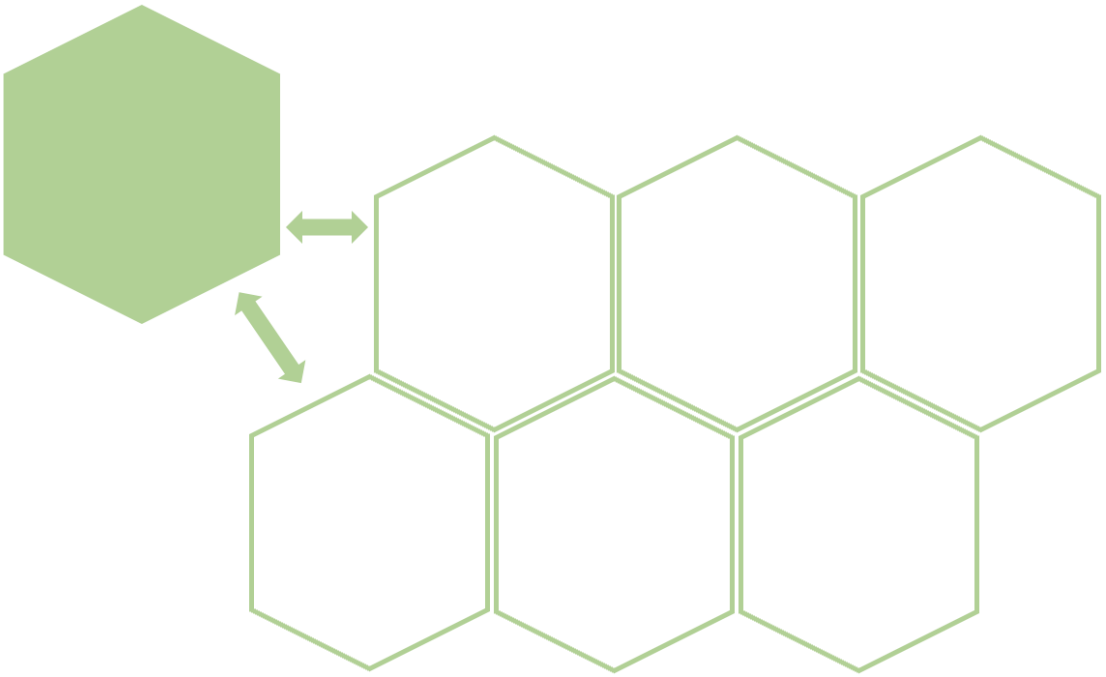
Framework JEE

Projet

Kévin LLOPIS

(kevin.llopis@carbon-it.com)



1. Cahier des charges

1.1. Contexte

Jusqu'à maintenant, l'entreprise ESGI-Rent avait pour activité la location de biens immobiliers, mais elle a désormais pour projet d'élargir ses activités et de s'ouvrir notamment à la location de voitures.

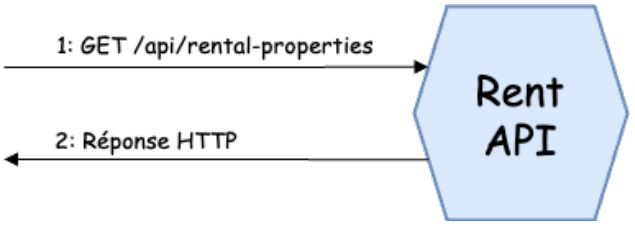
De ce fait, ESGI-Rent confie à votre équipe la mission de faire évoluer en conséquence son application web. Les objectifs sont les suivants :

- Réaliser la refonte de l'API REST existante pour la location de biens immobiliers
- Développer l'API REST correspondante à la location de voitures.
- Persister les biens immobiliers et les voitures en location dans une base de données

La suite du cahier des charges décrit l'architecture et les composants à développer.
Vous devrez respecter les contraintes décrites dans ce document en vue de ne pas être pénalisé durant l'évaluation.

1.2. Architecture du projet

Dans les TP précédents, l'architecture comprenait une seule application pour gérer les activités de location de biens immobiliers.



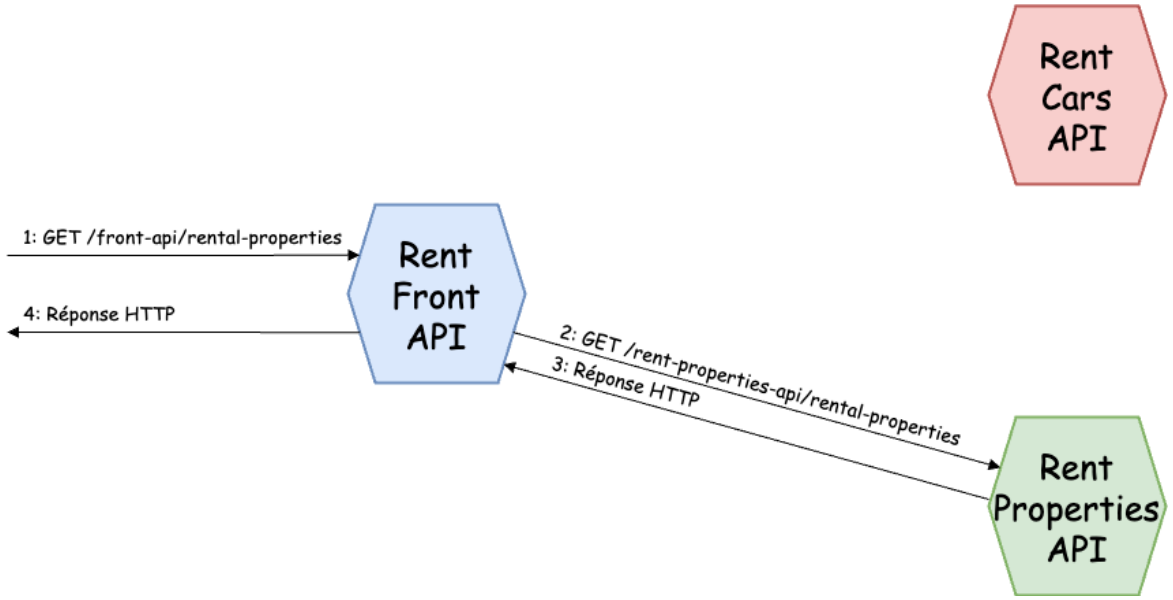
Dans ce projet, l'architecture comprendra 3 applications (micro-services) :

- **Rent Front API**
  Ce micro-service a pour objectif d'exposer une API REST à destination d'un client côté front et d'orchestrer les échanges avec les autres applications (Rent Properties API et Rent Cars API). L'API REST devra exposer des endpoints à la fois pour la location de biens immobiliers et de voitures. Il est important de noter que les requêtes du client devront être envoyées directement à Rent Front API et non pas aux autres applications ci-dessous.

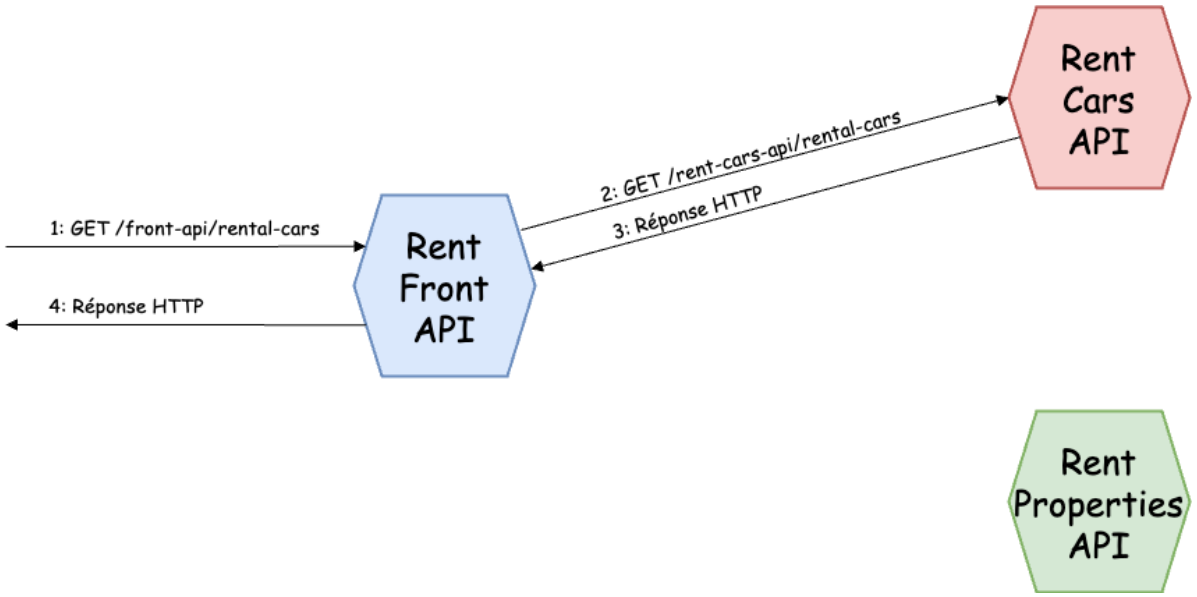
- **Rent Properties API**
  Il s'agit dans cette application de centraliser la gestion des biens immobiliers et d'exposer des endpoints REST pour leur consultation, leur création, leur mise à jour et leur suppression.

- **Rent Cars API**
  La nouvelle activité de location de voitures sera de la responsabilité de cette application en vue de réaliser les mêmes opérations que pour Rent Properties API mais appliquées à la location de voitures.

Chaque requête envoyée à Rent Front API sera relayée à l'une des deux autres applications. Par exemple, si l'endpoint **GET /front-api/rental-properties** est appelé (dédié à la location des biens immobiliers) dans Rent Front API, l'application interrogera Rent Properties API en appelant l'endpoint **GET /rent-properties-api/rental-properties**.



D'autre part, si l'endpoint **GET /front-api/rental-cars** est appelé (dédié à la location des voitures) dans Rent Front API, l'application interrogera Rent Cars API en appelant l'endpoint **GET /rent-cars-api/rental-cars**.



Remarque : Pour interroger une API REST depuis une autre application côté back, vous pouvez vous appuyer sur HttpClient.

1.3. Composants à développer

1.3.1. Rent Front API

1.3.1.1. Règles pour développer l'API REST

Vous devrez développer les endpoints de l'API REST en respectant :

- Les principes REST évoqués en cours (à l'exception d'HATEOAS) :
  - Stateless
  - Idempotent

- L'usage systématique de DTO pour les corps ("body") de requêtes/réponses aux endpoints.

- Les noms des champs décrits dans la documentation de l'API REST sont à respecter scrupuleusement. En revanche, le jeu de données pourra être adapté.

1.3.1.2. Documentation de l'API REST

- **GET /front-api/rental-properties**
  - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**GET /rent-properties-api/rental-properties**)
  - Statut de la réponse HTTP en cas de succès : 200
  - Exemple de corps de **réponse HTTP** à renvoyer :

```
[
  {
    "address":"77 Rue des roses",
    "area":37.48,
    "description":"Appartement spacieux avec vue sur l'ESGI",
    "propertyType":"FLAT",
    "rentAmount":750.9,
    "securityDepositAmount":1200.9,
    "town":"Paris"
  },
  {
    "address":"12 rue de la Pyramide",
    "area":62.5,
    "description":"Maison à louer dans banlieue calme et proche du RER",
    "propertyType":"HOUSE",
    "rentAmount":1050.9,
    "securityDepositAmount":1400.9,
    "town":"Champs-sur-Marne"
  }
]
```

- **GET /front-api/rental-properties/{id}**
  - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**GET /rent-properties-api/rental-properties/{id}**)
  - Statut de la réponse HTTP en cas de succès : 200
  - Statut de la réponse HTTP en cas de ressource introuvable : 404
  - Exemple de corps de **réponse HTTP** à renvoyer :

```
{
  "address":"12 rue de la Pyramide",
  "area":62.5,
  "description":"Maison à louer dans banlieue calme et proche du RER",
  "propertyType":"HOUSE",
  "rentAmount":1050.9,
  "securityDepositAmount":1400.9,
  "town":"Champs-sur-Marne"
}
```

- **POST /front-api/rental-properties**
  - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**POST /rent-properties-api/rental-properties**)
  - Statut de la réponse HTTP en cas de succès : 201
  - Statut de la réponse HTTP en cas de requête invalide : 400
    - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur jakarta.validation).
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
  - Exemple de corps de **requête HTTP** attendu :

```
{
    "description":"Appartement bien situé près du métro et des commerces",
    "town": "Neuilly-sur-Seine",
    "address": "90 rue de la Victoire",
    "propertyType": "FLAT",
    "rentAmount": 1040.9,
    "securityDepositAmount": 1250.9,
    "area": 50.69,
    "numberOfBedrooms": 3,
    "floorNumber": 2,
    "numberOfFloors": 5,
    "constructionYear": 1989,
    "energyClassification": "B",
    "hasElevator": true,
    "hasIntercom": true,
    "hasBalcony": true,
    "hasParkingSpace": true
}
```

- **PUT /front-api/rental-properties/{id}**
  - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**PUT /rent-properties-api/rental-properties/{id}**)
  - Statut de la réponse HTTP en cas de succès : 200
  - Statut de la réponse HTTP en cas de requête invalide : 400
    - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur jakarta.validation).
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
  - Exemple de corps de **requête HTTP** attendu :

```
{
    "description":"Appartement bien situé près du métro et des commerces",
    "town": "Neuilly-sur-Seine",
    "address": "90 rue de la Victoire",
    "propertyType": "FLAT",
    "rentAmount": 1040.9,
    "securityDepositAmount": 1250.9,
    "area": 50.69,
    "numberOfBedrooms": 3,
    "floorNumber": 2,
    "numberOfFloors": 5,
    "constructionYear": 1989,
    "energyClassification": "B",
    "hasElevator": true,
    "hasIntercom": true,
    "hasBalcony": true,
    "hasParkingSpace": true
}
```

- **PATCH /front-api/rental-properties/{id}**
  - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**PATCH /rent-properties-api/rental-properties/{id}**)
  - Statut de la réponse HTTP en cas de succès : 200
  - Statut de la réponse HTTP en cas de ressource introuvable : 404
  - Statut de la réponse HTTP en cas de requête invalide : 400
    - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur jakarta.validation).
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
  - Exemple de corps de **requête HTTP** attendu :

```
{
    "rentAmount": 1040.9
}
```

- **DELETE /front-api/rental-properties/{id}**
    - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**DELETE /rent-properties-api/rental-properties/{id}**)
    - Statut de la réponse HTTP en cas de succès : 204
    - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès

- **GET /front-api/rental-cars**
    - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**GET /rent-cars-api/rental-cars**)
    - Statut de la réponse HTTP en cas de succès : 200
    - Exemple de corps de **réponse HTTP** à renvoyer :

```
[
  {
    "brand": "BMW",
    "model": "Serie 1",
    "rentAmount": 790.9,
    "securityDepositAmount": 1550.9,
    "numberOfSeats": 5,
    "numberOfDoors": 4,
    "hasAirConditioning": true
  },
  {
    "brand": "Mercedes",
    "model": "Classe C Hybride",
    "rentAmount": 990.9,
    "securityDepositAmount": 2400.9,
    "numberOfSeats": 5,
    "numberOfDoors": 4,
    "hasAirConditioning": true
  }
]
```

- **GET /front-api/rental-cars/{id}**
    - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**GET /rent-cars-api/rental-cars/{id}**)
    - Statut de la réponse HTTP en cas de succès : 200
    - Statut de la réponse HTTP en cas de ressource introuvable : 404
    - Exemple de corps de **réponse HTTP** à renvoyer :

```
{
    "brand": "BMW",
    "model": "Serie 1",
    "rentAmount": 790.9,
    "securityDepositAmount": 1550.9,
    "numberOfSeats": 5,
    "numberOfDoors": 5,
    "hasAirConditioning": true
}
```

- **POST /front-api/rental-cars**
  - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**POST /rent-cars-api/rental-cars**)
  - Statut de la réponse HTTP en cas de succès : 201
  - Statut de la réponse HTTP en cas de requête invalide : 400
    - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur jakarta.validation).
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
  - Exemple de corps de **requête HTTP** attendu :

```
{
    "brand": "BMW",
    "model": "Serie 1",
    "rentAmount": 790.9,
    "securityDepositAmount": 1550.9,
    "numberOfSeats": 5,
    "numberOfDoors": 5,
    "hasAirConditioning": true
}
```

- **PUT /front-api/rental-cars/{id}**
  - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**PUT /rent-cars-api/rental-cars/{id}**)
  - Statut de la réponse HTTP en cas de succès : 200
  - Statut de la réponse HTTP en cas de requête invalide : 400
    - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur jakarta.validation).
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
  - Exemple de corps de **requête HTTP** attendu :

```
{
    "brand": "BMW",
    "model": "Serie 1",
    "rentAmount": 790.9,
    "securityDepositAmount": 1550.9,
    "numberOfSeats": 5,
    "numberOfDoors": 5,
    "hasAirConditioning": true
}
```

- **PATCH /front-api/rental-cars/{id}**
    - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**PATCH /rent-cars-api/rental-cars/{id}**)
    - Statut de la réponse HTTP en cas de succès : 200
    - Statut de la réponse HTTP en cas de ressource introuvable : 404
    - Statut de la réponse HTTP en cas de requête invalide : 400
        - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur jakarta.validation).
    - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
    - Exemple de corps de **requête HTTP** attendu :

```
{
    "rentAmount": 790.9
}
```

- **DELETE /front-api/rental-cars/{id}**
    - Objectif de l'endpoint : Relayer la requête du client vers l'endpoint de l'application dédiée (**DELETE /rent-cars-api/rental-cars/{id}**)
    - Statut de la réponse HTTP en cas de succès : 204
    - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès

1.3.1.3. Environnement technique à respecter

L'application doit être développée en respectant l'environnement technique suivant :

- JAX-RS (Java API for RESTful Web Services)
- CDI (Contexts and Dependency Injection)
- JUnit 5
- Mockito
- Spring et ses projets d'extension sont totalement interdits pour cette application

En cas de doute sur l'environnement technique, veuillez contacter l'enseignant, cela vous évitera d'être pénalisé dans les notes.

1.3.2. Rent Properties API

1.3.2.1. Règles pour développer l'API REST

Vous devrez développer les endpoints de l'API REST en respectant :

- Les principes REST évoqués en cours (à l'exception d'HATEOAS) :
  - Stateless
  - Idempotent

- L'usage systématique de DTO pour les corps ("body") de requêtes/réponses aux endpoints.

- Les noms des champs décrits dans la documentation de l'API REST sont à respecter scrupuleusement. En revanche, le jeu de données pourra être adapté.

1.3.2.2. Documentation de l'API REST

- **GET /rent-properties-api/rental-properties**
  - Objectif de l'endpoint : Interroger la base de données pour obtenir les enregistrements de la table **rental_property**.
  - Statut de la réponse HTTP en cas de succès : 200
  - Exemple de corps de **réponse HTTP** à renvoyer :


```
[
  {
    "address":"77 Rue des roses",
    "area":37.48,
    "description":"Appartement spacieux avec vue sur l'ESGI",
    "propertyType":"FLAT",
    "rentAmount":750.9,
    "securityDepositAmount":1200.9,
    "town":"Paris"
  },
  {
    "address":"12 rue de la Pyramide",
    "area":62.5,
    "description":"Maison à louer dans banlieue calme et proche du RER",
    "propertyType":"HOUSE",
    "rentAmount":1050.9,
    "securityDepositAmount":1400.9,
    "town":"Champs-sur-Marne"
  }
]
```

- **GET /rent-properties-api/rental-properties/{id}**
  - Objectif de l'endpoint : Interroger la base de données pour obtenir l'enregistrement correspondant de la table **rental_property**.
  - Statut de la réponse HTTP en cas de succès : 200
  - Statut de la réponse HTTP en cas de ressource introuvable : 404
  - Exemple de corps de **réponse HTTP** à renvoyer :

```
{
  "address":"12 rue de la Pyramide",
  "area":62.5,
  "description":"Maison à louer dans banlieue calme et proche du RER",
  "propertyType":"HOUSE",
  "rentAmount":1050.9,
  "securityDepositAmount":1400.9,
  "town":"Champs-sur-Marne"
}
```

- **POST /rent-properties-api/rental-properties**
  - Objectif de l'endpoint : Persister le nouveau bien immobilier dans la table **rental_property**.
  - Statut de la réponse HTTP en cas de succès : 201
  - Statut de la réponse HTTP en cas de requête invalide : 400
    - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur jakarta.validation).
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
  - Exemple de corps de **requête HTTP** attendu :

```
{
    "description":"Appartement bien situé près du métro et des commerces",
    "town": "Neuilly-sur-Seine",
    "address": "90 rue de la Victoire",
    "propertyType": "FLAT",
    "rentAmount": 1040.9,
    "securityDepositAmount": 1250.9,
    "area": 50.69,
    "numberOfBedrooms": 3,
    "floorNumber": 2,
    "numberOfFloors": 5,
    "constructionYear": 1989,
    "energyClassification": "B",
    "hasElevator": true,
    "hasIntercom": true,
    "hasBalcony": true,
    "hasParkingSpace": true
}
```

- **PUT /rent-properties-api/rental-properties/{id}**
  - Objectifs de l'endpoint :
    - Si l'enregistrement n'existait pas en base, le bien immobilier est persisté dans la table **rental_property**.
    - Si l'enregistrement existait en base, le bien immobilier est mis à jour dans la table **rental_property**.
  - Statut de la réponse HTTP en cas de succès : 200
  - Statut de la réponse HTTP en cas de requête invalide : 400
    - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur jakarta.validation).
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
  - Exemple de corps de **requête HTTP** attendu :

```
{
    "description":"Appartement bien situé près du métro et des commerces",
    "town": "Neuilly-sur-Seine",
    "address": "90 rue de la Victoire",
    "propertyType": "FLAT",
    "rentAmount": 1040.9,
    "securityDepositAmount": 1250.9,
    "area": 50.69,
    "numberOfBedrooms": 3,
    "floorNumber": 2,
    "numberOfFloors": 5,
    "constructionYear": 1989,
    "energyClassification": "B",
    "hasElevator": true,
    "hasIntercom": true,
    "hasBalcony": true,
    "hasParkingSpace": true
}
```

- **PATCH /rent-properties-api/rental-properties/{id}**
    - Objectif de l'endpoint : Le bien immobilier est mis à jour dans la table **rental_property**.
    - Statut de la réponse HTTP en cas de succès : 200
    - Statut de la réponse HTTP en cas de ressource introuvable : 404
    - Statut de la réponse HTTP en cas de requête invalide : 400
        - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur jakarta.validation).
    - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
    - Exemple de corps de **requête HTTP** attendu :

```
{
    "rentAmount": 1040.9
}
```

- **DELETE /rent-properties-api/rental-properties/{id}**
    - Objectif de l'endpoint : Le bien immobilier est supprimé dans la table **rental_property**.
    - Statut de la réponse HTTP en cas de succès : 204
    - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès

1.3.2.3. Environnement technique à respecter

L'application doit être développée en respectant l'environnement technique suivant :

- Spring
- Spring Boot
- Pour la couche de persistence des données, vous pouvez utiliser au choix :
    - spring-boot-starter-jdbc
    - spring-data-jdbc
    - spring-boot-starter-data-jpa
- Liquibase (facultatif)
- JUnit 5
- Mockito
- Les dépendances liées à Java EE / Jakarta EE (JAX-RS, CDI) sont totalement interdites pour cette application.

En cas de doute sur l'environnement technique, veuillez contacter l'enseignant, cela vous évitera d'être pénalisé dans les notes.

1.3.3. Rent Cars API

1.3.3.1. Règles pour développer l'API REST

Vous devrez développer les endpoints de l'API REST en respectant :

- Les principes REST évoqués en cours (à l'exception d'HATEOAS) :
  - Stateless
  - Idempotent

- L'usage systématique de DTO pour les corps ("body") de requêtes/réponses aux endpoints.

- Les noms des champs décrits dans la documentation de l'API REST sont à respecter scrupuleusement. En revanche, le jeu de données pourra être adapté.

1.3.3.2. Documentation de l'API REST

- **GET /rent-cars-api/rental-cars**
  - Objectif de l'endpoint : Interroger la base de données pour obtenir les enregistrements de la table **rental_car**.
  - Statut de la réponse HTTP en cas de succès : 200
  - Exemple de corps de **réponse HTTP** à renvoyer :

```
[
  {
    "brand": "BMW",
    "model": "Serie 1",
    "rentAmount": 790.9,
    "securityDepositAmount": 1550.9,
    "numberOfSeats": 5,
    "numberOfDoors": 4,
    "hasAirConditioning": true
  },
  {
    "brand": "Mercedes",
    "model": "Classe C Hybride",
    "rentAmount": 990.9,
    "securityDepositAmount": 2400.9,
    "numberOfSeats": 5,
    "numberOfDoors": 4,
    "hasAirConditioning": true
  }
]
```

- **GET /rent-cars-api/rental-cars/{id}**
    - Objectif de l'endpoint : Interroger la base de données pour obtenir l'enregistrement correspondant de la table **rental_car**.
    - Statut de la réponse HTTP en cas de succès : 200
    - Statut de la réponse HTTP en cas de ressource introuvable : 404
    - Exemple de corps de **réponse HTTP** à renvoyer :

```
{
    "brand": "BMW",
    "model": "Serie 1",
    "rentAmount": 790.9,
    "securityDepositAmount": 1550.9,
    "numberOfSeats": 5,
    "numberOfDoors": 5,
    "hasAirConditioning": true
}
```

- **POST /rent-cars-api/rental-cars**
    - Objectif de l'endpoint : Persister le nouvelle voiture dans la table **rental_car**.
    - Statut de la réponse HTTP en cas de succès : 201
    - Statut de la réponse HTTP en cas de requête invalide : 400
        - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur `jakarta.validation`).
    - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
    - Exemple de corps de **requête HTTP** attendu :

```
{
    "brand": "BMW",
    "model": "Serie 1",
    "rentAmount": 790.9,
    "securityDepositAmount": 1550.9,
    "numberOfSeats": 5,
    "numberOfDoors": 5,
    "hasAirConditioning": true
}
```

- **PUT /rent-cars-api/rental-cars/{id}**
  - Objectifs de l'endpoint :
    - Si l'enregistrement n'existait pas en base, la voiture est persistée dans la table **rental_car**.
    - Si l'enregistrement existait en base, la voiture est mise à jour dans la table **rental_car**.
  - Statut de la réponse HTTP en cas de succès : 200
  - Statut de la réponse HTTP en cas de requête invalide : 400
    - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur `jakarta.validation`).
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
  - Exemple de corps de **requête HTTP** attendu :

```
{
    "brand": "BMW",
    "model": "Serie 1",
    "rentAmount": 790.9,
    "securityDepositAmount": 1550.9,
    "numberOfSeats": 5,
    "numberOfDoors": 5,
    "hasAirConditioning": true
}
```

- **PATCH /rent-cars-api/rental-cars/{id}**
  - Objectif de l'endpoint : La voiture est mise à jour dans la table **rental_car**.
  - Statut de la réponse HTTP en cas de succès : 200
  - Statut de la réponse HTTP en cas de ressource introuvable : 404
  - Statut de la réponse HTTP en cas de requête invalide : 400
    - La requête est considérée comme invalide si les champs obligatoires ne sont pas renseignés (Vous pouvez vous appuyer sur `jakarta.validation`).
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès
  - Exemple de corps de **requête HTTP** attendu :

```
{
    "rentAmount": 790.9
}
```

- **DELETE /rent-cars-api/rental-cars/{id}**
  - Objectif de l'endpoint : La voiture est supprimée dans la table **rental_car**.
  - Statut de la réponse HTTP en cas de succès : 204
  - Aucun corps de réponse HTTP n'est à renvoyer en cas de succès

1.3.3.3. Environnement technique à respecter

L'application doit être développée en respectant l'environnement technique suivant :

- Spring
- Spring Boot
- Pour la couche de persistence des données, vous pouvez utiliser au choix :
    - spring-boot-starter-jdbc
    - spring-data-jdbc
    - spring-boot-starter-data-jpa
- Liquibase (facultatif)
- JUnit 5
- Mockito
- Les dépendances liées à Java EE / Jakarta EE (JAX-RS, CDI) sont totalement interdites pour cette application.

En cas de doute sur l'environnement technique, veuillez contacter l'enseignant, cela vous évitera d'être pénalisé dans les notes.

1.4. Base de données

Une base de données est requise pour stocker les données liées aux locations de biens immobiliers et de voitures.

1.4.1. Schéma

1.4.1.1. Table rental_property

Nom	Type	NOT NULL	Commentaire
id (Clé primaire)	INT	Oui	
description	VARCHAR(200)	Oui	
town	VARCHAR(100)	Oui	
address	VARCHAR(200)	Oui	
property_type_id (Clé étrangère)	INT	Oui	Référence la table “property_type”
rent_amount	DOUBLE	Oui	
security_deposit_amount	DOUBLE	Oui	
area	DOUBLE	Oui	
number_of_bedrooms	INT	Non	
floor_number	INT	Non	
number_of_floors	INT	Non	
construction_year	CHAR(4)	Non	
energy_classification_id (Clé étrangère)	INT	Non	Référence la table “energy_classification”
has_elevator	BIT(1)	Non	
has_intercom	BIT(1)	Non	
has_balcony	BIT(1)	Non	
has_parking_space	BIT(1)	Non	

1.4.1.2. Table property_type

Nom	Type	NOT NULL
id (Clé primaire)	INT	Oui
designation	VARCHAR(5)	Oui

1.4.1.3. Table energy_classification

Nom	Type	NOT NULL
id (Clé primaire)	INT	Oui
designation	CHAR(1)	Oui

1.4.1.4. Table rental_car

Nom	Type	NOT NULL
id (Clé primaire)	INT	Oui
brand	VARCHAR(100)	Oui
model	VARCHAR(100)	Oui
rent_amount	DOUBLE	Oui
security_deposit_amount	DOUBLE	Oui
number_of_seats	INT	Non
number_of_doors	INT	Non
has_air_conditioning	BIT(1)	Non

1.4.2. Environnement technique à respecter

Une base de données relationnelle est imposée dans ce projet.
Vous choisirez MySQL ou PostgreSQL.

Aucune autre base de données ne sera acceptée.

1.5. Tests

Dans toutes les applications, les tests unitaires devront être réalisés en appliquant les bonnes pratiques vues en cours. Veillez à ce que le coverage soit d'au moins 80% dans chacun des composants à développer.

1.6. Clean Code

En développant les différentes fonctionnalités, vous devrez respecter les bonnes pratiques de Clean Code et plus précisément :

- Un nommage des classes/méthodes/variables lisible et explicite
- Single Responsibility : Une et une seule responsabilité par classe

2. Starter

Aucune application "starter" ne sert de point de départ pour débuter le projet.
En effet, tout est à créer de votre côté.

Toutefois, vous trouverez des solutions à toutes les problématiques du projet dans :

- Les TP JEE
- Les TP Spring à venir du 22/05/2023 et 23/05/2023
- Les exemples de code fournit progressivement par l'enseignant pendant la durée du projet

3. Conseils pour réussir le projet

Ce projet comporte une charge de travail conséquente. De ce fait, il est primordial de prendre en compte les conseils suivants le plus tôt possible :

- Veiller à commencer le projet le plus tôt possible

  Au vu de la charge de travail et des différentes problématiques (base de données, échanges entre les composants, tests, clean code) pour arriver au bout du projet, il est fortement recommandé de débuter le projet dès le mois de Mai et sans attendre les prochains cours de la matière.

- Répartir la charge de travail équitablement entre tous les membres du projet

  - Dans un groupe de 3, la bonne stratégie est de confier le développement de chaque composant (micro-service) à un membre du groupe.

  - Dans un groupe de 2, un des composants devra être partagé entre les deux membres du groupe.

- Respecter les contraintes spécifiées dans le cahier des charges

- Appliquer les conseils vus en cours (et dans les corrections) pour réaliser les tests unitaires

- Poser des questions à l'enseignant sans modération

4. Modalités d'évaluation

4.1. Modalités du rendu

Vous aurez besoin de créer un repository GitHub/GitLab pour exposer le code du projet. Cela dit, veillez à ce que ce repository reste privé et ne soit pas visible des autres groupes.

Pour rendre le projet, il suffira de respecter les conditions suivantes :

- Déposez une archive de votre projet dans le dépôt MyGES prévu à cet effet
- Ajouter l'enseignant **kevin-llps** comme *collaborator (GitHub)/member (GitLab)* de votre repository avant le 21/06/2023.
- "Pusher" votre code dans le repository avant le 21/06/2023.

  Attention : Les commits poussés dans le repository à partir du 21/06/2023 ne seront pas pris en compte dans la notation pour des raisons d'équité entre les groupes.

- Ajouter un README qui indique :
  - La promotion
  - La liste des membres du groupe
  - Les difficultés rencontrées
  - La liste des fonctionnalités non implémentées (et la raison)

4.2. Déroulement des soutenances

Une soutenance de 18 minutes aura lieu le 21/06/2023 pour chaque groupe dont le déroulement sera le suivant :

- Une prise de parole de 10 minutes sera réservée au groupe, il s'agira de démontrer le fonctionnement de chaque fonctionnalité et passer en revue le code tout en expliquant les choix réalisés.
- En seconde partie de soutenance, l'enseignant posera des questions sur le code réalisé et fera des retours pour améliorer le code du projet.