

Rapport technique

ALNEZAMI Ibrahim

BRICAUD Dorian

FRAYSSE Cyrille

PEPIN Wilfried

PLESTAN Kévin

Introduction	4
Organisation	5
Gestion des tâches	5
Gestion des versions du code	6
Technique	8
Base de données	8
Ajout de classes	9
Les enums	9
Les DAO (Data Access Object, ou Objet d'Accès aux données)	9
Les services	9
Conception	11
Pour l'organisation du code source, nous sommes parties sur une architecture MVC (Model - View - Controller).	11
Model	11
Vue	11
Contrôleurs	11
Gestion des clients	12
Création	12
Liste	12
Afficher une fiche client	12
Modification	12
Suppression	12
Gestion des véhicules	12
Création	12
Liste	12
Afficher une fiche client	13
Modification	13
Suppression	13
Gestion des locations	13
Création	13
Liste	13
Paieement	14
Autres pages	14
Authentification	14
Déconnexion	14
Accueil	14
Statistiques	14
Pistes d'amélioration	14
Difficultés rencontrées	15
Conclusion	15

Introduction

Ce rapport décrit la réalisation de la partie JEE du projet UML-JEE.

Nous sommes repartis de ce que nous avons fait en UML pour commencer notre partie JEE. Par exemple, nous avons créé nos controllers en fonction de ceux définis dans notre partie UML et avons implémenté chacune des classes de notre diagramme de classes. Dans ce rapport, nous présenterons notre organisation, ainsi que la réalisation de ce projet

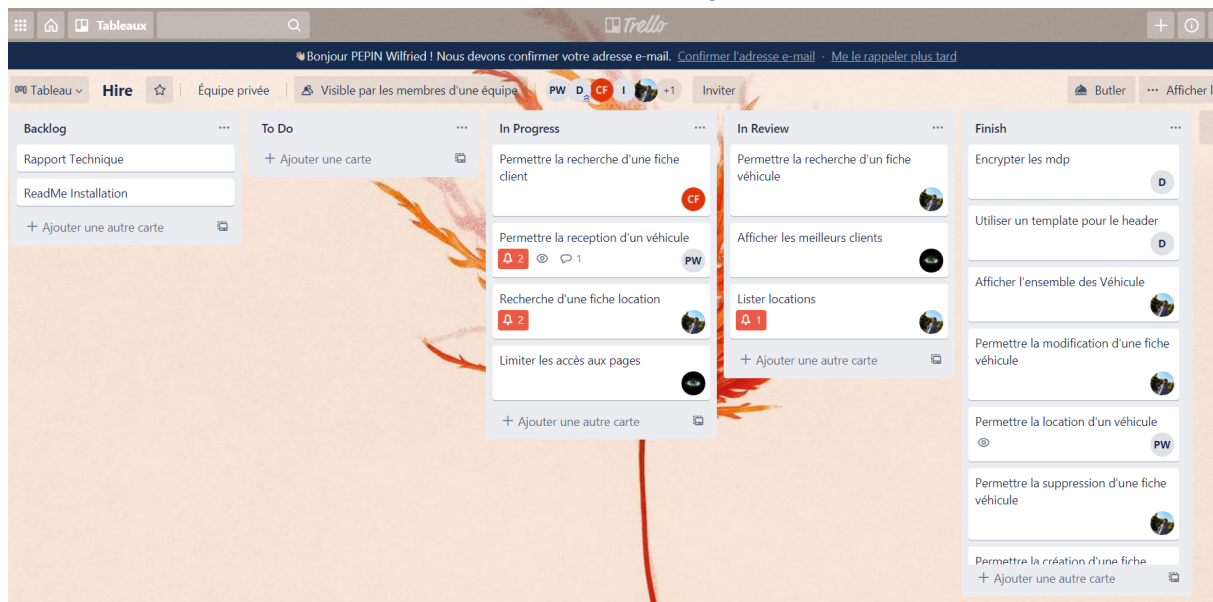
L'ensemble du code est disponible <https://github.com/Groupe-5-JEE/Hire/tree/main>. Afin de se connecter à l'application certains identifiants, mots de passe sont déjà définis :

- general / admin (profil Responsable de Locations)
- technical / azerty (profil Gestionnaire Technique)
- commercial / azerty (profil Gestionnaire Commercial)
- customer / azerty (profil Gestionnaire Client)
- employee / azerty (profil Standard)

Organisation

Gestion des tâches

Afin de nous répartir les tâches nous nous sommes organisés à l'aide du site Trello.



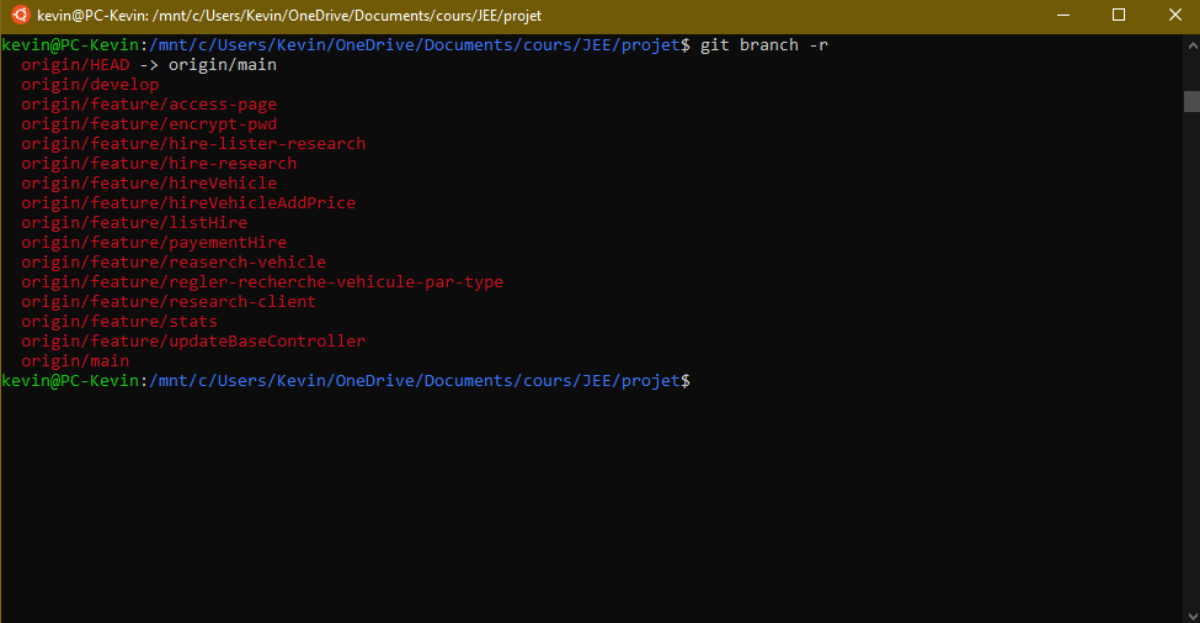
Comme vous pouvez le voir ci-dessus, nous avons créé plusieurs colonnes qui correspondent aux différentes étapes de la vie d'une partie de notre solution. Les différentes fonctionnalités à développer ont été listées dans la colonne "Backlog". Après priorisation et identifications des tâches à effectuer, étape par étape, le chef de projet les déplace dans la colonne "To do", lorsqu'elles sont prêtes à être réalisées.

À partir de cet instant, n'importe qui peut s'attribuer une des tâches "To do" et la mettre en "In progress". Une fois le développement terminé, la carte est déplacée dans la colonne "In review". Elle est en attente de vérification par un autre membre de l'équipe. Enfin, une fois validée, la carte est placée dans "Finish".

Cette organisation permettait une grande autonomie des différents membres du groupe, chacun pouvait développer les tâches restantes et en informer le groupe en faisant progresser cette dernière dans le trello. Enfin le code était relu avant d'être intégré dans le reste du projet afin d'uniformiser le code et s'assurer du fonctionnement du projet.

Gestion des versions du code

Pour la centralisation et la gestion du versionnage de notre code, nous avons utilisé le gestionnaire de version Git.



```
kevin@PC-Kevin: /mnt/c/Users/Kevin/OneDrive/Documents/cours/JEE/projet$ git branch -r
origin/HEAD -> origin/main
origin/develop
origin/feature/access-page
origin/feature/encrypt-pwd
origin/feature/hire-lister-research
origin/feature/hire-research
origin/feature/hireVehicle
origin/feature/hireVehicleAddPrice
origin/feature/listHire
origin/feature/payementHire
origin/feature/research-vehicle
origin/feature/regler-recherche-vehicule-par-type
origin/feature/research-client
origin/feature/stats
origin/feature/updateBaseController
origin/main
kevin@PC-Kevin: /mnt/c/Users/Kevin/OneDrive/Documents/cours/JEE/projet$
```

L'organisation du Git, et de ses différentes branches, s'est appuyé sur la méthodologie Git flow. Cette méthodologie définit plusieurs branches standard :

- master(ici main) : est la branche principale. Elle correspond au code de l'application à un instant donné.
- release : branche contenant le suivi des différentes livraisons. Chaque livraison à un commit apportant toutes les modifications et une étiquette, portant le nom de la release. Pour livrer la dernière version, cette branche est mergée dans la master.
- develop : branche principale de développement. Elle contient tous les ajouts de fonctionnalités jugés stables. Lorsque assez de fonctionnalités ont été ajoutées à l'application, develop est mergé dans la branche release.
- feature/* : branche de développement d'une nouvelle fonctionnalité. Toutes branches feature prennent comme base la version la plus récente de develop. Une fois la fonctionnalité développée et validée, elle est merge dans develop.

De cette façon, nous avons pu travailler chacun sur des fonctionnalités différentes sans trop de conflits, et nous permettait d'avoir une branche de développement propre.

Tout notre code est centralisé sur la plateforme Github à l'adresse suivante :

<https://github.com/Groupe-5-JEE/Hire/tree/main>. Grâce à la plateforme, nous avons, notamment, pu gérer les relectures de code grâce à la fonctionnalité de "Pull Request". Une Pull Request est une demande de fusion entre une branche (dans notre cas, branche de feature) et la branche de développement (develop). Lors de cette relecture, nous avons la possibilité de mettre des commentaires afin que le développeur de la fonctionnalité la

corrige. Enfin, le chef de projet, étant le seul ayant le droit de modifier les branches main et develop, confirme la fusion des deux branches.

Technique

Base de données

Pour la gestion de la relation entre notre application et la base de données, nous avons choisis de nous appuyer sur le framework Hibernate.

Hibernate est un ORM (Object-Relational Mapping), son travail est de faire le lien entre le monde objet (code Java) et le monde relationnel (base de données MariaDb).

Pour ce faire, nous définissons des classes de type POJO (Plain Old Java Object) qui n'ont que pour but de regrouper des données. Par exemple, nous avons une classe "Employee" qui contient toutes les informations d'un employé, tel que son nom, prénom, identifiant, mot de passe, Ces objets POJO (également appelés entités dans le monde des ORM), sont détectés par Hibernate, grâce aux annotations Java.

Après détection de ces classes, il peut générer un schéma de base de données correspondant et peut également le mettre à jour après modification d'une entité.

Ce dernier permet aussi de prendre des objets Java et de les enregistrer de manière transparente en base de données. Pareil pour la mise à jour des données d'un objet ou la suppression de celui-ci.

Enfin, il permet de récupérer les objets à partir de la base de données soit :

- Grâce à leur identifiant
- Soit grâce à un langage proche du SQL (JPQL) qui permet de faire des requêtes sur les objets. Ces requêtes sont ensuite converties dans le dialecte SQL propre au serveur de base de données utilisé

Hibernate s'adapte à un grand nombre de serveurs de base de données (Oracle, MySQL, MariaDb, PostgreSQL, ...) et génère donc les requêtes SQL adaptées au serveur auquel il est connecté.

Ajout de classes

Les enums

Nous avons fait le choix d'utiliser des enums pour faciliter la création des formulaires et uniformiser les valeurs au sein de l'application.

Ces enums sont utilisés au sein de certaines entités pour décrire un état. Nous les utilisons également pour afficher leur valeur pour construire les listes déroulantes lors de la création ou modification des entités.

Par exemple :

- StatePayment : le statut de la location. L'utilisateur a-t-il payé ? On a deux choix : NotPaid et Paid
- StateVehicle : L'état de la voiture : neuve, abîmée, passable.

Les DAO (Data Access Object, ou Objet d'Accès aux données)

Afin de mieux abstraire l'accès à nos données, nous avons mis en place une couche d'accès. Cette couche est constituée d'interface et d'implémentation de ces interfaces.

Ainsi, nous avons une interface par entité à manager. Ces interfaces définissent les différentes méthodes d'accès aux données pour l'entité à laquelle elle est rattachée.

Par exemple, nous avons des méthodes qui permettent de récupérer tous les véhicules (pour afficher la liste), récupération des véhicules selon un critère (utilisé pour la recherche), récupération de la liste des véhicules disponibles à la location sur un certain interval de temps (deux dates, utilisé pour afficher les véhicules disponible lors de la création d'une nouvelle location.).

Dans notre application, il y a, actuellement, une implémentation de ces interfaces. Ces implémentations se basent sur Hibernate (au travers de l'API JPA), pour récupérer les données, créer / modifier / supprimer des données en base de données. Il est facile d'ajouter une nouvelle implémentation de ces interfaces qui permettrait alors d'utiliser un autre système de stockage de données.

Les services

Enfin, nous avons mis en place des classes services au sein de notre application. Ces classes se découpe en grandes catégories :

- Classes utilitaires utilisables à plusieurs endroits
 - Classe de chiffrement de mot de passe (Encrypt)
 - Classe de filtrage des requêtes entrantes et sortantes pour indiquer que toutes les requêtes sont encodées avec l'encodage de caractères UTF-8 (CharacterSetFilter)
 - Classe donnant accès à l'instance partagée de l'EntityManager permettant de travailler avec Hibernate au sein de l'application (DBManager)
 - Vérification des droits d'accès (EmployeeServiceImp)
- Interface de service d'accès et gestion en relation avec les entités

- CustomerServiceInterface : permet l'accès et modification des entités de clients
 - EmployeeServiceInterface : permet l'accès et modification des entités d'employés ainsi que la vérification des droits d'un employé
 - HireServiceInterface : permet l'accès et modification des entités de location
 - VehicleServiceInterface : permet l'accès et modification des entités de véhicule
- Classes implémentant ces services. Ces classes utilisent les DAO utilisant Hibernate pour les accès et gestion des entités

Conception

Pour l'organisation du code source, nous sommes parties sur une architecture MVC (Model - View - Controller).

Model

Nous avons implémenté le modèle défini lors de notre partie UML, quelques modifications ont été apportées au fur à mesure de nos besoins, notamment l'ajout de nouveaux attributs. Dans ces classes nous avons dû ajouter les annotations permettant d'utiliser Hibernate. Toutes ces classes se trouvent dans le package *com.hire.model*.

Vue

Pour les vues nous avons rangé l'ensemble de nos jsp dans le dossier *WebContent/views*. Nous avons ensuite appliqué une arborescence proche de nos chemins urls, ainsi les vues concernant les clients sont stockées dans le dossier *customer*, pour les véhicules le dossier *vehicle*, etc...

Nous avons aussi mis en place des templates pour le header et le footer afin de faciliter l'uniformisation visuelle des vues. Malgré cela, certaines pages n'ont pas été remises en forme intégralement avant la fin du projet.

Contrôleurs

Nous avons une HttpServlet (controller) par URL.

Une servlet est liée à une URL. Par exemple, le contrôleur "CreateHireLocationController" gère la création d'une nouvelle location.

Un contrôleur peut gérer les différentes méthodes du protocole HTTP. Les méthodes que nous avons gérées dans l'application sont :

- La méthode GET
- La méthode POST

Ces chemins suivent une certaine logique, et se rapprochent dans le nommage de la convention REST.

Par défaut le /Hire gère les locations :

→ /Hire/create => créer une location

On a aussi /Hire/client pour la gestion des clients et /Hire/vehicle pour la gestion des véhicules.

→ /Hire/customer/sheet?id=6 ⇒ affiche la fiche client qui a l'id numéro 6

→ /Hire/vehicule/search ⇒ affiche la liste des véhicules

Afin de centraliser certains traitements et vérifications, nous avons une classe "BaseController" qui est héritée par tous nos contrôleurs. Celle-ci propose des méthodes comme la vérification du statut d'authentification d'un utilisateur (avec ou sans redirection vers la page de login en cas de non authentification), des méthodes de redirections vers nos

vues / page d'accueil et une méthode retournant l'employé actuellement connecté s'il y en a un.

Gestion des clients

Création

URL : /Hire/customer/create

C'est un formulaire qui permet de créer un client. On y entre les informations usuelles pour ce type de création.

Liste

URL : /Hire/customer/search

Pour le listing des clients, tout se passe dans "ResearchCustomerController". La servlet va chercher la liste des clients via le service CustomerServiceInterface, qui lui-même appelle le DAO CustomerDAOInterface qui fait la requête hibernate. Cette requête renvoie toutes les données sur tous les clients dans la base de données.

Afficher une fiche client

URL : /Hire/customer/sheet?id=<id_du_client>

Cette fiche est accessible à partir de la liste des fiches. Il suffit de cliquer sur afficher la fiche pour un client précis.

Dans cette fiche, on a toutes les informations essentielles d'un client, plus la possibilité de le modifier ou de le supprimer à l'aide du bouton correspondant.

Modification

URL : /Hire/customer/modify?id=<id_du_client>

Pour faire une modification, il faut d'abord cliquer sur le bouton "modifier" sur la fiche d'un client. Un formulaire vous est ensuite fourni pour que vous y fassiez vos modifications.

Suppression

URL : /Hire/customer/delete?id=<id_du_client> (Ne doit pas être utilisé comme tel à moins de connaître l'id d'un client)

La suppression se passe dans la fiche d'un client. Il suffit de cliquer sur le bouton "supprimer".

Gestion des véhicules

Les chemins d'accès pour la gestion des véhicules sont similaires à ceux d'un client.

Création

URL : /Hire/vehicle/create

C'est un formulaire qui permet de créer un véhicule. On y entre les informations usuelles pour ce type de création.

Liste

URL : /Hire/vehicle/search

Pour le listing des clients, tout se passe dans "ResearchVehiculeController". La servlet va chercher la liste des clients via le service VehicleServiceInterface, qui lui-même appelle le DAO VehicleDAOInterface qui fait la requête hibernate. Cette requête renvoie toutes les données sur tous les clients dans la base de données.

Afficher une fiche client

URL : /Hire/vehicle/sheet?id=<id_du_vehicle>

Cette fiche est accessible à partir de la liste des fiches. Il suffit de cliquer sur afficher la fiche pour un client précis.

Dans cette fiche, on a toutes les informations essentielles d'un client, plus la possibilité de le modifier ou de le supprimer à l'aide du bouton correspondant.

Modification

URL : /Hire/vehicle/modify?id=<id_du_vehicle>

Pour faire une modification, il faut d'abord cliquer sur le bouton "modifier" sur la fiche d'un client. Un formulaire vous est ensuite fourni pour que vous y fassiez vos modifications.

Suppression

URL : /Hire/vehicle/delete?id=<id_du_vehicle> (Ne doit pas être utilisé comme tel à moins de connaître l'id d'un client)

La suppression se passe dans la fiche d'un client. Il suffit de cliquer sur le bouton "supprimer".

Gestion des locations

Création

URL : /Hire/create

La création d'un véhicule vous propose deux formulaires :

Le premier vous permet de sélectionner la date de début et la date de fin de la location.

Cette saisie permet au serveur de chercher les véhicules qui sont libres pendant cette fourchette de date. En effet, un véhicule ne peut pas être loué deux fois en même temps.

Après cette saisie, le site va afficher un second formulaire, avec les informations pour la location (le client, le véhicule, le kilométrage prévisionnel). Le prix prévisionnel sera calculé dans le back. La location a aussi comme statut de paiement "non payée".

La création renvoie ensuite sur la liste des locations.

Liste

URL : /Hire/research

Affiche la liste des locations. On a les informations sur les locations. Le bouton "modifier" ne fonctionne pas. On a pas eu le temps de le finir.

Le bouton "Payer" renvoie vers un formulaire pour le paiement.

Paielement

URL : /Hire/return?id=<id_de_la_location>

Un formulaire s'affiche alors, il faut entrer le nombre de kilomètres réellement parcourus par le client. Le prix va se mettre à jour et le statut de paiement de la location va passer en "payée".

Autres pages

Authentification

URL : /Hire/

Une page d'authentification s'affiche, cette page redirige vers la page Home si on est déjà connecté.

On est redirigé vers cette page dès lors que l'on est pas connecté.

Déconnexion

URL : /Hire/logout

Permet de se déconnecter et redirige vers l'authentification.

Accueil

URL : /Hire/home

Page d'accueil permettant d'accéder aux autres pages.

Statistiques

URL : /Hire/stats

Permet d'afficher les pages statistiques.

Pistes d'amélioration

Parmi les différentes pistes d'amélioration, nous avons :

- Optimisation des différents services de recherches via plus de critères
- Uniformisation des visuels (not
- Nous n'avons pas pu faire toutes les fonctionnalités aussi complètes que ce que nous visions au départ, elles pourraient être reprise

Difficultés rencontrées

Durant le développement de ce site web, nous avons rencontré plusieurs difficultés :

- Lors de la prise en main du framework Hibernate et plus particulièrement lors :
 - du rajout du fichier "context.xml" au fichier war, pour fournir la ressource d'accès à la base de données pour Hibernate.
 - De l'utilisation de requêtes JPQL plus complexes (avec JOIN par exemple)
- Volonté de restructurer l'arborescence de l'application (ajout d'un package global) en fin de développement. Ceci a généré une grande quantité de "merge conflicts".
- Nous avons rencontré quelques difficultés avec la gestion des encodages de caractères. Notamment, lors de la mise en place du chiffage du mot de passe, qui nous a momentanément empêché de nous connecter, après sa mise en place, sur les PC Windows.
- Enfin ce projet nécessitait l'utilisation de git, nous n'avions pas tous l'habitude de l'utiliser, cela a parfois ralenti le développement.

Conclusion

Grâce au travail réalisé sur la partie UML nous avons pu réaliser le JEE sereinement. Nous avons implémenté les différents contrôleurs et classes selon ce qu'on avait prévu.

Bien entendu la mise en place a demandé de nombreux ajustements, sans doute à cause de notre manque d'expérience mais aussi pour des raisons d'organisation et d'optimisation.

La charge de travail était conséquente mais nous avons réussi à livrer à temps un code fonctionnel qui répond au cahier des charges. Si nous avions eu plus de temps nous aurions sans doute pu rajouter quelques fonctionnalités mais nous sommes satisfait du travail effectué et du rendu final.