



**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

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Contexte . . . . .	3
1.1.1	Client . . . . .	3
1.1.2	Solution existante . . . . .	3
1.2	Objectifs . . . . .	4
1.2.1	Court terme . . . . .	4
1.2.2	Long terme . . . . .	4
<b>2</b>	<b>Recueil d'information et Analyse</b>	<b>5</b>
2.1	User-stories . . . . .	6
2.1.1	Exemple . . . . .	7
2.1.2	Automatisation Trello . . . . .	8
2.2	Base de donnée . . . . .	8
<b>3</b>	<b>Méthodologie</b>	<b>10</b>
3.1	Agile . . . . .	10
3.2	Choix des technologies . . . . .	11
3.2.1	Backend . . . . .	11
3.2.2	Frontend . . . . .	11
3.2.3	Base de donnée . . . . .	12
3.2.4	Autre . . . . .	12
3.2.4.1	API . . . . .	12
3.2.4.2	Linter . . . . .	13
3.3	Outils . . . . .	13
<b>4</b>	<b>Développement</b>	<b>14</b>
4.1	Exemple concret . . . . .	14
4.2	Récapitulatif des fonctionnalités/US implémentées . . . . .	16
<b>5</b>	<b>Sécurité</b>	<b>18</b>
5.1	Contrôle d'accès . . . . .	18
5.2	Hachage et Chiffrement . . . . .	19
5.2.1	Données enregistrées . . . . .	19
5.2.2	HTTPS . . . . .	19
5.3	Security Headers . . . . .	20
5.4	Dependabot Github . . . . .	20
<b>6</b>	<b>Déploiement</b>	<b>21</b>
6.1	Première approche - Hosted Service . . . . .	21
6.2	Seconde approche - VPS . . . . .	21
6.2.1	Scripts . . . . .	21
6.2.2	Dockerisation . . . . .	21
6.2.3	Sécurité . . . . .	21
6.2.3.1	SSH . . . . .	21
6.2.3.2	Fail2ban . . . . .	21
6.2.3.3	Firewall (UFW) . . . . .	21
6.2.3.4	Firewall (OVH) . . . . .	21
6.3	Déploiement Continu . . . . .	21

<b>7</b>	<b>Migration données existantes</b>	<b>22</b>
7.1	Problématique . . . . .	22
7.2	Solution . . . . .	22
<b>8</b>	<b>Monitoring et backups</b>	<b>23</b>
<b>9</b>	<b>Pistes d'amélioration</b>	<b>24</b>
<b>10</b>	<b>Suite</b>	<b>25</b>
<b>11</b>	<b>Conclusion</b>	<b>26</b>
<b>12</b>	<b>Définitions</b>	<b>27</b>
<b>13</b>	<b>Bibliographie</b>	<b>28</b>

# 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 SI<sup>1</sup> 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 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. *"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]

## 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'encodage<sup>2</sup> 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"<sup>3</sup> 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-fonctionnalité
- 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 donnée 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 mockups<sup>4</sup>
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 Github<sup>5</sup> 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](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"<sup>6</sup> 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 continu<sup>7</sup>
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>

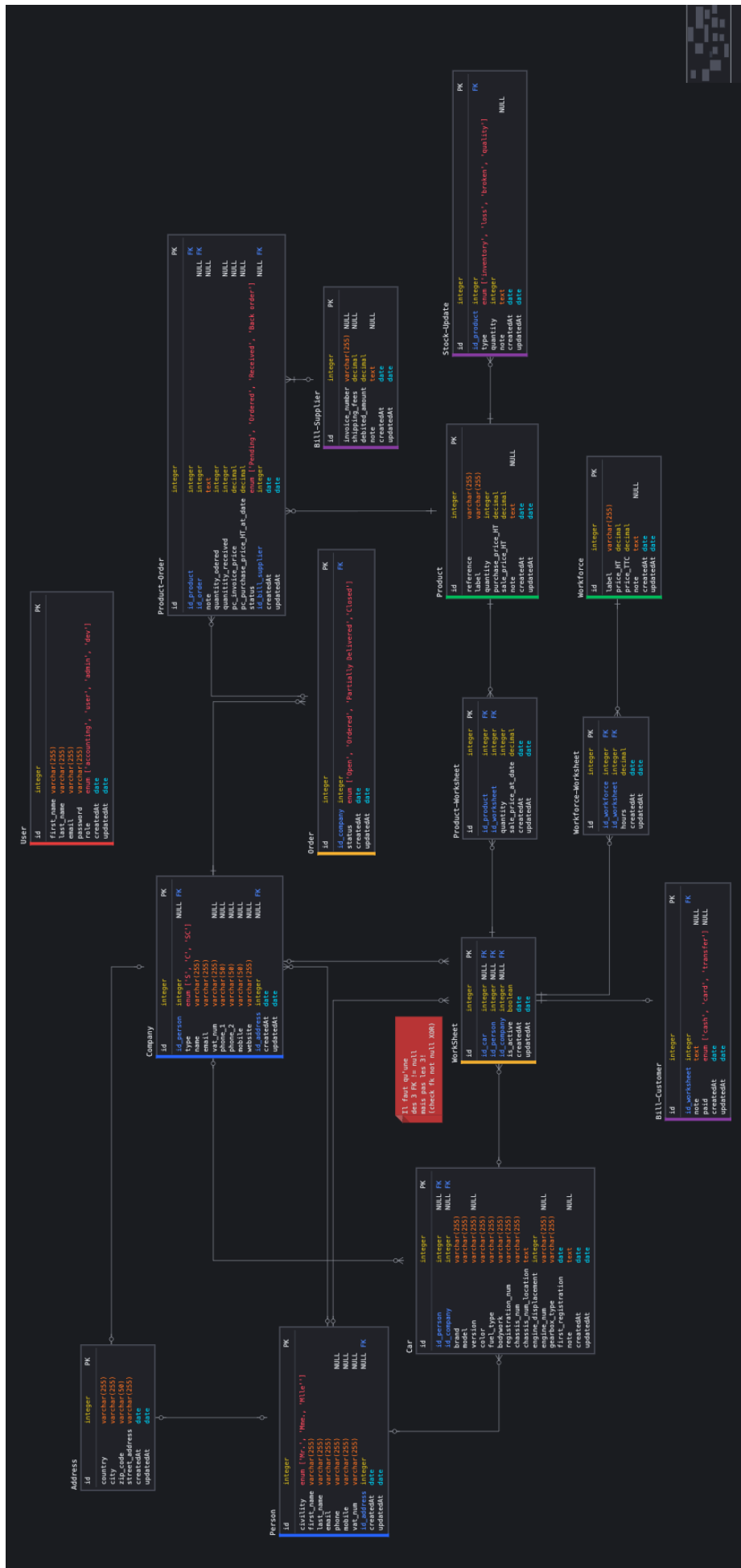


FIGURE 2 – Schéma de base de donnée

## 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 clients<sup>1</sup>. Elles s'appuient sur l'utilisation d'un cadre méthodologique léger mais suffisant centré sur l'humain et la communication<sup>2</sup>. 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.

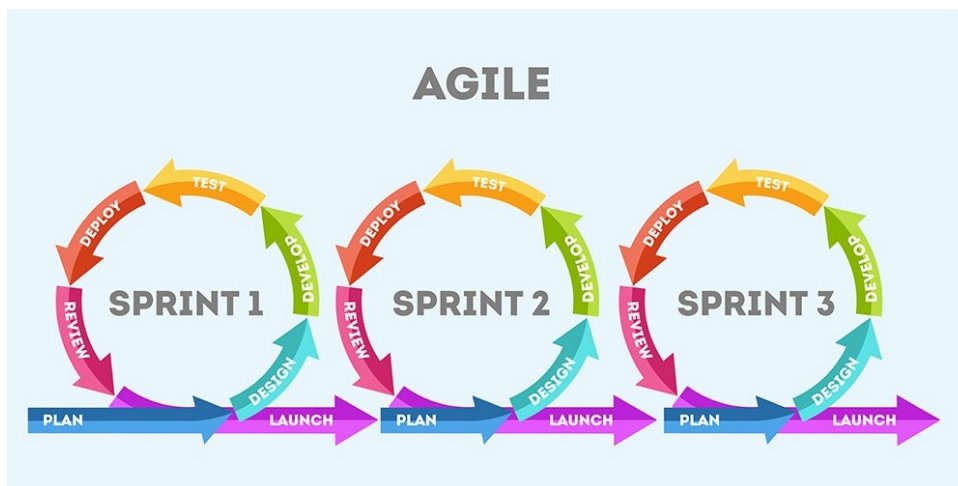


FIGURE 3 – Les raisons pour utiliser les méthodes Agile en entreprise de 300\_librarians

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 travaillé 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 telle envergure. J'ai dès 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**<sup>8</sup>. 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 donné 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-express<sup>9</sup>) me permet de générer automatiquement la documentation de mes endpoints sur base de quelques décorateurs<sup>10</sup> 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

## 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 le productivité :**

Afin de me simplifier le travail et/ou de ne pas avoir à exécuter des tâches répétitives, j'ai utilisé plusieurs outils tel qu'un linter<sup>11</sup> ou des extensions de mon IDE me proposant de l'autocomplétion avancée. Ce sont donc une série d'outils non-indispensables mais incontournables.

3. **implémentation des US**

4. **refactoring :**

Cette phase n'est pas une phase en tant que telle. 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"<sup>12</sup>. 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 détecter 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 équivalents 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 droits 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 certains droits ou non, l'application utilise des JSON Web Tokens (JWT)<sup>13</sup>. 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 clair <sup>14</sup>) 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) <sup>15</sup> afin de se protéger contre les attaques par table de correspondance (rainbow table) <sup>16</sup>. De plus elle est facilement paramétrable permettant ainsi d'augmenter la complexité de l'algorithme utilisé afin de palier aux attaques par force brute <sup>17</sup> 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).

---

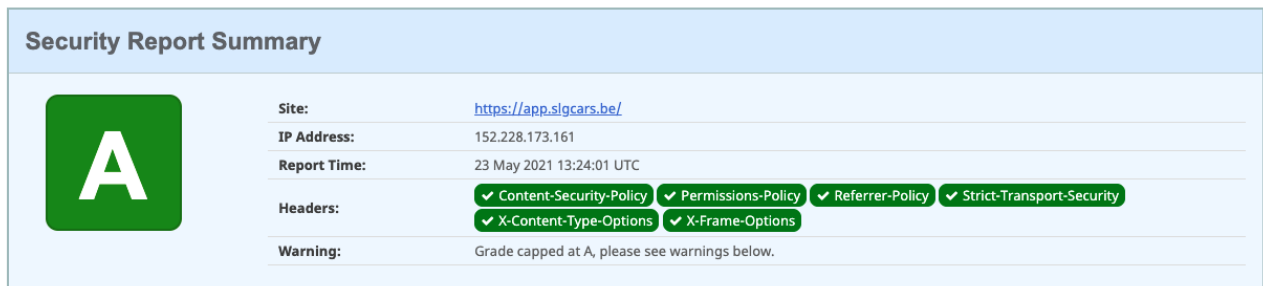
14. lisible et compréhensible par quiconque

15. *"Le salage, 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 Security Headers

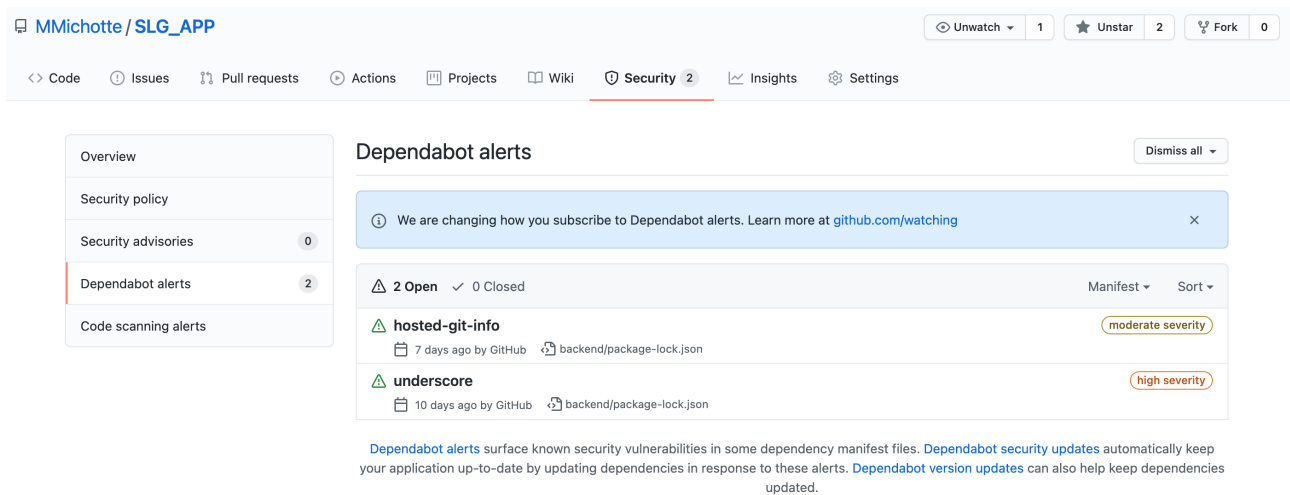


The image shows a 'Security Report Summary' for the website <https://app.slgcars.be/>. On the left is a large green square with a white letter 'A'. To the right, the report details are as follows:

Site:	<a href="https://app.slgcars.be/">https://app.slgcars.be/</a>
IP Address:	152.228.173.161
Report Time:	23 May 2021 13:24:01 UTC
Headers:	<div>✓ Content-Security-Policy ✓ Permissions-Policy ✓ Referrer-Policy ✓ Strict-Transport-Security ✓ X-Content-Type-Options ✓ X-Frame-Options</div>
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



The image shows the 'Dependabot alerts' page for the repository `MMichotte/SLG_APP`. The page has a navigation bar with links to Code, Issues, Pull requests, Actions, Projects, Wiki, Security (2), Insights, and Settings. On the left is a sidebar with links to Overview, Security policy, Security advisories (0), Dependabot alerts (2), and Code scanning alerts. The main content area is titled 'Dependabot alerts' and includes a 'Dismiss all' button. A blue notification banner states: 'We are changing how you subscribe to Dependabot alerts. Learn more at [github.com/watching](https://github.com/watching)'. Below this, there are two open alerts:

- hosted-git-info**: 7 days ago by GitHub, backend/package-lock.json, moderate severity.
- underscore**: 10 days ago by GitHub, backend/package-lock.json, high severity.

At the bottom, a note explains: 'Dependabot alerts surface known security vulnerabilities in some dependency manifest files. Dependabot security updates automatically keep your application up-to-date by updating dependencies in response to these alerts. Dependabot version updates can also help keep dependencies updated.'

FIGURE 5 – Alertes générées par "Dependabot"

## 6 Déploiement

### 6.1 Première approche - Hosted Service

### 6.2 Seconde approche - VPS

#### 6.2.1 Scirpts

#### 6.2.2 Dockerisation

#### 6.2.3 Sécurité

##### 6.2.3.1 SSH

##### 6.2.3.2 Fail2ban

##### 6.2.3.3 Firewall (UFW)

##### 6.2.3.4 Firewall (OVH)

### 6.3 Déploiement Continu

## 7 Migration données existantes

### 7.1 Problématique

### 7.2 Solution



## 8 Monitoring et backups

## 9 Pistes d'amélioration

## 10 Suite

## 11 Conclusion

## 12 Définitions

**US** : User Story

**CI** : Continuous Integration

**CD** : Continuous Deployment

**IDE** : Integrated Development Environment

## 13 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](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](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))