MEP 4 - Rapport

Présenté à :
Antoine Lefrancois

Dans le cadre du cours
Architecture logicielle (GLO-4003)

Présenté par l'équipe #4

Saad Rajeb 111 227 646
Reda Choukry 111-241-048
Maxime Payant 537 010 093
Thomas Tricaud 537 010 104
Mohamed Choukri Soukehal 111 183 890
Etienne Lachance-Perreault 111-178-219
Anes Daba 111-255-296
Jean-Marc Sohouli 111 182 569

Vendredi 16 décembre 2022

1. Récits

Aventure #1

- Récit 1 complété à 100%
- Récit 2 complété à 100%
- Récit 3 complété à 100%
- Récit 4 complété à 100%
- Récit 5 complété à 100%
- Récit 6 complété à 100%
- Récit 7 complété à 100%
- Récit 8 complété à 100%
- Réci obligatoire complété à 100%

Aventure #2

- Récit 1 complété à 100%

Aventure #3

- Récit 1 complété à 100%
- Récit 2 complété à 100%

2. Patrons

a. Factory

Le patron Factory est utilisé dans notre architecture afin de créer les objets du domaine et de l'api. Nous utilisons deux nomenclatures différentes pour nommer nos Factory. [*]Factory pour les factory créant des objets du domaine à partir des objets de l'api et [*]Assembler pour les factory créant des objets de l'api à partir des objets du domaine.

L'utilisation de Factory nous permet de respecter deux des principes SOLID. Le SRP car la création des objets est la seule responsabilité des Factory et aucune autre instance n'a cette responsabilité. Ainsi que l'OCP car il est possible d'ajouter une nouvelle fonction de création d'objet dans une Factory ou de créer une nouvelle Factory afin de correspondre à de nouveaux besoins utilisateur sans impacter le code déjà présent.

b. Repository

Le patron Repository est utilisé dans notre architecture afin de stocker et d'accéder aux différents objets du domaine. Dans notre projet, on utilise le repository pour stocker plusieurs têtes d'agrégats comme les utilisateurs, les abonnements, les voyages et les stations. Les Repository nous permettent de protéger la gestion des données lors de changement de technologie.

L'utilisation de Repository nous permet de respecter un principe SOLID : Le SRP car le stockage des objets est la seule responsabilité des Repository et aucune autre instance n'a cette responsabilité.

c. Strategy

Le patron Strategy est utilisé dans notre architecture afin de gérer efficacement les différents systèmes de paiement des passes.

L'utilisation de Strategy nous permet d'isoler la logique de paiement de l'implémentation des différents systèmes et de permuter la logique en fonction des possibilités de l'utilisateur. Le principe SOLID de l'OCP est respecté car Strategy nous permet d'ajouter ou enlever un système de paiement sans impacter le contexte.

d. Service Locator

L'objectif du patron Service Locator est de renvoyer les instances de service à la demande. Il a pour but de résoudre la problématique de chercher une instance d'un objet précis, et l'injection de dépendance.
Dans notre projet nous l'avons utilisé au démarrage de notre application pour éviter d'instancier les différentes « Memory » et respecter le SRP. On l'utilise en tant que "Repository Locator" techniquement mais la logique du patron reste identique.

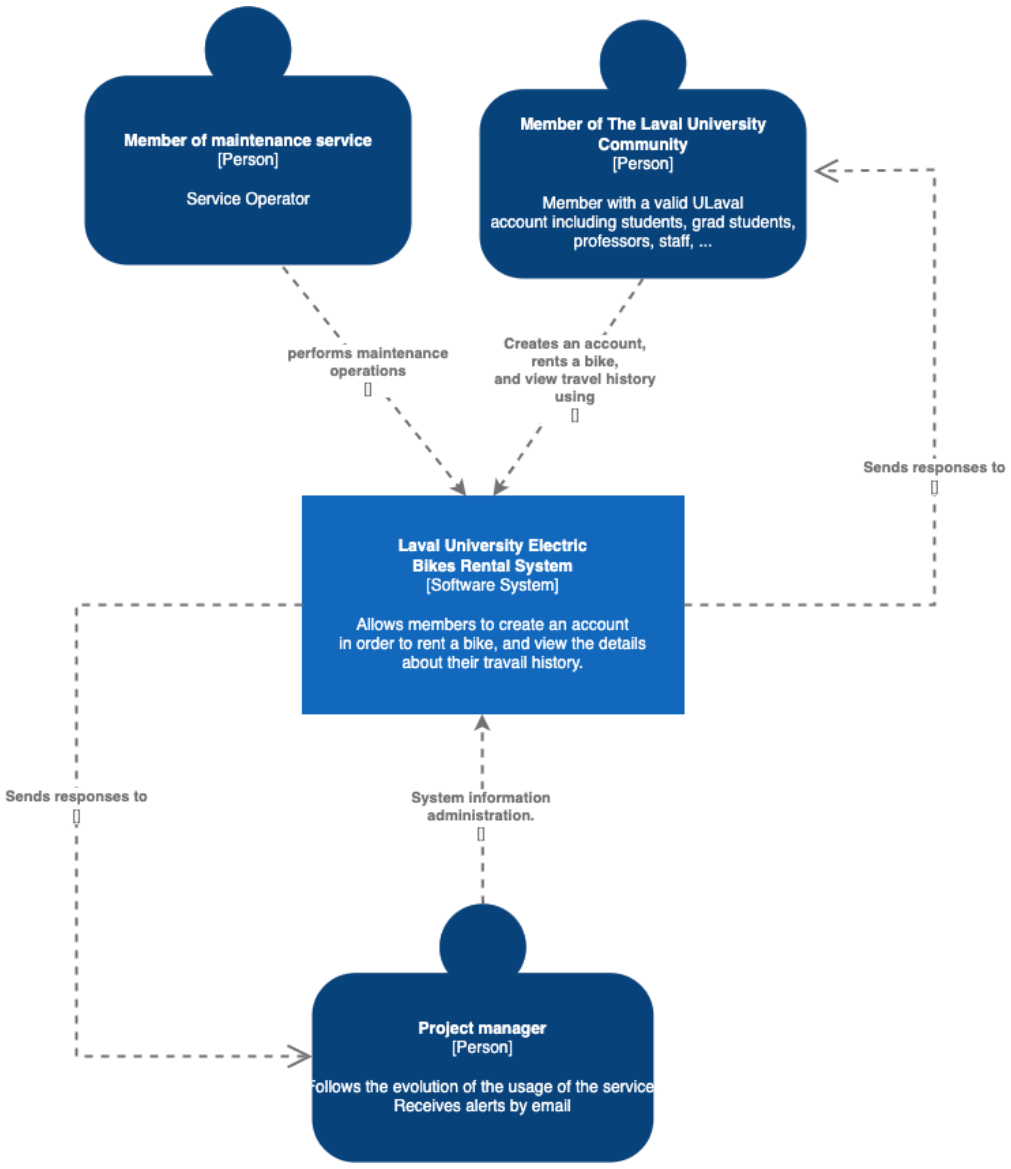
e. Chaîne de validation

Le patron de conception Chaîne de Validation permet de faire circuler des informations à travers différentes nodes validant ou non ces informations.

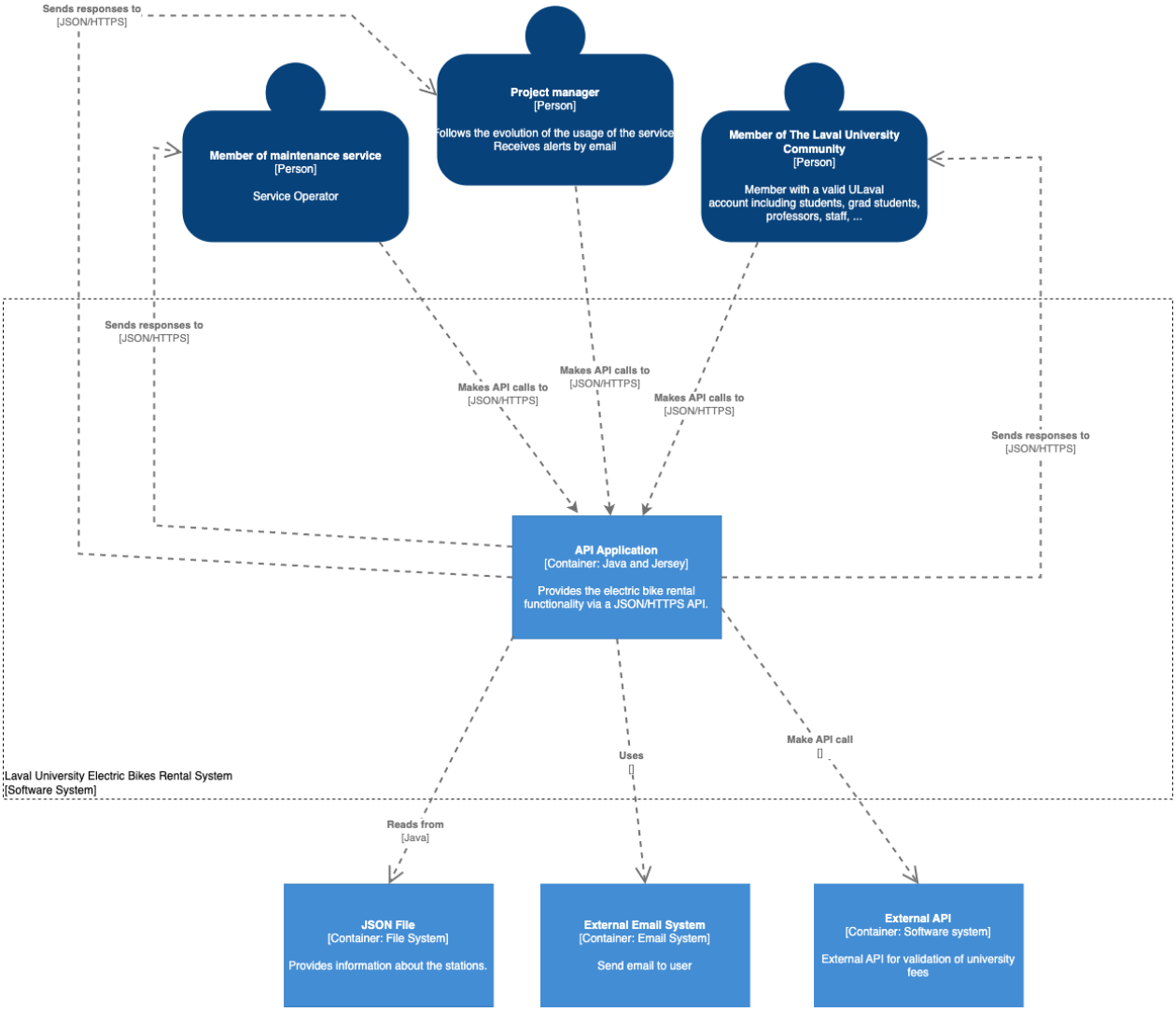
Dans notre projet, nous l'utilisons afin de valider les différents DTO reçus par les call API lors de la souscription à une passe, de l'inscription, la connexion et la déconnexion. Les nœuds étant réutilisables, peu importe les DTO, les différentes validations ne possèdent pas de code en commun. Chaque nœud possède sa propre logique de validation et est uniquement chargé de valider un type de valeur.

3. Diagramme C4

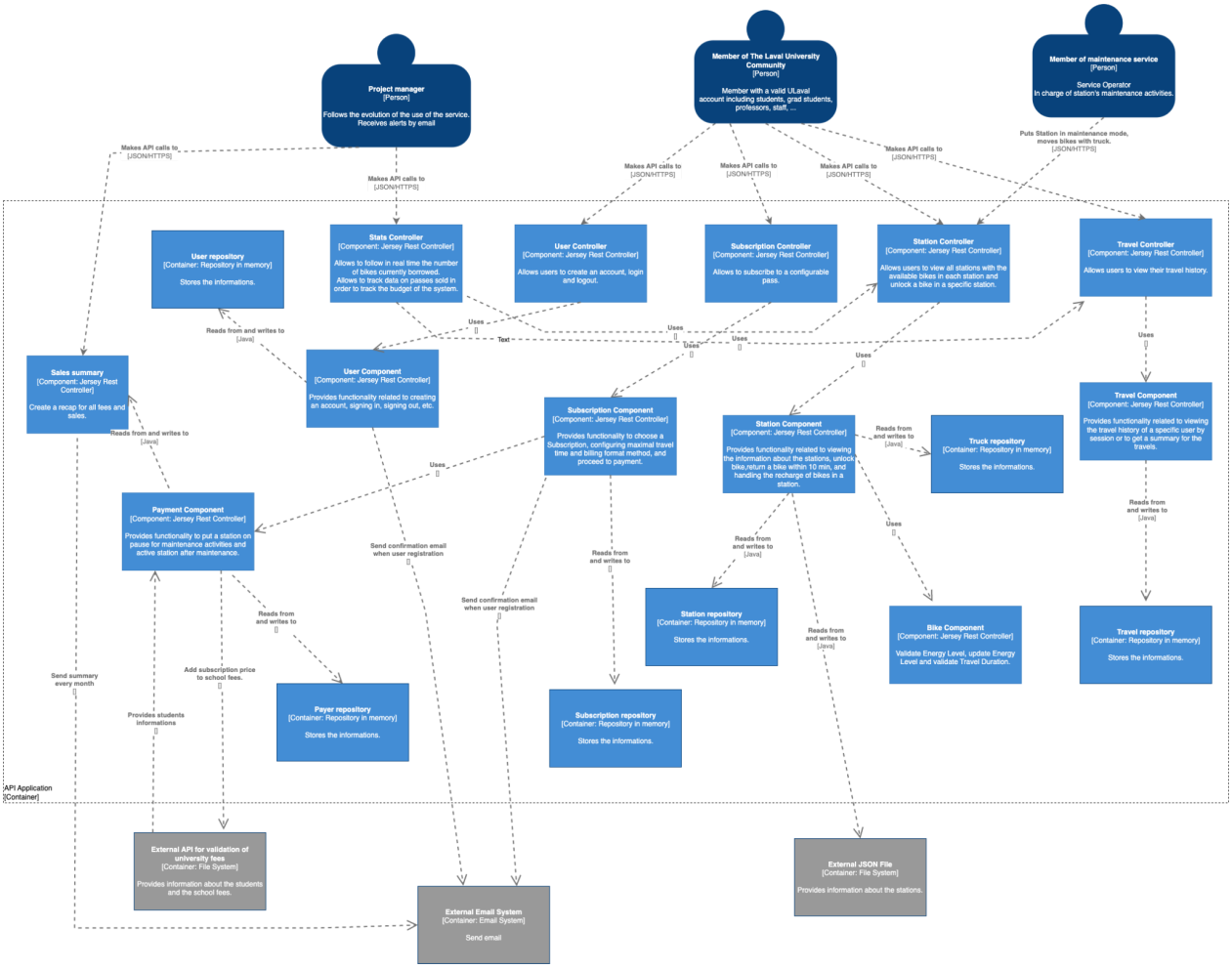
- **Context**



- **Container**


Sends responses to
[JSON/HTTPS]
Project manager
[Person]
Follows the evolution of the usage of the service
Receives alerts by email
Member of maintenance service
[Person]
Service Operator
Member of The Laval University Community
[Person]
Member with a valid ULaval account including students, grad students, professors, staff, ...
Sends responses to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Sends responses to
[JSON/HTTPS]
API Application
[Container: Java and Jersey]
Provides the electric bike rental functionality via a JSON/HTTPS API.
Laval University Electric Bikes Rental System
[Software System]
Reads from
[Java]
Uses
[]
Make API call
[]
JSON File
[Container: File System]
Provides information about the stations.
External Email System
[Container: Email System]
Send email to user
External API
[Container: Software system]
External API for validation of university fees

- **Component**


Project manager
[Person]
Follows the evolution of the use of the service. Receives alerts by email
Member of The Laval University Community
[Person]
Member with a valid ULaval account including students, grad students, professors, staff, ...
Member of maintenance service
[Person]
Service Operator
In charge of station's maintenance activities.
Puts Station in maintenance mode, moves bikes with truck.
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
User repository
[Container: Repository in memory]
Stores the informations.
Stats Controller
[Component: Jersey Rest Controller]
Allows to follow in real time the number of bikes currently borrowed. Allows to track data on passes sold in order to track the budget of the system.
User Controller
[Component: Jersey Rest Controller]
Allows users to create an account, login and logout.
Subscription Controller
[Component: Jersey Rest Controller]
Allows to subscribe to a configurable pass.
Station Controller
[Component: Jersey Rest Controller]
Allows users to view all stations with the available bikes in each station and unlock a bike in a specific station.
Travel Controller
[Component: Jersey Rest Controller]
Allows users to view their travel history.
Reads from and writes to
[Java]
Uses
[]
Text
Sales summary
[Component: Jersey Rest Controller]
Create a recap for all fees and sales.
User Component
[Component: Jersey Rest Controller]
Provides functionality related to creating an account, signing in, signing out, etc.
Subscription Component
[Component: Jersey Rest Controller]
Provides functionality to choose a Subscription, configuring maximal travel time and billing format method, and proceed to payment.
Station Component
[Component: Jersey Rest Controller]
Provides functionality related to viewing the information about the stations, unlock bike,return a bike within 10 min, and handling the recharge of bikes in a station.
Truck repository
[Container: Repository in memory]
Stores the informations.
Travel Component
[Component: Jersey Rest Controller]
Provides functionality related to viewing the travel history of a specific user by session or to get a summary for the travels.
Payment Component
[Component: Jersey Rest Controller]
Provides functionality to put a station on pause for maintenance activities and active station after maintenance.
Send confirmation email when user registration
[]
Reads from and writes to
[]
Station repository
[Container: Repository in memory]
Stores the informations.
Bike Component
[Component: Jersey Rest Controller]
Validate Energy Level, update Energy Level and validate Travel Duration.
Travel repository
[Container: Repository in memory]
Stores the informations.
Send summary every month
[]
Add subscription price to school fees.
[]
Provides students informations
[]
Payer repository
[Container: Repository in memory]
Stores the informations.
Subscription repository
[Container: Repository in memory]
Stores the informations.
API Application
[Container]
External API for validation of university fees
[Container: File System]
Provides information about the students and the school fees.
External Email System
[Container: Email System]
Send email
External JSON File
[Container: File System]
Provides information about the stations.

4. Burndown chart