Rapport Projet ISA/DevOps

Equipe R

Elias Merdaci

Aubin Rahmani

Wassim Baratli

Hadil Ferchichi

26 Janvier 2023

Université Côte d'Azur - Polytech Nice - SI4-ISA/DEVOPS

2023 - 2024

TABLES DES MATIÈRES

1. Introduction 3
2. Cas d'utilisation 3
3. Diagramme de classe 4
4. Diagramme de composants 5
5. Choix d'implémentation de la persistance 5
6. Forces et faiblesses 5
7. Axes d'améliorations 6
8. Répartition des points 7

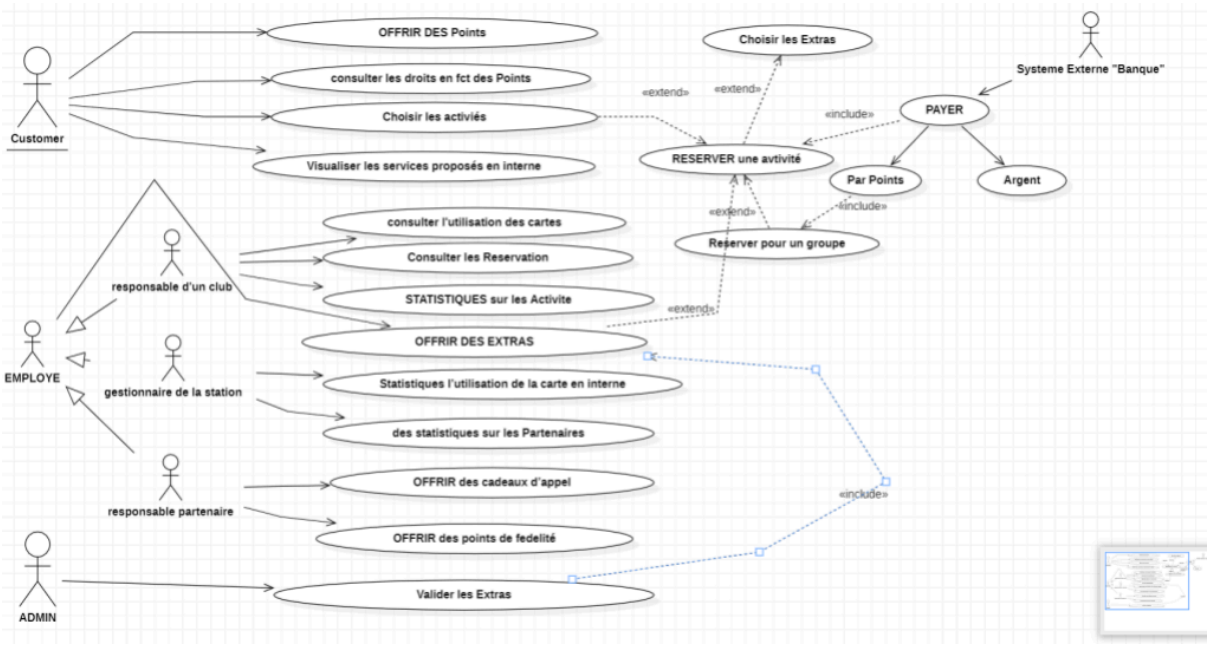
1. Introduction

Ce rapport présente l'architecture logicielle proposée pour un système de carte multi-activités destiné à un club de vacances. L'objectif principal de ce système est de fidéliser les hôtes du club en leur offrant la possibilité de réserver des séjours personnalisés et de bénéficier d'avantages exclusifs lors de leurs visites dans les stations de ski partenaires.

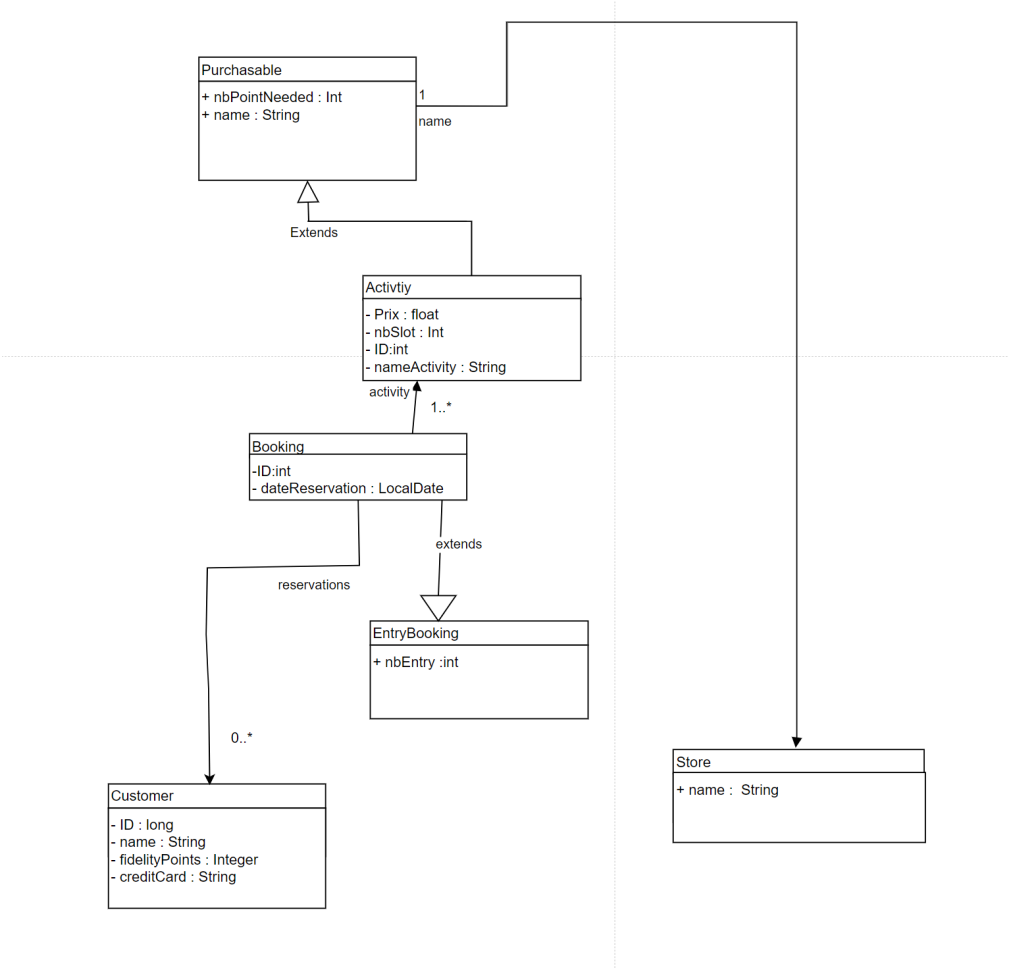
2. Cas d'utilisation

Le système de carte multi-activités répond à divers cas d'utilisation, notamment la réservation d'activités, l'accumulation et l'utilisation de points de fidélité, la consultation des avantages disponibles, la gestion des réservations et des stocks, ainsi que l'analyse de l'utilisation du système. Ces fonctionnalités sont conçues pour répondre aux besoins des différents acteurs, tels que les hôtes du club, les employés, les partenaires de la station et les administrateurs.

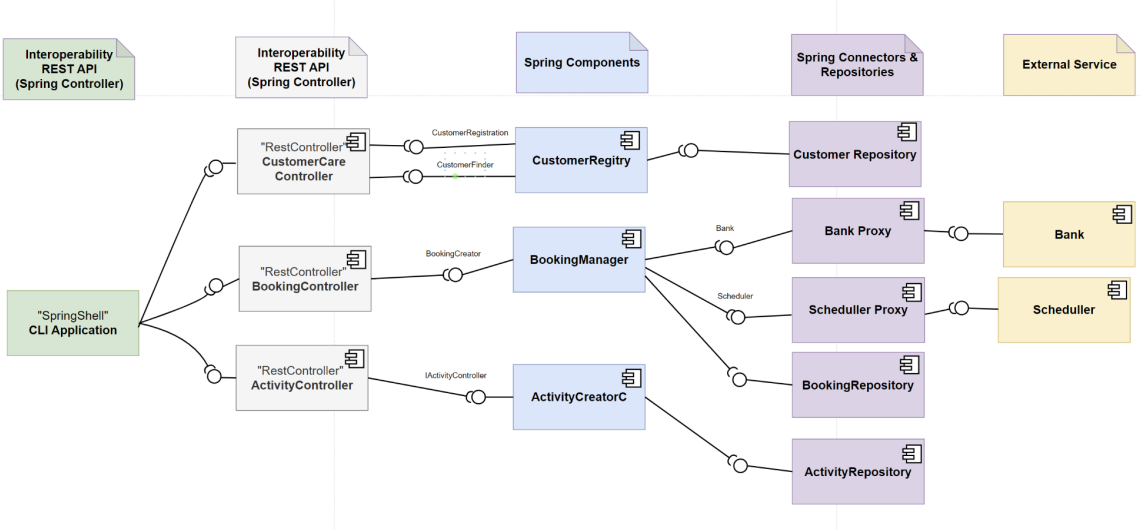
En plus des cas d'utilisation mentionnés précédemment, le système de carte multi-activités propose une flexibilité de paiement en permettant aux utilisateurs de régler leurs réservations et leurs achats d'avantages soit par carte bancaire, soit par utilisation de leurs points de fidélité cumulés. Cette intégration avec un système de paiement bancaire externe offre une option de paiement pratique et sécurisée pour les utilisateurs, leur permettant d'utiliser différents modes de paiement selon leurs préférences et leur situation financière.



3. Diagramme de classe



4. Diagramme des composants



5. Choix d'implémentation de la persistance

Dans notre système de carte multi-activités, PostgreSQL est choisi comme système de gestion de base de données, intégré dans un environnement Docker pour la gestion facilitée et la portabilité. À l'aide de **JPA** et **Spring Data JPA**, le schéma de la base de données est configuré avec des entités Java représentant les réservations, les clients et les activités.

Les annotations **JPA** telles que **@Entity**, **@Table**, et **@Column** sont utilisées pour définir la structure de la base de données, avec des relations établies via **@ManyToOne** et **@OneToMany.**

Dans la classe Booking (Réservation), nous avons utilisé les annotations **@ManyToOne** pour établir une relation entre chaque réservation et le client qui l'a effectuée, ainsi qu'avec l'activité réservée.

Ces annotations spécifient les clés étrangères **customer_id** et **activity_id** dans la table de réservation pour lier les réservations à leurs clients et activités respectifs.

De plus, dans la classe Customer (Client), l'annotation **@OneToMany** a été utilisée pour indiquer qu'un client peut avoir plusieurs réservations, avec la propriété bookings représentant cette relation.

En utilisant **mappedBy = "customer"**, nous spécifions que cette relation est gérée par la propriété customer dans la classe Booking.

Pour la classe **Activity (Activité)**, bien que les détails spécifiques ne soient pas fournis, nous supposons qu'elle établit également des relations avec d'autres entités telles que des réservations ou des prix associés à une activité. Nous avons utilisé l'option de cascading dans notre configuration d'entités de persistance pour automatiser les opérations de persistance associées aux relations entre les entités.

En utilisant **CascadeType.ALL** dans la relation entre **Customer (Client)** et **Booking (Réservation)**, les opérations de persistance effectuées sur un client sont également appliquées à toutes les réservations associées à ce client.

Cela simplifie la gestion des relations et assure la cohérence des données, en évitant la nécessité de gérer manuellement les opérations de persistance pour chaque entité associée

6. Forces et faiblesses de l'architecture

Forces

La principale force de notre architecture, réside dans son adaptabilité et dans sa capacité à ajouter par-dessus de nouvelles fonctionnalités. En effet, le petit périmètre que nous avons choisi de traiter, à savoir, le fait de pouvoir créer une activité, la payer et recevoir par la suite les points de fidélité, nous permet de faciliter l'implémentation future des fonctionnalités suivantes. Notre architecture a aussi un très faible couplage. L'utilisation des composants avec Spring permet de gérer les dépendances de manière très efficace. Cela signifie que chaque composant de notre système peut être

développé, testé et déployé de manière indépendante, ce qui réduit les risques associés aux futures modifications.

Faiblesses

Concernant les points négatifs, notre architecture présente un certain nombre de faiblesses. Tout d'abord, nous n'avons pas réalisé toutes les fonctionnalités requises pour le projet. Nous avons cependant un MVP fonctionnel, qui a de la valeur puisqu'il utilise la gestion des points de fidélité et de paiement des activités. Notre architecture contient aussi plusieurs incohérences au niveau du paiement par exemple. L'utilisateur ne peut pas voir la quantité d'argent qu'il a sur son compte.
Malheureusement, un grand nombre de nos tests tournaient au sein d'une certaine branche mais pas avec tous les autres, ce qui a posé problème au niveau de la fusion.

7. Axe d'améliorations

Dans ce projet, nous avons deux types de réservations qui n'ont pas encore été implémentées : la réservation TimeSlot et la réservation par forfait. Pour les mettre en place, nous aurions créé deux classes étendant Booking et mis en place un système de dates. La réservation par TimeSlot nécessite simplement l'ajout de deux dates. Ensuite, nous aurions implémenté la réservation par forfait, une combinaison entre la réservation par nombre d'entrées et celle par TimeSlot (ajout de deux dates et d'un nombre précis d'utilisations).

Après avoir implémenté ces fonctionnalités, Booking aurait été déclarée comme abstract, nous permettant ainsi de travailler sur l'affichage des réservations dans la CLI, en nous assurant que chaque réservation affiche correctement son ToString, et sur la création des commandes en ajoutant les nouveaux paramètres nécessaires. Cette approche aurait également permis de démontrer l'efficacité du scheduler. Nous aurions pu illustrer cela dans la démo en montrant une réservation avec une date trop ancienne qui aurait été rejetée par notre scheduler.

Concernant les Extras, nous avons commencé l'implémentation, mais elle n'est pas encore fonctionnelle. Il reste des bogues à déboguer et des détails à régler pour que tout fonctionne correctement.

En ce qui concerne les Statistiques, il aurait fallu créer un contrôleur utilisant les dépôts de chacune des classes pour recueillir les données voulues, puis les afficher à l'aide de commandes. Par exemple, faire une recherche par CustomerName dans le dépôt de Bookings afin d'obtenir toutes les recettes d'un client.

Pour finir, il faudrait modifier nos routes car elles ne respectent pas totalement la méthode REST. Il faudrait supprimer les /create et faire en sorte qu'un appel à bookings en

POST effectue la création, tandis qu'un appel à get donne toutes les listes au lieu d'une route /lists, etc. De plus, il serait judicieux d'ajouter une classe CatalogCommand (avec son contrôleur) pour gérer les affichages de toutes les entités (Extra, Bookings, Activity).

8. Répartition des points

Nous sommes tout d'abord fiers du travail que nous avons accompli durant ce projet. Nous avons beaucoup appris que ce soit en ISA et en DevOps. Bien évidemment, certains avaient plus de facilités sur certains domaines que d'autres. Néanmoins, chacun a participé à sa manière dans les différentes tâches du projet.

Nom & Prénom	Nombre de points
Rahmani Aubin	95
Merdaci Elias	120
Baratli Wassim	95
Ferchichi Hadil	90