

L'IndexedDB est un système de base de données qui permet de sauvegarder des informations de manière structurée dans le navigateur. Par exemple, lorsqu'il y a une quantité importante d'informations qu'il faut aller rechercher, comme des matériaux, ces données peuvent être stockées à cet endroit ci.

6.3.2 Implémentation

La récupération de données depuis l'API ne se fait plus systématiquement. L'application web va appliquer la logique expliquée dans la figure suivante.

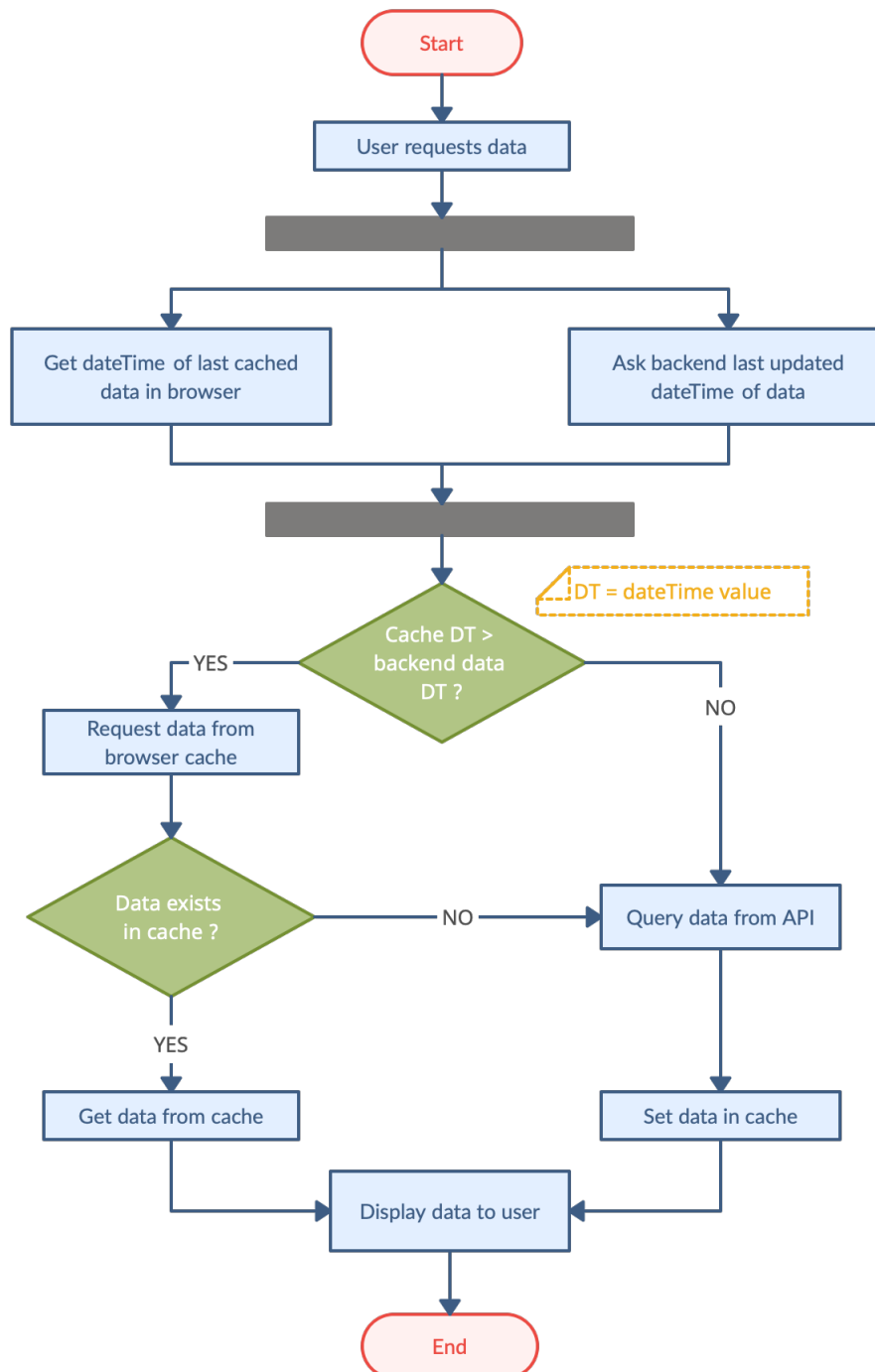


FIGURE 7 – Flux de récupération de données

Cette technique permet de réduire considérablement le temps d'accès aux données de l'application.

6.3.3 Difficultés rencontrées

Initialement, le backend renvoyait uniquement un status indiquant si les données d'une certaine table avaient été modifiées préalablement ou non. Ce status était remis à l'état initial (c.à.d. "non

modifiée") au moment d'une requête de type GET. Étant donnée qu'au frontend, après modification d'une donnée, l'ensemble des données mise à jours sont récupérées du backend, uniquement la personne ayant modifiée les données se verra recevoir le status "modifiée".

Les autres utilisateurs de l'application ne recevaient dès lors jamais les modifications.

Pour palier à ce problème, le backend renvoie dès à present la dernière date de modification des données de la table. Le frontend compare ensuite cette date à la date de création du cache pour ces données là et si la date de création du cache est inférieure à la dernière date de mise à jour de données, il ira rechercher les données au backend.

7 Sécurité

7.1 Frontend

7.1.1 Routage

7.2 Backend

7.2.1 Routage API

7.2.2 Base de données

7.2.3 En-têtes HTTPS

7.3 Web Server

7.3.1 Configuration de base

Lors de la location d'un VPS au près d'un fournisseur, il est essentiel de le configurer. Dans un premier temps, j'ai créer un utilisateur dédié pour mon application et supprimer l'utilisateur reçu lors de la location. Par la suite, j'ai reconfigurer le service SSH afin de interdire la connexion par mot de passe, d'interdire la connexion en tant que root et d'autoriser la connexion par clé publique/privé.

De plus, afin de sécuriser le VPS contre les attaques par brute forces sur le port part défaut de SSH (port 22), j'ai reconfigurer celui-ci sur le port 3333.

7.3.2 Firewall

7.3.3 Reverse-proxy

7.4 Autres

7.4.1 Libraries

Un projet de cette envergure nécessite l'installation de plusieurs dépendances. Lorsque qu'une vulnérabilité a été découverte au niveau d'une des dépendances, il faut absolument réaliser une mise à jour dans les plus brefs délais. C'est pourquoi l'utilisation d'un outil tel que **Dependabot** est nécessaire.

Dependabot est un outil intégré à github qui analyse de manière régulière les dépendances utilisées dans le projet afin de détecter d'éventuelles vulnérabilités. S'il en trouve une, cet outil envoie une alerte avec le niveau de danger qu'une la dépendance représente. Mais le plus intéressant, c'est qu'en plus des alertes, Dependabot crée une pull-request qu'il suffit d'accepter pour maintenir la dépendance à jour.

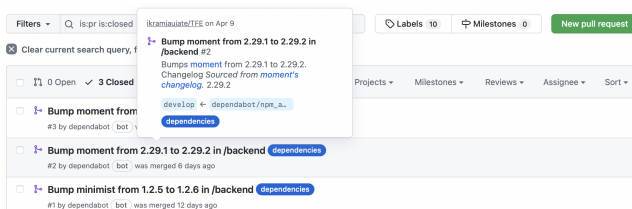


FIGURE 8 – Pull-Request créée par Dependabot

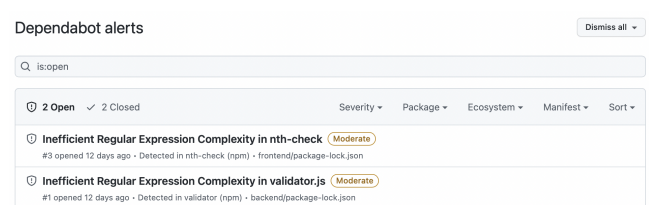


FIGURE 9 – Alertes générées par Dependabot

8 Monitoring

Il se peut qu'au cours du déploiement d'un site web, celui-ci subisse un temps d'arrêt. Cela signifie que si quelqu'un veut visiter le site, il ne pourra pas le faire. La mise en place d'un outil de monitoring est une tâche primordiale. L'outil de monitoring choisit pour ce projet est UpTimeRobot. Il permet de vérifier rapidement si les services sont opérationnels. De plus, cet outil réalise des vérifications de l'état des différents services à des intervalles réguliers. Pour ce projet, l'intervalle choisi est de 5 minutes étant donné que le site web ne devrait, en aucun cas, subir des interruptions.

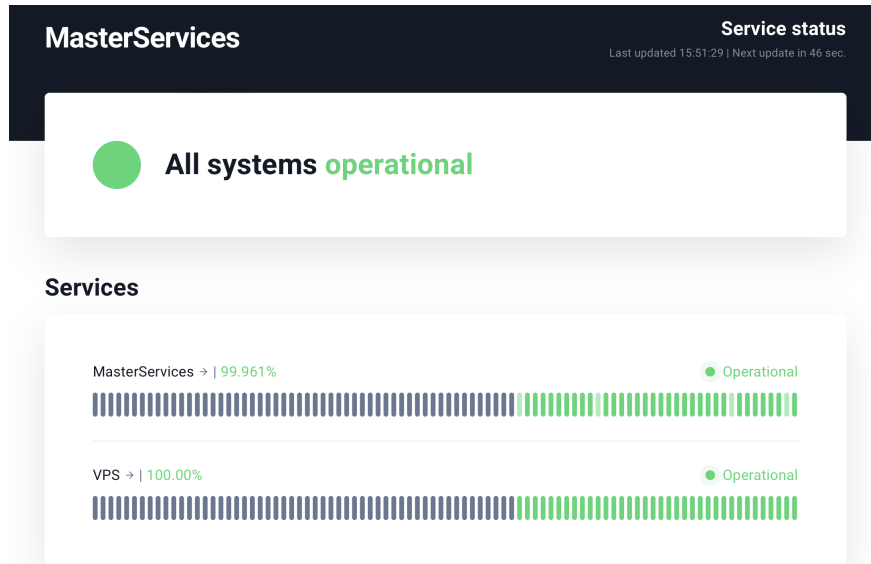


FIGURE 10 – Monitoring avec l'outil UpTimeRobot

9 Conclusion

10 Bibliographie

- [1] Smashing Magazine (2020, 2 mai), sur le site *Implementing Dark Mode In React Apps Using styled-components* Consulté le 20 janvier 2022
<https://www.smashingmagazine.com/2020/04/dark-mode-react-apps-styled-components/>
- [2] Atlassian (2021, 22 mars), sur le site *Beautiful and accessible drag and drop for lists with React* Consulté le 22 janvier 2022
<https://github.com/atlassian/react-beautiful-dnd>
- [3] User Story (2019), sur le site *User story - Wikipedia* Consulté le 24 novembre 2021
https://en.wikipedia.org/wiki/User_story
- [4] Stack Overflow (2020, 10 décembre), sur le site *Upgrading Node.js to latest version* Consulté le 22 décembre 2021
<https://stackoverflow.com/questions/10075990/upgrading-node-js-to-latest-version>
- [5] Tang, R. (2020, 10 décembre), sur le site *How to install Node JS and NPM* Consulté le 14 décembre 2021
<https://www.makersupplies.sg/blogs/tutorials/how-to-install-node-js-and-npm-on-the-raspberry->
- [6] Terzi, R.(2018, 16 novembre), sur le site *Qu'est-ce que l'approche CI/CD ?* Consulté le 22 janvier 2022
<https://www.redhat.com/fr/topics/devops/what-is-ci-cd>
- [7] Shadow-M-P (14 mai 2021), sur le site *Méthode agile — Wikipédia* Consulté le 23 janvier 2022
https://fr.wikipedia.org/wiki/Méthode_agile
- [8] PostgreSQL Tutorial (sans date), sur le site *PostgreSQL Transactions* Consulté le 3 décembre 2021
<https://www.postgresqltutorial.com/postgresql-transaction/>
- [9] PostgreSQL Tutorial (sans date), sur le site *PL/pgSQL For Loop* Consulté le 14 décembre 2021
<https://www.postgresqltutorial.com/plpgsql-for-loop/>
- [10] PostgreSQL Tutorial (22 mai 2021), sur le site *PostgreSQL CREATE PROCEDURE* Consulté le 3 janvier 2022
<https://developer.mozilla.org/fr/docs/Web/HTTP/Headers/X-Frame-Options>
- [11] Großgarten, G. (2021, 21 janvier), sur le site *Copy a folder to a remote server using SSH* Consulté le 10 novembre 2021
<https://github.com/garygrossgarten/github-action-scp>
- [12] OVH (sans date), sur le site *Qu'est-ce qu'un VPS ? Découvrez les avantages d'un VPS | OVHcloud* Consulté le 25 novembre 2021
<https://www.ovhcloud.com/fr/vps/definition/#:~:text=Grâce%20au%20VPS%2C%20vous%20profitez,en%20payant%20le%20juste%20prix.>
- [13] Adaobi, A. (2021, 27 octobre), sur le site *Making HTTP Requests in Node.js with node-fetch* Consulté le 25 novembre 2021
<https://stackabuse.com/making-http-requests-in-node-js-with-node-fetch/>
- [14] Laurent Hercé (2020, septembre 15), sur le site *Conteneurisation informatique : définition, avantages, différence virtualisation, solutions | Appvizer* Consulté le 26 décembre 2021
<https://www.appvizer.fr/magazine/services-informatiques/virtualisation/conteneurisation-informatique>

- [15] Stack Overflow (2011, 3 juillet), sur le site *Can I use return value of INSERT. . . RETURNING in another INSERT?* Consulté le 26 décembre 2021
<https://stackoverflow.com/questions/6560447/can-i-use-return-value-of-insert-returning-in-another-insert>