



Haute Ecole Economique et Technique

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

**Application web
de gestion de la supply-chain
pour la société SLG Classic Cars
(restauration de véhicules anciens)**

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

MICHOTTE Martin

Remerciements

Table des matières

1	Introduction	3
1.1	Contexte	3
1.1.1	Client	3
1.1.2	Solution existante	3
1.2	Objectifs	5
1.2.1	Court terme	5
1.2.2	Long terme	5
2	Recueil d'information et Analyse	6
2.1	User-stories	7
2.1.1	Exemple	8
2.1.2	Automatisation Trello	9
2.2	Base de donnée	9
3	Méthodologie	11
3.1	Agile	11
3.2	Choix des technologies	12
3.2.1	Backend	12
3.2.2	Frontend	12
3.2.3	Base de donnée	13
3.2.4	Autre	13
3.2.4.1	API	13
3.2.4.2	Linter	14
3.3	Outils	14
4	Développement	15
4.1	Exemple concret	15
4.2	Récapitulatif des fonctionnalités/US implémentées	17
5	Sécurité	19
5.1	Contrôle d'accès	19
5.2	Hachage et Chiffrement	20
5.2.1	Données enregistrées	20
5.2.2	HTTPS	20
5.3	HTTP Headers	21
5.4	Dependabot Github	22
6	Déploiement	23
6.1	Étude de marché	23
6.1.1	Avant-propos	23
6.1.2	Étude des solutions	24
6.1.2.1	Serveur Local	24
6.1.2.2	Heroku	25
6.1.2.3	Microsoft Azure	29
6.1.2.4	VPS OVH	31
6.2	Première approche - Heroku	33
6.3	Seconde approche - VPS	33
6.3.1	Dockerisation	33
6.3.2	Scirpts	34

6.3.3	Sécurité	35
6.3.3.1	SSH	35
6.3.3.2	Fail2ban	35
6.3.3.3	Firewall (UFW)	36
6.3.3.4	Firewall (OVH)	36
6.4	Intégration et Déploiement Continu (CI/CD)	37
7	Migration données existantes	38
7.1	Problématique	38
7.2	Solution	38
8	Monitoring et backups	39
9	Suite	40
10	Conclusion	41
11	Définitions	42
12	Bibliographie	43

1 Introduction

1.1 Contexte

1.1.1 Client

Le client, SLG Classic Cars, est une petite société familiale de restauration et entretien de voitures anciennes. Elle a été initialement fondée en 1976 et a été reprise par les fils en 2015. Ils sont désormais réputé dans leurs domaines et ne cessent de développer leurs activités. La société est basée sur deux sites, un atelier mécanique à Bierwart ainsi qu'un atelier carrosserie à Hingeon.

Dans le cadre de leurs activités, la société a besoin d'un SI¹ permettant de gérer de manière efficace les informations relatives aux :

- clients
- fournisseurs
- véhicules
- stock
- commandes
- devis
- factures
- fiches de Travail
- ...

1.1.2 Solution existante

Jusqu'à présent la société utilise une combinaison de deux logiciels :

GAD-Garag : *"Le logiciel garage GAD Garage est un logiciel garage professionnel de gestion commerciale pour la réparation de véhicule , GAD Garage est un logiciel Garage pour Auto , moto et la vente de pièces détachées automobiles (semi-grossiste)."*[3]

SLG-order-manager : Logiciel propriétaire permettant de générer et réceptionner des commandes. Ce logiciel interagit directement avec GAD-Garage et permet de simplifier l'encodage des informations relatives au stock.

En raison de divers problèmes liés au logiciel GAD-Garage et un manque de maintenance de celui-ci, la société souhaite développer une nouvelle solution informatique spécialement adaptée

1. *"Le système d'information (SI) est un élément central d'une entreprise ou d'une organisation. Il permet aux différents acteurs de véhiculer des informations et de communiquer grâce à un ensemble de ressources matérielles, humaines et logicielles. Un SI permet de créer, collecter, stocker, traiter, modifier des informations sous divers formats."*[2]

à leur façon de travailler et à leurs besoins cités ci-avant. Cette solution doit être conçue de façon à pouvoir être adaptée au fil des années en fonction des nouveaux besoins du client.

1.2 Objectifs

Au vue de l'envergure du projet et un temps relativement limité quant à la réalisation de ce travail de fin d'études, le client et moi-même avons décidé de définir des objectifs à court et long termes. Les objectifs à court termes sont prioritaires et sont ceux sur lesquels je travaillerai dans le cadre de ce travail de fin d'études. Si le résultat est concluant le projet sera étendu au delà du cadre scolaire et les objectifs à plus long termes seront réalisé.

Bien que les objectifs à long terme soient moins importants, ils ne sont pas indispensables pour autant. Dès lors, durant toute la durée de développement de ce projet, le client continuera d'utiliser ses solutions actuelles.

1.2.1 Court terme

- gestion des clients
- gestion des fournisseurs
- gestion du stock
- gestion de la main d'oeuvre
- gestion des commandes
- design "user-friendly"
- déploiement

Ces différents objectifs sont assez vaste et cachent une multitude d'objectifs secondaires tel que la gestion des droits aux ressources, la gestion de conflit d'encodage² et bien d'autres.

1.2.2 Long terme

- les autres fonctionnalités (voitures, facturation, ...)
- gestion avancée des utilisateurs de la web-App
- //TODO

2. Gérer la possibilité que deux utilisateurs distincte modifier simultanément une même donnée.

2 Recueil d'information et Analyse

Après avoir défini, avec le client, les grandes lignes du projet il a fallu définir et détailler de manière approfondie toutes les fonctionnalités. Pour ce faire, j'ai décidé de procéder en plusieurs étapes.

Premièrement, plusieurs réunions avec le client ont permis de dégrossir l'ensemble des fonctionnalités. Ainsi, chaque grosse fonctionnalité a été découpée en plus petites, a été détaillée en texte clair et a été contextualisée par rapport à l'ensemble du projet. Cette étape fut très importante car elle a permis de concrétiser les besoins du client et de mettre en avant les potentielles difficultés.

Dans un second temps, sur base de l'étape précédente, j'ai transformé le texte clair de chaque fonctionnalité en "user story" ³ Les "user stories" permettent de structurer l'ensemble des fonctionnalités en un format bien défini. Plus d'information quand à la réalisation de celles-ci se trouvent dans le sous-chapitre "User-stories" ci-après.

Pour finir, j'ai analysé et défini avec le client les besoins plus techniques tel que par exemple :

- les liens inter-fonctionnalités
- les données à devoir stocker
- les liens entre les différentes données
- les implications comptable (au niveau du stock et de la facturation)
- une estimation de la quantité de données à stocker
- les rôles des différents utilisateurs

3. Une "user story" est, dans le domaine du développement de logiciels, une description simple et structurée d'une fonctionnalité à développer.

2.1 User-stories

Comme expliqué précédemment, chaque fonctionnalité a été détaillée sous forme d'une "user story" (US). Dans un but organisationnel, j'ai décidé de grouper chaque US par grande fonctionnalité. Qu'entend-t-on par grande fonctionnalité ? Dans le cadre de ce projet, une grande fonctionnalité est un élément fictif faisant référence à l'ensemble des informations et manipulations relatives à un objet du système d'information. A titre d'exemple : la grande fonctionnalité "**Stock**" regroupe l'ensemble des fonctionnalités permettant de définir la façon dont un utilisateur va pouvoir consulter la quantité restante d'un produit, l'historique d'achat ou de vente d'un produit, la valeur cumulée de tous les produits, ajouter un nouveau produit, etc.

Afin de pouvoir facilement maintenir la multitude d'US, j'ai défini une structure type comportant les éléments suivants :

1. Code unique + Titre
2. Description (du type : *"En tant que X j'aimerais Y afin de Z"*)
3. Préconditions
4. Explication détaillée avec éventuellement des mockups⁴
5. Critères de validation

4. *"En informatique, le terme mock-up (qui vient du même mot anglais qui signifie une maquette à l'échelle 1 :1) désigne un prototype d'interface utilisateur."*[5]

2.1.1 Exemple

(ST02) Consulter le stock

En tant qu'utilisateur j'aimerais pouvoir consulter une liste des articles dans mon stock sous forme d'un tableau afin d'avoir un résumé des informations de chaque article.

🚩 Préconditions :

- **Technique :**
 - table `Product` doit exister
- **Logique :**
 - /

📄 Détail :

Quand l'utilisateur clique sur l'onglet `Stock` de la barre des menus, une requête `GET` est envoyée à l'API afin de récupérer les 25 premiers produits:

```
method : GET
url    : /api/products?show=0-24
```

En attendant la réponse du serveur, la page est chargée avec :

- la structure du tableau (les headers)
- un spinner à la place des données

✅ Si la requête abouti avec succès: les données sont chargées dans le tableau

❌ Si la requête échoue: un message d'erreur est affiché

S'il existe plus de 25 produits, des petites flèches en dessous du tableau permettent de charger les 25 produits suivants et ensuite remplacer les lignes du tableau existant par les "nouveaux" produits. Un compteur se trouvant à gauche des deux flèches permet de savoir la plage de produits actuellement affichée.

exemple :

1-10 < >

🔍 Critères de validation :

- Un utilisateur peut consulter une table reprenant tous les produits, chaque ligne de la table correspond à un produit.
- Si aucun produit existe, l'utilisateur voit un message indiquant qu'aucun produit n'a été trouvée et ce à la place du contenu de la table.

FIGURE 1 – Exemple de "user story"

L'ensemble des US peuvent-être consultées dans la wiki de la page Github⁵ de ce projet. Notons que toutes les fonctionnalités n'ont pas été transformées en US. J'ai décidé de travailler selon les méthodes de développement "agiles" qui préconisent de ne pas réaliser l'entièreté de l'analyse si cela n'est pas nécessaire au développement. Ma méthodologie de travail est décrite dans un chapitre ultérieur.

5. https://github.com/MMichotte/SLG_APP/wiki/US_0_home

2.1.2 Automatisation Trello

Une fois une US écrite, je la re-transcrivais sous forme de carte "Trello"⁶ afin de pouvoir utiliser celle-ci comme base de travail lors de l'implémentation de la fonctionnalité. Cette opération étant fastidieuse et rébarbative, j'ai décidé de développer une petite application en ligne de commande me permettant d'automatiquement générer mes cartes Trello sur base de mes US écrites dans la wiki du Github du projet.

J'ai conçu cette application avec comme objects :

1. imposer une structure d'US
2. utilisation simple
3. possibilité d'intégration dans une pipeline de déploiement continu⁷
4. ne doit pas avoir d'impact visuel sur les US dans le wiki de github
5. open-source

Bien que la conception de cette application m'ai pris un certain temps, cela est dérisoire par rapport au gain de temps que j'ai pu en tirer. Les fonctionnalités proposée par cette application sont néanmoins encore limités mais le projet étant open-source, tout le monde peut y contribuer.

Une explication plus détaillé n'ayant pas sa place dans ce rapport, elle est disponible sur le Github de ce mini-projet : <https://github.com/MMichotte/US-to-TrelloCard>.

2.2 Base de donnée

Un bon système d'information repose en grande partie sur la qualité de la base de donnée. La réalisation du schéma de la base de donnée est fortement dépendant du niveau de détail des fonctionnalités/US. Étant donnée que j'ai choisi de travailler en utilisant les méthodes "agile", il m'était dans un premier temps impossible de designer le schéma complet. Néanmoins, afin de ne pas devoir modifier l'ensemble de la structure de la base de donnée une fois en production, j'ai décidé de reprendre l'analyse de l'ensemble des fonctionnalités et d'en déduire un schéma de base de donnée le plus complet possible. Ce schéma a continué d'évoluer tout au long du projet en passant par plus de huit versions différentes.

En plus d'être un outil de documentation très important, ce schéma m'a permis de définir les éléments du projet plus complexe ainsi que de déceler certaines failles dans les descriptions des fonctionnalités.

6. Trello est un programme des gestion de taches permettant de créer différentes listes et cartes fin d'organiser le travail. Pour plus d'informations voir : <https://trello.com>

7. Une pipeline de déploiement continu est un ensemble d'actions permettant d'automatiser le déploiement du code depuis un environnement de développer vers un environnement de production. Pour de plus amples informations voir : <https://www.redhat.com/fr/topics/devops/what-is-ci-cd>

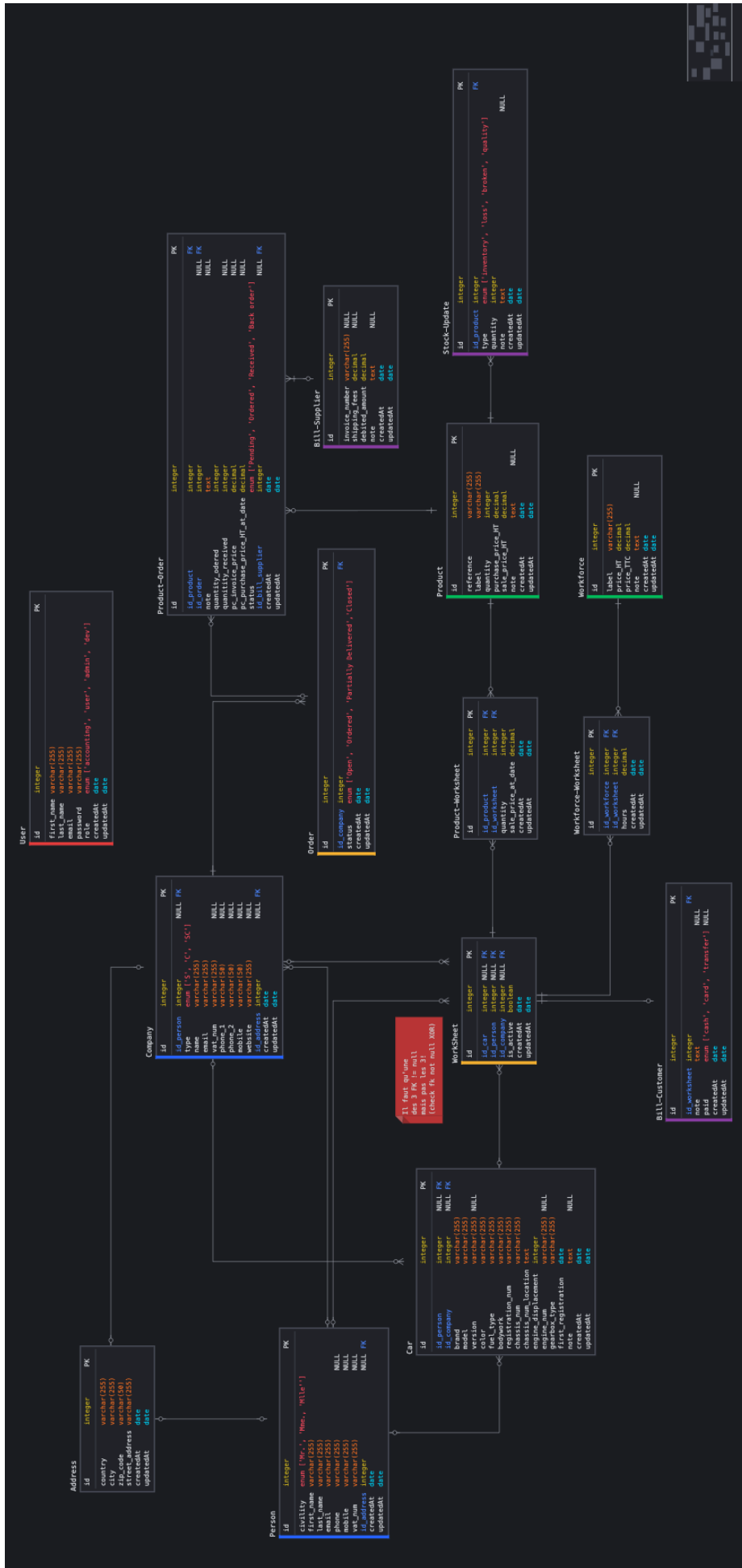


FIGURE 2 – Schéma de base de donnée

3 Méthodologie

3.1 Agile

Comme brièvement abordé précédemment, vu la l'envergure et la complexité du projet, j'ai décidé de travailler de manière "agile". Mais qu'est-ce que "l'agilité" dans le monde du développement informatique? *"En ingénierie logicielle, les pratiques agiles mettent en avant la collaboration entre des équipes auto-organisées et pluridisciplinaires et leurs clients¹. Elles s'appuient sur l'utilisation d'un cadre méthodologique léger mais suffisant centré sur l'humain et la communication². Elles préconisent une planification adaptative, un développement évolutif, une livraison précoce et une amélioration continue, et elles encouragent des réponses flexibles au changement"[7].*

Concrètement, dans le cadre de ce projet, cela à impliqué que j'ai travaillé de manière incrémentale comme l'illustre la *figure 3* ci-dessous.

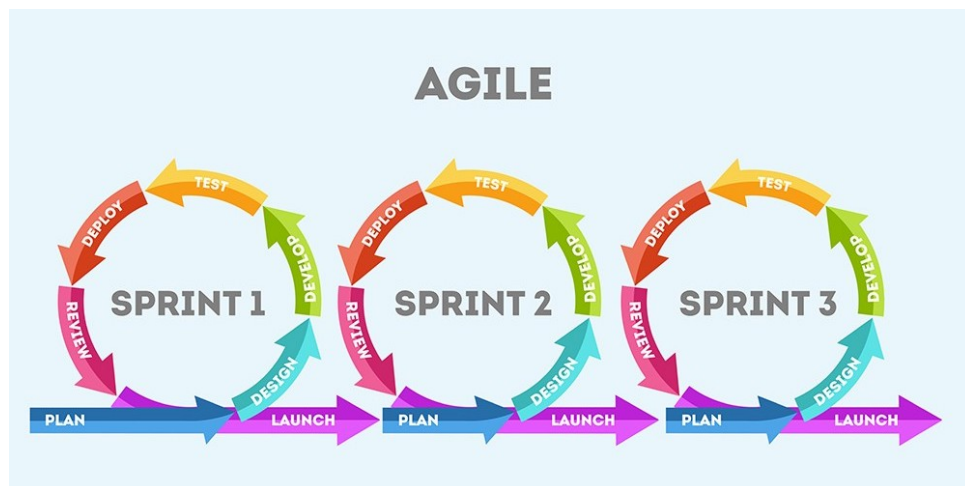


FIGURE 3 – Les raisons pour utiliser les méthodes Agile en entreprise de 300_librarians

Chaque fonctionnalité de ce projet peut être représentée par un "sprint". En d'autres termes j'ai, pour chaque fonctionnalité :

1. détaillé sous forme d'une US (PLAN + DESIGN)
2. implémenté (DEVELOP)
3. testé par le biais de tests automatisé ou manuellement (TEST)
4. déployé l'ensemble en production (DEPLOY)
5. demandé un feedback au client (REVIEW)
6. adapté la fonctionnalité en fonction du feedback

Cette Méthodologie a été très efficace et a permis une bonne collaboration avec le client.

3.2 Choix des technologies

3.2.1 Backend

Framework : NestJs

Language : TypeScript

Runtime environnement : NodeJs

ORM : TypeOrm



Tout d’abord, vu l’envergure du projet, il me paraissait important de veiller à une bonne structure et de garder en tête la maintenance du projet au fil du temps. Dans ces circonstances, il est important d’utiliser un framework. Quant au choix du framework, dans un premier temps mon choix s’est porté sur la suite NodeJs-ExpressJs. Ce choix était initialement justifié par le fait que j’avais déjà travailler avec ceux-ci. Après quelques semaines de développement, je me suis rendu compte que je préférais nettement utiliser le TypeScript pour un projet d’une tel envergure. J’ai des lors décidé de changer de framework et ai migré mon application NodeJs-Express vers du NestJs. Le NestJs utilise du Typescript et permet de bien structurer son code comme au frontend avec Angular. (voir ci-après)

3.2.2 Frontend

Framework : Angular

Languages :

- TypeScript
- HTML
- SCSS



Tout comme pour le backend, il est évident que l’utilisation d’un framework est indispensable. Le choix du framework s’est fait sur base de mon expérience personnelle. En effet, durant mes années d’études, j’ai eu l’occasion d’utiliser différents frameworks frontend tel que React, Vue et Angular. J’ai particulièrement bien aimé travailler avec ce dernier car il impose une certaine rigueur tout en restant relativement simple d’utilisation.

3.2.3 Base de donnée

Le choix de la base de données a été motivé par trois critères :

1. **SQL** : Comme expliqué lors de l'analyse des besoins du client dans la section "Base de donnée", vu le grand nombre de relations entre les différentes données, une base de données SQL est primordiale pour la bonne organisation du système d'information.
2. **Notoriété** : PostgreSQL est fortement utilisé dans le milieu professionnel ce qui lui oblige d'être mis à jour régulièrement et ce tant au niveau sécurité que en ajout de fonctionnalités. C'est donc une base de données fiable, utilisable à long terme et maîtrisée par de nombreux développeurs.
3. **Expérience** : Bien que toutes les bases de données SQL se ressemblent, le fait d'avoir déjà manipuler et de s'être familiariser avec PostgreSQL m'offre un gain de temps considérable.



3.2.4 Autre

3.2.4.1 API

Afin de tester les différents endpoints API de l'application, j'ai décidé d'utiliser **Insomnia**⁸. Insomnia permet non seulement de tester mon API en temps réel mais permet aussi d'écrire la documentation. Je n'utiliserai néanmoins pas cette dernière fonctionnalité étant donnée que le framework backend (NestJs) que j'utilise me permet de générer automatiquement et facilement une documentation API.

Une API sans documentation est pratiquement inutilisable. Il existe un grand nombre de technologies pour écrire de la documentation API. Afin de centraliser un maximum d'éléments, j'ai décidé d'utiliser un plugin pour NestJs. Ce plugin (swagger swagger-ui-express⁹) me permet de générer automatiquement la documentation de mes endpoints sur base de quelques décorateurs¹⁰ ajouté dans le code. En plus de cela, il me permet de déployer cette documentation directement avec l'application. Celle-ci est dès lors consultable ici :

<https://app.slgcars.be/api-docs/>.

8. <https://insomnia.rest>

9. <https://docs.nestjs.com/openapi/introduction>

10. Code non-essentiel au fonctionnement de l'application mais permettant d'écrire des annotations

3.2.4.2 Linter

Il est "facile" d'écrire du code mais beaucoup plus compliqué de le rendre cohérent, lisible et portable. Afin de palier à ces problèmes un linter est indispensable. Étant donnée que je travaille principalement avec du JS et du TS, j'ai opté pour ESLint. Ce linter est 100% configurable pour chaque projet et me permet de garantir, dans l'éventualité où dans le future un autre développeur venait à contribuer au projet, la cohérence de nommage des variables, la configuration de l'IDE et bien d'autre choses. Pour ce qui est du HTML et SCSS un linter est inclus dans le framework Angular.

3.3 Outils

Afin de travailler de travailler de manière efficace et dans un but de pouvoir garentir une maintenance efficace de ce projet, j'ai utilisé plusieurs outils tel que :

- **Git/Github** :
Outil de versioning et hébergement du code
- **VsCode** :
IDE
- **Trello** :
Outil de planification des tâches
Le Trello de ce projet est consultable ici : <https://trello.com/b/jxYKBrWG>
- **SQLDbm** :
Outil de design de schéma de base de donnée
- **DataGrip** :
Outil de manipulation de base de donnée

4 Développement

Bien que le développement à proprement parler ait constitué la majeure partie de ce projet, expliquer l'intégralité des fonctionnalités n'a pas sa place dans le cadre de ce rapport. Je tiens cependant à partager un exemple d'une analyse logique d'une fonctionnalité qui, à mes yeux, représente bien la complexité de ce projet (exemple ci-après).

Le développement à pour ce projet été divisé en quatre grandes phases :

1. **mise en place de la structure du projet :**

Cette phase a constitué à initialiser les différents modules du projet (backend, frontend, ...) ainsi qu'à choisir une structure de dossier appropriée.

2. **mise en place des outils facilitant la productivité :**

Afin de me simplifier le travail et/ou de ne pas devoir exécuter des tâches répétitives, j'ai utilisé plusieurs outils tel qu'un linter¹¹ ou des extensions de mon IDE me proposant de l'autocomplétion avancée. Ce sont donc une série d'outils non-indispensables mais incontournable.

3. **implémentation des US**

4. **refactoring :**

Cette phase n'est pas une phase en tant que tel. En effet, tout au long du projet, j'ai refactorisé le code que ce soit au niveau de la structure ou de la logique.

4.1 Exemple concret

Contexte : Dans le cadre de la US *"CO07 - Réceptionner une commande"*, une multitude de cas sont à étudier. En voici l'analyse :

Pour chaque produit réceptionné :

1. **Le produit à été reçu avec : quantité reçue == quantité commandé**

- (a) mise à jour de l'information du produit
- (b) le statut du produit passe à "RECEIVED"

2. **Le produit à été reçu avec : quantité reçue < quantité commandé**

- (a) Le produit est affiché avec deux status : "RECEIVED" et "BO"¹². La quantité manquante est affichée dans le statut "BO".

11. Un linter est un outil d'analyse statique de code permettant d'uniformiser le style du code et de déceler des potentiels erreurs.

12. Back Order

(b) Lors de la sauvegarde de la facture :

- i. un nouveau produit reprennent les mêmes informations que le produit d'origine est ajouté à la commande courante. La quantité commande de ce nouveau produit est égale à la quantité manquante définie précédemment. Le status de ce produit est "BO".
- ii. le status produit d'origine passe à "RECEIVED".

3. Le produit à été reçu avec : quantité reçue > quantité commandé

(a) Le système cherche tous les produits équivalent à celui-ci venant du même fournisseur et dont le status est "BO".

— **le système trouve un produit :**

- i. Récupération de la commande concernée par le produit trouvé
- ii. Traitement de ce produit comme s'il faisait partie de la commande actuelle (il passe par les mêmes étapes décrites ci-autour)

— **le système ne trouve pas produit :** On continue la procédure ci-dessous

(b) mise à jour de l'information du produit

(c) le statut du produit passe à "RECEIVED"

4. Le produit n'a pas été reçu

(a) le statut du produit passe à "BO"

5. Le produit ne faisait pas partie de la commande concernée

(a) l'utilisateur a la possibilité d'ajouter un produit non-présent dans la commande à l'aide d'un bouton.

(b) (ajout du produit dans la commande, voir US CO03)

(c) retour à l'étape n° 3

La réception d'une commande génère une facture. Cette facture a une valeur comptable et se doit donc d'être correcte. En plus de la création d'une facture, la réception d'une commande met à jour le stock. Le stock étant d'une grande importance financière et commerciale, celui-ci se doit également d'être le plus correcte possible avec la réalité. Dans ces circonstances, il était impératif que toutes les actions exécutées sur tous les produits d'une commande soient Des lors, j'ai du utiliser un système de transaction afin de garantir l'intégrité des données. Soit l'intégralité des données sont correcte et cohérente et sont alors ajoutée en base de donnée, soit il y a une erreur et l'entièreté de l'action est annulée, aucun changement n'a lieu dans la base de données.

4.2 Récapitulatif des fonctionnalités/US implémentées

L'ensemble des fonctionnalités prévues dans les objectifs à "court terme" ont été réalisées. Voici un récapitulatif non-détaillé de "user stories" implémentées :

Général

- ✓G01 - Connexion utilisateur

Stock

- ✓ST01 - Onglet Stock
- ✓ST02 - Consulter le stock
- ✓ST03 - Ajouter nouvel article dans le stock
- ✓ST07 - Rechercher un article
- ✓ST09 - Supprimer un article
- ✓ST04 - Consulter/modifier le détail d'un article
- ✓ST06 - Consulter récapitulatif entrée/sortie d'un article

Main d'oeuvre

- ✓MO01 - Onglet Main d'oeuvres
- ✓MO02 - Consulter la liste des main d'oeuvres
- ✓MO03 - Ajouter un nouveau tarif de main d'oeuvre
- ✓MO05 - Rechercher un main d'oeuvre
- ✓MO06 - Supprimer une main d'oeuvre
- ✓MO04 - Consulter/Modifier le détail d'un main d'oeuvre

Client

- ✓CL01 - Onglet Clients
- ✓CL02 - Consulter liste clients
- ✓CL03 - Ajout nouveau client
- ✓CL09 - Rechercher un client
- ✓CL10 - Supprimer un client
- ✓CL04 - Consulter/modifier détail d'un client

fournisseur

- ✓FO01 - Onglet Fournisseurs
- ✓FO02 - Consulter liste des fournisseurs
- ✓FO03 - Ajouter nouveau fournisseur
- ✓FO04 - Consulter/Modifier le détail d'un fournisseur
- ✓FO06 - Rechercher un fournisseur
- ✓FO07 - Supprimer un fournisseur

Commandes

- ✓CO01 - Onglet Commandes
- ✓CO02 - Consulter la liste des commandes
- ✓CO03 - Créer une nouvelle commande
- ✓CO04 - Consulter le détail d'une commande
- ✓CO05 - Rechercher une commande
- ✓CO06 - Supprimer une commande
- ✓CO07 - Réceptionner une commande

Factures

- ✓FA01 - Onglet Factures

5 Sécurité

5.1 Contrôle d'accès

Cette application web est destinée à être utilisée par plusieurs personnes avec des rôles différents au sein de la société. Chaque rôle n'a pas les mêmes droit quant à la consultation ou manipulation des données de l'application. On peut ainsi distinguer cinq catégories d'utilisateurs :

1. **Inconnu** : utilisateur non-authentifié. Celui-ci n'a aucun droit sur les données et peut uniquement accéder à la page de connexion de l'application.
2. **Comptabilité** :
 - a tous les droits de l'inconnu
 - peut uniquement consulter l'ensemble des données
 - peut générer des pdf (factures, commandes, ...)
3. **Atelier** :
 - a tous les droits de la comptabilité
 - peut créer/modifier une fiche de Travail
 - peut créer/modifier une commande
4. **Administration** :
 - peut tout faire hormis supprimer les utilisateurs de type "Développeur"
5. **Développeur** :
 - peut tout faire et a accès aux logs et métriques

Afin d'authentifier un utilisateur et donc de lui donner certain droits ou non, l'application utilise des JSON Web Tokens (JWT) ¹³. Les tokens sont générés lors de chaque nouvelle connexion réussie et sont valides pendant 48h. Toute action effectuée à l'aide d'un token non-valide déconnecte automatiquement l'utilisateur et le re-dirige vers la page de connexion.

13. Comment cela fonctionne-t-il ? Explication en vidéo : <https://www.youtube.com/watch?v=7Q17ubqLfaM>

5.2 Hachage et Chiffrement

Avant toute chose il me semble bon de rappeler la différence entre le **hachage** et le **chiffrement** :

— **hachager** :

Utilisation d'un algorithme de hachage uni-directionnel afin de transformer les données de manière irréversible et de longueur fixe. Il est donc possible de comparer deux hash.

— **chiffrer** :

Encoder des données afin qu'uniquement la personne ayant la clé de déchiffrement puisse les déchiffrer. Cette action est donc réversible.

5.2.1 Données enregistrées

Toute donnée susceptible d'être un risque pour l'intégrité et la protection de l'application ou des autres données est stockée sous forme de hash (et non en clair¹⁴) dans la base de donnée (par exemple les mots de passes des utilisateurs).

Afin de hasher ces données, j'ai utilisé la librairie "bcrypt". La librairie "bcrypt" est une des librairies de hashage les plus réputées. Elle permet d'incorporer un sel (salt)¹⁵ afin de se protéger contre les attaques par table de correspondance (rainbow table)¹⁶. De plus elle est facilement paramétrable permettant ainsi d'augmenter la complexité de l'algorithme utilisé afin de palier aux attaques par force brute¹⁷ utilisant des machines à puissances de calcul importante.

5.2.2 HTTPS

Stocker les données à risque de manière chiffrée dans la base de donnée est une chose, cependant, avant que les données n'arrivent jusque là elles sont potentiellement envoyées en clair sur le réseau (internet). Afin d'éviter cela, je n'autorise que les connexion HTTPS et redirige les connexions HTTP vers le HTTPS. Ceci permet de garantir que l'entièreté des données envoyées sont chiffrées et dès lors beaucoup moins vulnérables aux attaques de type "Man In The Middle" (MITM).

14. lisible et compréhensible par quiconque

15. *"Le selage, est une méthode permettant de renforcer la sécurité des informations qui sont destinées à être hachées (par exemple des mots de passe) en y ajoutant une donnée supplémentaire afin d'empêcher que deux informations identiques conduisent à la même empreinte (la résultante d'une fonction de hachage)"*[8]

16. Attaque consistant à comparer un hash à un table reprenant un grand nombre de paires de text clair et le hash correspondant.

17. Attaque consistant à essayer un grand nombre de données générées automatiquement en espérant trouver la bonne.

5.3 HTTP Headers

Les headers HTTP permettent de sécuriser un site-web contre la plupart des attaques les plus répandues et ce de manière relativement simple. Voici en résumé les principaux headers que j'ai configuré :

- **Strict-Transport-Policy (STP) :**

Header permettant d'indiquer au navigateur que le site-web ne peut être accèder qu'en utilisant HTTPS au lieu de HTTP.

- **Content-Security-Policy (CSP) :**

"L'en-tête de réponse HTTP Content-Security-Policy permet aux administrateurs d'un site web de contrôler les ressources que l'agent utilisateur est autorisé à charger pour une page donnée. Bien qu'il y ait quelques exceptions, ces règles impliquent la plupart du temps de définir les origines du serveur et les points d'accès pour les scripts. Cet en-tête aide à se protéger contre les attaques de cross-site scripting."[9]

- **X-Frame-Options :**

"L'en-tête de réponse HTTP X-Frame-Options peut être utilisé afin d'indiquer si un navigateur devrait être autorisé à afficher une page au sein d'un élément <frame>, <iframe>, <embed> ou <object>. Les sites peuvent utiliser cet en-tête afin d'éviter les attaques de clickjacking pour s'assurer que leur contenu ne soit pas embarqués dans d'autres sites."[10]

- **X-Content-Type-Policy :**

"L'entête X-Content-Type-Options est un marqueur utilisé par le serveur pour indiquer que les types MIME annoncés dans les en-têtes Content-Type ne doivent pas être modifiés ou et suivis. Cela permet de se détacher du sniffing de type MIME, ou, en d'autres termes, c'est une façon de dire que les webmasters savaient ce qu'ils faisaient."[11]

Après avoir configuré ces différents headers, j'ai testé le bon fonctionnement de ceux-ci en lançant une analyse en ligne. Comme le montre la *figure 4* l'ensemble des headers ont bien été configurés et protègent désormais le site-web.


Security Report Summary	
	Site: https://app.slgcars.be/
	IP Address: 152.228.173.161
	Report Time: 23 May 2021 13:24:01 UTC
	Headers: ✔ Content-Security-Policy ✔ Permissions-Policy ✔ Referrer-Policy ✔ Strict-Transport-Security ✔ X-Content-Type-Options ✔ X-Frame-Options
	Warning: Grade capped at A, please see warnings below.

FIGURE 4 – Résultat analyse de sécurité des headers fait sur <https://securityheaders.com/>

5.4 Dependabot Github

Durant le développement de l'application, afin de ne pas réinventer la roue, il est fréquent d'utiliser des packages/composants externes créés par d'autres développeurs. Ces packages étant utilisés par un grand nombre d'utilisateurs, ils sont une proie "facile" pour des personnes malveillantes qui souhaiteraient infecter un grand nombre de site-web. Ces packages sont dès lors régulièrement mis à jour par les créateur afin de garantir une sécurité maximale.

Mais comment savoir quand une nouvelle version d'un packages est disponible pour des raison de sécurité? Heureusement, Github propose plusieurs services permettant de réaliser des analyse de sécurité statique sur le code y étant hébergé. Ainsi j'ai utilisé "Dependabot" qui scan périodiquement l'ensemble des dépendances du projet afin d'y détecter de potentielles failles. Si celui-ci en trouve, il m'envoie un email me prévenant de la nature de la faille et de sa dangerosité. Il ne me reste alors plus qu'à mettre à jour le(s) package(s) concerné(s) manuellement ou d'accepter une pull-request¹⁸ générée par "Dependabot".

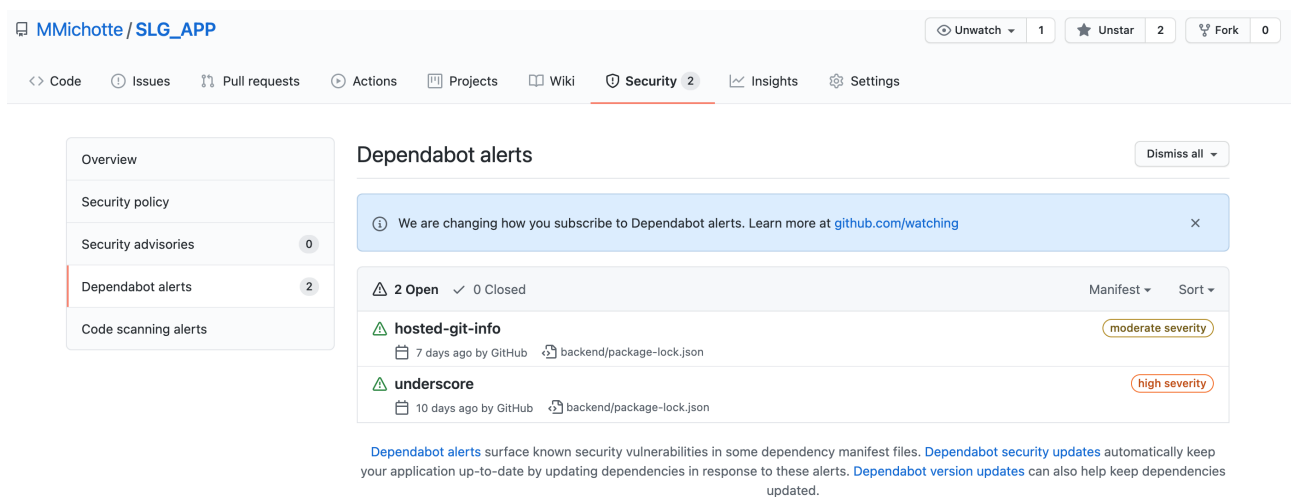


FIGURE 5 – Exemple d'alertes générées par "Dependabot"

18. Demande de modification du code source par une tierce personne. Pour plus d'information, voir : <https://www.atlassian.com/git/tutorials/making-a-pull-request>

6 Déploiement

6.1 Étude de marché

6.1.1 Avant-propos

Il existe de nos jours une multitude de solutions afin d'héberger une application web. Toutes les solutions offrent divers avantages et inconvénients, aussi bien d'un point de vue technologique que financier. Afin de ne pas s'encombrer avec une quantité importante de possibilités, j'ai sélectionné 3 options qui me paraissent les plus adéquates :

- Serveur local
- Heroku (<https://www.heroku.com/home>)
- Microsoft Azure (<https://azure.microsoft.com/en-us/>)
- Virtual Private Server (VPS) chez OVH (<https://www.ovhcloud.com/fr/vps/>)

Dans le cadre de ce projet, il est important de préciser les points suivant :

- le client n'a **pas d'informaticien** à temps pleins
- le client souhaite **limiter les dépenses** au maximum
- le **trafic** vers l'application sera **faible** car utilisé uniquement par une ou deux personnes et pendant 2-3h par jour maximum
- dans le futur, un web-shop pourrait être intégré à l'application et donc générer un trafic plus important -> **évolutivité**

La solution doit pouvoir :

- Héberger une application NodeJs
- Héberger une base de données SQL

(Idéalement les deux services sont hébergé sur la même plateforme mais ce n'est pas obligatoire.)

6.1.2 Étude des solutions

6.1.2.1 Serveur Local

Description :

Le client utilise actuellement un serveur tournant sous Windows 10 pro. Ce serveur fait tourner leur programme de gestion mais sert également d'ordinateur "classique" pour tous les employés. Notons qu'aucun système de backup automatique n'est mis en place. La base de données incluse dans leur programme de gestion devant être accessible depuis le site de la carrosserie, une règle de port-forwarding est mise en place dans le firewall de leur routeur.

Avantages :

- Coût négligeable car infrastructure existante
- Maîtrise totale des données

Inconvénients :

- Os du serveur inadapté (Windows 10 pro)
- Maintenance plus compliqué à mettre en place
- Scalability (évolutivité) compliquée voir impossible sans investissement conséquent
- Déploiement continue plus complexe à mettre en place
- Accessibilité extérieur require plus de sécurité
- Nécessite la mise en place d'une meilleur gestion des backups
- Pas de redondance

Prix :

L'infrastructure physique déjà existante et l'utilisation de software gratuit et open-source tel qu'Apache, PostgreSQL, fail2ban,... rendraient les coûts négligeables.

Coût estimé en production : ± 0 €/mois

Conclusion

Au vu des nombreux désavantages, je pense que cette solution, bien que peu coûteuse, ne soit pas envisageable.

6.1.2.2 Heroku

Déscription :

Heroku est un PaaS(Platform as a Service) fortement utilisé de par sa simplicité et sa compatibilité avec des langages modernes tel que Node, Ruby, Python et bien d'autres.

Bien que souvent utilisé pour des projets de petite à moyenne taille, certaines grandes entreprises comme Toyota Europe ou Dubsmash l'utilisent également.

Avantages :

- Intégration très facile avec git
- Entièrement gratuite pour le développement
- Inclus une base de données PostgreSQL
- Coût en production fixe
- Maintenance facile et accessible à distance
- Portabilité élevée : il est très facile d'arrêter le service de de migré vers une autre solution.
Le code n'est pas lié à Heroku
- Expérience : j'ai déjà plusieurs fois eu l'occasion de travailler avec Heroku, je connais assez bien la plateforme et la façon de travailler avec
- Métriques inclus
- Sécurité

Inconvénients :

- Scalabilité moyenne car il faut changer de plan tarifaire en fonction du trafic
- Gestion de la base de données moins facile (pas de rôle différents)
- Cold-starts d'une à deux minutes du site-web (uniquement avec la version gratuite)

Prix :

Heroku fonctionne sur base de plans tarifaire prédéfini mais modulable. En fonction des besoins du client, j'ai sélectionné 4 plans tarifaire :

— Hobby - gratuit :

Ce plan tarifaire offre pratiquement toutes les fonctionnalités dont nous avons besoin. Nous serons néanmoins restraints par le nombre de requêtes, la taille de la base de données, les cold-starts, ... Cette solution me semble plus que suffisante durant le développement de l'application et pourquoi pas durant les premiers mois ou premières années d'utilisation. Attention, la DB gratuite est limitée à 10K lignes.

Heroku Pricing Estimate		Est. monthly cost \$0	
Created: December 12, 2020 Available until: June 6, 2021		1 app	
App 1	my-app-estimate-1		\$0
	Dynos		
	Free	1 instance	\$0
	Data Services		
	Postgres Hobby Dev	1 instance	\$0
	Redis Hobby Dev	1 instance	\$0
Included	• Ability to add collaborators • Heroku Pipelines and Heroku Review Apps		

— Hobby - basic :

Ce plan tarifaire est identique au précédent sauf que la DB peut accueillir 10M de lignes.

Heroku Pricing Estimate		Est. monthly cost \$9	
Created: December 12, 2020 Available until: June 6, 2021		1 app	
App 1	my-app-estimate-1		\$9
	Dynos		
	Free	1 instance	\$0
	Data Services		
	Postgres Hobby Basic	1 instance	\$9
	Redis Hobby Dev	1 instance	\$0
Included	• Ability to add collaborators • Heroku Pipelines and Heroku Review Apps		

— Hobby - avancé :

Ce plan tarifaire est à peu de choses prêt équivalent au plan gratuit. Il permet néanmoins de supprimer complètement les cold-starts. Il offre également un les métriques du site pour les dernières 24h.

Heroku Pricing Estimate		Est. monthly cost \$16	
Created: December 12, 2020 Available until: June 6, 2021		1 app	
App 1	my-app-estimate-1		\$16
	Dynos		
	Hobby	1 instance	\$7
	Data Services		
	Postgres Hobby Basic	1 instance	\$9
	Redis Hobby Dev	1 instance	\$0
Included	<ul style="list-style-type: none">• Ability to add collaborators• Heroku Pipelines and Heroku Review Apps		

— Production :

Cet plan offre tout ce que les plans précédents offraient mais augmente considérablement les capacité de gestion de trafic, augmente la taille maximale de la base de données, offre des métriques détaillées aussi bien pour la base de données que pour le site-web, permet des roll-backs sur une période de 7 jours.

Heroku Pricing Estimate		Est. monthly cost \$75	
Created: December 12, 2020 Available until: June 6, 2021		1 app	
App 1	my-app-estimate-1		\$75
	Dynos		
	Standard 1X	1 instance	\$25
	Data Services		
	Postgres Standard 0	1 instance	\$50
Included	<ul style="list-style-type: none">• Up to 5 Heroku Teams with 1-5 members each (additional cost for larger Teams)• Heroku Pipelines and Heroku Review Apps• Standard support during business hours with 1+ day response		

Conclusion

Sur base de ces options, je pense qu'il est possible de partir dans un premier temps sur le plan tarifaire n°2. Néanmoins le jour où un web-shop est ajouté à l'application, il faudra probablement passer sur un autre plan tarifaire tel que le n°3.

En plus du coût du plan n°2, il est bon de prendre une petite marge de sécurité afin de ne pas être surpris lors d'éventuels coûts supplémentaires tel qu'un nom de domaine, un autre certificat SSL,...

Coût estimé en production : 18 €/mois

6.1.2.3 Microsoft Azure

Déscription :

Microsoft Azure est un des leaders dans le domaine des cloud service providers. La plateforme offre plus de 200 produits couvrant une multitudes de domaines allant de la location de ressources de calculs pour du Machine Learning à l'hébergement de base de données en passant par la gestions de conteneurs Docker.

Microsoft Azure est utilisé par un très grands nombre d'entreprises tel que 3M, Airbus, Avid, BMW et bien d'autres.

Avantages :

- Scalabilité extrêmement performante
- Compartimentation de chaque service
- Documentation et communauté très active
- Grandes flexibilité de configuration
- Maintenance facile et accessible à distance
- Métrics inclus
- Sécurité

Inconvénients :

- Coût très variable et difficile à prévoir à l'avance
- Complexe à configurer correctement
- Plus de choses à configurer manuellement
- Je n'ai aucune expérience

Prix :

A l'inverse de Heroku, Azure ne fonctionne pas sur base de plans tarifaire spécifique mais fonctionne sur base du principe Pay-as-you-go. Le coût dépend donc fortement du trafic, de la taille de la base de données, de la taille des requêtes etc.

- **Service Web** : Hébergement de l'app sur une machine Linux :
 - version gratuite pendant 12 mois
 - version payante : 11.081€/mois
- **Base de donnée** :
 - 5GB (5GB étant le minimum configurable) d'espace de stockage à 0.1155€/GB/mois soit : ±5.8€/mois

— 1 vCore à 0.4840€/heure, si utilisation 2h/j -> 60h/mois soit : ± 29 €/mois

Coût estimé en production : 50 €/mois

Conclusion

Bien que Microsoft Azure offre une très grande flexibilité et environnement très professionnel, le coût semble fort élevé et pas très compétitif pour une application de cette envergure. Néanmoins, le jour où un web-store est ajouté, cette solution peut être retenue !

6.1.2.4 VPS OVH

Déscription :

Qu'est ce qu'un VPS ? *"Un serveur privé virtuel est un environnement isolé, créé sur un serveur physique à partir d'une technologie de virtualisation. Cette solution offre tous les avantages d'un serveur standard, profitant de ressources allouées et d'une administration complète."*[12]

Il existe plusieurs entreprises de Cloud Computing offrant des services VPS tel que AWS, IBM et OVH. Ayant eu de bons échos d'OVH et ayant déjà travaillé avec leurs services, j'ai choisi de limiter ma recherche d'offre à OVH.

Avantages :

- Maîtrise totale des données
- Facile à mettre en place
- Grandes flexibilité de configuration
- Maintenance facile et accessible à distance
- Scalability (évolutivité) compliquée
- Métriques inclus

Inconvénients :

- Nécessite la mise en place de la sécurité du VPS
- Nécessite la mise en place d'une gestion des backups
- Redondance plus compliqué à mettre en place

Prix :

OVH propose une multitude de plans tarifaires pour leurs VPS (voir *figure 6* ci-dessous). Notre application étant réservé à un usage interne, le trafic généré sera faible. Le plan tarifaire le moins cher nous sera dès lors plus que suffisant. De plus il est toujours possible de changer de plan par la suite.

Specifications techniques

	Value 5 € HT/mois 4,60 € HT/mois Commander	Essential 10 € HT/mois 9,20 € HT/mois Commander	Comfort 20 € HT/mois 18,40 € HT/mois Commander	Elite À partir de 30 € HT/mois 27,60 € HT/mois Commander
Processeur	1 vCore	2 vCore	4 vCore	8 vCore
Mémoire	2 Go	4 Go	8 Go	De 8 Go à 32 Go
Storage	40 Go SSD NVMe	80 Go SSD NVMe	160 Go SSD NVMe	De 160 Go à 640 Go SSD NVMe
Bande passante	250 Mbit/s illimité*	500 Mbit/s illimité*	1 Gbit/s illimité*	2 Gbit/s illimité*

FIGURE 6 – Tarifs VPS OVH

Coût estimé en production : ± 5 €/mois

Conclusion

Cette solution offre pratiquement tous les avantages d'un serveur local sans devoir se soucier de l'aspect hardware et ce à un prix très raisonnable.

6.2 Première approche - Heroku

Dans un but de réduire au maximum les coûts et de simplifier les choses durant le développement de l'application, le client et moi-même avons décidé de déployer, dans un premier temps, celle-ci sur Heroku avec le plan gratuit. Cette solution fut plus que suffisante pendant les deux-trois premiers mois du développement. Néanmoins, une fois l'application plus complète, plusieurs facteurs tel que :

- la lenteur de mise en production (temps nécessaire entre l'action de déploiement et l'accessibilité du site)
- le maximum de 10k lignes d'entrées dans la base de donnée

nous ont fait revoir les solutions de déploiement.

6.3 Seconde approche - VPS

Pour les raisons précédemment abordées, nous avons décidé de migrer l'application hébergée sur Heroku vers un VPS chez OVH. Ceci nous a permis de nous affranchir totalement des contraintes liés à la base de données mais impliquait une plus grande configuration initiale.

En sachant que l'application pouvait potentiellement encore être migrée dans le futur, j'ai décidé de mettre en place une multitude d'éléments afin de rendre ces transitions les plus simples possible. Ainsi l'entièreté de l'application est conteneurisée¹⁹ et toutes les configurations du VPS sont au maximum scriptées.

6.3.1 Dockerisation

Dans le but de faciliter la configuration des différents services nécessaires au bon fonctionnement de l'application, je les ai chacun mis dans un conteneur docker. Ainsi l'application est composée de 3 conteneurs distincts :

1. **slg-db** : La base de donnée PostgreSQL. Port exposé : **5434**.
2. **slg-app** : L'application NodeJs²⁰ en tant que tel, composée du backend et frontend. Port exposé **8080**.
3. **caddy** : Serveur web Caddy²¹ utilisé en tant que reverse-proxy²² et gestionnaire des certificats https. Ports exposés : **80** et **441**.

19. "La conteneurisation informatique permet de packager tous les services, scripts, API, librairies dont une application a besoin. L'objectif : en permettre l'exécution sur n'importe quel noyau compatible." [14]

20. NodeJs est un environnement d'exécution open-source et multi-plateforme permettant de faire tourner du code Javascript.

21. <https://caddyserver.com>

22. Un reverse-proxy est un intermédiaire de communication réseau permettant de transmettre une requête au serveur cible en fonction du nom de domaine de cette requête.

Ces trois conteneurs fonctionnent en symbiose et sont dépendant les uns des autres. L'application node ne démarrera par exemple pas si la base de donnée n'a pas démarrée avec succès. Afin de contrôler l'environnement dans lesquels ces conteneurs sont lancés ainsi que de pouvoir facilement les démarrer/arrêter, j'ai créé un fichier docker-compose²³.
(Voir : https://github.com/MMichotte/SLG_APP/blob/master/docker/docker-compose.yml)

6.3.2 Scripts

J'ai principalement écrit 2 scripts, le premier et plus consistant me permet de configurer automatiquement un nouveau VPS. Dans les faits, celui-ci exécute :

1. Mise à jour des packages
2. Mise à jour de la time-zone
3. Création d'un nouvel utilisateur
4. Modification des paramètres de connexion SSH
5. Configuration et activation de Fail2ban²⁴
6. Installation de docker-compose

Le second script est bien moins complexe et me permet de configurer et démarrer le backup (local) journalier de la base de donnée. Ces backups sont réalisés périodiquement à l'aide de crontab²⁵.

Il s'avère que ces scripts m'ont été très rapidement utiles. Seulement 6h après la configuration complète du VPS, un incendie a ravagé plusieurs data-centers chez OVH et a détruit, entre autres, notre VPS.²⁶ Après quelques jours d'inaccessibilité, j'ai pu re-configurer un VPS dans un autre data-center. Cette configuration n'a pas duré plus de 10 minutes.

23. docker-compose permet d'orchestrer un ensemble de conteneurs dans le but de simplifier la gestion et la communication de ceux-ci.

24. Fail2ban est un programme de prévention permettant de bloquer les tentatives d'intrusion frauduleuses

25. *"Crontab est un outil qui permet de lancer des applications de façon régulière, pratique pour un serveur pour y lancer des scripts de sauvegardes, etc."*[13]

26. Plus d'information sur l'incident : <https://www.ovh.com/fr/news/presse/cpl1785.dernieres-informations-notre-site-strasbourg>

6.3.3 Sécurité

6.3.3.1 SSH

Une connexion SSH avec le VPS est indispensable pour pouvoir le configurer et y configurer les différents services. Cependant c'est aussi une des principales porte d'entrée pour les personnes malveillantes. Afin de minimiser au maximum les risques j'ai configuré le service ssh de tel sorte que :

- il soit disponible sur le port 62222 et non 22. Un grand nombre d'attaques par force brute se basent sur le fait que par défaut le port utilisé par SSH est le 22. Dès lors changer le port par défaut permet d'éviter toute une série d'attaques automatisées.
- il interdise la connexion par mot de passe. Il est désormais uniquement possible de se connecter par pair de clés SSH.
- il interdise la connexion en tant que root.

6.3.3.2 Fail2ban

Qu'est ce que Fail2ban ? *"fail2ban est une application qui analyse les logs de divers services (SSH, Apache, FTP...) en cherchant des correspondances entre des motifs définis dans ses filtres et les entrées des logs. Lorsqu'une correspondance est trouvée une ou plusieurs actions sont exécutées. Typiquement, fail2ban cherche des tentatives répétées de connexions infructueuses dans les fichiers journaux et procède à un bannissement en ajoutant une règle au pare-feu iptables ou nftables pour bannir l'adresse IP de la source."*[15]

Comme expliqué précédemment, un des points d'entrées les plus vulnérable est le service SSH. Dans le but de renforcer son imperméabilité, j'ai ajouté une règle fail2ban limitant le nombre de tentatives de connexion à 6. Après 6 connexions échouées, l'adresse IP source est bloquée pour une durée de 1h. La durée d'un ban augmente de manière exponentielle si la même IP re-tente une multitudes de connexions après s'être faite bannir.

6.3.3.3 Firewall (UFW)

Afin de pouvoir contrôler le flux d'informations circulant entre le VPS et le réseau internet j'ai mis en place un firewall sur le VPS. Celui-ci intégrant par défaut les iptables²⁷, j'ai choisi d'utiliser "UFW" car celui-ci s'intègre parfaitement avec les iptables.

```
#!/bin/bash

sudo ufw disable

sudo ufw allow 62222
sudo ufw allow 80
sudo ufw allow 443
sudo ufw allow from [redacted].34/32 to any port 5434

sudo ufw default deny incoming

sudo ufw enable
```

FIGURE 7 – Script d'ajout règles UFW

6.3.3.4 Firewall (OVH)

En plus du firewall interne au VPS, OVH met à disposition un firewall en amont du VPS. J'ai configuré celui-ci afin d'être le plus restrictif possible (voir *figure 8*).

Priority ⓘ	Action	Protocol	Source IP	Source port	Destination port	Options	Status	
0	Authorise	TCP	all			established	Enabled	🗑
1	Authorise	TCP	all		62222		Enabled	🗑
2	Authorise	TCP	all		80		Enabled	🗑
3	Authorise	TCP	all		443		Enabled	🗑
4	Authorise	TCP	[redacted].34/32		5434		Enabled	🗑
18	Authorise	ICMP	all				Enabled	🗑
19	Refuse	IPv4	all				Enabled	🗑

FIGURE 8 – Configuration du firewall en amont du VPS

Notons que dans les deux firewall (UFW et OVH) le port 5434 est ouvert, ce port est utilisé par la base de donnée et rendre se port accessible publiquement est potentiellement une faille de sécurité. Néanmoins j'ai restreint l'accès à ce port à uniquement une adresse IP spécifique (la mienne en l'occurrence). Une fois l'application terminée, cette règle sera supprimée rendant la base de données totalement inaccessible depuis l'extérieur.

27. iptables est le pare-feu installé par défaut sur les système d'exploitation linux.

6.4 Intégration et Déploiement Continu (CI/CD)

La méthodologie Agile se basant sur une succession de "sprints" aboutissant à chaque fois à une nouvelle fonctionnalité, il est important que celle-ci puisse rapidement être testée par le client afin d'en avoir un feedback et de pouvoir y apporter ou non des modifications.

Dans ces conditions il est indispensable d'avoir un système d'intégration et de déploiement continu. L'intégration continue consiste à automatiser les phases de tests tandis que le déploiement continu permet d'automatiser la mise en production de la nouvelle fonctionnalité.

Le code de ce projet étant hébergé sur Github, j'ai pu profiter des "Github Actions"²⁸ permettant de définir un ensemble d'actions à exécuter sur le code et ce sur base de certaines conditions et dans un environnement isolé. pour ce projet j'ai dès lors défini le workflow suivant :

Lors d'une modification du code sur la branche "master" :

1. lancement en simultané des tests unitaires frontend et backend

Si tous les tests sont passés avec succès :

2. déploiement :

- (a) Exécution d'un script bash permettant de compiler le code source
- (b) Création des variables d'environnement
- (c) Envoie des fichier sur le serveur
- (d) Lancement des différents services (docker-compose)

Exemple de résultat du CI/CD lors d'une résolution d'un bug.

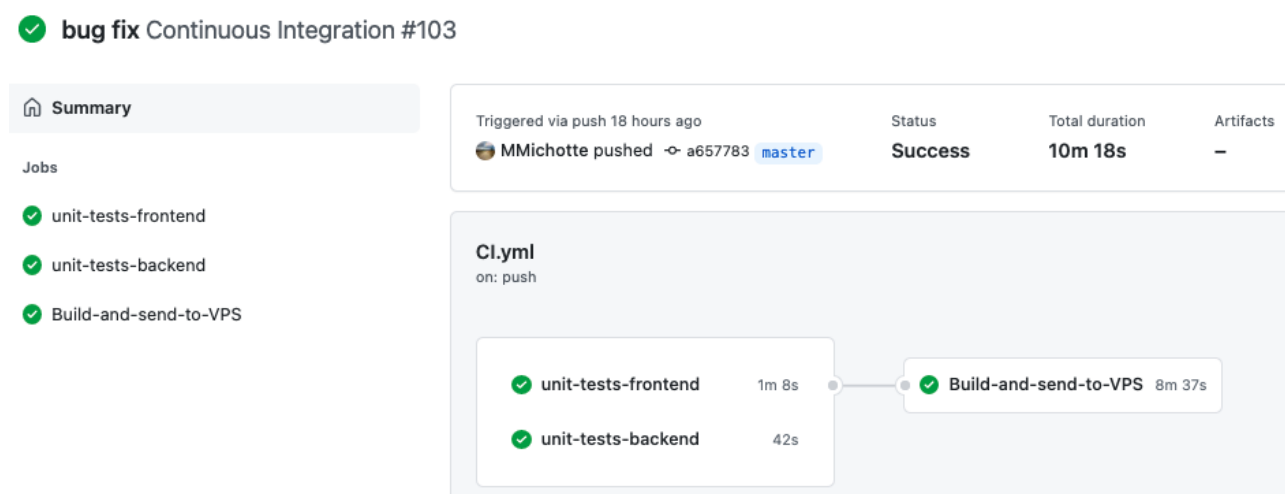


FIGURE 9 – Exemple résultat du CI/CD

28. Plus d'informations voir : <https://github.com/features/actions>

7 Migration données existantes

7.1 Problématique

7.2 Solution

8 Monitoring et backups

9 Suite

10 Conclusion

11 Définitions

US : User Story

CI : Continuous Integration

CD : Continuous Deployment

IDE : Integrated Development Environment

API : Application Programming Interface

12 Bibliographie

- [1] Tristan Slegers (sans date), sur le site *SLG Classic Cars : La voiture ancienne. Côté passion*. Consulté le 19 mai 2021.
<https://slgcars.be>
- [2] Syloé (2021), sur le site *syloe : Système d'information* Consulté le 20 mai 2021
<https://www.syloe.com/glossaire/systeme-dinformation/>
- [3] E.U.R.L ADELIE (2021), sur le site *Logiciel Garage / Logiciel Garage Professionnel , GAD Garage* . Consulté le 20 mai 2021
<https://www.logiciel-garage.fr>
- [4] Walter Görlitz (21 février 2021), sur le site *User story - Wikipedia* Consulté le 22 mai 2021
https://en.wikipedia.org/wiki/User_story
- [5] Verdy P (31 janvier 2021), sur le site *Mock-up — Wikipédia* Consulté le 22 mai 2021
<https://fr.wikipedia.org/wiki/Mock-up>
- [6] Rob Terzi (16 novembre 2018), sur le site *Qu'est-ce que l'approche CI/CD ?* Consulté le 22 mai 2021
<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 mai 2021
https://fr.wikipedia.org/wiki/Méthode_agile
- [8] Pautard (08 avril 2021), sur le site *Salage (cryptographie) — Wikipédia* Consulté le 23 mai 2021
[https://fr.wikipedia.org/wiki/Salage_\(cryptographie\)](https://fr.wikipedia.org/wiki/Salage_(cryptographie))
- [9] JNa0 (22 mai 2021), sur le site *Politique de sécurité de contenu - HTTP / MDN* Consulté le 24 mai 2021
<https://developer.mozilla.org/fr/docs/Web/HTTP/Headers/Content-Security-Policy>
- [10] DeusExNihilo (22 mai 2021), sur le site *X-Frame-Options - HTTP / MDN* Consulté le 24 mai 2021
<https://developer.mozilla.org/fr/docs/Web/HTTP/Headers/X-Frame-Options>
- [11] tchioubak (22 mai 2021), sur le site *X-Content-Type-Options - HTTP / MDN* Consulté le 24 mai 2021
<https://developer.mozilla.org/fr/docs/Web/HTTP/Headers/X-Content-Type-Options>
- [12] OVH (sans date), sur le site *Qu'est-ce qu'un VPS ? Découvrez les avantages d'un VPS / OVHcloud* Consulté le 25 mai 2021
<https://www.ovhcloud.com/fr/vps/definition/#:~:text=Grâce%20au%20VPS%2C%20vous%20profitez,en%20payant%20le%20juste%20prix.>

- [13] linuxTricks (sans date), sur le site *Cron et crontab : le planificateur de tâches!* - Wiki - Wiki Consulté le 25 mai 2021
<https://www.linuxtricks.fr/wiki/cron-et-crontab-le-planificateur-de-taches>
- [14] Laurent Hercé (15 septembre 2020), sur le site *Conteneurisation informatique : définition, avantages, différence virtualisation, solutions* / Appvizer Consulté le 26 mai 2021
<https://www.appvizer.fr/magazine/services-informatiques/virtualisation/conteneurisation-informatique>
- [15] n/c (19 mai 2021), sur le site *fail2ban* [Wiki ubuntu-fr] Consulté le 26 mai 2021
<https://doc.ubuntu-fr.org/fail2ban>