Rapport technique
JEE

Corentin BOULC'H Lucas DANSAY Alexis CHOUPAULT
Julien GOUBET Erwan LE PRADO1

Sommaire

Cahier des charges

L’objectif du projet est de développer une application web en JEE répondant au cahier des charges suivant. La liste est organisée par objet et méthodes.

- Utilisateur
  - Création
  - Connexion
  - Déconnexion
- Client
  - Consulter
  - Top 10
  - Ajouter
  - Supprimer
  - Modifier
- Véhicule
  - Ajouter
  - Modifier
  - Supprimer
- Location
  - Ajouter
  - Finaliser
  - Recherche

Conception

La conception s’est faite dans le cadre de la matière UML-UP. La totalité des diagrammes ont été réalisés pour chaque fonctionnalité et objet.
Afin de rappeler les concepts clefs du projet, à suivre le diagramme des classes.

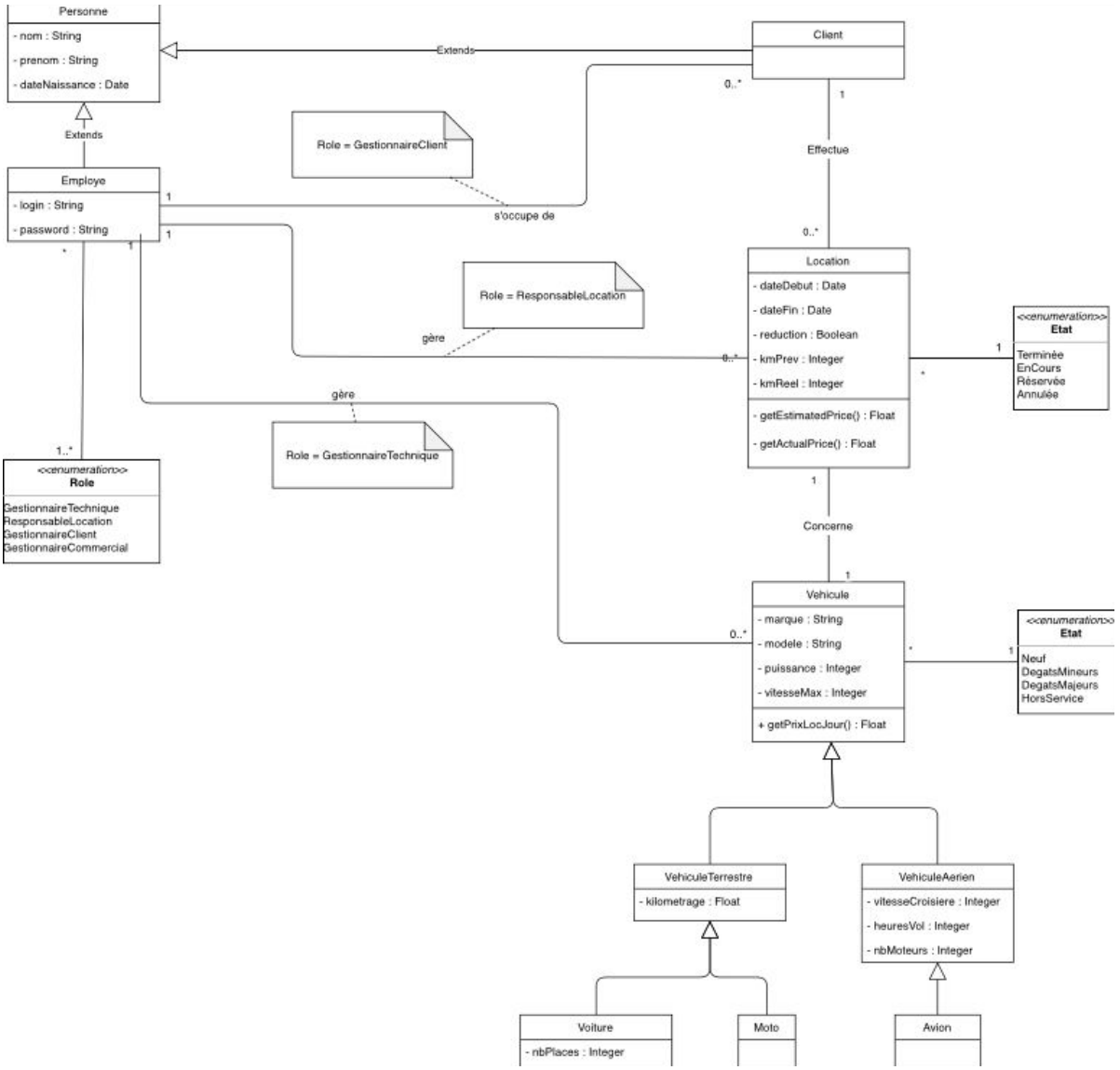

Fig 1 : Diagramme de classe du projet

Architecture

Il a été choisi d'adopter une architecture classique en MVC. Ainsi, chaque vue est reliée à un servlet. Les servlets délèguent la responsabilité de la logique aux services. Ces derniers se connectent à la base de données via une couche DAO. Il est possible de résumer l'architecture choisie par la figure ci-dessous.

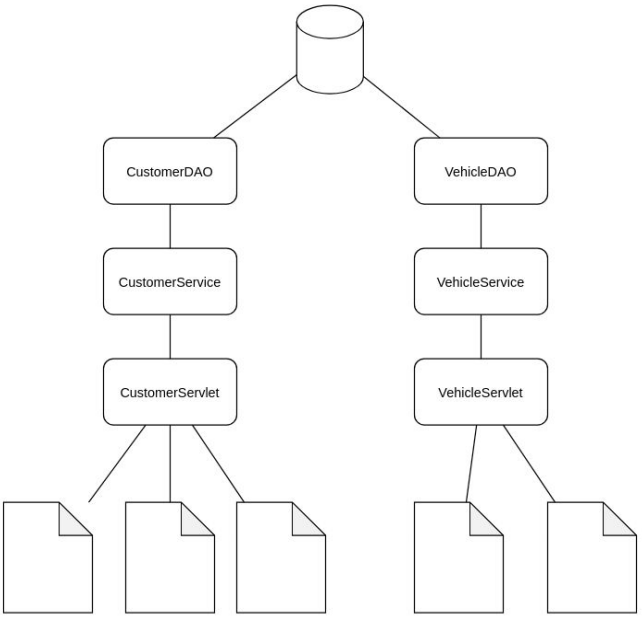

Fig 2 : Architecture globale du projet

Le principal intérêt de cette découpe du projet est de limiter les responsabilités de chaque entité. En effet, les vues sont chargées de la communication entre l'utilisateur et le serveur. Les servlets doivent gérer les routes ainsi que répondre aux requêtes HTTP effectuées par les vues. Une fois les requêtes reçues, les servlets transforment la donnée brute en une information exploitable par le système d'information. Toutefois, ce n'est pas à cette couche de traiter l'aspect métier de ce qui a été demandé par l'utilisateur. De ce fait, le traitement est effectué par les services et dès lors que des actions en lien avec de la persistance de données sont requises, les différents DAO communiquent avec la base de données.

Il est important de noter que la découpe des servlets/services/DAO s'est faite principalement en fonction du diagramme de classe préalablement établi. Il sera donc possible de retrouver un servlet Customer qui répondra aux requêtes sur cet objet ainsi qu'un service et une couche DAO dédiés.

Ces choix permettent d'assurer d'un côté une maintenabilité et de l'autre une évolutivité simple de mise en œuvre.

La description des sources

La base de données

Cette partie va traiter de la méthode choisie pour traiter la persistance des données. En effet, les choix de conception ainsi que les outils utilisés seront exposés.

Hibernate

La connexion à la base de données n'est pas effectuée directement via du code mais par le framework Hibernate. Ce dernier apporte une solution générique et facile d'implémentation. Un simple fichier de configuration contenant les identifiants, URL de connexion et dialecte SQL suffit à la connexion. Cependant, dans un contexte professionnel cette solution ne serait pas acceptable car les identifiants peuvent-être trouvés depuis un dépôt Git. De plus, il n'y aurait pas de distinction entre les différents environnements. Pour contrer ces deux problèmes, l'utilisation de variables d'environnement mises à jour lors des déploiements pourrait être une solution.

L'apport principal d'Hibernate réside dans le fait que les requêtes effectuées par ce framework renvoient directement des objets correspondant au modèle mis en place au préalable. De plus, le framework permet de définir une validation des différents attributs des classes avec l'aide de décorateurs.

Il est également important de noter que le dialecte utilisé pour requêter n'est pas du SQL standard mais du HQL, la syntaxe est donc légèrement modifiée. Toutefois Hibernate traduit automatiquement les requêtes HQL vers le dialecte choisi en configuration.

Les sources

Les servlets

Les servlets sont un point majeur du fonctionnement de JEE. Ils indiquent globalement deux choses : le point d'entrée (via l'URL) et la méthode HTTP utilisée.
Ainsi un servlet qui à pour route : /test et comme méthode : GET pourra traiter la demande de l'utilisateur : http://monsiteweb.com/test.
Les autres méthodes sont principalement utilisées de façon transparente à l'utilisateur. On peut citer POST qui sert à transporter des données du client au serveur par exemple.
En d'autre terme les méthodes sont conventionné :

- GET : pour l'affichage simple de données.
- POST : pour la création d'objets.
- PUT : pour la mise à jour des objets.
- DELETE : pour la suppression des objets.

Les servlets ont pour caractéristique de n'avoir aucun ou très peu de logique. L'idée est assez simple : le servlet récupère par l'intermédiaire des services toutes les données requises à l'affichage de la vue et fait en sorte qu'elles soient accessibles par cette dernière.

```java
@WebServlet(urlPatterns = {"/", "/accueil"})
public class HomeServlet extends HttpServlet {

    public static final String INDEX_VIEW = "/index.jsp";

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws Servlet
        this.getServletContext().getRequestDispatcher(INDEX_VIEW).forward(request, response);
    }

}
```

Fig 3 : Exemple d'un servlet (HomeServlet)

Par exemple : HomeServlet est joignable par la route / ou /accueil de notre site web. Elle n'est disponible que par la méthode GET (**doGet()**) et affichera la vue : index.jsp.

Les services

Les services traitent de la partie logique du site web. Ils font souvent l'objet d'intermédiaire entre les servlets et la base de données.

Le service va se référer à un objet qui lui est propre (Ex : Voiture) pour proposer la logique haut niveau de ce dernier et dans le plus souvent des cas la logique CRUD : CREATION, READ, UPDATE or DELETE.

Ainsi, n'importe quel service propose des méthodes simples : create, update, delete, findById, findAll… Toutes permettent une manipulation simple des objets sans se soucier de la base de données ou de la structure du modèle.
À savoir que ces méthodes sont des méthodes qui utilisent des méthodes du DAO. Il faut voir le service comme un intermédiaire plus haut niveau que DAO.

Dans notre projet, nous avons fait le choix d'avoir une classe Service qui sert de base à l'ensemble des autres services. Cela permet de garder une constance pour la convention d'écriture du code. En d'autre termes, les autres services héritent de Service.

Les DAO

Concernant la conception de la connexion à la base de données, l'utilisation du design pattern DAO semblait évidente tant il est utilisé par de nombreux projets. Dans ce contexte, une classe générique `DAO` a été créée. En effet, lors de l'instanciation de cette classe, le type d'objet que va manipuler `DAO` est spécifié. Ainsi des méthodes génériques d'interaction avec la base telles que `findById`, `delete` ou `merge` sont écrites dans cette classe.

Dès lors que des besoins spécifiques en fonction d'un objet du modèle sont requis, une classe héritant de `DAO` est créée. Par exemple, l'instanciation de `EmployeeDAO` va donc créer une instance de `DAO` avec `Employee.class` en paramètre. Les méthodes supplémentaires de `EmployeeDAO` correspondent à des aspects métiers. Un employé étant censé pouvoir se connecter à l'application, des méthodes de vérification de mot de passe ainsi que login sont ajoutées.

Les modèles

Les modèles sont la dernière étape avant d'arriver à la base de données.
Les modèles sont la définition d'un objet en base de données. Un simple objet java interprété par Hibernate permettent la traduction d'objet à SQL et SQL à objet.
Le DAO traite directement avec le modèle, c'est lui qui s'instancie ou l'injecte en base de données.

Les vues

Les vues sont des fichiers au format JSP. Elles représentent la partie visuel du projet. C'est donc le servlet qui s'occupe d'injecter des données dans celles-ci.
Les vues sont en fait des fichiers au contenu multiple. Son contenu y sera envoyé à l'utilisateur, libre à l'utilisateur de savoir comment traiter ceci (Exemple : JSON, XML ou du HTML).
Dans notre cas et dans l'esprit du projet que l'on a appliqué, nos vues sont des fichiers HTML.

JSP propose alors quelques outils afin de simplifier l'exploitation des données.

```
<html>
<head>
    <title>Création de nouvel utilisateur</title>
</head>
<body>
    <% if(result != null){%>
        <% if(result){%>
            <span>L'utilisateur a bien été créé</span><br/>
        <% } else { %>
            <span>Impossible de créer l'utilisateur</span><br/>
        <% } %>
    <%}%>
    <% if(errorsList != null){%>
        <% for(String err : errorsList){%>
            <span><%=err%></span><br/>
        <%}%>
    <%}%>
    <form method="post" action="createEmployee">
```

Fig 4 : Exemple partiel d'une page JSP

Grâce à des instructions équivalentes au java, nous pouvons ou non afficher des informations différentes en fonction de la donnée injectée. Ici dans le cas d'un résultat nul, on affiche une balise de type span différente.

Cheminement MVC

Pour récapituler :
Un utilisateur appelle un service qui affiche un client à l'adresse suivante : GET /client/1.
Le servlet configuré pour écouter sur /client en GET va s'occuper de récupérer l'identifiant du client dans l'URL.
Le servlet récupère l'objet client avec un findById(1) du service associé au client : CustomerService. Customer étant client en anglais.
Le service client est déjà instancié avec le DAO client. Permettant ainsi de par les méthodes CRUD du service d'utiliser les méthodes du DAO.
Ainsi, la méthode findById du service appelle la méthode au même nom du DAO.
Comme le DAO est configuré pour résoudre l'objet Customer (objet du dossier modèles), la méthode nous renvoie bien un objet Customer.
En fin de compte, le service renvoie un client ou "null" en fonction de la réponse.
Le servlet s'occupe de transférer la variable par une clef à la vue pour que celle-ci la récupère.
La vue appliquera une stratégie d'affichage approprié à la page.

Les filtres

Les filtres ajoutent une couche de protection des routes. Ainsi, chaque servlet décoré par un filtre peut se retrouver bloqué.
Un filtre très utilisé est celui de la connexion. Cela permet de ne pas autoriser les requêtes vers des servlets protégés. Si l'utilisateur non connecté tente d'accéder à une page protégé, le filtre va s'occuper de soit indiquer une erreur soit le rediriger vers la page /login.
La connexion est gérée par un cookie de session, à la fin de toutes les vérifications on détermine si le cookie de session existe. Si oui, c'est qu'il est connecté.

D'autres filtres peuvent servir. Un filtre d'admin est utilisé par exemple pour n'autoriser que les admins à interagir avec des pages spécifiques. Ce filtre va regarder à partir de l'identifiant utilisateur trouvé par la session, si le rôle d'administration lui est attribué en base de données.

Les utils

Les utils sont des fonctions utilisées n'ayant que pour seule fonction des tâches simples.
Les utils peuvent être de toute nature comme formater une date en une chaîne de caractères lisible ou bien appliquer un algorithme de recherche sur une liste par exemple.
Utiles car elles sont réutilisables bien plus qu'une seule fois et pour n'importe quel contexte.

Les méthodes développement

La gestion de projet

Afin d'être rigoureux sur les tâches à accomplir, nous avons, et ce depuis la rédaction des diagrammes, utilisé trello. Trello est une application web mettant en pratique la méthode kanban.

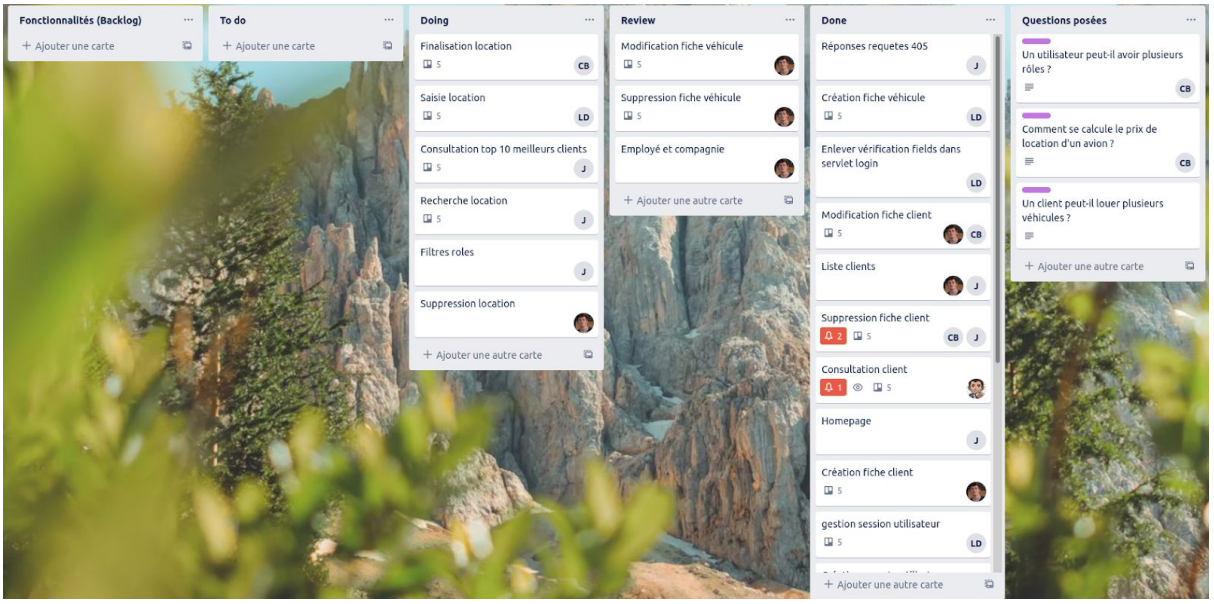

Fig 5 : Tableau Trello du projet JEE

Grâce à cette pratique, nous pouvions voir à tout moment l'avancée d'une tâche : par qui, quand et comment ?

Logiciels utilisés

Pour développer

Afin de rendre le développement assez souple, nous avons opté non pas pour la machine virtuelle proposée mais pour un système de conteneurs.
En effet, avec cette méthode nous avons gardé les performances natives de nos ordinateurs, couplées à notre IDE favori pour développer le projet.
Notre IDE pouvait in fine produire un fichier war (fichier étant le livrable final de l'application).
Il ne manquait plus qu'un serveur Tomcat fourni par un fichier docker compose. Ce fichier docker compose déposé à la racine de notre projet s'occupait d'interpréter le fichier war depuis l'extérieur du conteneur vers son intérieur (un "bind" de fichier).
Une simple commande permettait donc de lancer un serveur Tomcat et de tester notre projet.

Gestion du git

Nous avons utilisé un dépôt GitLab pour notre code. Très rapidement, nous avons mis en place une convention très stricte pour la validation du code.
Cela commence par le gitflow. Afin de simplifier le développement, tout se passait sur la branche main (anciennement master).

Dès lors que nous voulions créer une fonctionnalité, nous avions à créer une branche préfixée par feature/ en indiquant le nom de la fonctionnalité.
De même pour des “fix”, la branche devait contenir ce mot précisément pour indiquer sa nature.

Une fois que le responsable de la branche jugeait sa branche opérationnelle, il devait demander un merge request. Toute fusion entre une branche de feature vers main, directement était interdite. La fusion devait être validée obligatoirement par deux personnes.
C’est l’occasion de relire le code proposé s’il répond bien aux exigences de convention de nommage ou vérifier s’il n’y pas de conflit.

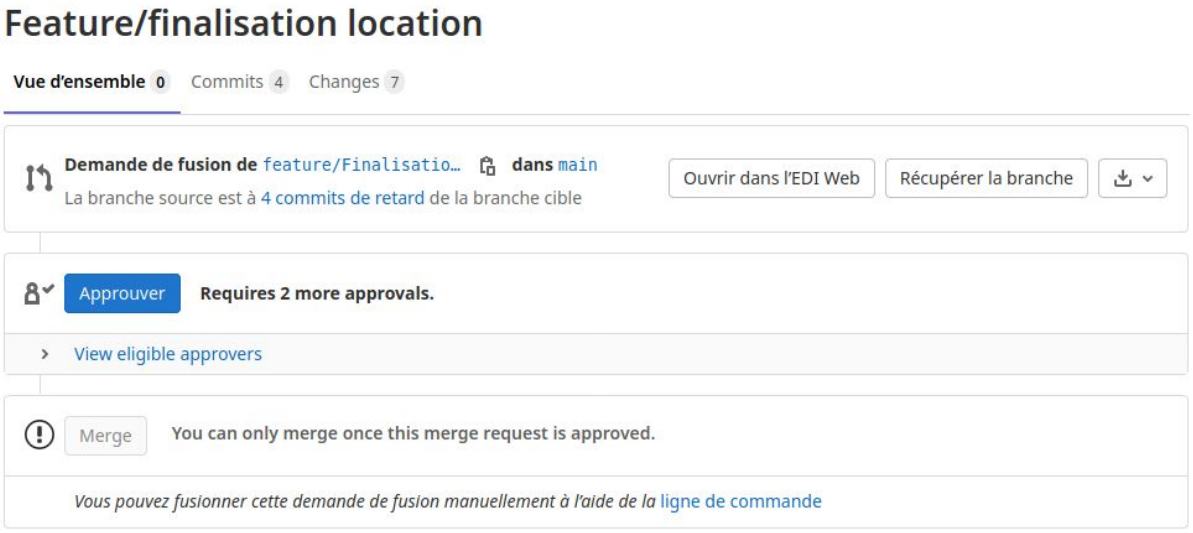

Fig 6 : Demande de fusion d’une feature

Chacun utilisait une interface graphique ou non pour git. Dans certains cas, la gestion s'est faite avec le logiciel gitkraken. Gitkraken propose une gestion graphique très intuitive de git et aide par exemple à résoudre des conflits très simplement.
Son autre avantage est la visualisation du gitflow claire.

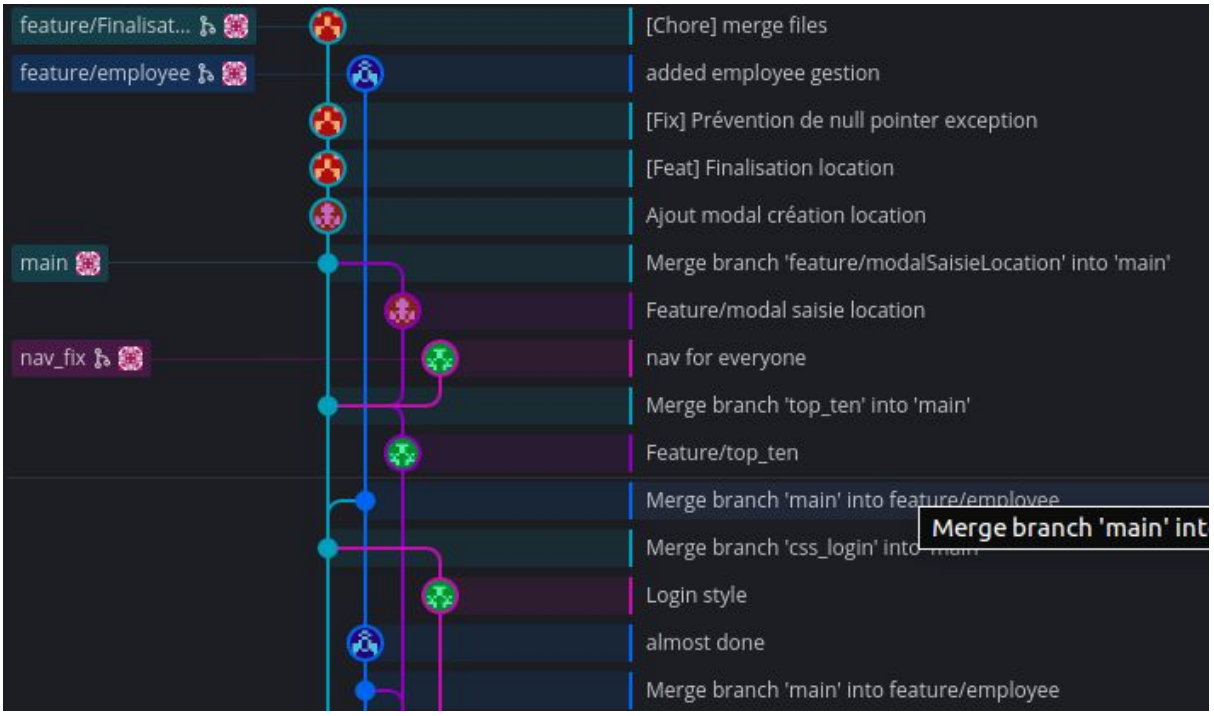

Fig 7 : Le gitflow du projet (via gitkraken)

Esthétique et design de l’application

Pour développer un design esthétique simplement et rapidement nous avons opté pour un système bootstrap couplé au material design de chez Google.
Nous sommes donc partis sur mdBootstrap (https://mdbootstrap.com/) qui propose un large choix pour avoir de jolies pages très rapidement et simplement.

Fig 8 : Page login du projet

Conclusion

Le projet JEE nous a imposé un rythme assez soutenu d'apprentissage. Même si la courbe d'apprentissage de JEE est atteignable, il a fallu trouver une organisation robuste rapidement. Nos expériences dans certains domaines nous ont permis d'être très rapidement prêt pour les sprints de création de fonctionnalités.
Tous les domaines de compétences ont été mis à contribution : Gestion de projet (Trello, gitlab), connaissance de java (IntelliJ, Java, JEE) et docker (avec docker compose). Chacun s'est mis à niveau très rapidement.
Malgré cela, la deadline de rendu est arrivée très vite. Le projet est certes terminé et fonctionnel mais ne sera pas abouti comme nous le souhaitons.