2025

PLR conseil X epsi l'école d'ingénierie informatique

RAPPORT D'ACTIVITÉ

Alexandre Levenez

SOMMAIRE

REMERCIEMENTS

Avant de commencer ce rapport, je tiens à remercier toutes les personnes qui m'ont accompagné et soutenu tout au long de mon stage.

Un grand merci à Paul Larsonneur, mon maître de stage, pour son accueil, sa disponibilité et ses précieux conseils. Grâce à lui, j'ai découvert le monde professionnel sous un nouvel angle et appris énormément sur le développement d'une application en partant de zéro. Son encadrement m'a permis de monter en compétences et de gagner en autonomie, tout en travaillant dans un cadre motivant.

Je remercie également toute l'équipe pour sa bienveillance et sa bonne humeur. Travailler à leurs côtés a été une expérience super enrichissante, où j'ai non seulement développé de nouvelles compétences techniques, mais aussi découvert un environnement de travail dynamique et collaboratif. Ce stage a renforcé ma passion pour le développement et m'a permis d'avoir une vision plus concrète du métier.

INTRODUCTION

Durant ma deuxième année d'études à l'EPSI, j'ai eu l'opportunité d'effectuer un stage de cinq semaines au sein de l'entreprise PLR CONSEIL, fondée par Paul Larsonneur le 11 août 2021.

Spécialisée dans la programmation informatique, cette entreprise se consacre à la conception, au développement et à la commercialisation de logiciels, de sites web et d'outils de communication. Elle propose également des services de gestion de projets informatiques, de conseil en programmation et en stratégie digitale, ainsi que la distribution et l'exploitation des solutions développées.

Pendant ces cinq semaines, j'ai eu l'occasion de travailler sur un projet assez conséquent : la mise en place d'une application nommée *Casa Bonita*. Ce projet a été divisé en trois parties :

- La mise en place d'une base de données
- La mise en place d'un backend
- La mise en place d'un frontend

Durant mon stage, j'ai eu l'opportunité d'approfondir mes compétences en développement. Cette expérience m'a permis de mettre en pratique les connaissances acquises en cours, notamment en programmation, en gestion de bases de données et en méthodologies de travail. J'ai pu travailler sur des projets concrets, manipuler des technologies utilisées en entreprise et mieux assimiler les bonnes pratiques du développement.

Au-delà de l'aspect technique, ce stage m'a également appris à collaborer avec une équipe, à m'adapter aux contraintes d'un projet réel et à gagner en rigueur dans l'écriture du code. Cette immersion dans le monde professionnel m'a offert une vision plus claire des attentes du métier et des compétences à développer pour évoluer dans ce domaine.

Présentation de Casa Bonita

Avant de présenter *Casa Bonita*, il est important de poser un peu de contexte.
Dans de nombreuses régions du monde, faire appel à des services de ménage ou de jardinage est une pratique courante. Que ce soit pour gagner du temps, alléger son quotidien ou répondre aux besoins des entreprises, ces services sont devenus essentiels, notamment dans les grandes villes où le rythme de vie est soutenu. Dans certains pays, cette habitude est encore plus répandue, car ces prestations sont plus accessibles et mieux intégrées au mode de vie local.

C'est en partant de ce constat que PLR CONSEIL a développé *Tauturu Fare*, une application web et mobile permettant de commander facilement des services de ménage et de jardinage. Destinée à la Polynésie française, cette plateforme met en relation des professionnels qualifiés avec des particuliers et des entreprises à la recherche de solutions rapides et efficaces.

Après environ un an, l'équipe a décidé de créer une version adaptée à un nouveau marché : le Mexique. C'est ainsi qu'est née *Casa Bonita*.

Casa Bonita fonctionne sur le même principe que *Tauturu Fare* : elle permet aux particuliers et aux entreprises de commander des services de ménage et de jardinage rapidement et facilement. La grande différence, c'est que *Casa Bonita* est exclusivement destinée au marché mexicain.

Cependant, ce n'est pas une simple copie de *Tauturu Fare*. L'application a été adaptée pour répondre aux spécificités locales, notamment :

- Les habitudes et attentes des utilisateurs mexicains,
- Les différences de tarifs et les particularités du marché,
- Les règles et réglementations locales,
- La mise en place d'un réseau de prestataires adaptés.

Avec *Casa Bonita*, l'objectif est d'offrir une solution simple et efficace aux particuliers et aux entreprises mexicaines cherchant des prestations de qualité en ménage et en jardinage.

Cadre de travail

Ce stage était une expérience assez particulière. Non seulement il s'est déroulé entièrement en télétravail, mais en plus, j'avais 7 heures de décalage avec mon maître de stage, qui est basé au Mexique. Ce décalage horaire a parfois compliqué les échanges, nécessitant une bonne organisation pour communiquer efficacement.

Contrairement à mon stage de première année, où j'étais en entreprise, j'ai cette fois-ci opté pour un stage en quasi-totalité à distance. Je n'avais qu'une seule obligation de présence sur le campus d'HEP Nantes, le mercredi. J'ai donc voulu profiter de cette opportunité pour tester le télétravail à 100 %, afin de voir si j'étais capable de gérer mon emploi du temps et mon travail de manière autonome.

Ce mode de travail présente plusieurs avantages :

Tout d'abord, la flexibilité des horaires : j'ai pu organiser mes journées de manière à ne pas être trop décalé par rapport à mon maître de stage, Paul. Ensuite, le confort : travailler de chez soi permet d'être dans un environnement plus agréable. Un autre point important est la réduction des déplacements, ce qui permet de gagner du temps, d'économiser de l'énergie et de réduire son impact écologique. J'ai aussi pu travailler sur un PC fixe avec plusieurs écrans, ce qui est bien plus efficace qu'un simple ordinateur portable. De plus, le télétravail m'a permis d'éviter les distractions sociales, car en entreprise, les discussions entre collègues peuvent parfois nuire à la concentration. Enfin, ce mode de travail est particulièrement adapté à mon tempérament, car, n'étant pas très sociable, le fait de rester dans un cadre qui me convient mieux, sans trop d'interactions, m'a aidé à être plus productif.

Malgré ses nombreux avantages, le télétravail comporte aussi quelques inconvénients :

Travailler chez soi signifie être entouré de nombreuses sources de distraction (réseaux sociaux, jeux, confort du domicile...), ce qui peut nuire à la productivité. De plus, même si cela peut être un avantage pour certains, le manque d'interactions humaines peut rendre le travail un peu plus monotone et limiter les échanges informels, qui sont parfois utiles.

Bilan :

Au final, mon choix du télétravail a été très bénéfique. J'ai pu profiter d'une grande liberté dans mon organisation, d'un environnement confortable et surtout d'une première expérience proche du mode de vie d'un développeur freelance. Ce stage m'a permis de mieux comprendre les enjeux et défis du travail à distance, tout en me confortant dans l'idée qu'une carrière en télétravail ou en freelance pourrait me correspondre.

MISSION CASA BONITA

Contexte

Mon stage s'est principalement concentré sur une seule mission de grande envergure : mettre en place l'application Casa Bonita à partir de zéro.

Pour structurer mon travail et savoir exactement où j'allais, je devais suivre plusieurs livrables définis en amont :

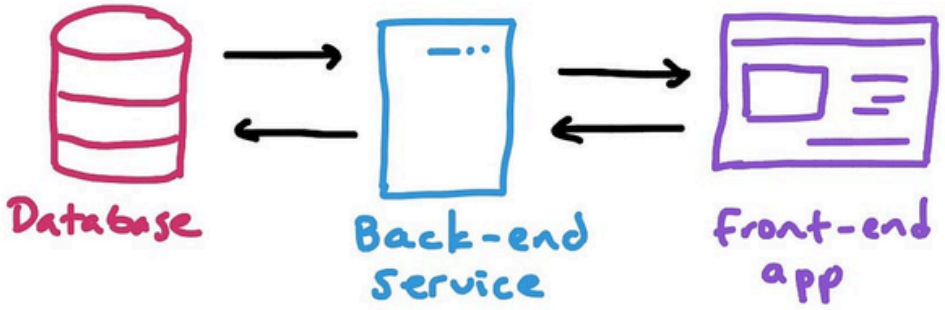
- Un backend Java opérationnel, développé avec le framework Spring Boot et utilisant jOOQ comme ORM.
- Un front-end fonctionnel, développé avec un framework JavaScript de mon choix, avec un script de lancement (npm run start), et pouvant être déployé en option.
- Des Cloud Functions, testables localement ou directement déployables.
- Une documentation détaillée (README), expliquant toutes les étapes nécessaires pour lancer le backend, le front-end et les Cloud Functions, ainsi que toute configuration spécifique.
- Un serveur de base de données PostgreSQL, avec un système de migrations géré par Liquibase, garantissant une base conforme au modèle de données fourni.

J'avais une certaine liberté quant à l'ordre dans lequel je pouvais aborder ces tâches. Pour faciliter l'avancement du projet, plusieurs ressources étaient mises à ma disposition sur GitLab. Elles contenaient des instructions précises et des points clés à respecter pour assurer la cohérence et la qualité du projet.

MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL

Pour débuter correctement mon stage, j'ai d'abord analysé les livrables afin de mettre en place et d'installer tout ce dont j'avais besoin.

J'ai donc décidé de commencer par le livrable lié à la base de données, puis de passer au backend, et enfin au front-end. Cette approche me semblait plus logique et facilitait la gestion du projet.



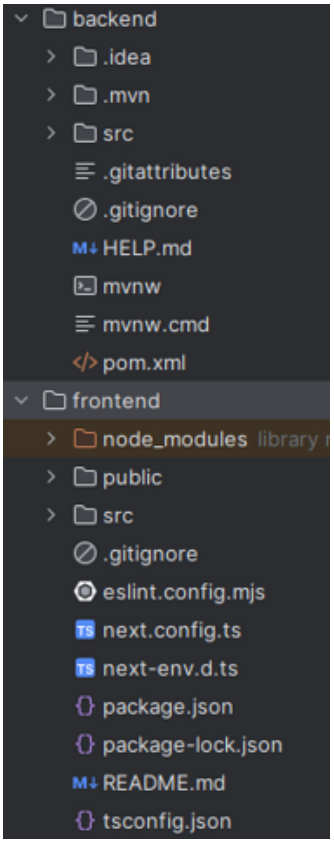
Pour gérer la base de données, j'ai donc dû installer PostgreSQL, un système de gestion de base de données open source, reconnu pour sa stabilité et ses nombreuses fonctionnalités avancées en SQL.

Pendant l'installation de PostgreSQL, j'ai commencé à créer le projet. Cela permettait d'avoir une vision globale de l'ensemble et de poser une structure de base avant de commencer le développement. L'objectif, à ce stade, était simplement de définir l'architecture générale du projet.

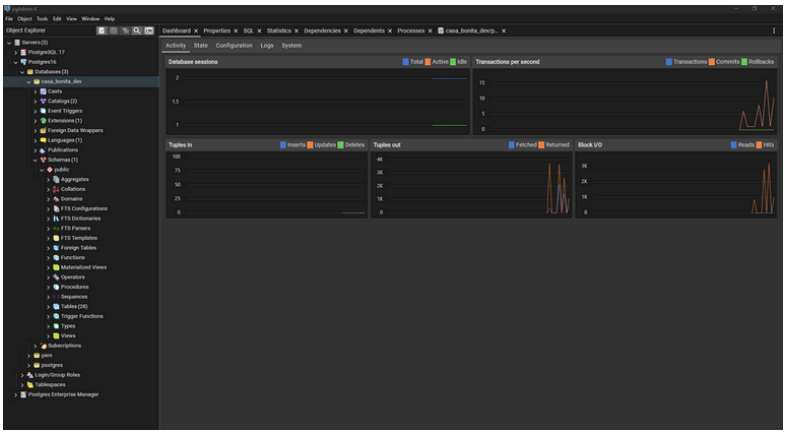
J'ai donc commencé par installer le backend, qui repose sur Java avec Spring Boot et utilise Maven comme gestionnaire de dépendances.

Ensuite, après une discussion avec Paul, nous avons décidé d'utiliser React, un framework JavaScript populaire. Nous avons opté pour React en raison de sa grande flexibilité, permettant de structurer l'interface utilisateur comme on le souhaite. C'est un framework puissant, souvent comparé à un bac à sable, qui laisse une grande liberté d'utilisation et d'organisation.

J'ai donc installé React via Node.js, ce qui a permis de poser une première base solide pour le projet. Cette structure initiale facilite ensuite l'organisation et la gestion du développement, en assurant une bonne séparation entre le backend et le frontend.



Une fois le tout installé, j'ai lancé PostgreSQL avec un logiciel appelé pgAdmin. Ce logiciel permet de gérer les bases de données plus facilement et d'injecter du code SQL directement.
Ensuite, il m'a suffi de créer une base de données.



```xml
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
</dependency>
```

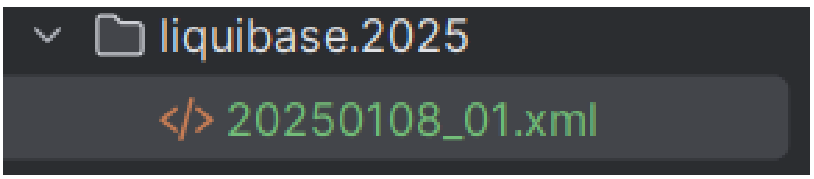
Après cela, il a fallu que j'apprenne rapidement à utiliser Liquibase. Liquibase est un outil qui permet de gérer les mises à jour de la base de données de manière structurée et versionnée.
J'ai donc d'abord ajouté Liquibase dans le fichier pom.xml (le fichier qui gère les dépendances). Cela permet de gérer l'ensemble dans le backend du projet.

Il a ensuite fallu, à partir du backend, ajouter du SQL. Pour ce faire, il est nécessaire d'ajouter des propriétés dans un fichier nommé application.properties.
Il est important d'y indiquer l'URL de la base de données, le nom d'utilisateur et le mot de passe associés. Une fois cela fait, on peut commencer à utiliser la base de données depuis le backend.

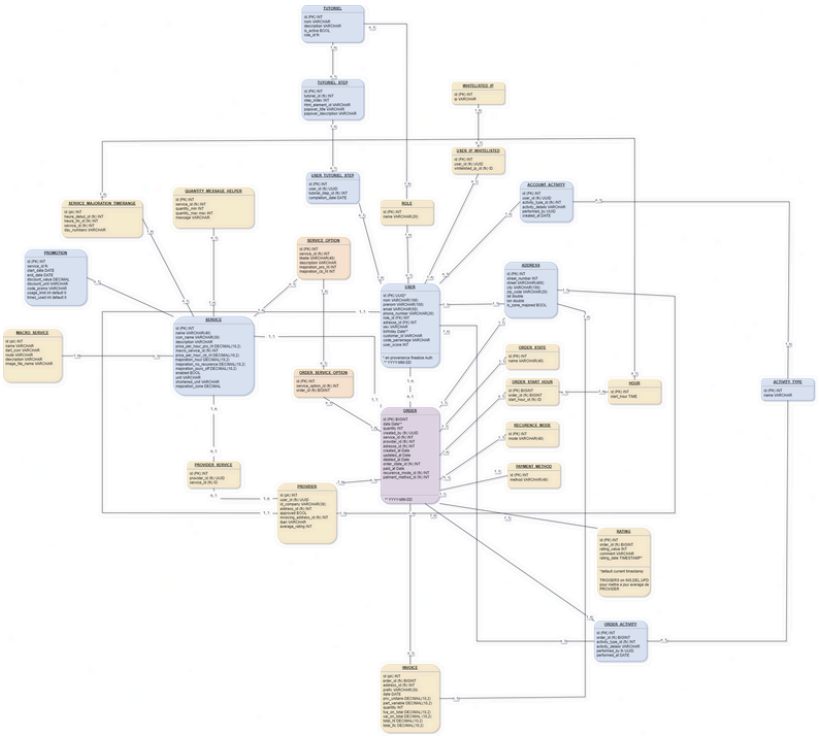
Une fois cette étape terminée, il a fallu créer un fichier nommé db.changelog-master.yaml.
Dans ce fichier, il faut spécifier l'emplacement où l'on va chercher les différentes migrations.

```yaml
databaseChangeLog:
  - includeAll:
      path: 'liquibase/2025'
```

Une fois créé, il suffit de créer un fichier .xml dans le dossier liquibase/2025 comme ceci :

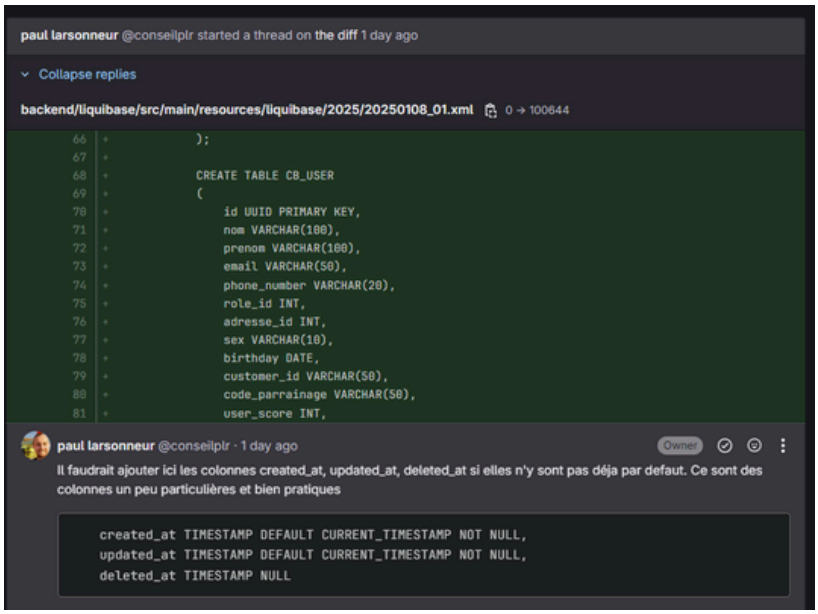


Une fois toute cette préparation effectuée, j'ai dû m'occuper de la partie SQL de la base de données.
Pour ce faire, je me suis appuyé sur Tauturu Fare.
En effet, j'avais à disposition le schéma de la base de données de Tauturu Fare :



Une fois que j'ai pris connaissance de ce schéma, j'ai pu rédiger le code SQL, puis l'ajouter dans le fichier de migration.
J'ai ensuite dû récupérer un code SQL déjà existant, mais il a fallu le remettre à jour pour qu'il soit compatible avec PostgreSQL.
Après l'avoir adapté, je l'ai ajouté dans une seconde migration. Cela permet de remplir la base de données créée précédemment avec la première migration.

Une fois cela fait, il ne me restait plus qu'à créer une merge request sur GitLab.
Paul m'a ensuite fait des retours comme celui-ci :



Après avoir appliqué les retours, la merge request fut acceptée.
Cela a permis d'ajouter ma première pierre à ce projet.

CRÉATION DU BACKEND

Une fois la partie base de données validée, j'ai pu entamer la partie backend.
Pour rappel, la partie backend permet de récupérer des informations de la base de données pour les envoyer via HTTP.
Pour ce faire, nous avons décidé d'utiliser quatre couches :

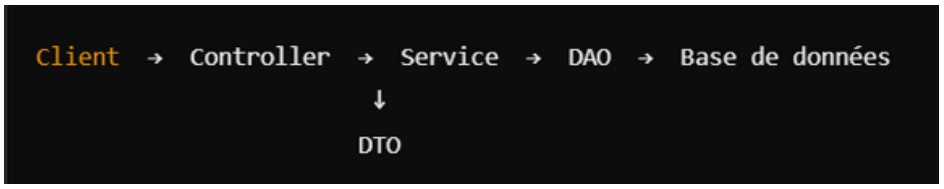
- La couche Controller : C'est la couche qui gère les requêtes HTTP (GET, POST, PUT, DELETE). Elle reçoit les requêtes des clients (navigateur, mobile, autre API), les transmet à la couche Service et renvoie la réponse.

- La couche Service : Elle contient la logique métier de l'application. Elle fait le lien entre la couche Controller et la couche DAO. Elle applique les règles métiers (par exemple : validation, calculs, transformations de données).

- La couche DTO : Elle sert d'intermédiaire entre la couche Service et la couche Controller. Les DTO permettent de cacher la structure des entités de la base de données et d'éviter d'exposer trop d'informations.

- La couche DAO : C'est la couche qui interagit directement avec la base de données. Elle effectue les opérations CRUD (Create, Read, Update, Delete).

Ce qui, schématisé, donne ceci :



Une fois avoir réfléchi à la gestion des couches, je me suis mis à utiliser un ORM.
Un ORM permet de travailler avec la base de données en utilisant du code orienté objet plutôt que des requêtes SQL brutes.

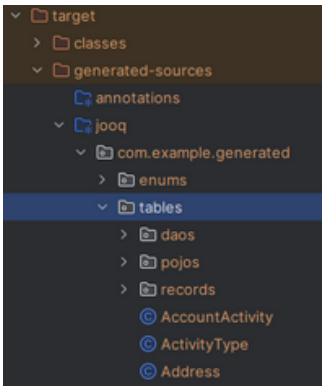
Utiliser un ORM présente de nombreux avantages, tels que :

- Gain de productivité
- Sécurité
- Lisibilité
-

Pour ce projet, nous avons décidé d'utiliser l'ORM jOOQ en raison de ses nombreux avantages, tels que :

- Fort typage & sécurité
- Performance optimisée
- Facilité d'intégration avec Spring Boot
- Lisibilité et maintenance

Après avoir pris la décision d'utiliser jOOQ, j'ai pu commencer à créer la première couche pour le backend.
J'ai donc commencé par créer la couche DAO.
Pour cela, j'ai utilisé jOOQ, qui m'a permis de générer automatiquement cette couche, ainsi que les entités et les POJOs (Plain Old Java Objects). Cela m'a donné une base solide pour la suite du projet, car j'ai pu bénéficier d'une structure préconfigurée, tout en me concentrant sur la logique métier et la gestion des données.



Une fois ces éléments créés, j'ai dû créer à nouveau des DAO, mais cette fois-ci, elles seraient gérées manuellement.
En effet, les couches DAO créées par jOOQ offrent une excellente base, mais il pourrait potentiellement manquer certains éléments. C'est pour cette raison qu'il a fallu les recréer à la main. Pour ce faire, il suffit d'appeler ce qui est déjà présent dans les couches générées par jOOQ, comme ceci :

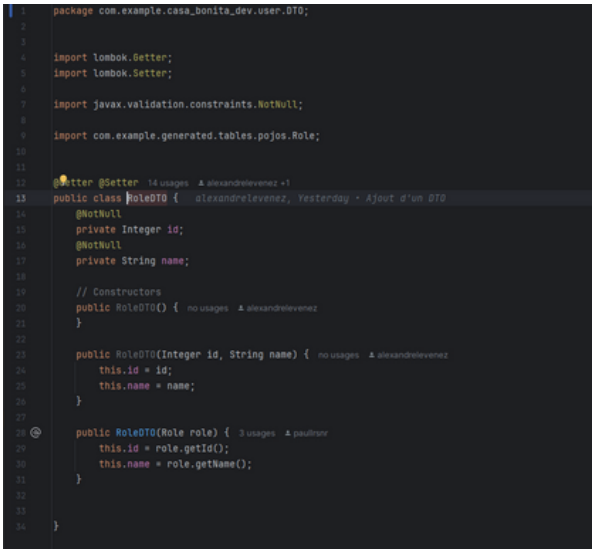
```java
package com.plrconseil.casabonita.address.DAO;

import com.example.generated.tables.daos.AddressDao;
import org.jooq.DSLContext;
import org.springframework.stereotype.Repository;

@Repository  alexandre.levenez
public class AddressDAO extends AddressDao{

    public AddressDAO(DSLContext dsl) { super(dsl.configuration()); }
}   alexandre.levenez, 24/01/2025 07:08 · Mise en place du backend
```

Ici, on peut donc ajouter des éléments qui n'ont pas été créés par jOOQ, si nécessaire.

Après avoir créé cette couche, j'ai entamé la création de la couche DTO.

J'ai donc créé, pour chaque table de la base de données, un fichier DTO.
Ces fichiers DTO vont permettre de représenter les données sous forme d'objets. Ils sont utilisés pour transférer des données entre les différentes couches de l'application, notamment entre les Services et les Controllers.
Pour structurer ces données sous forme d'objets, j'ai utilisé des getters et des setters.
Les getters et setters sont des méthodes qui permettent d'accéder aux valeurs des champs d'une classe et de les modifier. Ils sont utilisés pour encapsuler les champs d'une classe et garantir l'encapsulation des données.
Pour gagner du temps et améliorer la lisibilité du code, j'ai utilisé la bibliothèque Lombok pour générer automatiquement ces getters et setters.
En effet, au lieu de les créer manuellement, il m'a suffi d'importer cette bibliothèque et d'ajouter les annotations @Getter et @Setter à la création de la classe.

En effet, au lieu de les créer à la main, il m'a suffi d'importer cette bibliothèque et d'ajouter les annotations @Getter et @Setter à la création de la classe.
Comme ceci :

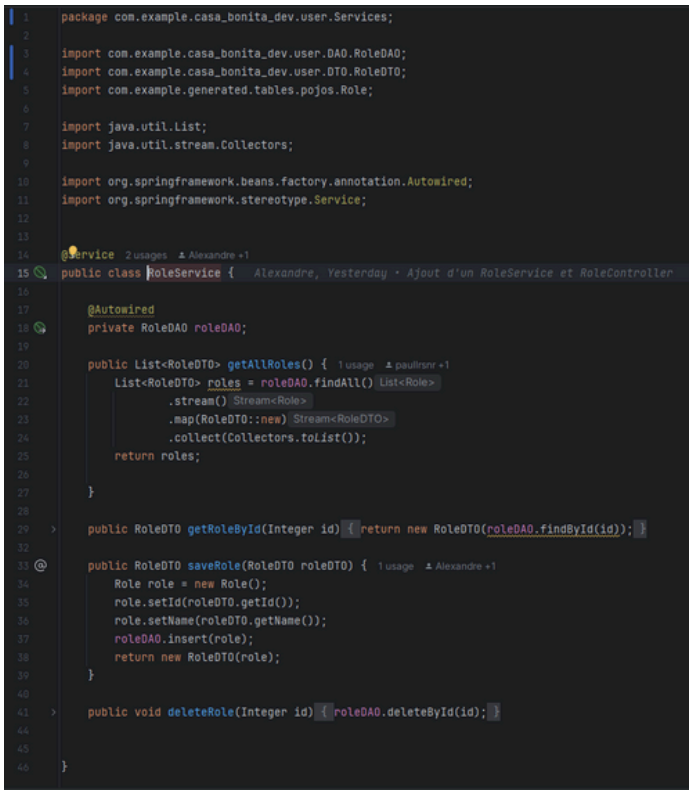


```java
package com.example.casa_bonita_dev.user.DTO;


import lombok.Getter;
import lombok.Setter;

import javax.validation.constraints.NotNull;

import com.example.generated.tables.pojos.Role;


@Getter @Setter 14 usages  alexandrelevenez +1
public class RoleDTO {   alexandrelevenez, Yesterday • Ajout d'un DTO
    @NotNull
    private Integer id;
    @NotNull
    private String name;

    // Constructors
    public RoleDTO() {  no usages  alexandrelevenez
    }

    public RoleDTO(Integer id, String name) {  no usages  alexandrelevenez
        this.id = id;
        this.name = name;
    }

    public RoleDTO(Role role) {  3 usages  paullrsnr
        this.id = role.getId();
        this.name = role.getName();
    }


}
```

L'exemple ci-dessus montre comment un DTO est fait. Il faut juste faire attention à bien inclure les bons éléments dans les classes et les constructeurs des autres DTO.
Une fois ceux-ci créés, je suis passé à la couche Service.

Pour créer cette couche, il faut dans un premier temps récupérer les couches DAO et DTO.
Le DAO est utilisé pour interagir avec la base de données, ce qui permet d'écrire un code plus propre et plus facile à maintenir. Grâce à cette séparation, les fichiers Service peuvent se concentrer sur la logique métier sans avoir à gérer directement les requêtes SQL.
C'est à ce niveau que l'on peut vraiment apprécier l'utilité de jOOQ, car nous utilisons des fonctions qui ont été créées avec ce framework. Cela nous a permis de gagner un temps précieux et d'améliorer la lisibilité du code.

J'ai également utilisé les DTO, qui permettent d'éviter d'exposer directement les entités de la base de données.



```java
package com.example.casa_bonita_dev.user.Services;

import com.example.casa_bonita_dev.user.DAO.RoleDAO;
import com.example.casa_bonita_dev.user.DTO.RoleDTO;
import com.example.generated.tables.pojos.Role;

import java.util.List;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;


@Service  2 usages  Alexandre +1
public class RoleService {   Alexandre, Yesterday • Ajout d'un RoleService et RoleController

    @Autowired
    private RoleDAO roleDAO;

    public List<RoleDTO> getAllRoles() {  1 usage  paullrsnr +1
        List<RoleDTO> roles = roleDAO.findAll() List<Role>
                .stream() Stream<Role>
                .map(RoleDTO::new) Stream<RoleDTO>
                .collect(Collectors.toList());
        return roles;

    }

    public RoleDTO getRoleById(Integer id) { return new RoleDTO(roleDAO.findById(id)); }

    public RoleDTO saveRole(RoleDTO roleDTO) {  1 usage  Alexandre +1
        Role role = new Role();
        role.setId(roleDTO.getId());
        role.setName(roleDTO.getName());
        roleDAO.insert(role);
        return new RoleDTO(role);
    }

    public void deleteRole(Integer id) { roleDAO.deleteById(id); }


}
```

Une fois les services terminés, j'ai enfin pu passer à la dernière étape, qui était de créer la couche Controller.

Pour rappel, la couche Controller gère les requêtes HTTP et renvoie les réponses sous forme de JSON.
Pour gérer tout cela, j'ai utilisé une API REST.
Le Controller reçoit les requêtes HTTP et fait appel aux méthodes des Services.
J'ai utilisé les annotations Spring (@GetMapping, @PostMapping, @DeleteMapping) pour définir des endpoints permettant d'afficher, d'ajouter et de supprimer des rôles. Grâce à cette architecture en couches, le code est mieux organisé, plus clair et plus facile à maintenir.
Le tout donne ceci :

```java
package com.example.casa_bonita_dev.user;

import ...

@RestController  Alexandre *
@RequestMapping("/api/roles")
public class RoleController {

    @Autowired
    private RoleService roleService;

    @GetMapping  Alexandre
    public List<RoleDTO> getAllRoles() { return roleService.getAllRoles(); }

    @GetMapping("/{id}")  Alexandre
    public RoleDTO getRoleById(@PathVariable Integer id) { return roleService.getRoleById(id); }

    @PostMapping  Alexandre  You, 29 minutes ago · Uncommitted changes
    public RoleDTO createRole(@RequestBody RoleDTO roleDTO) { return roleService.saveRole(roleDTO); }

    @DeleteMapping("/{id}")  Alexandre
    public void deleteRole(@PathVariable Integer id) { roleService.deleteRole(id); }

}
```

Avec toutes ces couches terminées, j'ai pu effectuer des essais directement via des URLs pour vérifier si je pouvais injecter des données dans la base de données.
Pour faire les tests, j'ai utilisé ce genre d'URL :
http://localhost:8081/api/insertOrderServiceOption?order_id=1&service_option_id=2

Après avoir vérifié que les tests fonctionnaient, j'ai créé une nouvelle merge request sur le projet pour obtenir des commentaires et des retours sur ce qui avait été fait. Une fois les vérifications effectuées, j'ai pu ajouter le tout au projet et ainsi terminer mon second livrable.
Après avoir géré toutes les corrections, il a fallu résoudre un conflit qui existait avec la merge request sur GitLab.

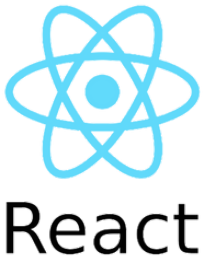
Ce fut une première pour moi.
J'ai constaté que pour régler un conflit avec Git, ce n'était pas très difficile.
Il suffit, dans un premier temps, de rassembler tous les commits d'une branche en un seul.
Ensuite, on pousse cette branche. Cette étape permet de résoudre les conflits plus facilement, sans avoir à examiner chaque commit pour identifier où le problème pourrait se situer.
Enfin, après avoir résolu les conflits, on fait un rebase de la branche, puis on pousse les modifications effectuées.

Avec tout cela, j'ai pu régler le conflit Git. La merge request a donc pu être acceptée, le livrable validé, et j'ai pu passer au frontend.

CRÉATION DU FRONTEND

Pour cette partie, je n'ai pas travaillé sur la vue utilisateur. En effet, nous n'avons reçu les maquettes que deux ou trois jours avant la fin du stage, ce qui ne m'a pas permis de créer une interface soignée pour l'application.
Cependant, l'objectif de cette étape était de permettre, à partir du frontend, de récupérer ou modifier les données de la base de données via le backend (comme le schéma de la page 8 le montre).

Pour réaliser ce frontend, j'ai utilisé React, un framework JavaScript. Au départ, je souhaitais utiliser du React pur, car il permettait une gestion des routes simple et très flexible, selon mon point de vue.

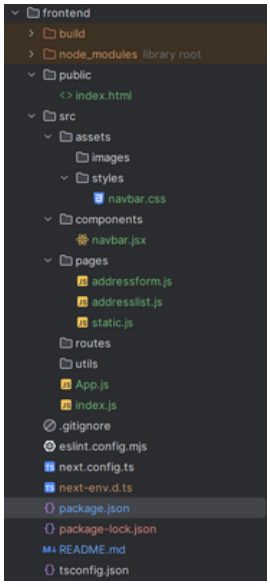


À ce moment-là, j'ai commencé à développer mon projet en React sans utiliser de framework spécifique. Pour tester l'affichage et comprendre comment les routes fonctionnaient, j'ai commencé par créer un fichier très simple qui affiche juste un texte sur la page web. Cela m'a permis de voir comment React gérait les routes.

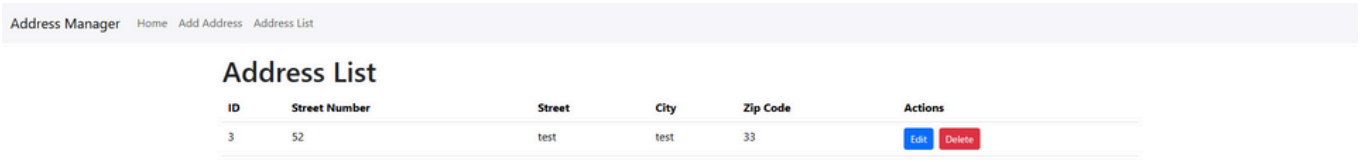
Ensuite, j'ai intégré la gestion du frontend avec le backend en utilisant Axios.
Axios est une bibliothèque JavaScript qui permet de faire des requêtes HTTP pour récupérer ou envoyer des données à un serveur. Elle simplifie le traitement des réponses en convertissant automatiquement les données en JSON et en gérant les erreurs de manière plus efficace. Axios est souvent utilisé dans les applications web modernes pour communiquer avec une API REST.

Dans mon projet, j'ai utilisé Axios pour échanger des données avec le backend. En plus de sa simplicité, Axios permet de configurer des headers globaux, ce qui est utile pour l'authentification. Il offre aussi une gestion avancée des erreurs et la possibilité d'intercepter les requêtes avant qu'elles ne soient envoyées, ce qui m'a permis de sécuriser et d'optimiser les échanges entre mon application React et le backend. Grâce à Axios, j'ai pu structurer mon code de manière plus propre.
Une fois le tout mis en place, j'ai pu tester l'application, qui était fonctionnelle.

La structure du projet ressemblait à cela :



Et la première page dynamique ressemblait à ça:



Address List

ID	Street Number	Street	City	Zip Code	Actions
3	52	test	test	33	Edit Delete

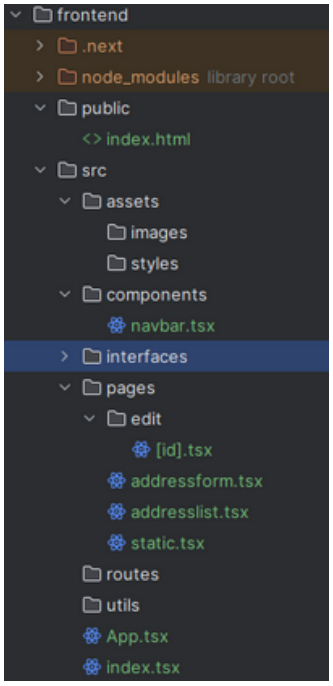
Mais après une réunion avec Paul, on a décidé d'utiliser un framework de React qui est Next. On a principalement décidé d'utiliser ce framework car il permet de gérer plus facilement le SEO.

En plus de devoir passer à Next.js, j'ai dû migrer mes fichiers JavaScript en TypeScript.

Pour ce faire, j'ai d'abord installé toutes les dépendances nécessaires dans le package.json. Ensuite, j'ai dû modifier la gestion des routes. En React pur, les routes se gèrent assez simplement, il suffit de les assigner comme ceci :

```jsx
import React from "react";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import Navbar from "./components/navbar";
import AddressList from "./pages/addresslist";
import Addressform from "./pages/addressform";
import Static from "./pages/static";

function App() {  Show usages  new *
    return (
        <Router>
            <Navbar />
            <div className="container mt-3">
                <Routes>
                    <Route path="/" element={<AddressList />} />
                    <Route path="/add" element={<Addressform />} />
                    <Route path="/edit/:id" element={<Addressform />} />
                    <Route path="/static" element={<Static />} />
                </Routes>
            </div>
        </Router>
    );
}

export default App;  Show usages  new *          You, 26 minutes ago · Uncommit
```

En Next, il faut créer un nouveau dossier nommé 'pages' dans le dossier 'src' du projet. Ensuite, on place les pages que l'on souhaite afficher dans ce dossier.
Il est important de bien faire attention, car le nom des fichiers correspondra à ce qui apparaîtra dans l'URL.

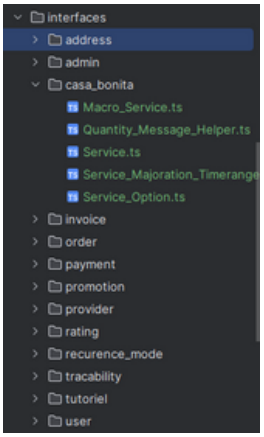


Si on prend un exemple, dans la structure qu'on peut voir à droite, on a plusieurs fichiers dans le dossier 'pages'. Pour les afficher dans un navigateur, il faut se baser sur le nom des fichiers.
Par exemple, si on veut afficher la page 'addresslist' dans le navigateur, il faudra taper l'URL suivante :
http://localhost:3000/addresslist

Le port 3000 correspond au port local du front, et 'addresslist' est le nom du fichier dans le dossier 'pages'.

C'est un sujet qui m'a posé un peu de problèmes, car malgré mes recherches, j'ai eu du mal à comprendre le fonctionnement de Next au niveau des routes. Mais une fois que c'était compris, je n'ai pas eu trop de difficultés à gérer le projet sous Next.

Une fois cela terminé et fonctionnel, j'ai mis les interfaces des pages dans des fichiers à part, car cela sera plus simple pour les réutiliser et évitera de devoir les recopier à chaque fois.



Après avoir reçu les maquettes, j'ai pu commencer à les réaliser, mais à cause du manque de temps, je n'ai pas pu les faire correctement, surtout que j'utilisais Tailwind CSS (qui est un framework CSS). Comme je découvrais Tailwind, je n'ai pas eu le temps de faire des maquettes suffisamment complètes.

Pendant toute la durée de la création du backend et du frontend, j'ai pu terminer le 4e livrable, qui était de rédiger la documentation de ce que j'avais réalisé sur ces sujets dans le fichier README.md.

Cela clôture donc les grands axes de mon stage, mais il reste encore deux sujets intéressants à aborder.

FIREBASE / STRIPE

Contexte

Pendant la partie frontend du projet, et étant donné qu'on n'avait pas les maquettes pour créer un visuel utilisateur, j'ai pu traiter deux sujets vraiment intéressants.
Ces sujets sont :

- Firebase

Firebase est une plateforme développée par Google qui propose plusieurs services pour aider au développement d'applications web et mobiles. Elle offre notamment une base de données en temps réel, une authentification utilisateur (avec Google, Facebook, email, etc.), un hébergement et des outils pour envoyer des notifications push. Son principal avantage est qu'elle permet de gérer facilement le backend d'une application sans avoir besoin de configurer un serveur, ce qui accélère le développement.

- stripe

Stripe est une solution de paiement en ligne qui permet aux entreprises et aux développeurs d'intégrer facilement des paiements sécurisés dans leurs applications ou sites web. Il prend en charge les cartes bancaires, Apple Pay, Google Pay, et d'autres méthodes de paiement. Stripe simplifie la gestion des transactions grâce à une interface simple et des API performantes, tout en assurant un haut niveau de sécurité grâce au chiffrement des données.

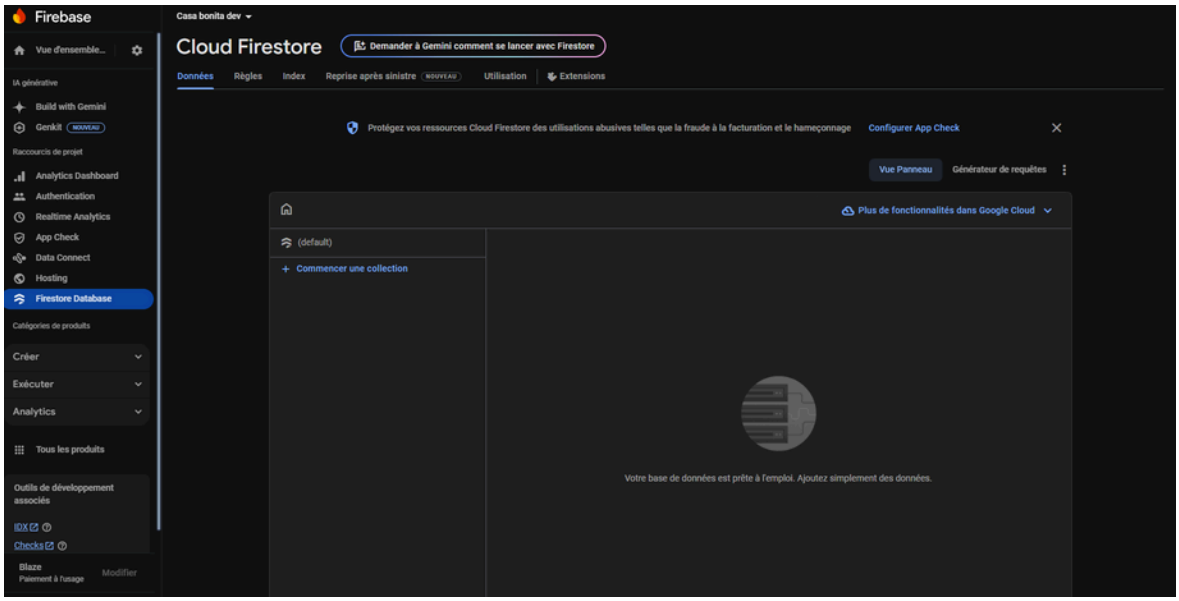
MISE EN PLACE DE FIREBASE

Firebase est probablement l'outil qui m'a posé le plus de problèmes et qui m'a pris le plus de temps durant ce stage. En effet, c'est un outil qui peut paraître terrifiant au premier abord, car il propose de nombreux services et il faut un peu de temps pour comprendre comment le mettre en place.

Pour ce stage, j'ai seulement géré la partie hosting, ce qui permet de déployer le site directement sur le web.

Pour bien débuter avec Firebase, j'ai dû énormément me renseigner, que ce soit en consultant des documentations ou en regardant des vidéos trouvées sur YouTube. Après plusieurs recherches, je n'ai pas réussi à trouver la bonne manière d'initialiser le projet, surtout que je ne voulais pas faire d'erreurs.

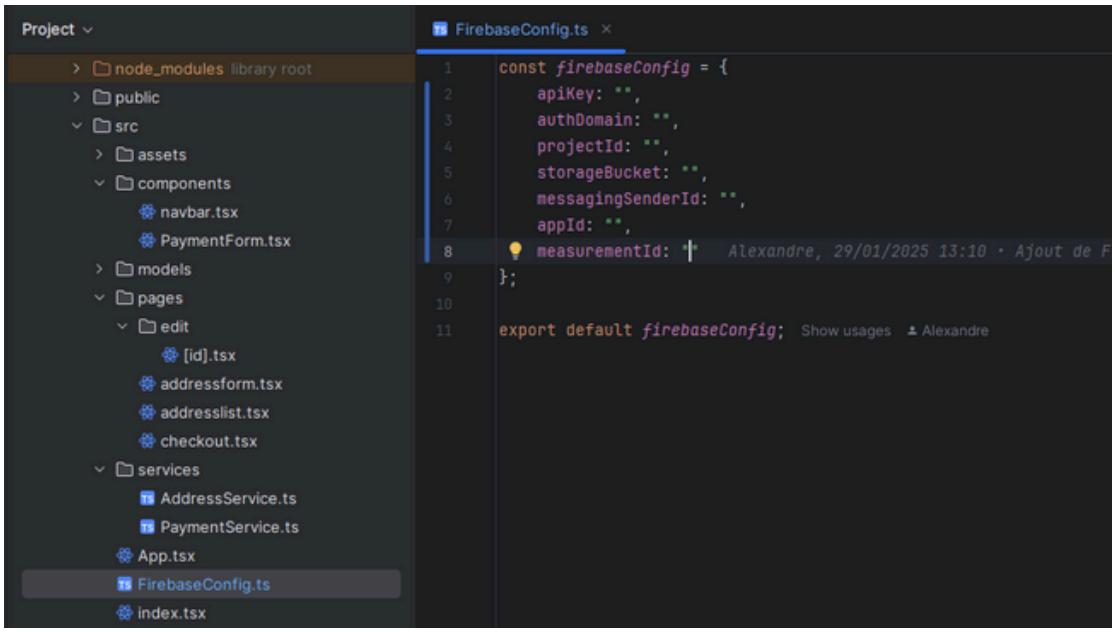
Lors de la réunion quotidienne, nous avons donc pu initialiser correctement le projet.
Pour initialiser la partie Hosting de Firebase, il faut dans un premier temps gérer la base de données de l'outil.
Comme ici :



Une fois que la base de données a été créée, nous avons dû créer la partie Hosting de Firebase.

Pour ce faire, plusieurs étapes étaient nécessaires, notamment ajouter la base de données que l'on venait de créer au Hosting. Nous avons également dû, par exemple, connecter et configurer des fichiers à Firebase.
Évidemment, les champs montrés sur la capture d'écran seront vides pour des raisons évidentes.

Exemple de configuration:



Une fois toute cette configuration effectuée, il a fallu faire en sorte que le projet soit en SSR, car jusqu'à ce moment, le projet était statique.
Pour rappel, le SSR est une technique qui permet de générer les pages web côté serveur avant de les envoyer au navigateur.

Pour gérer cette partie, il a fallu modifier directement la configuration de Next afin qu'il génère des fichiers SSR.

Une fois ce changement effectué, j'ai dû supprimer tous les fichiers temporaires générés par Node, à savoir : le dossier out, le dossier firebase et le dossier .next.
Une fois cela fait, j'ai buildé le projet avec Node.

Enfin, il ne me restait plus qu'à héberger le projet. Pour ce faire, il m'a suffi d'exécuter cette commande :

firebase deploy --only hosting

Tout cela m'a permis de terminer le 5e livrable.

MISE EN PLACE DE STRIPE

En plus des livrables réalisés précédemment, j'ai pu apprendre à gérer les paiements pour un projet.

C'est un sujet assez important lorsqu'on crée un projet, car l'objectif à terme est qu'il y ait du profit pour l'entreprise.

J'ai étudié deux façons d'utiliser Stripe, car je n'avais pas bien compris la première méthode qui m'a été expliquée. J'ai donc vu la manière dont Paul voulait gérer les paiements.

Dans un premier temps, j'ai utilisé Stripe pour effectuer des paiements directs. C'est un outil vraiment pratique, car il propose de nombreuses fonctionnalités, comme des champs préconfigurés ainsi que des cartes de test pour simuler des paiements réussis ou différentes erreurs possibles.

Pour gérer cette partie de manière directe, il suffit de créer un fichier dans le front.

Dans ce fichier, il faut inclure une clé API qui permettra de gérer Stripe via leur API. Ensuite, on crée un objet auquel on associe un prix et une description.

Enfin, on ajoute les champs à remplir grâce à la dépendance "@stripe/react-stripe-js", qui permet d'avoir des champs préconfigurés comme ceux-ci :

Enfin, il ne reste plus qu'à récupérer les cartes de test sur la documentation de Stripe pour faire des tests et vérifier si les paiements directs fonctionnent.

Dans un second temps, j'ai utilisé Stripe pour gérer des intentions de paiement. Cela a un peu compliqué les choses, car je devais aussi gérer une partie backend et travailler avec un projet à 3 couches (DTO, Service et Controller).

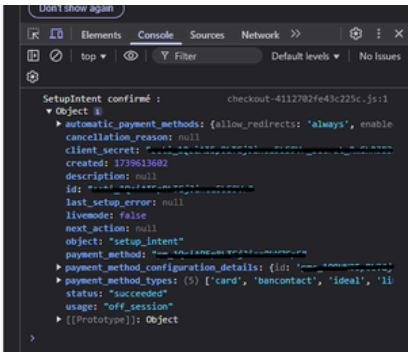
Il a donc fallu gérer le paiement dans le backend, mais je ne vais pas revenir sur les couches.
Cependant, il a tout de même fallu adapter le backend à l'API Stripe, car une clé spéciale pour le backend a dû être ajoutée.

Cette clé m'a fait perdre le plus de temps dans l'utilisation de Stripe, car j'avais mal copié la clé et utilisé une incorrecte, ce qui faisait que l'intention de paiement ne fonctionnait pas correctement.
Une fois que tout le backend a été terminé, je suis reparti sur le frontend.

Le but du frontend est assez simple : récupérer les éléments du controller du backend avec Axios. Ensuite, on définit le client secret et on vérifie qu'il correspond bien à celui du backend.

Enfin, il ne reste plus qu'à ajouter les éléments du formulaire pour tester si l'intention de paiement fonctionne correctement en vérifiant les informations directement dans la console.

Comme ceci:



Tout cela clôture donc ce qui a été fait durant mon stage et permet à PLR Conseil d'avoir une très bonne base sur laquelle s'appuyer pour Casa Bonita.

BILAN DES MISSIONS, DES COMPÉTENCES UTILISÉS & CONCLUSION

BILAN DES MISSION

Pour ce stage, ma mission a été de mettre en place un projet de A à Z, en créant toute la partie base de données, backend et frontend. De plus, j'ai également créé des parties importantes du projet avec Firebase et Stripe.

Pour construire toute la base du projet, j'avais des livrables à respecter pour la fin du stage, et tous ont été validés avec succès.

Dans un premier temps, j'ai géré la base de données en installant PostgreSQL sur ma machine.

Ensuite, j'ai dû utiliser Liquibase pour créer des migrations et pouvoir commencer le projet.
Pour la structure de la base de données, je me suis basé sur un schéma existant et ai refait tout le SQL du schéma pour l'appliquer.

Ensuite, j'ai travaillé sur le backend en utilisant ce système de couches. D'abord, j'ai récupéré les données de la base de données grâce à la couche DAO. Puis, j'ai créé la couche DTO pour ajouter une certaine sécurité pour les couches suivantes. Ensuite, j'ai géré les services, qui permettent d'appeler les fonctions devant être utilisées dans la dernière couche.

Enfin, les controllers ont été réalisés pour gérer les requêtes HTTP et ainsi manipuler les données. Cela m'a permis de gérer les données via HTTP.

Dans un troisième temps, j'ai géré la partie frontend. L'objectif était de communiquer avec le backend pour interagir avec la base de données via une interface.
Pour cela, j'ai utilisé le framework React Next.js. Grâce à cela, j'ai pu créer des pages dynamiques et fonctionnelles permettant de gérer les données avec des interfaces.

En parallèle, grâce à Firebase, j'ai pu héberger le projet, ce qui permet d'avoir une présence en ligne et, à l'avenir, de gérer simplement des éléments comme les utilisateurs ou les statistiques générales du projet.

Enfin, j'ai utilisé Stripe, qui permet de gérer facilement les paiements. Pour ce faire, j'ai dû revoir une partie du backend et adapter certains fichiers frontend.

En résumé, ce stage m'a permis de découvrir les points clés de la création d'un projet. Il m'a véritablement donné une vue d'ensemble de ce que doit gérer un développeur fullstack.
J'ai pu mettre en pratique des connaissances déjà acquises, mais j'ai surtout énormément appris durant ce stage.

BILAN DES COMPÉTRENCES UTILISÉES

Ce stage a été une véritable mine d'or en termes d'acquisition de compétences.
En effet, j'ai travaillé sur énormément de sujets très différents et, grâce à ce stage, j'ai pu beaucoup évoluer dans mes compétences techniques.

Dans un premier temps, si l'on parle purement technique, j'ai appris de nouvelles technologies que je ne connaissais pas et qui sont largement utilisées dans le milieu professionnel.
J'ai pu découvrir PostgreSQL, qui est un système de gestion de base de données relationnelle. Pendant mes études, j'ai déjà travaillé avec d'autres SGBD (MySQL, SQL Server...), mais je n'avais pas encore eu l'occasion d'utiliser PostgreSQL.

Ensuite, pour le backend, j'ai découvert Java SpringBoot, utilisé avec Maven pour gérer les dépendances. Je ne savais pas non plus utiliser Java. Cela a donc été l'occasion de découvrir ce langage et, en plus de cela, j'ai pu revoir une compétence importante étudiée en cours : le système de couches utilisé pour les API. C'est une notion très importante en développement backend et je savais que j'avais des difficultés avec cela. Grâce à ce stage, j'ai pu mieux comprendre son fonctionnement.

J'ai également découvert l'un des plus grands frameworks JavaScript : React avec Next.js. Comme pour le reste, je ne connaissais pas React avant de commencer ce stage, et maintenant je me débrouille plutôt bien avec ce framework.

En plus de tout cela, j'ai pu découvrir un outil très utile : PrimeReact. PrimeReact est une bibliothèque de composants UI pour React, qui permet de créer facilement des interfaces modernes et réactives sans avoir à tout coder soi-même. C'est un outil que j'ai découvert grâce à ce stage et qui m'a beaucoup aidé.

J'ai aussi pu me familiariser avec deux outils qui seront extrêmement importants pour mes futurs projets. En effet, avoir appris à utiliser Firebase et Stripe sera très utile pour de futurs projets et me fera gagner énormément de temps.

J'ai également pu utiliser des compétences non techniques, comme l'organisation, surtout quand je fais du télétravail. J'ai aussi utilisé des compétences en recherche et en communication. Ce sont des compétences importantes, car lorsqu'on travaille sur un projet partant de zéro, il est essentiel de savoir trouver les meilleurs outils à utiliser. Quant à la communication, il est crucial de pouvoir argumenter sur le choix des outils et de convaincre que tel ou tel outil est le plus adapté.

En résumé, ce stage m'a permis de retravailler certaines compétences, mais surtout d'en apprendre de nouvelles. Il m'a aussi permis de voir où je pouvais progresser et quelles compétences je devais développer pour que ma carrière professionnelle soit la plus réussie possible.

BILAN PERSONNEL

Ce stage a été très enrichissant, tant sur le plan personnel que professionnel. Il m'a permis d'acquérir de nouvelles compétences et de consolider celles que j'avais déjà. J'ai également pris conscience de certaines difficultés sur lesquelles je dois travailler, notamment le travail en équipe, qui reste un domaine assez compliqué pour moi, surtout dans un contexte de développement informatique. Collaborer avec d'autres développeurs m'a demandé une grande adaptabilité et une meilleure communication pour comprendre et intégrer les différentes visions du projet. Cela m'a aidé à mieux appréhender les attentes en entreprise et à améliorer ma manière de travailler avec les autres.

Grâce à ce stage et celui de première année, je sais désormais où me diriger et ce que je souhaite faire. Après mon dernier stage, je voulais me concentrer davantage sur le backend, car j'avais fait énormément de frontend, mais ce n'était pas ce que je préférais. Je me suis donc dit que je voulais surtout travailler sur le backend.

J'ai ensuite eu l'opportunité de faire ce stage dans cette entreprise. Cela m'a permis de me conforter dans l'idée que je préférais faire du backend, mais m'a aussi permis de découvrir un côté frontend qui m'a tout autant plu, grâce à des technologies que je n'avais pas eu l'occasion de voir auparavant. Je peux donc dire que je me suis un peu réconcilié avec le frontend.
Maintenant, je sais exactement où me diriger pour mon alternance. Cela va me permettre de chercher une entreprise idéale pour mon profil et, par la même occasion, mettre en avant le meilleur de ce que je peux proposer.

En conclusion, ce stage m'a permis de mieux comprendre mes forces et mes faiblesses, tout en clarifiant mon projet professionnel. J'ai gagné en expérience et je sais maintenant dans quelle direction je souhaite avancer pour la suite de ma formation et de ma carrière. Je ressors de cette expérience avec une plus grande confiance en mes capacités et une motivation renforcée pour continuer à apprendre et à progresser.

CONCLUSION

Ce stage m'a permis de clarifier mon projet professionnel, d'acquérir et de renforcer de nombreuses compétences, tout en étant très bien entouré par des professionnels passionnés par ce métier, ce qui a renforcé mon envie de faire carrière dans ce domaine.

Ce stage a donc été très bénéfique. C'est pourquoi je tiens à remercier encore PLR Conseil de m'avoir accueilli pour ce stage, car ils m'ont grandement aidé.