



CERTIFICATION PROFESSIONNELLE

Titre professionnel : Concepteur développeur d'applications  
RNCP: 37873

## DOSSIER PROJET

PROJET : MyCarTeam

Rédacteur : HUBERT Clément

Date : 24/02/2025

## SOMMAIRE

<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>2. CAHIER DES CHARGES.....</b>	<b>2</b>
2.1. Présentation du projet.....	2
2.2. Benchmarking.....	2
2.3. Description graphique et ergonomique.....	3
2.3.1. Charte graphique.....	3
2.3.1.1. Logo.....	3
2.3.1.2. Couleurs.....	3
2.3.1.3. Typographie.....	4
2.3.2. Zoning, wireframe et maquettes.....	5
2.3.2.1. Zoning.....	5
2.3.2.2. Wireframes.....	6
2.3.2.3. Maquettes.....	7
2.4. Besoins fonctionnels “métiers”.....	8
2.4.1. Public cible.....	8
2.4.1.1. Gérants TPE/PME.....	8
2.4.1.2. Responsables des ressources humaines.....	8
2.4.1.3. Salariés.....	9
2.4.2. Inventaire des besoins fonctionnels.....	9
2.4.2.1. Authentification et gestion des utilisateurs.....	9
2.4.2.2. Gestion de ses salariés.....	9
2.4.2.3. Gestion de son compte.....	10
2.4.2.4. Gestion des trajets.....	10
2.4.2.5. Gestion des concours.....	11
2.4.2.6. Notifications.....	11
2.4.2.7. Communication entre les utilisateurs.....	12
<b>3. GESTION DE PROJET.....</b>	<b>13</b>
3.1. Méthode SCRUMBAN.....	13
3.2. Le product Backlog.....	13
3.3. Les sprints.....	14
3.4. Outil de suivi d’avancement du sprint : Kanban.....	15
<b>4. SPÉCIFICATIONS FONCTIONNELLES.....</b>	<b>16</b>
4.1. Architecture logicielle du projet.....	16
4.2. Diagrammes UML.....	18
4.2.1. Diagramme de cas d’utilisation.....	18
4.2.2. Diagramme de séquence.....	19
4.2.3. Diagramme d’activité.....	20
4.2.4. Diagramme entité/relation.....	21
<b>5. SPECIFICATIONS TECHNIQUES.....</b>	<b>22</b>
5.1. Stack technique.....	22
<b>6. RÉALISATION DE LA FONCTIONNALITÉ RÉSERVATION.....</b>	<b>25</b>
6.1. Interface utilisateur.....	25

6.2. Route API : POST /booking.....	29
6.3. Controller.....	30
6.4. Service : la logique métier.....	31
6.5. Entité : logique entreprise.....	32
6.6. Repository : composant d'accès aux données.....	33
<b>7. ÉLÉMENTS DE SÉCURITÉ DE L'APPLICATION.....</b>	<b>34</b>
7.1. Faille SQL.....	34
7.2. Faille XSS.....	35
7.3. Faille CSRF.....	35
7.4. Hachage de mot de passe.....	36
7.5. Broken Access Control.....	37
7.5.1. Côté backend.....	37
7.5.2. Côté frontend.....	39
<b>8. PLAN DE TESTS.....</b>	<b>41</b>
<b>9. JEU D'ESSAI.....</b>	<b>42</b>
9.1. Description du scénario testé.....	42
9.2. Scénarios et résultats.....	42
<b>10. DÉMARCHE DEVOPS.....</b>	<b>44</b>
10.1. Introduction.....	44
10.2. Architecture et infrastructure.....	44
10.3. Mise en place du pipeline CI/CD.....	45
10.4. Focus sur les tests automatisés.....	48
<b>11. VEILLE SUR LES VULNÉRABILITÉS DE SÉCURITÉ.....</b>	<b>52</b>
<b>12. CONCLUSION.....</b>	<b>54</b>

# 1. INTRODUCTION

Ce dossier projet a pour objectif de présenter l'application **MyCarTeam**. Il s'agit d'un projet réalisé pendant la formation **Concepteur Développeur d'applications (CDA)** de l'établissement de formation **3W Academy**.

Ce dossier a été réalisé en tenant compte des exigences d'évaluation du titre professionnel "**Concepteur développeur d'applications**" (RNCP n° 37873).

Il a pour objet de démontrer l'acquisition des compétences suivantes:

- **Développer une application sécurisée** (voir point 7 pour la sécurité)
  - *Installer et configurer son environnement de travail en fonction du projet*
  - Développer des interfaces utilisateurs (voir point 6.1)
  - *Développer des composants métier* (voir point 6..)
  - *Contribuer à la gestion d'un projet informatique* (voir point 3)
- **Concevoir et développer une application sécurisée organisée en couches**
  - *Analyser les besoins et maquetter une application* (voir points 2.3.2 et 2.4)
  - *Définir l'architecture logicielle d'une application* (voir point 4.1)
  - *Concevoir et mettre en place une base de données relationnelle* (voir point 4.2.4)
  - *Développer des composants d'accès aux données SQL et NoSQL* (voir point 6.6)
- **Préparer le déploiement d'une application sécurisée**
  - *Préparer et exécuter les plans de tests d'une application* (voir points 8 et 9)
  - *Préparer et documenter le déploiement d'une application* (voir point 10)
  - *Contribuer à la mise en production dans une démarche DevOps* (voir point 10)

Je tiens à remercier l'ensemble des formateurs pour leur passion communicative.

Un remerciement particulier à Monsieur ROUSSEAU Romain, mon tuteur d'alternance, pour la chance qu'il m'a donnée en m'acceptant comme alternant, il y a deux ans. Cette alternance m'a apporté énormément professionnellement comme humainement.

## **2. CAHIER DES CHARGES**

### **2.1. Présentation du projet**

MyCarTeam est une application de covoiturage professionnel permettant la mise en relation de salariés d'une même entreprise souhaitant covoiturer.

L'application doit être simple et ludique à l'utilisation.

En effet, les utilisateurs seront des salariés. Ces derniers n'ont que peu de temps à accorder à l'organisation de leurs trajets professionnels en raison de leurs activités. Il faut donc que l'application soit la plus simple possible à l'utilisation afin de minimiser le temps passé à proposer ou s'inscrire à des trajets en covoiturage.

De plus, elle doit être ludique pour donner envie au salarié d'utiliser le service dans le temps. Cette application ayant pour objectif de réduire l'empreinte carbone directe et indirecte de l'entreprise en modifiant les habitudes de transport de leurs salariés, il faut inciter ces derniers à se connecter et utiliser régulièrement MyCarTeam.

D'après le ministère de la Transition écologique, de la Biodiversité, de la Forêt, de la Mer et de la Pêche, 70% des trajets domicile-travail sont réalisés en véhicule individuel et seulement 3% en covoiturage<sup>1</sup>. Ce projet répond à cette problématique.

### **2.2. Benchmarking**

Une recherche sur les mots clés "covoiturage professionnel" fait apparaître plusieurs acteurs sur ce marché comme Klakit (Groupe Blablacar) ou Karos. Mais aucune ne propose aux entreprises de créer un espace pour leurs salariés. Cet élément différenciant sera pris en compte dans la conception de l'application.

---

<sup>1</sup> . Ministère de la Transition écologique, de la Biodiversité, de la Forêt, de la Mer et de la Pêche (2019). *Le covoiturage en France, ses avantages et la réglementation en vigueur.*. [EN LIGNE]. <https://www.ecologie.gouv.fr/politiques-publiques/covoiturage-france-ses-avantages-reglementation-vigueur> (Page consultée le 16/02/2025).

## 2.3. Description graphique et ergonomique

### 2.3.1. Charte graphique

#### 2.3.1.1. Logo



Le logo reprend le nom de l'application agrémenté d'un point pour représenter l'idée que l'application suffit en elle-même pour le covoiturage professionnel. Il s'agit d'une application tout en un.

Il reprend aussi les deux couleurs principales de l'application sous forme de deux ronds qui semblent fusionner. L'idée exprimée ici est que MyCarTeam a pour objectif de favoriser la rencontre entre un conducteur et un passager.

#### 2.3.1.2. Couleurs

<b>27AA68</b> <small>Jade</small>	<b>FFD012</b> <small>Jonquil</small>	<b>000000</b> <small>Black</small>	<b>FFFFFF</b> <small>White</small>
--------------------------------------	---	---------------------------------------	---------------------------------------

La couleur “*jade*” est utilisée pour les éléments importants comme les titres (<h1>, <h2>, etc.).

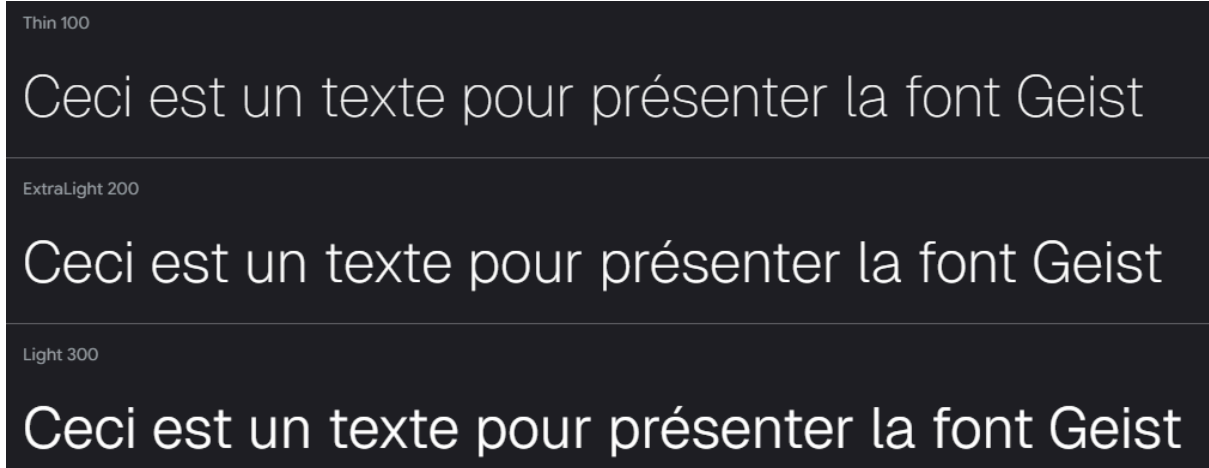
La couleur “*jonquil*” est utilisée pour attirer l'attention des utilisateurs. Il sera principalement utilisé pour les éléments <button>.

Les couleurs noire et blanche sont utilisées pour les textes.

La couleur blanche sera utilisée pour le fond.

### 2.3.1.3. Typographie

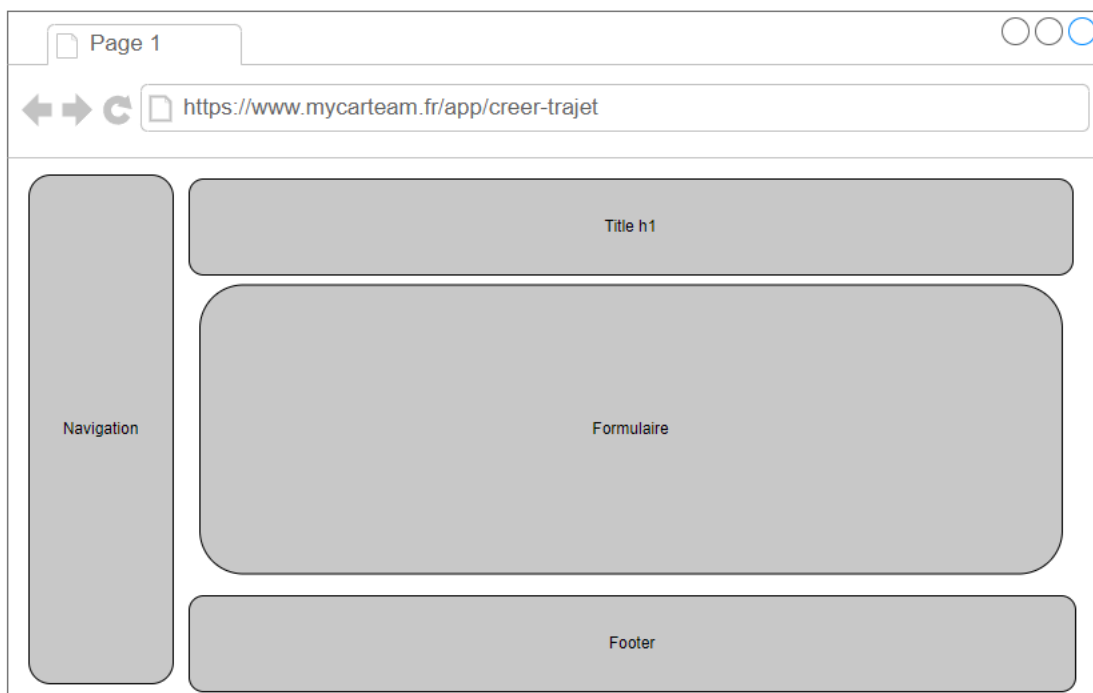
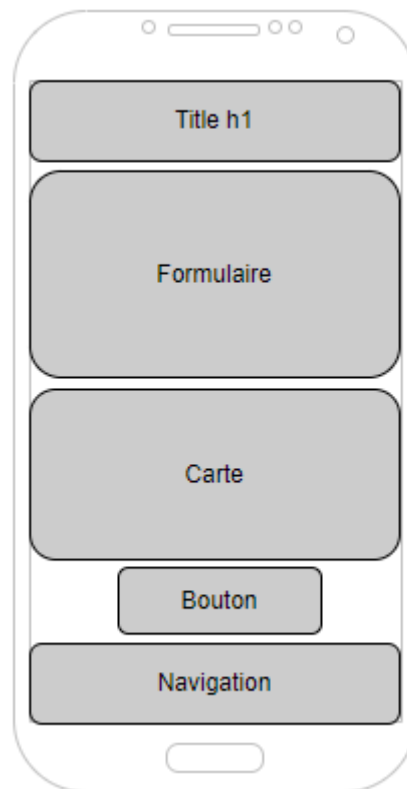
La typographie utilisée est la **Geist Sans**. Elle a été choisie pour sa simplicité, sa lisibilité sur écran et ses 9 variantes de graisse (weights).



## 2.3.2. Zoning, wireframe et maquettes

### 2.3.2.1. Zoning

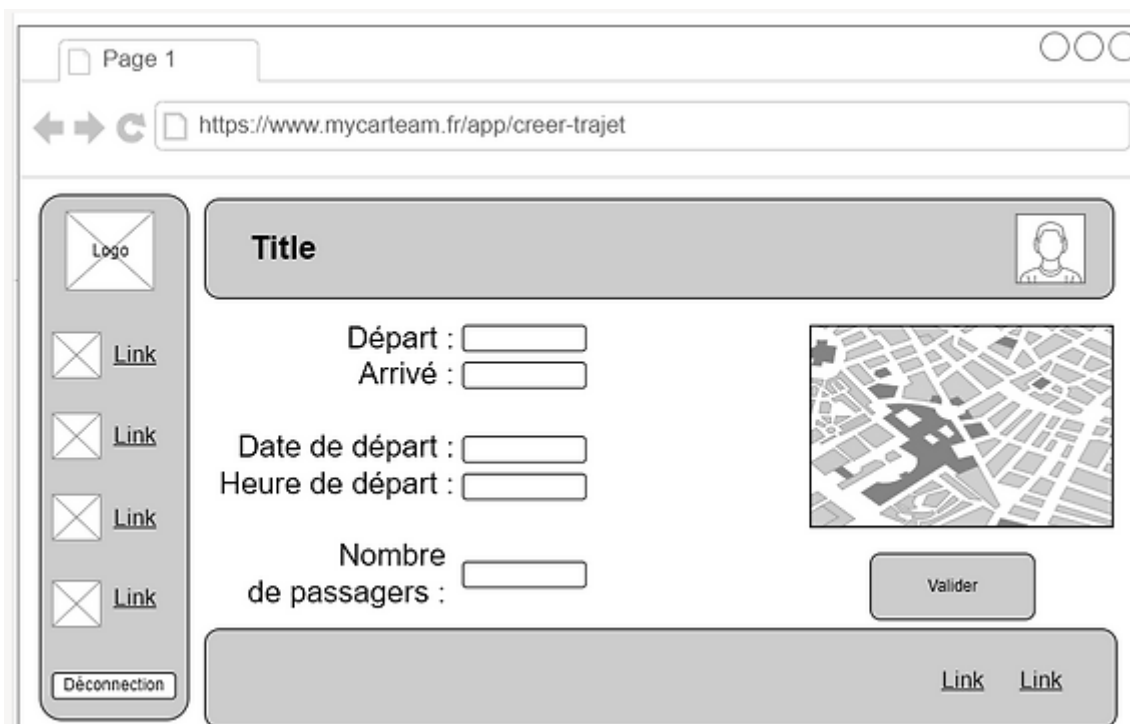
Zoning réalisés avec le logiciel **Draw.io**.





### 2.3.2.2. Wireframes

Wireframes réalisés avec le logiciel **Draw.io**.



### 2.3.2.3. Maquettes

Réalisée avec l'application web **Figma**.

The image displays two wireframe mockups of a web application interface for proposing a new trip. Both mockups feature a green header bar with the text "Proposer un nouveau trajet".

**Left Mockup:**

- Header:** "Proposer un nouveau trajet" in white text on a green background.
- Form Fields:**
  - Départ:** A text input field with the placeholder "Entrez une adresse".
  - Arrivé:** A text input field with the placeholder "Entrez une adresse".
  - Date et heure de départ:** A date and time picker field with the placeholder "jj/mm/aaaa --:--".
  - Nombre de passagers (conducteur exclu):** A numeric input field with the value "1".
  - Prix:** A text input field.
- Map:** A map of Paris showing the city grid and landmarks.
- Buttons:** A yellow "Valider" button at the bottom right.
- Footer:** A green bar with three icons: a person, a house, and a location pin.

**Right Mockup:**


- Header:** "Proposer un nouveau trajet" in white text on a green background, with a "CH" button in the top right corner.
- Form Fields:**
  - Départ:** A text input field with the placeholder "Entrez une adresse".
  - Arrivé:** A text input field with the placeholder "Entrez une adresse".
  - Date et heure de départ:** A date and time picker field with the placeholder "jj/mm/aaaa --:--".
  - Nombre de passagers (conducteur exclu):** A numeric input field with the value "1".
  - Prix:** A text input field.
- Map:** A map of Paris showing the city grid and landmarks.
- Buttons:** A yellow "Valider" button at the bottom right.
- Footer:** A green bar with the text "©MyCarTeam Accueil Mentions légales" and a red "Deconnexion" button in the bottom left corner.

## 2.4. Besoins fonctionnels “métiers”


### 2.4.1. Public cible

MyCarTeam cible les entreprises et leurs salariés. Les entreprises peuvent être représentées par leurs gérants, si TPE/PME, ou par un responsable de ressources humaines pour les plus grandes structures.


#### 2.4.1.1. Gérants TPE/PME

 <b>Sandrine R.</b>  Gérante d'une agence immobilière 47 ans Licence en immobilier Taille de l'entreprise : 7 salariés	Besoins spécifiques	<ul style="list-style-type: none"><li>• Modifier rapidement ses trajets</li><li>• Centraliser les documents légaux (permis, assurance).</li></ul>
	Scénarios d'utilisation	<ul style="list-style-type: none"><li>• Ajout d'un nouveau salarié</li><li>• Modifications de ses trajets</li></ul>
	Attentes en termes d'expérience utilisateur	<ul style="list-style-type: none"><li>• Application ergonomique pour faire ses actions rapidement.</li><li>• Assistance réactive en cas de problème</li></ul>

#### 2.4.1.2. Responsables des ressources humaines

 <b>Maeva U.</b>  Responsable des ressources humaines 32 ans Master en RH. Taille de l'entreprise : 42 salariés	Besoins spécifiques	<ul style="list-style-type: none"><li>• Gestion des utilisateurs</li><li>• Gestion de concours</li></ul>
	Scénarios d'utilisation	<ul style="list-style-type: none"><li>• Créer des employés</li><li>• Supprimer des employés.</li><li>• Voir les résultats des concours</li></ul>
	Attentes en termes d'expérience utilisateur	<ul style="list-style-type: none"><li>• Fiabilité</li><li>• Sécurité</li></ul>

### 2.4.1.3. Salariés

 <p><b>Alexandre P.</b></p> <p>Employé - Développeur web 22 ans Titre RNCP CDA Taille de l'entreprise : 15 salariés</p>	Besoins spécifiques	<ul style="list-style-type: none"><li>• Rechercher des trajets</li></ul>
	Scénarios d'utilisation	<ul style="list-style-type: none"><li>• Rechercher un trajet et réserver une place sur trajet travail-domicile</li></ul>
	Attentes en termes d'expérience utilisateur	<ul style="list-style-type: none"><li>• Simplicité</li><li>• Rapidité</li></ul>

## 2.4.2 Inventaire des besoins fonctionnels

### 2.4.2.1. Authentification et gestion des utilisateurs.

- Inscription

En tant que visiteur, je souhaite créer un compte pour pouvoir utiliser l'application.

- Connexion

En tant qu'utilisateur non connecté, je souhaite me connecter à l'application avec mon identifiant et mon mot de passe afin d'accéder à mon tableau de bord.

### 2.4.2.2. Gestion de ses salariés

- Création de comptes salariés

En tant que gestionnaire de compte d'entreprise, je souhaite créer des comptes pour les salariés de mon entreprise afin qu'ils puissent utiliser l'application.

- Modification des salariés

En tant que gestionnaire de compte d'entreprise, je souhaite modifier les informations liées à un employé afin de corriger des erreurs ou prendre en compte un changement de statut.

- Consultation des salariés

En tant que gestionnaire de compte d'entreprise, je souhaite consulter la liste des employés de mon entreprise ayant un compte sur l'application afin de voir les informations les concernant.

- Suppression des salariés

En tant que gestionnaire de compte d'entreprise, je souhaite supprimer les comptes des employés (en cas de bannissement ou de démission) afin de mettre à jour la liste des salariés.

#### 2.4.2.3. Gestion de son compte

- Modification informations de connexion

En tant qu'utilisateur, je souhaite pouvoir modifier mon identifiant et mon mot de passe afin de sécuriser les accès si nécessaire.

- Téléversement des documents légaux

En tant que salarié, je souhaite pouvoir téléverser mon permis de conduire, ma carte grise et mon attestation d'assurance afin d'obtenir le droit de proposer des trajets de covoiturage en tant que conducteur.

En tant que gestionnaire de compte d'entreprise, je souhaite pouvoir voir les documents téléversés par les employés de mon entreprise afin de leur accorder le droit de proposer des trajets.

#### 2.4.2.4. Gestion des trajets

- Création de trajet

En tant que salarié, je souhaite créer un trajet en précisant le lieu de départ, la destination, la date et l'heure afin de proposer un service de covoiturage à mes collègues.

- Consultation et recherche des trajets

En tant que salarié, je souhaite consulter et rechercher les trajets disponibles en fonction de critères tels que le lieu et la date afin de trouver un trajet adapté à mes besoins.

- Validation des passagers

En tant que salarié, je souhaite valider les demandes de covoiturage des passagers à mes trajets afin de confirmer leur participation.

#### 2.4.2.5. Gestion des concours

- Paramétrage des concours

En tant que gestionnaire de compte d'entreprise, je souhaite paramétrer les concours en définissant la durée et les récompenses afin de motiver la participation des salariés de mon entreprise.

- Consultation des concours

En tant que salarié, je souhaite consulter les concours en cours et passés afin de connaître les résultats et les récompenses en jeu.

- Suppression des concours

En tant que gestionnaire de compte d'entreprise, je souhaite supprimer les concours obsolètes ou annulés afin de maintenir une liste de concours à jour.

#### 2.4.2.6. Notifications

- Notification de nouveau passager

En tant que salarié, je souhaite recevoir une notification lorsqu'un nouveau passager s'inscrit à un trajet afin d'être informé et pouvoir le valider à temps.

- Notification d'acceptation de la demande de covoiturage.

En tant que salarié, je souhaite recevoir une notification dès que ma demande de covoiturage est acceptée afin de suivre l'évolution de ma réservation.

#### 2.4.2.7. Communication entre les utilisateurs

- Envoie de messages

En tant qu'utilisateur, je souhaite envoyer des messages aux autres utilisateurs afin de faciliter la communication concernant les trajets et l'administration de l'application.

- Lecture de messages

En tant qu'utilisateur, je souhaite consulter mes messages reçus afin d'y répondre si nécessaire.

## 3. GESTION DE PROJET

### 3.1. Méthode SCRUMBAN

Le choix a été fait d'utiliser les approches **SCRUM** et **KANBAN** (**SCRUMBAN**) pour la gestion du projet. Il s'agit d'une **méthode agile**. Elle a été choisie pour son **approche itérative** et sa **capacité d'adaptation** aux changements. Comme toute méthode issue du manifeste agile, elle met l'accent sur les individus et leurs interactions, la collaboration avec les clients, la réponse au changement et tend à obtenir un logiciel fonctionnel.

Il s'agit d'un projet personnel réalisé en autonomie, j'endosse donc les différents rôles : le **Scrum Master**, le **Product Owner** et l'équipe de **développeurs**.

En tant que **Scrum Master**, je suis concentré au respect des principes et valeurs du SCRUM.

Lorsque je suis dans le rôle du **Product Owner**, je définis les spécificités fonctionnelles de l'application par rapport aux objectifs de l'application. J'édite le **backlog** et le tiens à jour, tout au long du projet.

En tant que **développeur** je respecte la répartition des tâches (en sprint) et tiens les délais.

### 3.2. Le product Backlog

Il s'agit de la première étape. En tant que Product Owner j'**analyse les besoins** par rapport au résumé du projet (résumé qui ici peut représenter le discours du client).

Une fois cette analyse terminée, j'**identifie toutes les fonctionnalités** de l'application. C'est ce qu'on appelle les **user stories**. (cf. *inventaires des besoins fonctionnels*).

Exemple de backlog :

#### **PB-9 : Création de trajet**

*User Story :*

"En tant que salarié, je souhaite créer un trajet en précisant le lieu de départ, la destination, la date et l'heure afin de proposer un service de covoiturage à mes collègues."

*Description :* Formulaire de création de trajet avec les champs requis.



*Critères d'acceptation:*

- Validation des champs obligatoires
- Confirmation de création et affichage dans le tableau de bord rubrique “mes trajets”

### 3.3. Les sprints

Une fois le backlog défini, je sélectionne, les users stories et tâches à réaliser pendant le sprint. Il s'agit du **sprint meeting**.

Ici, la durée de chaque sprint a été définie à une semaine.

Le **but** à atteindre à la fin de chaque sprint : la **livraison** d'une démonstration fonctionnelle. Cette **approche itérative** permet un retour rapide du client qui permettra de modifier les sprints au fur et à mesure. Cette approche explique la démarche DevOps et la mise en place (CI/CD) (cf. partie *DevOps*).

En cours de sprint, chaque jour, je prenais le temps (15 minutes maximum) de faire le point sur ce qui a été fait la veille, les problématiques que j'ai rencontrées et sur ce que je vais faire aujourd'hui. C'est le **daily scrum**.

Lors du sprint j'ai utilisé un tableau **Kanban** (cf. Outil de suivi d'avancement du sprint : KaNban) pour m'organiser visuellement.

A la fin du sprint, je teste les fonctionnalités (normalement rôle de l'équipe de développement et du Product Owner). Puis je livre ce qui a été développé et je teste à nouveau mais dans une vision client.

Exemple du Sprint 1 :

Tâches :

- Mise en place de l'environnement de développement ainsi que des outils comme Docker, GitHub)
- Définition de l'architecture logicielle
- Configuration de l'environnement de test et d'intégration continue.

User stories :

- En tant que visiteur, je souhaite créer un compte pour pouvoir utiliser l'application.
- En tant qu'utilisateur non connecté, je souhaite me connecter à l'application avec mon identifiant et mon mot de passe afin d'accéder à mon tableau de bord.

Critères d'acceptations :

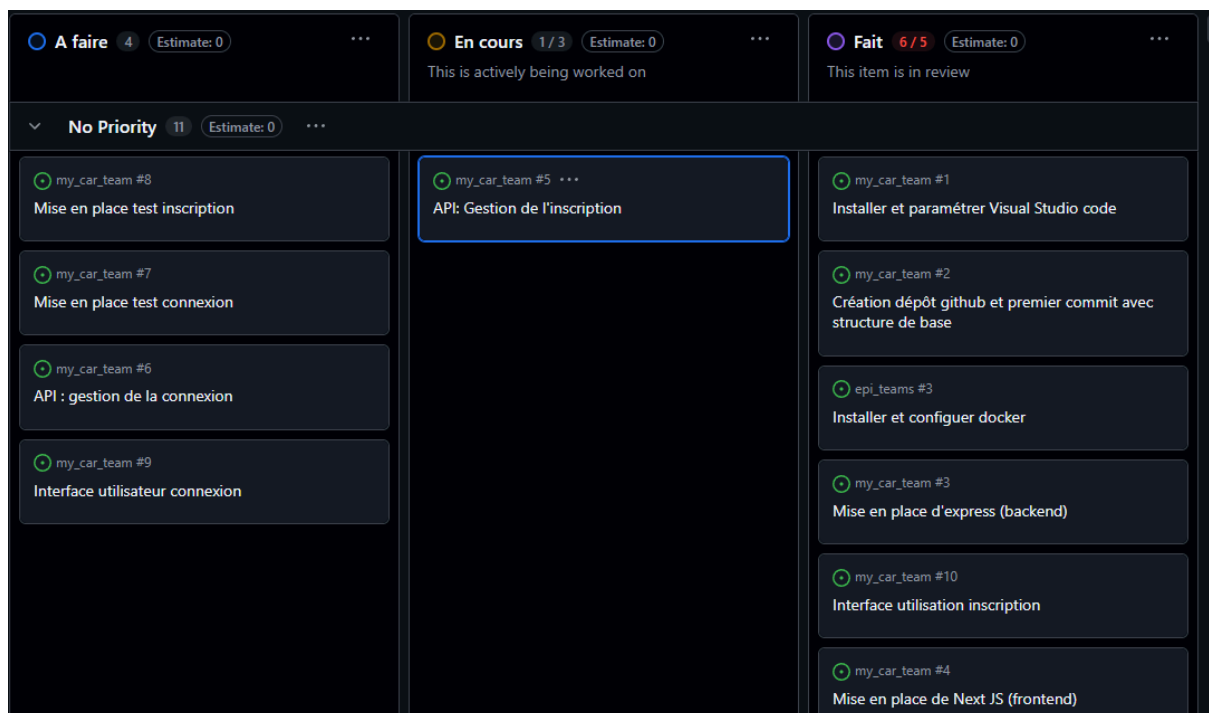
- Si l'utilisateur remplit mal le formulaire d'inscription, il voit un message l'invitant à modifier ses entrées.
- Si l'utilisateur est correctement connecté, alors il est redirigé vers son tableau de bord.

### 3.4. Outil de suivi d'avancement du sprint : Kanban.

À chaque sprint est mis en place un tableau Kanban qui permet une **visualisation du flux de travail**. Il est composé de trois colonnes : A faire, en cours et terminé. Chaque colonne contient des **cartes** (appelées aussi **tickets**). Chaque carte représente une tâche.

J'utilise "Projects" dans **GitHub** pour faire le tableau Kanban.

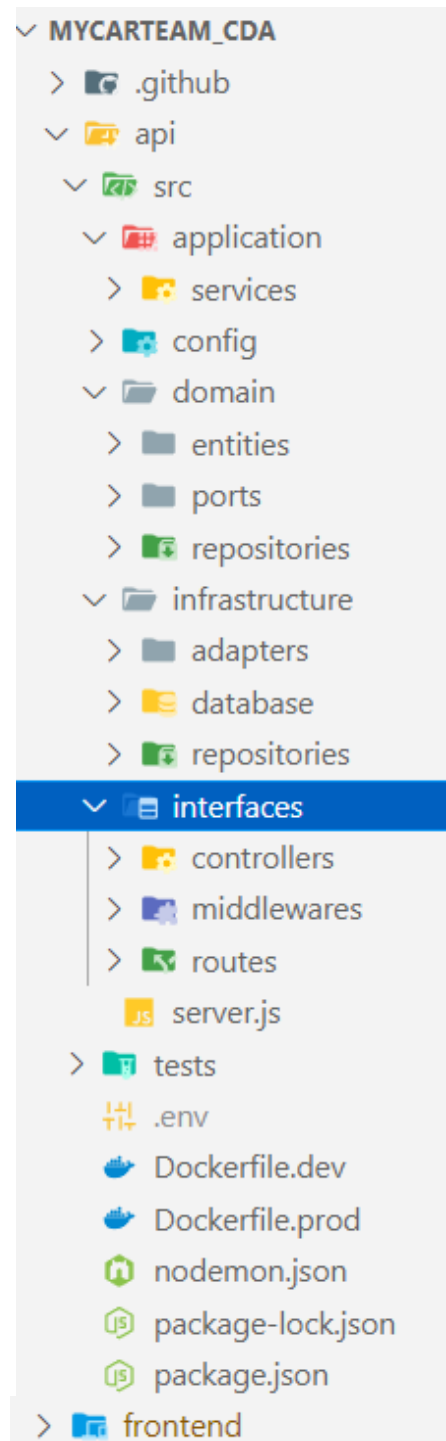
Exemple du tableau **Kanban** au cours du sprint 1.



## 4. SPÉCIFICATIONS FONCTIONNELLES

### 4.1. Architecture logicielle du projet

L'application est **organisée en couches** et respecte les principes de la **Clean Architecture**.



**/api** : API Express.

**/api/src/domain** : *Entities*

Contient le coeur métier et les règles entreprises

**/api/src/application** : *Use Cases*

Contient la logique applicative. Les cas d'utilisation orchestrent les entités.

**/api/src/interface** : *Interface Adapters*

Son objectif est de récupérer les données extérieures afin d'être utilisables par les uses cases.

Cette couche va aussi formater les réponses des cas d'utilisation pour le client.

**/api/src/infrastructure** : *Frameworks and Drivers*

Fournit les outils nécessaires pour le fonctionnement de l'API.

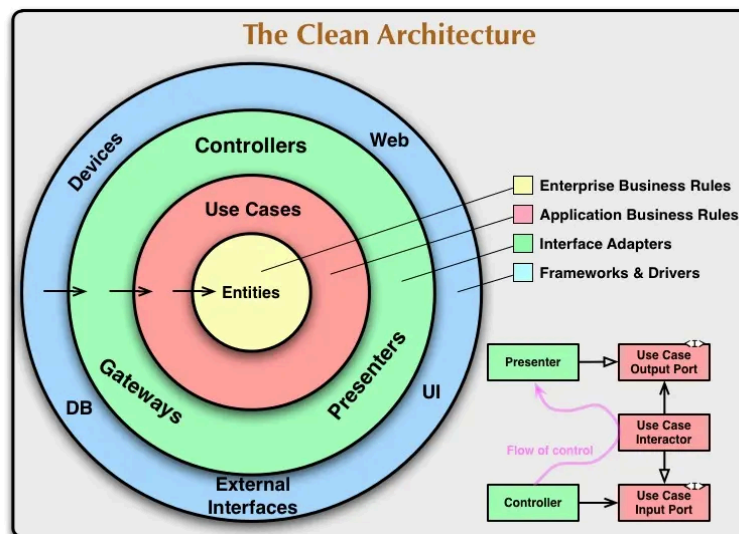
**/frontend** : *couche de présentation*.

Framework utilisé : Next.JS.

C'est ici qu'est développée les interfaces utilisateurs.

Pourquoi ce choix d'architecture ? Car elle respecte les principes SOLID, par exemple :

- **Single Responsibility Principle** : chaque couche a une responsabilité et une seule. Par exemple, la **couche Entities** contient la logique métier alors que la **couche Interface Adapters** a pour responsabilité de formater les données entrantes et sortantes.
- **Open/Closed Principle**: on peut ajouter un nouveau type de base de données en créant de nouvelles implémentations d'interface sans toucher à la couche Entities, par exemple.
- **Liskov Substitution Principle**: une classe dérivée doit pouvoir être substituée à sa classe mère sans modifier le programme. Ici, le code métier dépend d'interfaces abstraites. On peut facilement changer les implémentations.
- **Dependency Inversion Principle**: les dépendances pointent toujours vers l'intérieur (voir schéma ci-dessous).

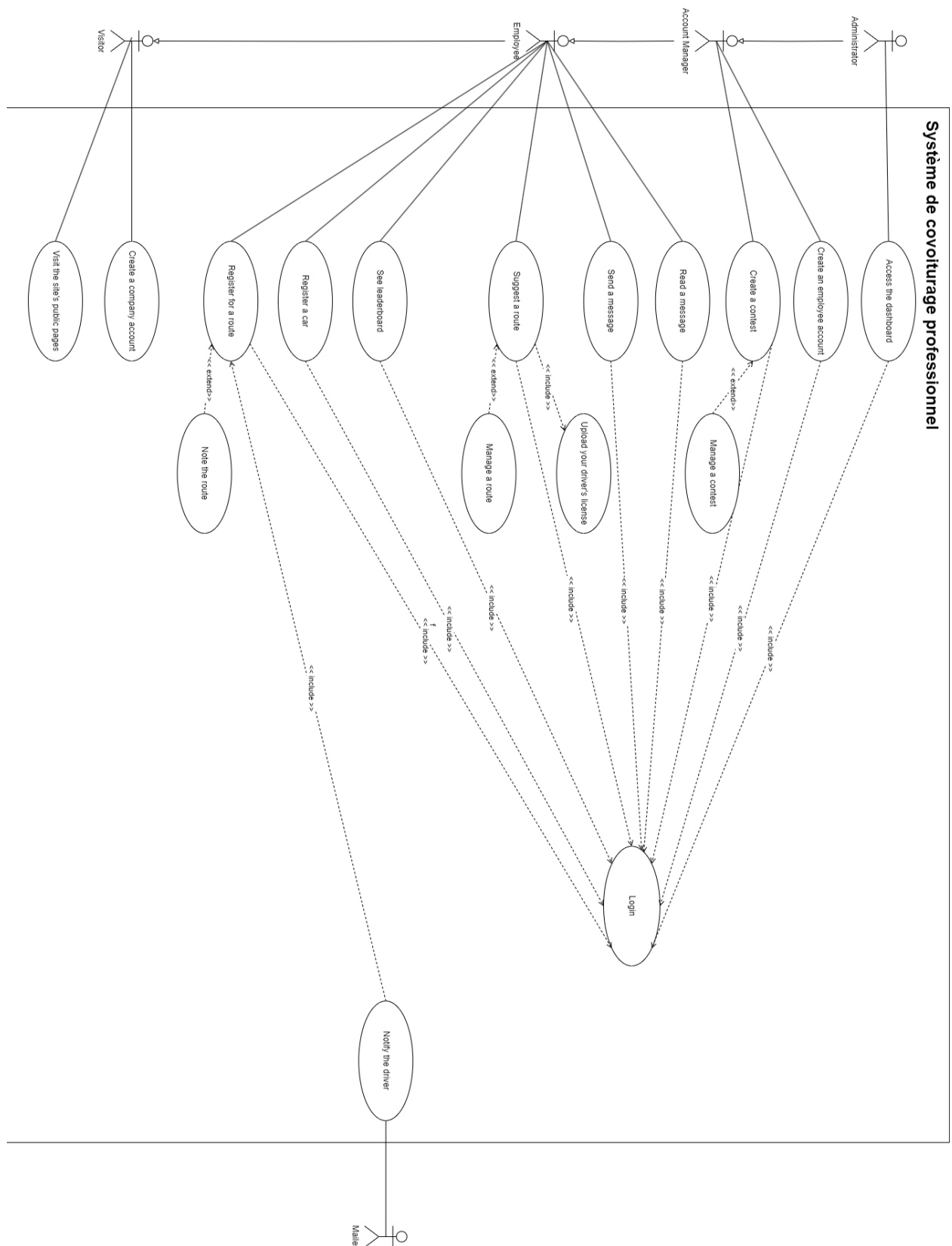


Robert C. Martin Clean Architecture Diagram - Clean Architecture  
A Craftsman's Guide to Software Structure and Design ed. Addison-Wesley

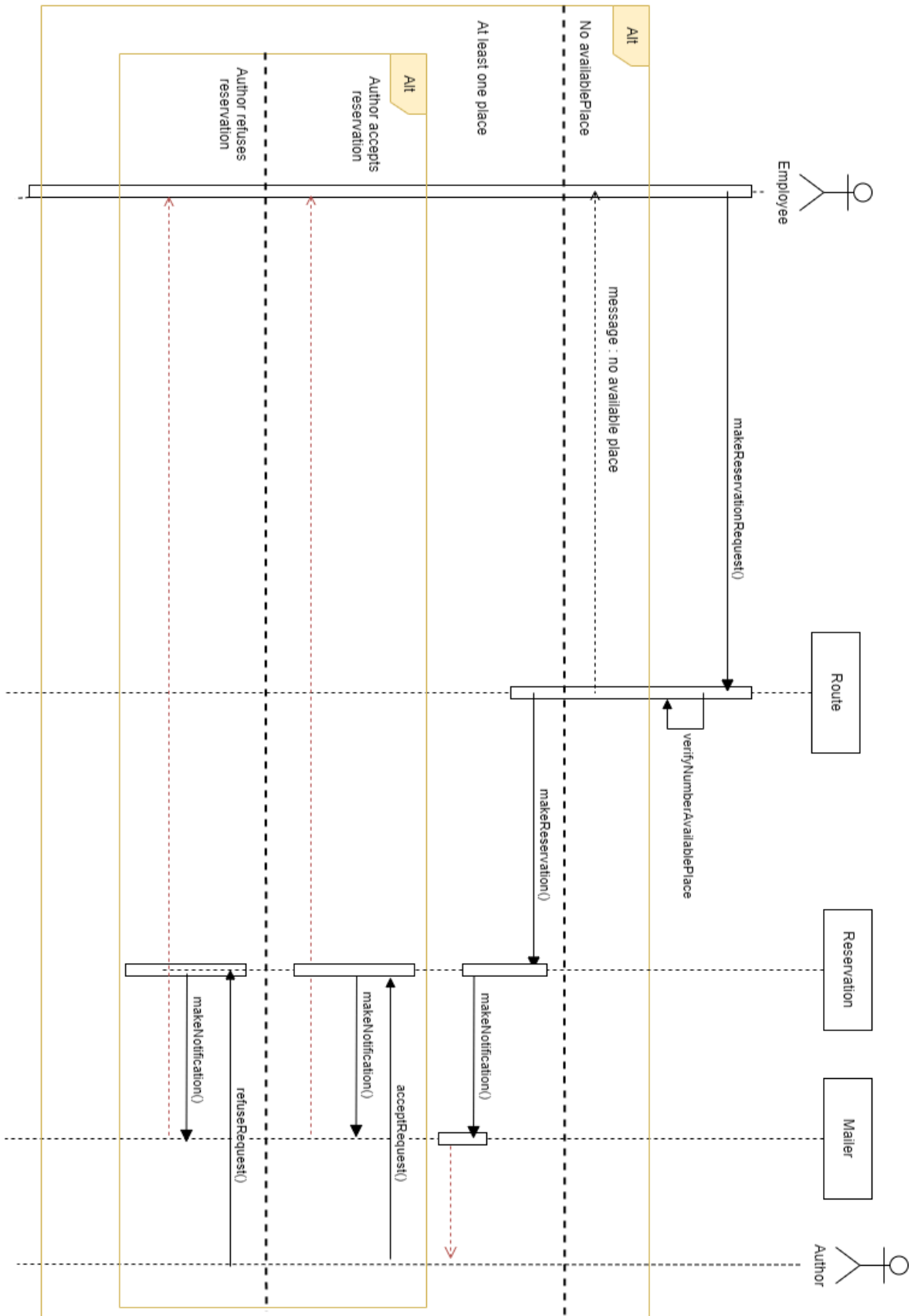
Le respect de ces principes permet d'avoir une application plus facilement testable, maintenable, indépendante des frameworks, bases de données, de l'UI etc.

## 4.2. Diagrammes UML

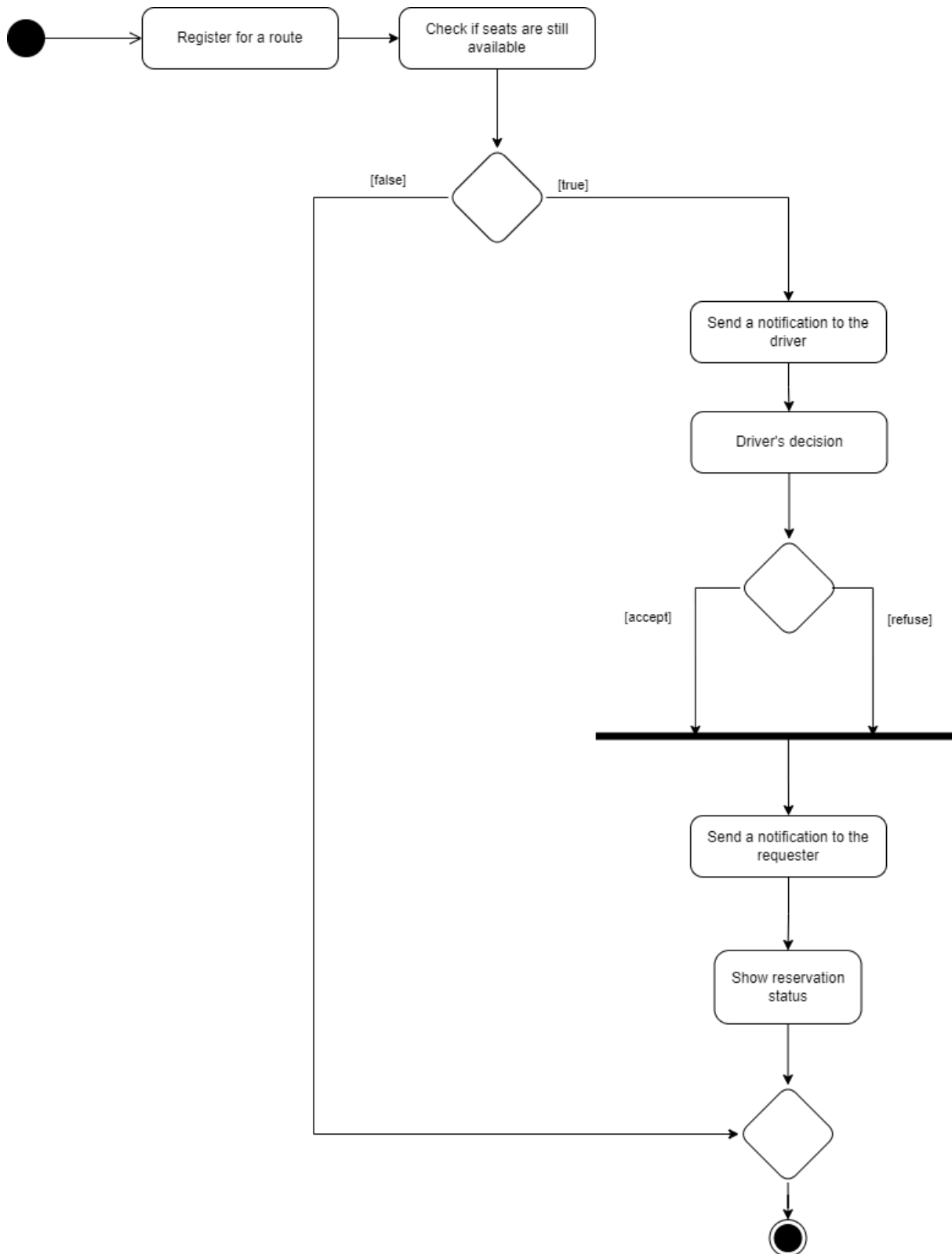
### 4.2.1. Diagramme de cas d'utilisation



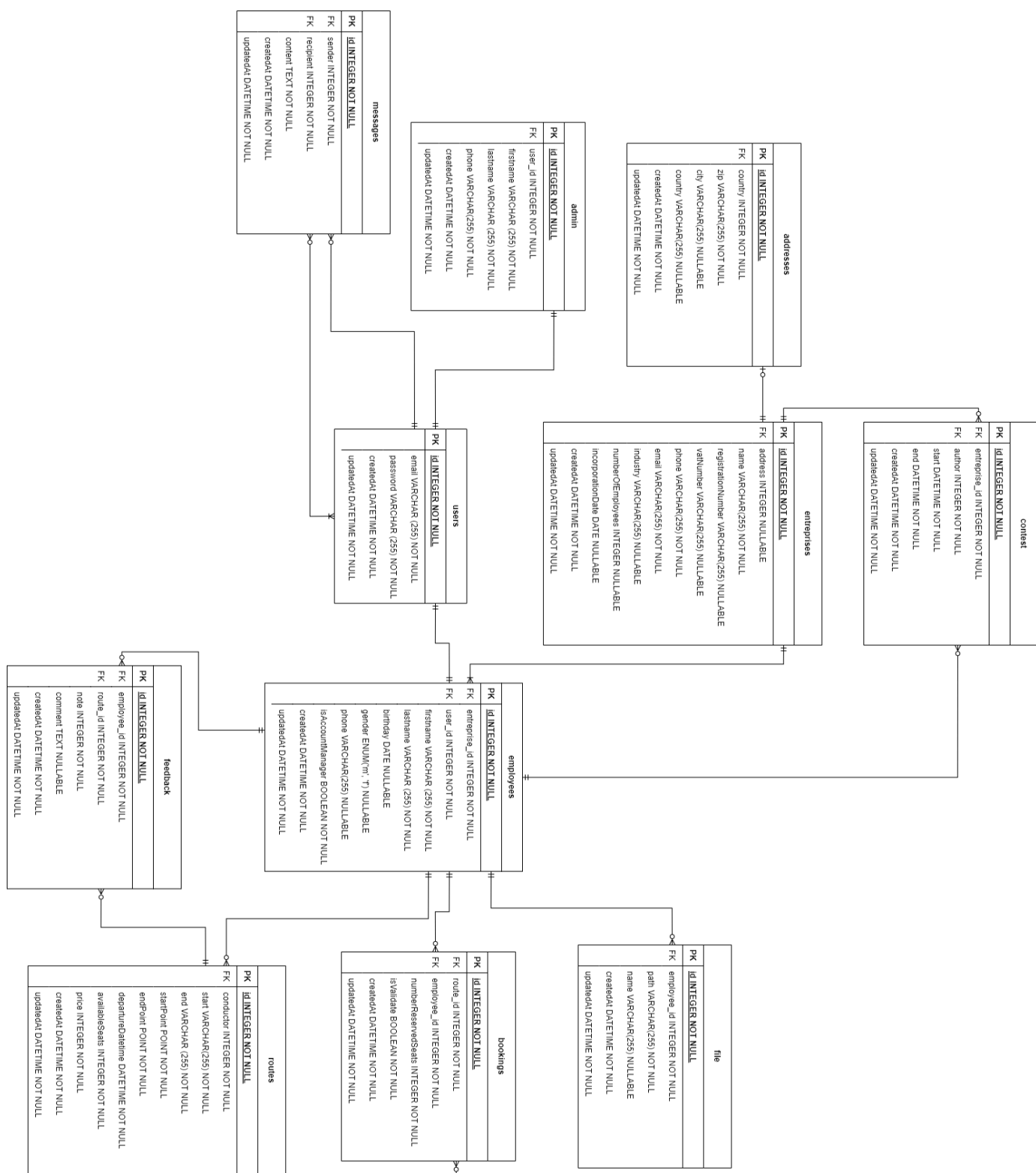
#### 4.2.2. Diagramme de séquence



### 4.2.3. Diagramme d'activité






#### 4.2.4. Diagramme entité/relation



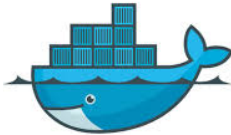






## 5. SPECIFICATIONS TECHNIQUES

### 5.1. Stack technique

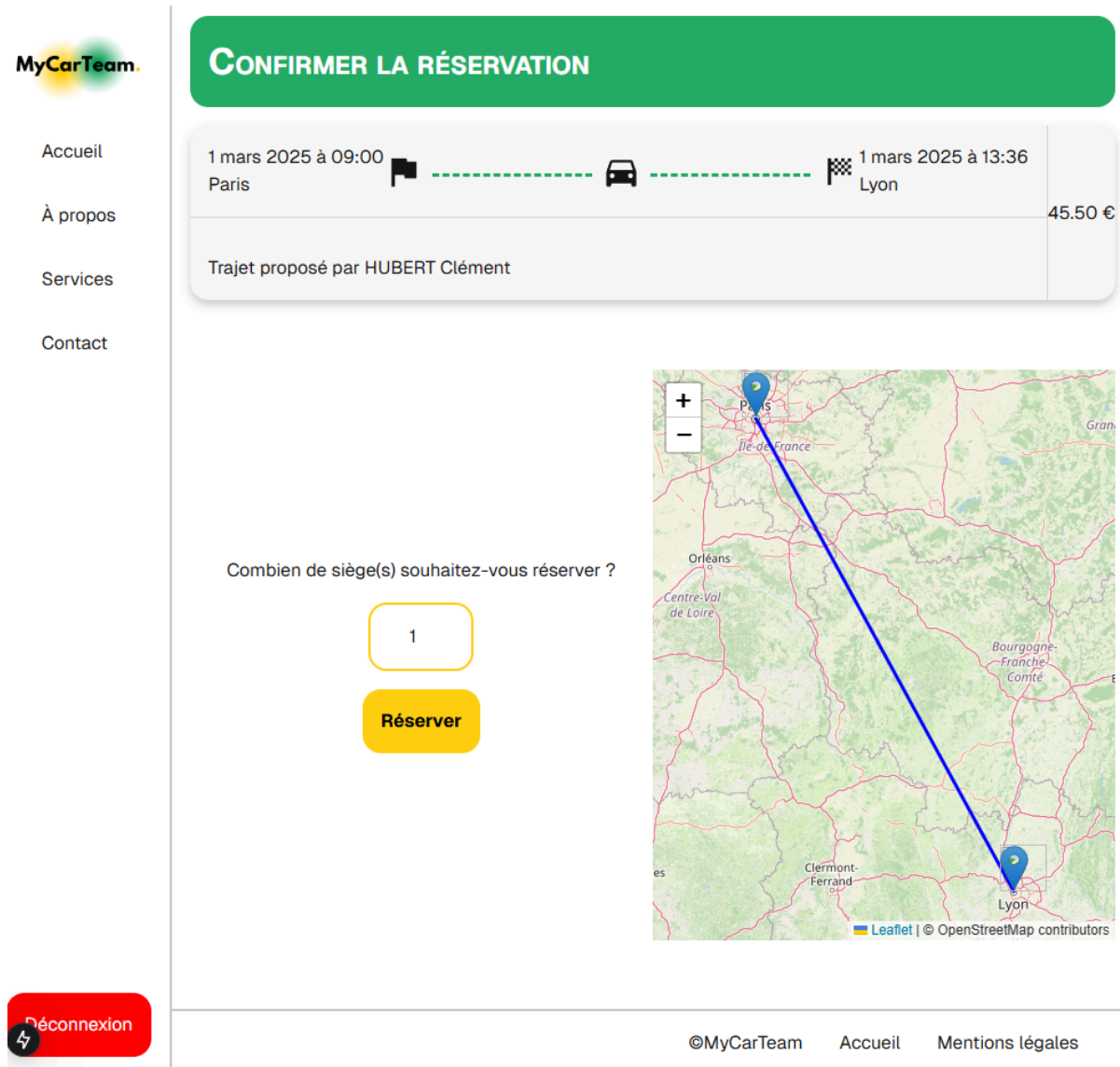
Framework frontend	 Next.js v. 15.1.0	<p>Framework frontend basé sur <b>React</b>. Open source.</p> <p>Choix porté sur ce framework plutôt que React en raison d'une de ses fonctionnalités principales : <b>rendu côté serveur (SSR)</b> permettant aux robots d'indexation des moteurs de recherche d'analyser le site.</p> <p>De plus avec la fonctionnalité App Router, la gestion du routing est simplifiée.</p>
Framework backend	 Express.js v.4.21.2	<p>Framework backend léger permettant de gérer les requêtes HTTP sous <b>Node.js</b>. Open source.</p> <p>Architecture asynchrone et non bloquante de Node.js permet d'optimiser les performances en gérant efficacement un grand nombre de connexions simultanées.</p> <p>Il bénéficie d'une large communauté assurant un suivi régulier.</p>
Système de Gestion de Base de Données (SGBD)	 MySQL	<p>SGBD relationnelle.</p> <p>Choix porté sur ce SGBD pour son ancienneté (v. 1 en 2001) et ses options avancées relatives à la sécurité (accès et permissions).</p> <p>Niveau performance, permet des requêtes efficaces via le système de jointure.</p> <p>Maintenu par Oracle.</p>

Système de gestion de version	 GitHub	<p>Système de collaboration basé sur <b>Git</b>. Permet de stocker et partager son code tout en gérant les versions.</p> <p>Le choix s'est porté sur GitHub en particulier pour sa fonctionnalité <b>GitHub Actions</b> utilisée pour automatiser les workflows de test et de déploiement (CI/CD).</p>
Infrastructure	 Droplet DigitalOcean	<p>Serveurs hébergés sur des droplets DigitalOcean (Ubuntu 22.04, 1 vCPUs, 1 GB RAM, 10 GB en SSD).</p> <p>Permet de déployer et mettre en route des conteneurs Docker.</p> <p>L'un des leaders du marché, le choix a été porté sur DigitalOcean pour son prix, son interface intuitive et sa documentation précise.</p> <p>Contrairement à certaines solutions Cloud, la tarification est à l'heure et non à l'utilisation permettant une gestion budgétaire simplifiée.</p> <p>En un clic, on peut augmenter les caractéristiques de l'instance afin de suivre l'évolution de l'application (<b>scalabilité</b>).</p>
Plateforme de conteneurisation	 Docker	<p>Docker permet d'exécuter des applications dans des environnements isolés (conteneurs).</p> <p>La conteneurisation de l'application permet de s'assurer que l'ensemble des développeurs travaillent dans le même environnement.</p> <p>De plus, elle rend l'application facilement portable sur plusieurs systèmes supportant Docker (comme les droplets DigitalOcean) sans devoir modifier la</p>

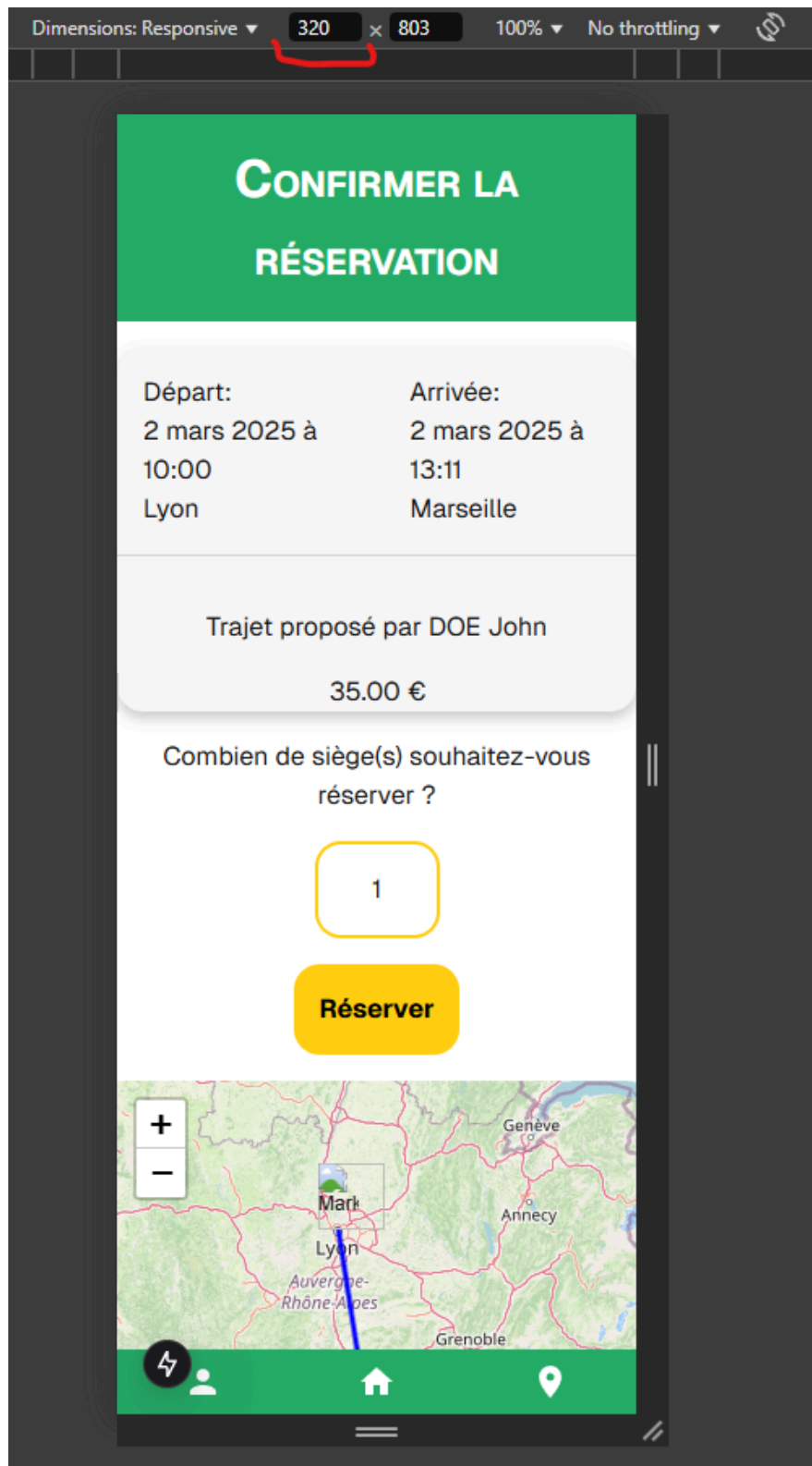
		<p>configuration pour chaque service d'hébergement.</p> <p>Dans le cadre de la démarche DevOps, les images Docker sont hébergées au sein de <b>Docker Hub</b>.</p>
Editeur de code	 <p>VS Code</p>	<p>Éditeur de code développé par Microsoft.</p> <p>Choix porté pour sa très grande bibliothèque d'extensions. Par exemple, lors du développement de l'application, <b>SonarQube</b> (ex SonarLint) qui aide à détecter les bugs du code avant de l'exécuter.</p>
Outil d'édition de digrammes	 <p>draw.io (diagrams.net)</p>	<p>Outil gratuit utilisé pour éditer les différents diagrammes UML lors de la conception de l'application.</p>

## 6. RÉALISATION DE LA FONCTIONNALITÉ RÉSERVATION

### 6.1. Interface utilisateur



*Impression écran format desktop*



*Impression écran format mobile (320px)*

```

'use client'

import RouteCard from '@app/components/RouteCard/RouteCard';
import styles from './page.module.css';
import { useParams } from 'next/navigation';
import { useEffect, useState } from 'react';
import apiClient from '../../../services/api.js';
import RouteMap from '@app/components/RouteMap';

export default function ConfirmReservation({ params }) {
  const { routeId } = useParams();

  const [route, setRoute] = useState(null);
  const [currentUser, setCurrentUser] = useState(null);

  useEffect(() => { ...
  }, []);

  async function fetchCurrentUser() { ...
  }

  async function fetchRoute() { ...
  }

  async function onSubmitBooking(event) {
    event.preventDefault();

    const seats = event.target.seats.value;


    try {
      const results = await apiClient.post('/booking', {
        route_id: routeId,
        employee_id: currentUser.employee.id,
        numberReservedSeats: seats
      });
      console.log(results);
    } catch (error) {
      console.error(error);
    }
  }

  return (
    <div className={styles.confirmReservation}>
      <h1>Confirmer la réservation</h1>

      <section className={styles.infos}>
        {route && <RouteCard route={route} displayReservationBtn={false} />}
      </section>

      <div className={styles.confirmReservationContainer}>
        <form className={styles.confirm} onSubmit={e => onSubmitBooking(e)}>
          <label htmlFor="seats">Combien de siège(s) souhaitez-vous réserver?</label>
          {route && <input type="number" name="seats" id="seats" min={1} max={route.availableSeats} defaultValue={1} />}
          <button type="submit">Réserver</button>
        </form>
        <section className={styles.map}>
          {route && <RouteMap itineraire={{ start: route.startPoint.coordinates.join(','), end: route.endPoint.coordinates.join(',') }} />}
        </section>
      </div>
    </div>
  );
}

```



Les documents ci-dessus représentent des impressions écrans de l'interface utilisateur (format desktop et mobile) et le code correspondant.

Le code se situe dans le fichier:

/frontend/src/app/app/reservation/confirmation/[routeId]/page.js.

Cette organisation permet avec la fonctionnalité App Router de Next.JS d'avoir une interface utilisateur qui s'affiche à l'adresse : [mycarteam.fr/app/reservation/confirmation/3](https://mycarteam.fr/app/reservation/confirmation/3)

Ci-dessous, un extrait du code CSS avec les medias queries permettant une interface utilisateur responsive.

```
frontend > src > app > app > reservation > confirmation > [routeId] > page.module.css >
63
64
65 @media (min-width: 480px) {
66   ... .confirmReservation {
67     ... padding: 1rem;
68   }
69
70   ... .confirmReservation h1 {
71     ... border-radius: 16px;
72   }
73 }
74
75 @media (min-width: 768px) {
76   ... .confirmReservationContainer {
77     ... flex-direction: row;
78   }
79
80   ... .map {
81     ... width: 50%;
82   }
83
84   ... .confirm {
85     ... width: 50%;
86   }
87 }
```

Le formulaire, lorsqu'il est soumis par l'utilisateur, exécute la fonction asynchrone *onSubmitBooking* qui va effectuer une requête POST vers le chemin */booking* de l'api.

## 6.2. Route API : POST /booking

```
api > src > interfaces > routes > js bookingRoutes.js > ...
1  import express from 'express';
2  import BookingController from '../interfaces/controllers/BookingController.js';
3  import { bookingService, tokenAuthentication } from '../config/dependencyInjector.js';
4  import verifyTokenMiddleware from '../interfaces/middlewares/verifyTokenMiddleware.js';
5
6  const router = express.Router();
7
8  const bookingController = new BookingController(bookingService);
9
10 const verifyToken = verifyTokenMiddleware(tokenAuthentication);
11
12
13 router.post('/booking', verifyToken, (req, res) => bookingController.create(req, res));
14
15
16 export default router;
17
```

On peut voir ici comment la route a été défini le endpoint en utilisant l'objet Router d'express.

Cet objet prend en charge la méthode post qui correspond à la requête HTTP. POST. Cet méthode prend en paramètres :

- une chaîne de caractères qui définit le chemin qui sera **api.mycarteam/booking**
- un middleware. En l'occurrence **verifyToken**, qui va vérifier le token de l'utilisateur pour l'authentifier.
- un callback. Ici qui renvoie vers le **controller** qui a pour responsabilité de renvoyer la réponse. Ici on appelle la méthode create de l'objet **bookingController**, instance de la classe du même nom.



### 6.3. Controller

```
api > src > interfaces > controllers > JS BookingController.js > BookingController > create
1  class BookingController {
2
3      constructor(bookingService) {
4          this.bookingService = bookingService;
5      }
6
7      async create(req, res) {
8          try {
9              const bookingData = req.body;
10
11              const booking = await this.bookingService.create(bookingData);
12
13              res.status(201).json(booking);
14          } catch (error) {
15              console.error(error);
16              res.status(500).json({ error: error.message });
17          }
18      }
19  }
20
21  export default BookingController;
22
```

L'impression écran ci-dessus présente l'objet **BookingController** et sa méthode `create` qui est appelée par la route (voir cf. 6.2).

Cette méthode prend deux paramètres :

- `req` : objet Express qui contient l'ensemble des informations relatives à la requête HTTP. On peut ainsi accéder aux données, comme ici au corps de la requête (le `body`).
- `res` : objet Express qui permet de construire et envoyer la réponse HTTP. Ici on appelle la méthode `status` permettant de paramétrer le code HTTP. Le code HTTP 201 (Created) qui permet d'indiquer la réussite de l'opération de création ou le code HTTP 500 (Internal Server Error) indiquant une erreur côté serveur.  
La méthode `json` permet d'envoyer la réponse au format json.

Au sein de cette méthode, j'appelle la méthode `create` de l'objet **BookingService** qui contient la logique métier pour la création d'une réservation.

## 6.4. Service : la logique métier

```
api > src > application > services > BookingService.js > BookingService > create
1  import Booking from '../domain/entities/Booking.js';
2
3  class BookingService {
4  > constructor(bookingRepository, routeRepository, sequelize) { ...
8  }
9
10 async create({ employee_id, route_id, numberReservedSeats }) {
11   const transaction = await this.sequelize.transaction();
12
13   try {
14     // On récupère la route pour vérifier la capacité de la voiture
15     const route = await this.routeRepository.getById(route_id, { transaction });
16
17     // Afin de connaître le nombre de places déjà réservées sur ce trajet on récupère toutes les réservations concernant ce trajet
18     const results = await this.bookingRepository.findByRouteId(route_id, { transaction });
19     const totalReservedSeats = results.reduce((acc, booking) => acc + booking.numberReservedSeats, 0);
20
21     // On crée une nouvelle réservation
22     const booking = Booking.create(
23       employee_id,
24       route_id,
25       numberReservedSeats,
26       route.dataValues.availableSeats,
27       totalReservedSeats
28     );
29
30     // On enregistre la réservation en base de données
31     const newBooking = await this.bookingRepository.create(booking, { transaction });
32
33     await transaction.commit();
34
35     return newBooking;
36   } catch (error) {
37     await transaction.rollback();
38     throw error;
39   }
40 }
41
42 export default BookingService;
```

La classe **BookingService** contient une méthode `create` qui va implémenter la logique métier en orchestrant des appels à plusieurs repositories.

Dans un premier temps, je récupère l'information du nombre de places disponibles sur le trajet qui fait l'objet d'une demande de réservation.

Puis, je récupère l'ensemble des réservations qui ont déjà été effectuées pour ce trajet pour ensuite calculer le nombre total de sièges déjà réservés. Nous verrons que la logique : réservation impossible si aucune place disponible a été implémentée au sein de l'entité **Booking**.

J'appelle ensuite la méthode statique `create` de la classe **Booking** qui va instancier un nouvel objet **booking** en vérifiant préalablement s'il reste des places disponibles.

Ensuite j'appelle la méthode `create` de l'objet **bookingRepository** en lui passant l'objet **booking** instancié plus haut.

## 6.5. Entité : logique entreprise

```
api > src > domain > entities > Bookingjs > ...
1  export default class Booking {
2      constructor(employee_id, route_id, numberReservedSeats) {
3          this.id = null;
4          this.employee_id = employee_id;
5          this.route_id = route_id;
6          this.numberReservedSeats = numberReservedSeats;
7          this.isValidate = false;
8      }
9
10     /**
11     * Création de réservation
12     * @param {number} employee_id -- Identifiant de l'employé.
13     * @param {number} route_id -- Identifiant de l'itinéraire.
14     * @param {number} numberReservedSeats -- Nombre de places à réserver.
15     * @param {number} availableSeats -- Nombre total de places disponibles sur la route.
16     * @param {number} totalReservedSeats -- Nombre total de places déjà réservées.
17     * @returns {Booking} Une instance de Booking si la réservation est valide.
18     * @throws {Error} Si la réservation dépasse la capacité disponible.
19     */
20     static create(employee_id, route_id, numberReservedSeats, availableSeats, totalReservedSeats) {
21         if (Number(totalReservedSeats) + Number(numberReservedSeats) > Number(availableSeats)) {
22             throw new Error("Il n'y a plus de places disponibles");
23         }
24         return new Booking(employee_id, route_id, numberReservedSeats);
25     }
26 }
```

Ici, la classe *Booking* a une méthode statique *create* qui vérifie s'il y a assez de places disponibles pour la réservation. Si oui : instantiation de l'objet. Sinon, elle propage une erreur.

On y ajoute des commentaires au standard JSDoc afin de documenter le code.

## 6.6. Repository : composant d'accès aux données

```
api > src > infrastructure > repositories > SequelizeBookingRepository.js > SequelizeBookingRepository > create
1 import IBookingRepository from '../../domain/repositories/IBookingRepository.js';
2 import BookingModel from '../../database/models/BookingModel.js';
3
4 class SequelizeBookingRepository extends IBookingRepository {
5   async create(booking, options = {}) {
6     try {
7       const created = await BookingModel.create({
8         employee_id: booking.employee_id,
9         route_id: booking.route_id,
10        numberReservedSeats: booking.numberReservedSeats,
11      }, {
12        transaction: options.transaction
13      });
14      return created;
15    } catch (error) {
16      console.error(error);
17    }
18  }
19
20  async findByRouteId(route_id, options = {}) { ...
32  }
33 }
34
35 export default SequelizeBookingRepository;
```

La classe **SequelizeBookingRepository** possède une méthode `create` qui a pour responsabilité de persister en base de données la réservation.

Cette méthode appelle la méthode `create` de **Sequelize** attachée au modèle de Booking et ainsi crée en base de données la nouvelle réservation.

## 7. ÉLÉMENTS DE SÉCURITÉ DE L'APPLICATION

Une attention particulière a été portée à la sécurité de l'application. En effet, il s'agit d'une application web, de ce fait plusieurs risques sont encourus pour l'intégrité du site ainsi que celles des utilisateurs et de leurs données.

Il est à noter qu'il s'agit d'une **obligation réglementaire**. En effet, le Règlement du Parlement européen et du Conseil relatif à la protection des personnes physiques à l'égard du traitement des données à caractère personnel et à la libre circulation de ces données, dit **RGDP**, impose une “*obligation de sécurité des données personnelles*” (article 32)<sup>2</sup>. Les sanctions peuvent atteindre jusqu'à 10 M€ ou 2% du chiffre d'affaires annuel mondial.

Cette section a pour objet de présenter les moyens mis en place au sein de l'application pour se prémunir de 3 failles : la faille SQL, la faille XSS et la faille CSRF.

De plus, sera présenté un autre élément de sécurité indispensable : hachage de mot de passe avant son enregistrement en base de données.

### 7.1. Faille SQL

Dans le cas où une application concatène directement une chaîne de caractères fournie par l'utilisateur, un hacker peut injecter des portions de code SQL malveillant et détourner le fonctionnement normal de la base de données en effaçant l'ensemble des tables, par exemple.

Pour se protéger, il convient de faire des requêtes préparées qui sépare la commande SQL des données utilisateurs.

En l'occurrence, l'application utilise l'ORM **Sequelize** gère cela automatiquement.

Par exemple pour cette ligne de code :

```
// api/src/infrastructure/repositories/SequelizeUserRepository

const created = await UserModel.create({name: userEntity.name,
email: userEntity.email, password: userEntity.password});
```

---

<sup>2</sup> CNIL. (2008). *Le règlement général sur la protection des données - RGPD CHAPITRE IV - Responsable du traitement et sous-traitant*. [EN LIGNE]. <https://www.cnil.fr/fr/reglement-europeen-protection-donnees/chapitre4#Article32> (Page consultée le 09/02/2025).

La requête SQL paramétrée correspondante est :

```
INSERT INTO users (name, email, password, createdAt, updatedAt)
VALUES (?, ?, ?, ?, ?);
```

## 7.2. Faille XSS

La faille XSS (**Cross-Site-Scripting**) est un type d'attaque où le hacker va tenter d'injecter du code malveillant dans une page web. Ce script sera exécuté par le navigateur.

Pour se protéger il ne faut jamais faire confiance aux données reçues en les échappant et en validant les données utilisateurs.

NextJS permet, par défaut, de se protéger contre les failles XSS. En effet, il s'agit d'un framework React. Or React protège contre ces injections malveillantes.

Par contre en cas d'utilisation de **dangerouslySetInnerHTML** (=innerHTML en HTML), il convient de bien valider et échapper les données. En effet, cette propriété permet d'injecter dans un élément une chaîne de caractères contenant du HTML brut. En l'espèce, le projet n'utilise pas cette propriété.

## 7.3. Faille CSRF

La faille CSRF (**Cross-Site Request Forgery**) est un type de vulnérabilité permettant à un hacker de faire exécuter à un utilisateur authentifié des actions sur l'application où il a une session en cours. Par exemple, l'attaquant va envoyer un formulaire avec une requête qui vise l'API de l'application. Cette requête, sans les précautions suivantes, sera exécutée car l'utilisateur a une session valide sur le site.

Pour s'y prévenir, lors de l'envoi du cookie contenant le token *jsonwebtoken*, il a été paramétré ainsi :

```
//api/src/interfaces/UserController.js

async login(req, res) {

    //...

    res.cookie('token', user.token, {
        maxAge: 24 * 60 * 60 * 1000, // Durée de validité 24 heures
        httpOnly: true, // Le cookie n'est pas accessible via JavaScript
        côté client (sécurité).
        secure: config.appEnv === 'production', // utilisation HTTPS
        (sécurité)
        sameSite: 'strict', // protection faille CSRF : empêche l'envoi
        du cookie dans des requêtes cross-site.
    });

    // ...

}
```

## 7.4. Hachage de mot de passe

Dans le cas où la base de données est compromise, en l'absence d'hachage du mot de passe, hacker pourrait avoir accès aux mots de passe en clair de l'ensemble des utilisateurs du site.

L'application utilise la librairie bcrypt.

```
// api/src/application/services/EntrepriseService.js
class EntrepriseService {

    // ...

    async createEntreprise({ compagnyName, compagnyPhone,
    compagnyEmail, employeeEmail, password, lastname, firstname }) {

        // ...

    }

}
```

```

    // Créer User
    const hashedPassword = await
this.passwordHasher.hash(password);

    // ...
}

// api/src/infrastructure/adapters/BcryptPasswordHasher.js
import bcrypt from 'bcryptjs';
import IPasswordHasher from
'../../domain/ports/IPasswordHasher.js';

class BcryptPasswordHasher extends IPasswordHasher {
  async hash(password) {
    return bcrypt.hash(password, 10);
  }

  async compare(password, hash) {
    return bcrypt.compare(password, hash);
  }
}

```

## 7.5. Broken Access Control

Le **Broken Access Control** (*contrôle d'accès défaillant* en français) est une vulnérabilité où il y a une défaillance des contrôles d'accès. Par exemple, un utilisateur non authentifié arrive à accéder au tableau de bord de l'administrateur et ainsi prendre le contrôle de l'application.

Afin de se prémunir de ce risque, il a été mis en place un contrôle des autorisations lorsqu'un utilisateur tente d'accéder à une ressource. Cela peut être un endroit de l'api (7.5.1) ou une page spécifique côté frontend (7.5.2).

### 7.5.1. Côté backend

Par exemple, je souhaite qu'un utilisateur non authentifié ne puisse pas accéder au endpoint GET /user/current.

Pour ce faire, on met un place à la source du dossier un middleware. Puis on le passe sur la route à protéger. Le middleware va s'exécuter avant la fonction de rappel, permettant de



vérifier l'authentification de l'utilisateur. Si il est authentifié on exécute la fonction de rappel (*next()*). Sinon on envoie au client le code HTTP 403 qui indique que le serveur a bien reçu la requête mais qu'il refuse de l'autoriser.

```
// /api/src/interfaces/middlewares/verifyTokenMiddleware.js

// On passe en paramètre le module tokenAuth qui contient la
// fonction verifyToken de jsonwebtoken.
// Cette fonction retourne un middleware qui vérifie le token dans
// les cookies de la requête.
export default function verifyTokenMiddleware(tokenAuth) {

  return function verifyTokenMiddleware(req, res, next) {

    const token = req.cookies.token;
    if (!token) {
      return res.status(401).json({ message: 'Token manquant' });
    }
    try {
      const decoded = tokenAuth.verifyToken(token);
      // Si le token est valide, on ajoute l'objet décodé à la
      // requête.
      req.user = decoded;

      // Permet de passer au middleware suivant.
      next();
    } catch (error) {
      return res.status(403).json({ message: 'Token invalide' });
    }
  };
}

// Exemple d'utilisation sur la route /user/current

import config from "../env.config.js";
import TokenAuthentication from
  "../infrastructure/adapters/TokenAuthentication.js";

const verifyToken = verifyTokenMiddleware(tokenAuthentication);
const tokenAuthentication = new
```

```
TokenAuthentication(config.jsonWebTokenSecret);

// verifyToken est le middleware. Si le token n'est pas valide, il
renvoie une erreur 403. Sinon il renvoie code 200 avec l'objet
décodé.
router.get('/user/current', verifyToken, (req, res) => {
  res.status(200).json(req.user);
});
```

### 7.5.2. Côté frontend

Par exemple, je souhaite qu'un utilisateur non connecté ne puisse pas accéder à la page /app (qui est la page du tableau de bord).

Pour ce faire, on met en place à la source du dossier /frontend/src un fichier middleware. Le middleware va s'exécuter avant que chaque requête atteigne la logique de rendu (ici les pages). Cela permet, avant que le rendu soit effectué, de mettre en place la vérification de l'authentification de l'utilisateur.

```
// frontend/src/middleware.js
import { NextResponse } from 'next/server';
import { cookies } from 'next/headers';
import { jwtVerify } from 'jose';

// Middleware s'exécutant dans un env edge car cookie en httpOnly
export default async function middleware(req) {

  // Je récupère le chemin de la requête
  const path = req.nextUrl.pathname;
  // Je souhaite protéger les routes commençant par /app
  const isProtectedRoute = path.startsWith('/app');

  let session = null;
  // Je récupère le token stocké dans le cookie nommé 'token'
  const token = (await cookies()).get('token')?.value;

  try {
    if (token) {
      // Encodage du secret issu des variables d'environnement en
      format Uint8Array
      const secret = new TextEncoder().encode(process.env.JWT_SECRET);
```

```

    // Vérifie le token et récupère les données du payload
    const { payload } = await jwtVerify(token, secret);
    session = payload;
  }
} catch (error) {
  session = null;
  console.error('Erreur de vérification du token:', error);
}

// Si la route est protégée (/app) et si aucune session n'est
trouvée, on redirige vers la page de login
if (isProtectedRoute && !session) {
  return NextResponse.redirect(new URL('/login', req.nextUrl));
}

return NextResponse.next();
}

```

## 8. PLAN DE TESTS

**Version du logiciel** 0.1.0

**Domaine fonctionnel** : Réservation d'un trajet de covoiturage

**Contexte du plan de test** : Ce plan de test permet de vérifier le bon fonctionnement du processus de réservation d'un trajet de covoiturage

**Scope:**

- Création de compte
- Création de trajets
- Login
- Affichage des réservations en liste
- Réservation d'un trajet
- Sécurité (Access Broken Control).

**Hors scope:**

- Notification par email
- Validation de la réservation par le conducteur

**Environnement** : Développement

**Données de test:**

- Compte utilisateurs valides et invalides pour vérifier l'authentification
- Enregistrement d'entreprises, de trajets et d'employés pour tester l'affichage des trajets et la réservation.

**Risques:**

Les données de tests sont fictives et pourraient ne pas déceler des erreurs humaines. Il sera nécessaire de prévoir des tests avec de réels utilisateurs.

**Calendrier** : Tests à réaliser avant la livraison de la démonstration technique au client.

## 9. JEU D'ESSAI

### 9.1. Description du scénario testé

Le scénario testé est ici la réservation d'un trajet de covoiturage par un salarié de la sélection d'un trajet à la confirmation de réservation.

### 9.2. Scénarios et résultats

#### *Scénario 1 :*

- Le salarié sélectionne un trajet disponible.
- Il valide la réservation

Résultat attendu : Un message de confirmation de la prise en compte de sa réservation s'affiche. Un enregistrement en base de données est effectué.

#### *Scénario 2 :*

- Le salarié sélectionne un trajet et réserve plus de places que disponibles
- Il valide la réservation

Résultat attendu : Un message indique que sa demande de réservation n'a pas pu être prise en compte en raison d'un nombre de places disponibles insuffisant. Il n'y a pas d'enregistrement en base de données.

#### *Scénario 3 :*

- Un utilisateur non connecté tente d'accéder à la page de réservation

Résultat attendu : redirection vers la page login.

## **RÉSULTATS DES TESTS**

Le scénario 1 et 2 : il y a bien un enregistrement en base de données lorsqu'il reste des places disponibles. En cas d'absence de places disponibles l'enregistrement n'est pas effectué. Les messages ne s'affichent pas

Le scénario 3 est un succès.

## **ACTIONS À ACCOMPLIR**

Fixer le bug d'affichage des messages d'alertes. Important pour l'expérience utilisateur (UX). Priorité forte.

Puis relancer les tests pour le scénario 1 et 2.

## 10. DÉMARCHE DEVOPS

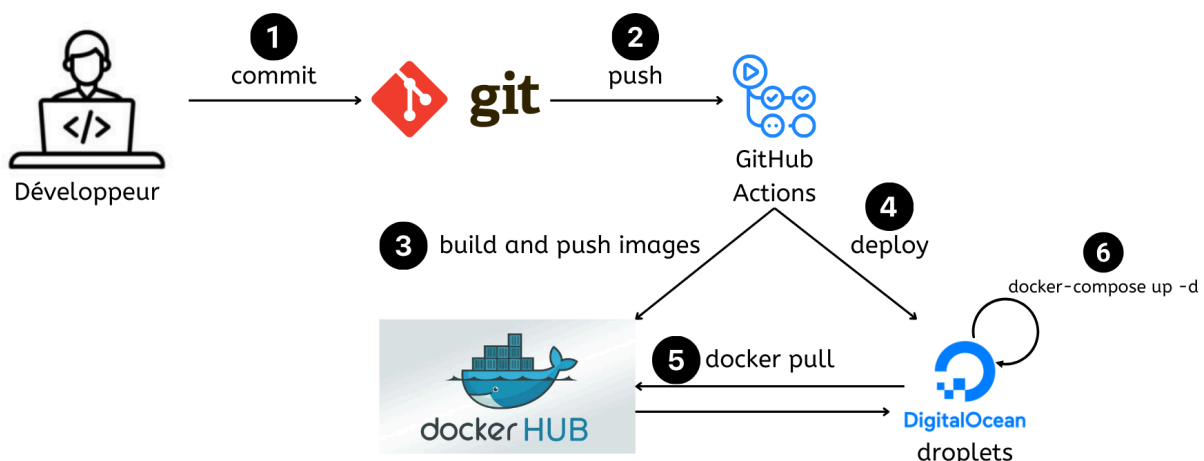
### 10.1. Introduction

Le DevOps est un mouvement apparu à la fin des années 2000 en Belgique sous l'impulsion de **Patrick Debois**. Ce mouvement a un lien fort avec les méthodologies Agiles. En effet, avant que P. Debois n'introduit le terme de DevOps, la démarche était appelée **“Agile Infrastructure & Operations”**, appellation utilisée pour la première fois lors d'une conférence “Agile” à Toronto (2008)<sup>3</sup>.

Le terme DevOps reprend les termes anglais : *development* et *operations*. L'idée est de réunir l'équipe de développement et d'exploitation afin de garantir une collaboration efficace tout au long de la vie du projet afin de répondre à plusieurs problématiques :

- **Accélération de la livraison** du logiciel. En automatisant l'intégration continue (CI) et le déploiement continu (CD).
- **Amélioration de la qualité et de la fiabilité** en favorisant les retours grâce au CI/CD qui permet de ne pas livrer d'un coup le projet.
- Favoriser la communication entre deux équipes traditionnellement séparées (équipes développement et exploitation).
- Automatisation des tests et du déploiement.

### 10.2. Architecture et infrastructure



<sup>3</sup> Patrick DEBOIS. (2008). *Agile Infrastructure & Operations*. [EN LIGNE].  
<https://www.jedi.be/presentations/agile-infrastructure-agile-2008.pdf> (Page consultée le 09/02/2025).

### 10.3. Mise en place du pipeline CI/CD

L'application utilise **Github Actions**. Le workflow est paramétré au sein du fichier `.github/workflows/ci-cd.yml`.

En résumé, au moment du push sur la branche main du répertoire GitHub de l'application plusieurs jobs vont être lancés à la suite. Si l'un des jobs échoue, l'application ne sera pas déployée.

1. **tests**. Ce job va exécuter l'ensemble des tests. Si l'un des tests échoue, le workflow s'interrompt.
2. **build-and-push**. Ici les images Docker vont être construites puis pushées sur DockerHub. Si le job rencontre une erreur, le workflow s'interrompt.
3. **deploy**. Déploiement sur DigitalOcean Droplet via SSH. Les images sont pullées sur le serveur via DockerHub puis les conteneurs seront lancés à partir de ces nouvelles images.

```
name: CI/CD Pipeline for MyCarTeam

on:
  push:
    branches:
      - main

jobs:
  tests:
    name: Run tests
    runs-on: ubuntu-latest
    env:
      APP_ENV: test
      NODE_ENV: test
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Configuration de Node.js
        uses: actions/setup-node@v3
        with:
          node-version: "16"

      - name: Installation des dépendances
```



```

    run: |
        cd api
        npm install

- name: Lancement des tests de l'API
  run: |
    cd api
    npm run test

build-and-push:
  runs-on: ubuntu-latest
  needs: tests
  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Configuration de Node.js
      uses: actions/setup-node@v3
      with:
        node-version: "16"

    - name: Build de l'image Docker de l'API
      run: |
        docker build -f ./api/Dockerfile.prod -t ${
secrets.DOCKER_USERNAME }}/api:latest ./api

    # --- Construction de l'image Frontend ---
    - name: Build de l'image Docker du Frontend
      run: |
        docker build --no-cache --build-arg
NEXT_PUBLIC_API_BASE_URL=https://api.mycarteam.fr -f
./frontend/Dockerfile.prod -t ${
secrets.DOCKER_USERNAME }}/frontend:latest ./frontend

    # --- Authentification à Docker Hub ---
    - name: Connexion à Docker Hub
      uses: docker/login-action@v2
      with:
        username: ${
secrets.DOCKER_USERNAME }
        password: ${
secrets.DOCKER_PASSWORD }

```

```

# --- Pousser l'image API ---
- name: Push de l'image Docker de l'API
  run: |
    docker push ${ secrets.DOCKER_USERNAME }/api:latest

# --- Pousser l'image Frontend ---
- name: Push de l'image Docker du Frontend
  run: |
    docker push ${ secrets.DOCKER_USERNAME
}}/frontend:latest

deploy:
  runs-on: ubuntu-latest
  needs: build-and-push
  steps:
    - name: Déploiement sur DigitalOcean Droplet via SSH
      uses: appleboy/ssh-action@v0.1.5
      with:
        host: ${ secrets.SERVER_HOST }
        username: ${ secrets.SERVER_USER }
        key: ${ secrets.SERVER_SSH_KEY }
        timeout: "900s"
        script: |
          # Authentification Docker Hub
          docker login --username ${ secrets.DOCKER_USERNAME }}
--password ${ secrets.DOCKER_PASSWORD }}

          # Télécharger les dernières images
          docker pull ${ secrets.DOCKER_USERNAME }/api:latest
          docker pull ${ secrets.DOCKER_USERNAME
}}/frontend:latest

          # Aller dans le dossier où se trouve votre
docker-compose.yml sur le Droplet
          cd /root/myCarTeam

          # Redémarrer les services
          docker-compose down
          docker image prune -f
          docker-compose up -d --force-recreate

```

## 10.4. Focus sur les tests automatisés

Lors du job appelé “run test” la commande `npm run test` est exécutée au sein d’un environnement de test.

La base de donnée de cet environnement de test est configuré ainsi :

```
// /api/src/config/database.test.js

export default {
  dialect: 'sqlite',
  storage: ':memory:',
  logging: false,
};
```

Ainsi, lorsque la connexion à la base de données est effectuée, la configuration sequelize en environnement de développement est substituée par celle de test.

```
// /api/src/infrastructure/database/index.js

import Sequelize from 'sequelize';
import config from '../../config/env.config.js';
import testConfig from '../../config/database.test.js';

const databaseOptions = config.appEnv === 'test' ? testConfig : {
  host: config.dbHost,
  dialect: 'mysql',
  port: config.dbPort,
  logging: false,
}

const sequelize = new Sequelize(config.dbName, config.dbUser,
config.dbPassword, databaseOptions);

export default sequelize;
```

Pourquoi ? Car SQLite en mode “in-memory” permet d’avoir une base de données **simplement**, entièrement stockée en mémoire ce qui permet une **exécution des tests plus rapide** et surtout cela garantit **une isolation des tests**.

La commande `npm run test`, permet le lancement du script suivant :

```
// /api/package.json

{
  "scripts": {
    "test": "node --experimental-vm-modules ./node_modules/jest/bin/jest.js  
--config ./src/config/jest.config.js",
    "dev": "nodemon ./src/server.js",
    "start": "node ./src/server.js"
  },
}
```

Ce script lance les **tests unitaires et fonctionnels** se trouvant dans le dossier `/api/tests/`

Les tests unitaires permettent de vérifier le comportement d'une unité de code isolée comme une fonction. Ainsi, on peut se rendre compte des erreurs ou régressions en cas de changement de code.

Par exemple, le code ci- dessous teste le service `UserService` en tant qu'unité de code isolée :

```
// /api/tests/unit/UserService.test.js

import UserService from "../../src/application/services/UserService";
import { jest } from '@jest/globals';

describe('UserService', () => {
  let userRepository;
  let passwordHasher;
  let tokenAuthentication;
  let userService;

  beforeEach(() => {
```

```

        userRepository = {
            findByEmail: jest.fn()
        };
        passwordHasher = {
            compare: jest.fn()
        };
        tokenAuthentication = {
            generateToken: jest.fn()
        };
        userService = new UserService(userRepository, passwordHasher,
tokenAuthentication);
    });

    describe('login', () => {
        it('should throw an error if the user does not exist', async ()
=> {
            userRepository.findByEmail.mockResolvedValue(null);

            await expect(userService.login({ email: 'test@example.com',
password: 'password' }))
                .rejects
                .toThrow("L'utilisateur n'existe pas");
        });

        // ...
    });

```

Ici on simule une réponse null par la méthode `findByEmail`. On vérifie ensuite qu’une erreur “L’utilisateur n’existe pas” est bien lancée.

Les tests fonctionnels eux s’intéressent aux exigences fonctionnelles en simulant un cas d’utilisation.

Par exemple, le code ci- dessous teste le cas d’utilisation relatif à la création d’un nouvel utilisateur.

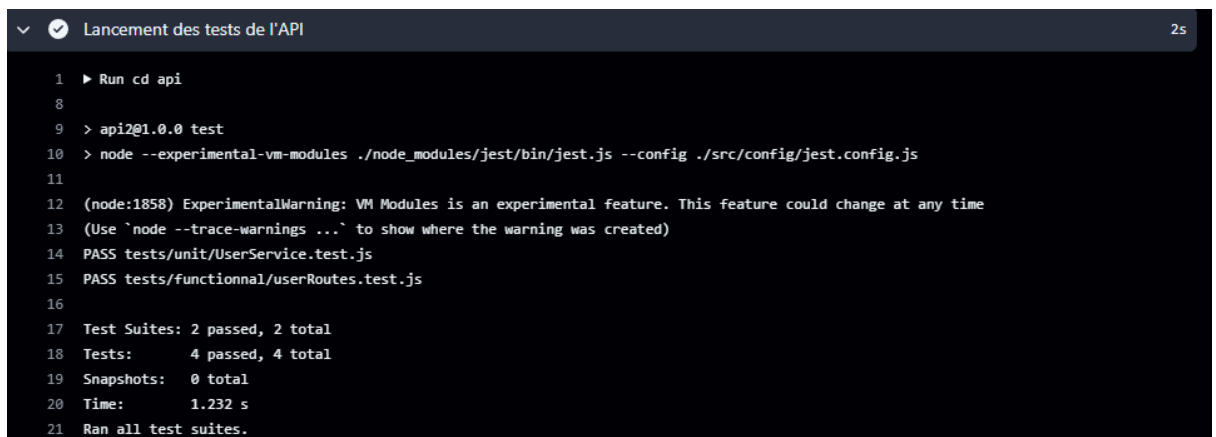
```
import request from 'supertest';
import app from '../src/config/express.config';

describe('User Routes', () => {
  test('POST /user should create a new user', async () => {
    const response = await request(app)
      .post('/user')
      .send({ compagnyName: '3WA', compagnyPhone: '0164318515',
        compagnyEmail: '3wa@mail.com', employeeEmail: 'clement@mail.com',
        password: 'myPassword', lastname: 'HUBERT', firstname: 'Clément' });

    expect(response.status).toBe(201);
    expect(response.body).toHaveProperty('id');
  });
});
```

Ici on simule l'envoi d'un body contenant l'ensemble des informations nécessaires à la création d'un utilisateur, et on vérifie que l'API renvoie bien un code HTTP 201 (Created).

Ci-dessous, le rapport d'exécution des tests généré par la pipeline GitHub Actions.



```

1  ▶ Run cd api
8
9  > api2@1.0.0 test
10 > node --experimental-vm-modules ./node_modules/jest/bin/jest.js --config ./src/config/jest.config.js
11
12 (node:1858) ExperimentalWarning: VM Modules is an experimental feature. This feature could change at any time
13 (Use `node --trace-warnings ...` to show where the warning was created)
14 PASS tests/unit/UserService.test.js
15 PASS tests/fonctionnal/userRoutes.test.js
16
17 Test Suites: 2 passed, 2 total
18 Tests:      4 passed, 4 total
19 Snapshots:  0 total
20 Time:       1.232 s
21 Ran all test suites.

```

Pour des raisons de contrainte de temps, les test end-to-end n'ont pas été mis en place. Mais cela pourrait être intéressant car il s'agit d'une approche de test qui valide le fonctionnement global d'une application de l'interface utilisateur jusqu'à la base de données.

## 11. VEILLE SUR LES VULNÉRABILITÉS DE SÉCURITÉ

La sécurité d'une application web est un domaine en constante évolution. En effet, la prévention des attaques s'apparente à un jeu du chat et de la souris. Lorsque des attaquants arrivent à trouver une faille et l'exploitent, des correctifs sont apportés, puis le correctif peut être contourné, etc. Il est donc nécessaire de se mettre à jour constamment.

Dans le cadre de mon projet, j'ai mis en place une veille active en me rendant sur plusieurs sources au moins une fois par semaine (en début de sprint).



### **Agence nationale de la sécurité des systèmes d'information.**

<https://cyber.gouv.fr/>

Site officiel de l'agence gouvernementale chargée d'assister le gouvernement.

Permet de se tenir à jour sur les actualités générales liées à la cybersécurité.



### **Computer Emergency Response Team (CERT)**

<https://www.cert.ssi.gouv.fr/>

Site internet édité par l'ANSSI répertoriant en temps réel les alertes de sécurité en cours et passées.

Chaque bulletin d'alerte expose les risques, les systèmes affectés, un résumé de la menace et, les solutions à mettre en place.



### **Open Web Application Security Project (OWASP)**

<https://owasp.org/>

Contrairement aux deux premières sources, il ne s'agit pas d'un site gouvernemental. OWASP est une communauté en ligne ouverte et accessible à tous publiant des recommandations de sécurisation Web.

La communauté propose un top 10 des risques de sécurité relatifs aux applications Web.

En complément, je me rends régulièrement sur la documentation des différents outils, frameworks et bibliothèques que j'utilise afin de m'assurer d'avoir toujours la dernière mise à jour.

```

PS D:\dev\3WA\myCarTeam_CDA\frontend> npm audit
# npm audit report

next 15.0.0 - 15.1.1
Severity: moderate
Next.js Allows a Denial of Service (DoS) with Server Actions - https://github.com/advisories/GHSA-7m27-9
fix available via `npm audit fix --force`
Will install next@15.1.7, which is outside the stated dependency range
node_modules/next

1 moderate severity vulnerability

To address all issues, run:
  npm audit fix --force

```

Enfin, régulièrement j'exécute la commande `npm audit` me permettant de m'assurer qu'il n'y a pas de menaces liées à mes dépendances. Ci-dessous un exemple de la commande `npm audit` mettant en lumière une faille :

Et pour m'assurer d'être à jour sur mes anciens projets, je suis inscrit aux **GitHub security alert digest** qui est un rapport synthétique des analyses automatiques de mes répertoires GitHub et de leurs dépendances.

Exemple pour un projet qui n'est pas en production :



### GitHub security alert digest

clhubdev's repository security updates from the week of **Feb 4 - Feb 11**

 clhubdev's personal account

---

 **clhubdev / 3wa\_node2\_eval**

Known security vulnerabilities detected

Dependency	Version	Upgrade to
semver	>= 7.0.0 < 7.5.2	-> 7.5.2
Defined in package-lock.json		
Vulnerabilities CVE-2022-25883 High severity		
ip	= 2.0.0	-> 2.0.1
Defined in package-lock.json		
Vulnerabilities CVE-2023-42282 Low severity		
express	< 4.19.2	-> 4.19.2
Defined in package-lock.json		



## 12. CONCLUSION

Ce projet est l'aboutissement d'un an de formation en conception et développement d'applications et plus largement d'un projet de reconversion mûrement réfléchi.

Il m'a permis de réaliser l'importance de la conception avant de se lancer dans le développement d'une application web.

Auparavant, je me lançais directement dans le développement, pensant que consacrer du temps à la conception était inutile – une erreur que j'ai commise, car je me retrouvais ensuite à modifier ma base de données. En effet, sans réflexion préalable, le modèle envisagé ne permettait pas d'intégrer une fonctionnalité ajoutée en fin de projet.

Dans le cadre de ce projet, ce ne fut pas le cas. Le temps passé à la conception a été regagné lors de la phase de développement.

Par ailleurs, j'ai pu apprendre des méthodes de gestion de projet comme SCRUM. J'ai eu le plaisir de commencer à la mettre en place au sein de mon alternance après en avoir parlé à mon tuteur de stage. On a gagné en efficacité.

Le principal problème que j'ai rencontré lors du développement est la prédictibilité du temps nécessaire au développement d'une fonctionnalité. C'est un aspect sur lequel je travaille activement, et avec l'expérience, cette problématique devrait s'atténuer, bien qu'il semble impossible de prédire parfaitement le temps requis pour chaque tâche.

Pour pallier ce problème en entreprise, j'ai instauré l'ajout d'une marge de sécurité d'environ 20 % entre le temps estimé et le temps communiqué.

En conclusion, comme le disait Virgile (poète latin du 1er siècle avant JC.) qui a compris le quotidien du développeur avant même l'apparition du premier ordinateur : *“On se lasse de tout, excepté d'apprendre.”*