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

Rapporteur : M-N. VROMAN

Année académique 2020-2021

Remerciements

Table des matières

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 societé a besoin d'un SI [1] 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 intéragit 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 [2] 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 besoin 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 claire de chaque fonctionnalité en "user story" [3] Les "user stories" permettent de structurer l'ensemble des fonctionnalité 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 analyser 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 roles 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

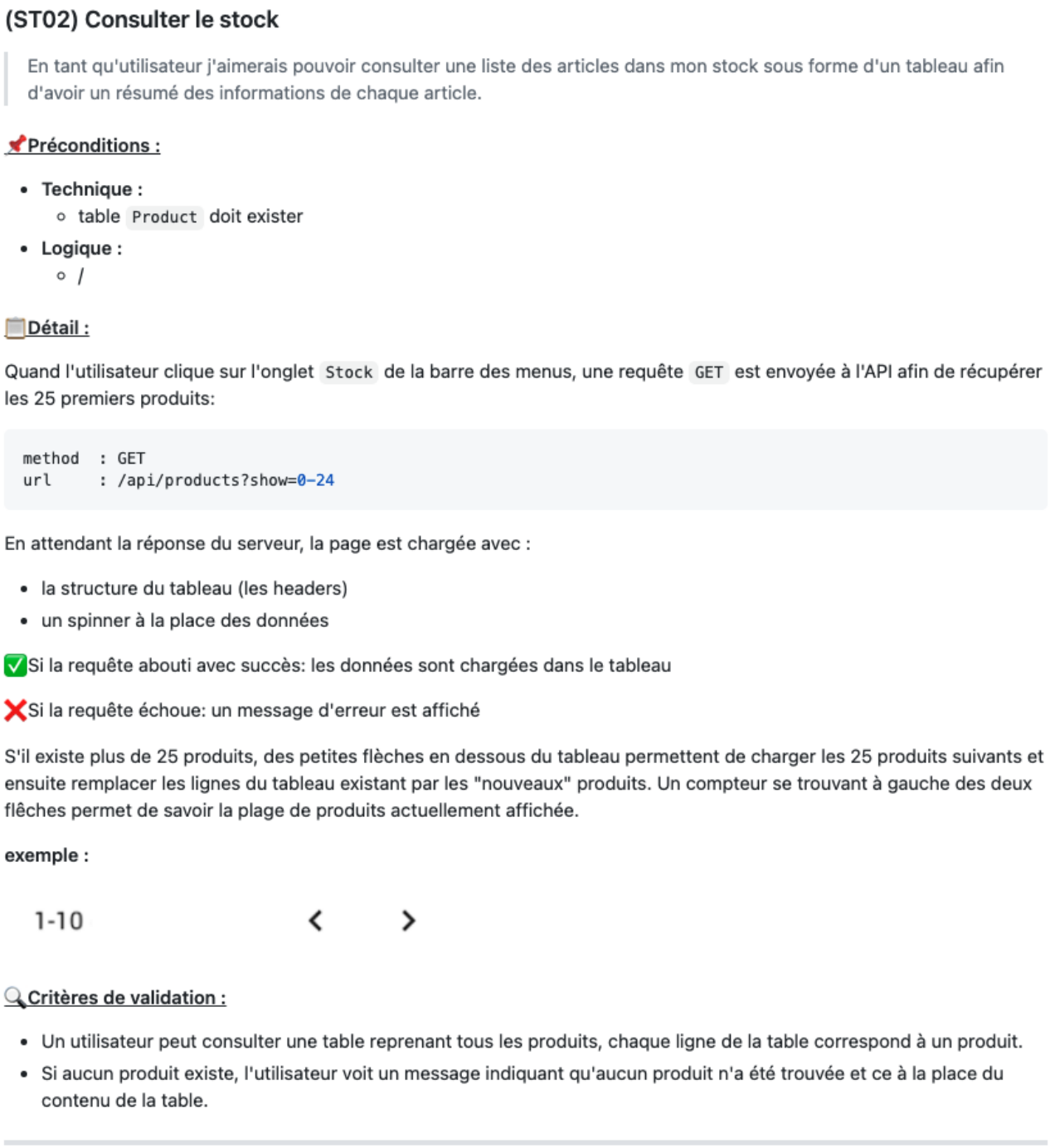
Comme expliqué précédemment, chaque fonctionnalité à été détaillée sous forme d'une "user story" (US). Dans un but organisationnel, j'ai décidé de grouper chaque US par grande fonctionnalité. Qu'entend-t-on par grande fonctionnalité ? Dans le cadre de ce projet, une grande fonctionnalité est un élément 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 un structure type comportant les éléments suivants :

1. Code unique + Titre
2. Description (du type : *"En tant que X j'aimers Y afin de Z"*)
3. Préconditions
4. Explication détaillée avec éventuellement des mockups [4]
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 flêches 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 [5] 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-transcrivait 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 continu [7]
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. //TODO

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`

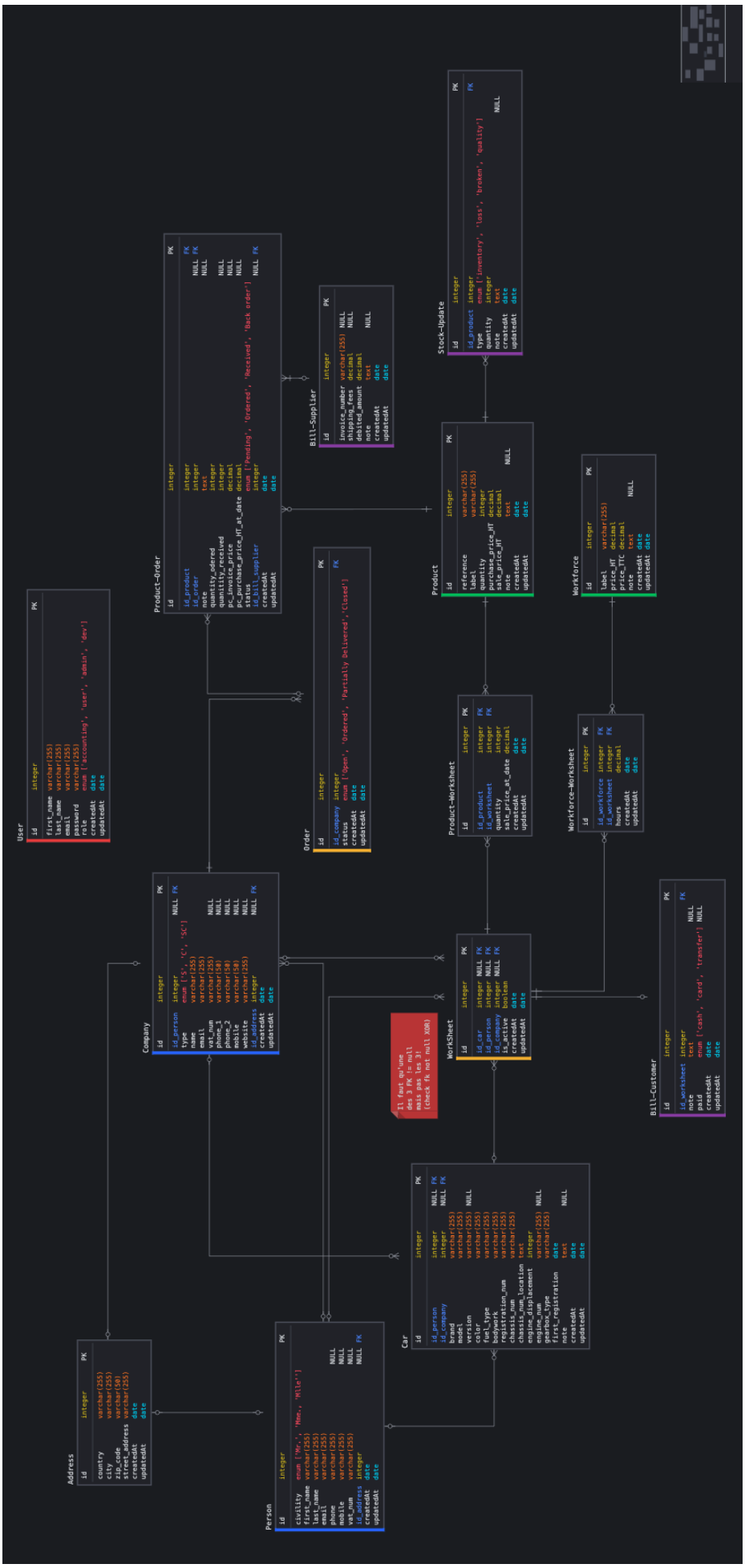


FIGURE 2 – Schéma de base de donnée

3 Méthodologie

3.1 Agile

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 préférait 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 différent 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 profésionnel 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 ressemble, 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-express [9]) me permet de générer automatiquement la documentation de mes endpoints sur base de quelques décorateurs [10] ajouté dans le code. En plus de cela, il me permet de de déployer cette documentation directement avec l'application. Celle-ci est dès lors consultable ici :
`https://app.slgcars.be/api-docs/`.

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

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

4 Développement

5 Sécurité

5.1 Chiffrement

5.2 Security Headers

5.3 Dependabot Github

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 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`