POLYTECH NICE SOPHIA | UNIVERSITÉ CÔTE D'AZUR

Rapport final de projet

Architectures Logicielles

M. Guilhem Molines & M. Philippe Collet

Groupe C
Paul-Marie DJEKINNOU
Paul KOFFI
Florian AINADOU
Djotiham NABAGOU

08 Janvier 2021 | Département des Sciences Informatiques (AL) 5A

PLAN

Récapitulatif de l'existant

1. Rappel du sujet traité et des fonctionnalités implémentées

Pour ce projet, il nous a été attribué la variance V1 du sujet général qui consiste à créer deux applications: une web et une mobile au travers desquelles les utilisateurs auraient la possibilité d'effectuer des réservations de billets de train.

Les fonctionnalités implémentées pour le POC sont les suivantes :

- La possibilité offerte à l'utilisateur aussi bien sur l'application web que l'application mobile d'acheter un billet de train.
- Nous avons considéré que la liste des trains disponibles était fournie par un système extérieur. Ces données ont donc été moquées.
- Les places de trains n'ont que les états "payé" ou "non payé".
- La répartition des places est faite de façon aléatoire.
- La banque étant un service externe, nous avons supposé dans un premier temps que toutes les opérations seraient acceptées par la banque.
- La prise en compte du compte client authentifié et effectuant des opérations de réservation.

Au travers de ces fonctionnalités, nous avons décidé d'implémenter les **user stories** suivantes pour le POC :

- Création de compte utilisateur
- Consultation de la liste des trains en fonction d'un trajet
- Visualisation du récapitulatif de ma réservation
- Paiement d'une réservation
- Visualisation de mon billet de train (réservation déjà effectuée)
- Annulation d'une réservation par un utilisateur
- Finaliser sur mobile une réservation commencée sur l'application web

2. Architecture implémentée

Suite aux fonctionnalités choisies, nous avons mis en place l'architecture suivante représentée par un diagramme de composants et un diagramme technique mettant en avant les technologies utilisées et l'intégration continue :

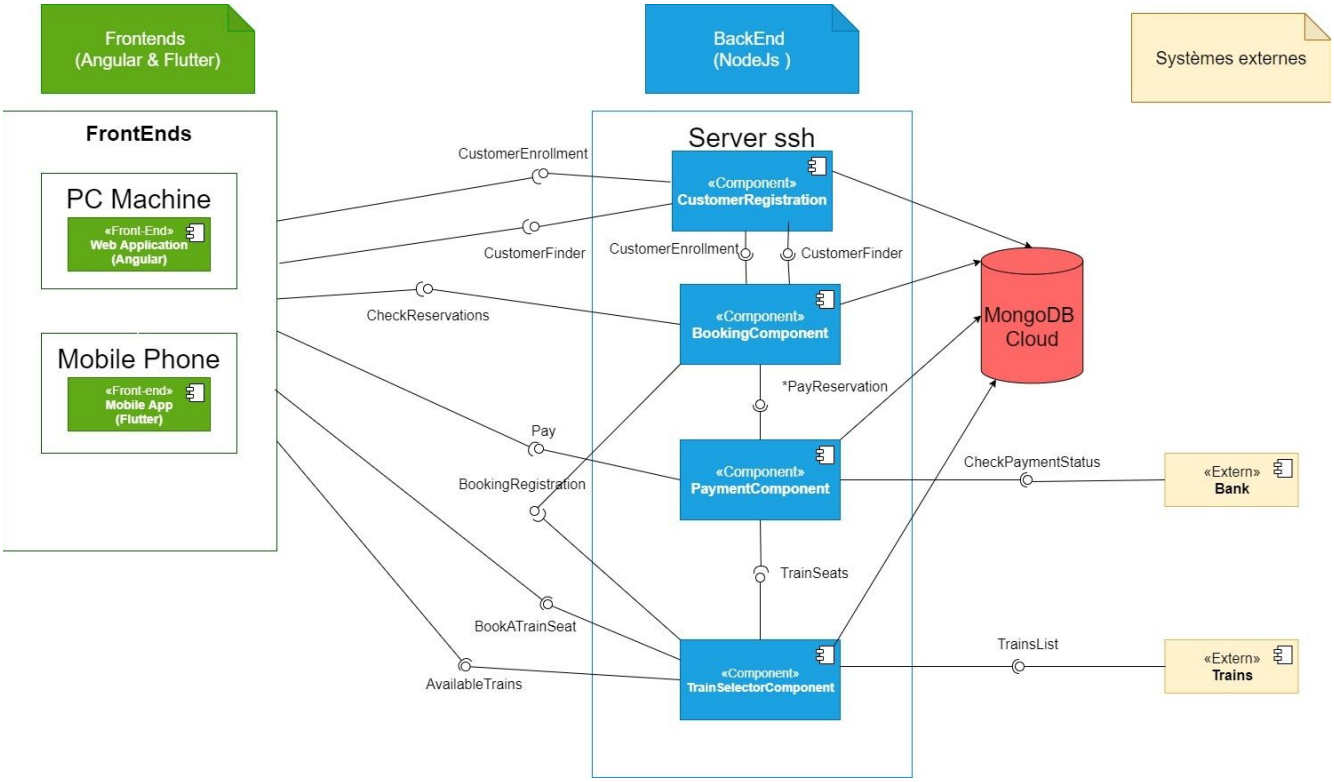

Figure 1 : Architecture des composants du système train-booking

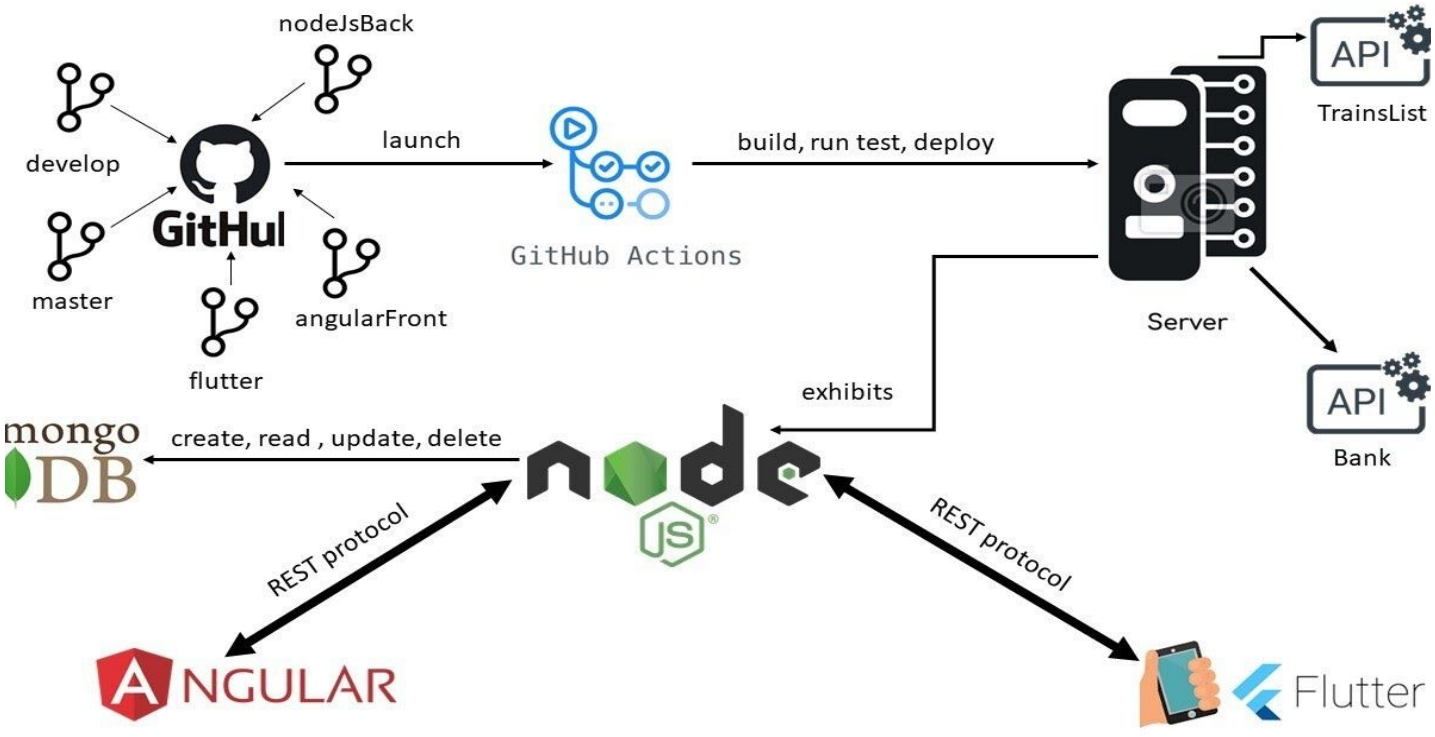

Figure 2 : Architecture général du système train-booking

Analyse de l'existant

1. Forces

Une base de données sur le cloud

Dans la solution proposée, nous avons fait le choix d'utiliser une base de données sur le cloud. Cela nous garantit un stockage de données centralisé qui permet à chaque membre de l'équipe de travailler librement sans avoir à dupliquer notre environnement de BD en local via un autre processus (Dockerisation par exemple). Nous avons donc les MAJ en temps réel. De plus, cela nous permet d'assurer la disponibilité de notre solution à moins que notre fournisseur CLOUD ne soit lui-même sujet à une panne.

Une architecture monolithique certes, mais facilement basculable

Pour la réalisation de notre POC, nous avons opté pour une architecture monolithique au détriment d'une architecture en micro-services. Les fonctionnalités à implémenter ne nous demandaient pas la souplesse d'utiliser des micro-services. Nous avons néanmoins pris soin de faire une structure plus ou moins fine de notre système monolithique dans le sens où chacun de nos composants est constitué :

- d'une couche modèle
- d'une couche fonctionnalité
- et enfin d'une couche gateway interne ou "routes" qui permet d'accéder à nos fonctionnalités.

Nos composants sont donc faits à la manière d'une architecture (Model - Service - Controller).

L'architecture est monolithique parce que notre BackEnd principal tourne donc sur un seul serveur qui présente l'API GATEWAY qui sert de routage et de point d'entrée à notre système. Pour basculer à une architecture en micro-services, il nous suffira juste de lancer chacun de nos différents composants sur un serveur qui leur est propre, et le tour sera joué. Ayant pris soin d'utiliser nos différentes fonctionnalités à travers des routes pour faire abstraction , basculer nos composants en micro-services est donc facile.

Un BackEnd centralisé et déployé continuellement

Au vu de notre sujet, l'un de nos défis techniques principaux était d'assurer une cohérence des données entre l'application mobile et notre client Web. Pour ce faire, notre premier choix fort était celui d'une BD en ligne et ensuite celui de centraliser notre BackEnd en le déployant sur un serveur ssh distant. Nos 2 applications à tout moment s'adressaient donc au même serveur distant qui représentait l'API de notre solution. Cela nous permet aussi dans le futur de pouvoir connecter facilement d'autres clients potentiels.

De plus à chaque nouvelle modification du BackEnd, une série de test s'exécute pour vérifier le bon fonctionnement de celui-ci et ensuite le BackEnd est mis à jour sur le

serveur distant. On s'assure donc à tout moment d'avoir notre backEnd distant dans un état viable. En détail, tout notre système monolithe est copié sur le serveur et grâce à PM2 qui est un outil de gestion de processus pour Javascript, nous redémarrons le serveur.

Nos choix technologiques

Pour notre application mobile , nous avons choisi Flutter pour profiter de son côté multiplateforme. En effet, une application de réservation de trains va concerner à long terme une multitude d'utilisateurs qui n'utilisent sûrement pas un seul type de système mobile. En optant pour une techno multiplateforme on s'assurait donc de ne pas avoir un blocage technologique dûe à cette divergence après en fournissant ainsi une application accessible à tous.

Avec l'évolution de nos fonctionnalités à long terme, l'introduction d'un système de gestion de notifications tels que **Firebase notifications** serait indispensable pour les alertes et les modifications en temps réel dans notre application. Nous nous sommes donc assurés à l'avance que notre technologie mobile soit compatible avec **Firebase notifications** ainsi que notre technologie BackEnd pour gérer les notifications en amont.

2. Faiblesses

La scalabilité

Pour le moment, notre application est un monolithe chargé de fonctionnalités. Dans le cas où notre application serait en proie à un grand nombre d'utilisateurs, nous avons peu de certitudes sur la capacité de notre architecture à être mise à l'échelle convenablement. En effet, la scalabilité ne serait pas réellement gérée correctement, vu que nous n'aurions pas la possibilité de faire passer à l'échelle les fonctionnalités qui connaîtraient une forte demande utilisateur notamment:

- Le **TrainSelectorComponent** qui permet d'obtenir à partir d'un formulaire, la liste des trains permettant d'effectuer un certain trajet
- Le **BookingComponent** qui est chargé d'attribuer des places dans un train et de gérer le système de réservations.

La disponibilité

Dans notre choix de base de données, nous avons choisi une base de données de type CP: mongoDB. Ce choix nous a fait sacrifier la disponibilité des données. C'est un choix que nous avons fait afin de garantir la tolérance à la partition pour une application qui devra passer à l'échelle. Néanmoins, ce choix pourrait affecter l'expérience utilisateur.

La gestion du système en cas de panne ou d'erreur

Un des principaux problèmes avec notre application actuelle est la gestion des pannes et des erreurs. En effet, en cas de panne, l'application n'est pas encore assez solide pour se relever convenablement. Nous n'avons pas encore mis en place un système de gestion d'erreurs

complexes dans le système. Aussi, pour ce qui est des retours utilisateurs, quelques améliorations restent à être mises en place afin de garantir une meilleure expérience utilisateur.

Des tests avancés

Nous avons des tests, mais nous n'en avons pas suffisamment. En effet, nous avons réalisé des tests afin de garantir le fonctionnement de notre application. Nous n'avons néanmoins pas encore assez de tests pour des scénarios plus poussés et plus complexes. Nous devons également faire d'avantages de tests sur le fonctionnement indépendant des composants.
Par ailleurs, nous devrons aussi effectuer des tests utilisateurs et des tests automatisés sur les applications Web et Mobile afin de garantir le fonctionnement de bout en bout de l'application.

L'utilisation d'un langage ubiquitous

Nous n'avons pas encore pris le temps de nous assurer que notre application emploie correctement les termes utilisés dans le domaine du transport ferroviaire. Cela pourrait permettre sur le long terme de nous assurer lors des tests utilisateurs avec des professionnels de nous assurer que nous respectons bien le fonctionnement de réservations de billets de train dans le métier.

Fonctionnalités futures

Au cours du second bimestre nous allons apporter les fonctionnalités suivantes :

- La prise en compte du positionnement dans le train :

Nous allons proposer un ou plusieurs algorithmes de placement des utilisateurs en fonction des places disponibles et de certains paramètres du client comme par exemple être assis à côté de d'une fenêtre, d'une personne dont il connaît la place, en première ou seconde classe. Pour implémenter cette fonctionnalité il faudra écrire des algorithmes avec en entrés les préférences du client dans notre composant "**BookingComponent**" qui se chargera de dérouler l'un d'entre eux pour attribuer une place au client. Ainsi ce dernier pourra voir la place qui lui est attribuée.

- La notion de fidélité et d'avantage :

Le client pourra utiliser des bons de réductions sur l'achat de ses billets en fonction de sa fidélité (par exemple par rapport à son nombre d'achat) ou d'autres avantages comme des promos ou comme dans le cas de la SNCF des cartes avantages. Il faudrait compter le nombre de billets achetés par le client afin de lui proposer un bon réduction créé par notre système donc sûrement un autre composant qu'il pourra appliquer. S'il renseigne une carte avantage il faudra récupérer la réduction puis l'appliquer automatiquement sur le prix de tous les billets. Le client aura la possibilité de regarder la réduction sur le prix avant de choisir sa place.

- Un mail de confirmation sera reçu une fois le paiement validé par la banque

Un mail sur l'état de sa commande qu'elle soit annulée, validée ou réservée afin qu'il puisse avoir une trace de ses commandes.

- La notion de délais pour une réservation

Le client après avoir réservé un billet, peut le payer plus tard dans le délai imparti sinon le billet réservé est retiré et donc la place libérée. Relancer le client par des mails afin de lui rappeler de payer avant la fin du temps imparti. Définir un délais par défaut pour toute réservation et relancer automatiquement le client par rapport à l'avancée du temps qui lui reste et annuler sa réservation en cas de non paiement (donc libérer la place afin que d'autre la récupère)

- Mettre en place une alerte pour être notifié si une place se libère

Lorsque le client veut réserver une place dans le un train qui est complet, il demande à être notifié qu'une place se libère en cas d'annulation d'un billet ou de non paiement d'une réservation. Garder un tracker sur les places de ce train et notifier le ou les client(s) concerné(s) par cette place (utilisation de FireBase Notifications pour les notifications).

- Système de 2nde et 1ère classe

Les trains proposeront des places à différents prix pour indiquer les places de seconde ou première classe. Proposer un algorithme de prix qui prend en compte les gares de départ et d'arrivées pour proposer un prix en fonction pour la 2nde et 1ère classe.

- Tester la scalabilité de notre système en envoyant plusieurs requêtes de réservation de places pour voir comment notre système réagit et si cela correspond à nos attentes.

- Mettre en place un système de gestion de pannes et des erreurs

Notre système devant être résilient aux pannes et aux erreurs, nous mettrons en place un système de compensation ou d'autres patterns afin de pallier cela.

- Faire des tests d'intégrations de notre backend beaucoup plus complexes afin de voir si nous n'avons pas des erreurs que nous n'avons pas traitées.

Rétrospective

L'aboutissement de ce projet nous a permi de nous rendre compte de certains aspects de notre projet que nous aurions pu aborder autrement :

- La priorité dans l'implémentation des fonctionnalités :

Nous avons priorisé d'implémenter certaines user stories au détriment d'autres lors des premières semaines et parmi celles-ci, nous avions choisi par exemple la création de compte utilisateur et l'authentification qui au final se sont avérées moins importantes et nous ont ralenti dans notre développement. L'implémentation a certes été rapide sur le frontend web mais a ralenti le développement des fonctionnalités sur le frontend mobile. Si c'était à refaire, nous implémenterions directement les fonctionnalités phares de l'application tel que la réservation des billets de train qui a pris plus de temps que prévu sur le frontend mobile.

- Soutenance :

Notre groupe a été aussi surpris par la note de soutenance "**13**" qui au final s'est avérée être la note la plus faible de toutes les soutenances. Nous nous demandions toujours si cela était uniquement dû à **l'utilisation de Prezi** qui a perturbé la présentation de la soutenance. Au final nos principaux reproches étaient surtout sur l'utilisation de cet outil de présentation et aussi parce que l'on avait pas assez mis en évidence les différentes interactions sur notre serveur externe (choix d'utilisation des couleurs sur la console et n'avoir pas pris le temps de montrer en détails quelles étaient les différents logs). L'utilisation de Prezi est donc à proscrire.

- Suivi de projet :

De même, notre note de suivi de projet a été impactée par nos 2 semaines de retard pendant lesquelles nous avons mal représenté notre diagramme d'architecture que nous avons fait dans un premier temps à haut niveau plutôt que d'opter pour une architecture montrant en détail les composants ainsi que les interactions. Cela a entraîné par la suite un autre retard dans nos justifications technologiques et dans le développement. A la fin nous nous en sommes plutôt bien sortis au vu de la réalisation de toutes les fonctionnalités prévues et de nos justifications technologiques. Néanmoins notre note de suivi de projet s'en est trouvée impactée car nous n'avons pas été réguliers du début à la fin et on aurait préféré logiquement éviter cela en partant sur de bonnes bases dès le début.