

|  |
| --- |
| 1civil’IT |
| WATTSWAP |
| DOCUMENT DE LIVRAISON |

|  |
| --- |
| 12/03/2025 |

SOMMAIRE

[LA LIVRAISON 3](#_Toc193261343)

[GITHUB 3](#_Toc193261344)

[LES FICHIERS ANNEXES 3](#_Toc193261345)

[LE FRONT 3](#_Toc193261346)

[La maquette 3](#_Toc193261347)

[L’écran d’accueil 3](#_Toc193261348)

[Xxxx 3](#_Toc193261349)

[LE BACK 3](#_Toc193261350)

[L’ACCES A L’API 3](#_Toc193261351)

[L’AUTHENTIFICATION 3](#_Toc193261352)

[LE CRUD USER 3](#_Toc193261353)

[LE CRUD DES ADRESSES 4](#_Toc193261354)

[GEOLOCALISATION DE L’ADRESSE 5](#_Toc193261355)

[LE CRUD DES BORNES 5](#_Toc193261356)

[LA GEOLOCALISATION DES BORNES 6](#_Toc193261357)

[LE CRUD DES RESERVATIONS 6](#_Toc193261358)

[LE PAIEMENT DE LA RESERVATION 7](#_Toc193261359)

[SUIVI DE LA CONSOMMATION 7](#_Toc193261360)

[LES FONCTIONNALITES DU CAHIER DES CHARGES N’AYANT PU ETRE DEVELOPPEES 8](#_Toc193261361)

[LES TESTS UNITAIRES 8](#_Toc193261362)

[LES NOTIFICATIONS 8](#_Toc193261363)

[LA GEOLOCALISATION DE L’UTILISATEUR 8](#_Toc193261364)

[LE CRUD ET LE SUIVI DES CONNEXIONS 8](#_Toc193261365)

[LE PAIEMENT DE LA CONSOMMATION 8](#_Toc193261366)

[ASSISTANCE ET RECHERCHE DE GARAGISTE 8](#_Toc193261367)

[NOS IDEES COMPLEMENTAIRES 8](#_Toc193261368)

[AIDE A LA SAISIE DE LA BORNE (DUPLICATION DE BORNE) 8](#_Toc193261369)

[LES AVIS 9](#_Toc193261370)

[LE MCD 10](#_Toc193261371)

[LES ENTITYs 10](#_Toc193261372)

[BORNE 10](#_Toc193261373)

[AVAILABILITY 11](#_Toc193261374)

[ADDRESS 11](#_Toc193261375)

[USER 12](#_Toc193261376)

[HABITER 13](#_Toc193261377)

[RESERVATION 13](#_Toc193261378)

[CONSUMPTION 14](#_Toc193261379)

[ROLE 14](#_Toc193261380)

[DAY 15](#_Toc193261381)

[Payment 15](#_Toc193261382)

[CONNEXION 16](#_Toc193261383)

[GARAGE 16](#_Toc193261384)

[PAYS 16](#_Toc193261385)

[LES DTOS 17](#_Toc193261386)

[ADDRESSDTO 17](#_Toc193261387)

[AVAILABILITYDTO 17](#_Toc193261388)

[BORNEDTO 17](#_Toc193261389)

[CONSUMPTIONDTO 18](#_Toc193261390)

[CONSUMPTIONRESPONSEDTO 18](#_Toc193261391)

[FILTERDTO 18](#_Toc193261392)

[GEOBORNEDTO 18](#_Toc193261393)

[GEOFILTERDTO 19](#_Toc193261394)

[USERDTO 19](#_Toc193261395)

[RESERVATIONDTO 19](#_Toc193261396)

[RESERVATIONSUMMARYDTO 20](#_Toc193261397)

[PaymentRequestDTO 20](#_Toc193261398)

[PaymentResponseDTO 20](#_Toc193261399)

1. LA LIVRAISON

# LA LIVRAISON

## GITHUB

L’adresse du projet sur github est : **https://github.com/AlexandreNT1/wattswap.git**

## LES FICHIERS ANNEXES

En annexe sont livrés dans le répertoire du projet GITHUB : **https://github.com/AlexandreNT1/wattswap.git/annexe.**

* Le script de création de base POSTGRE
* Le document ci-présent.
* Le document des modifications apportées à l’environnement initial pour faire fonctionner l’application

1. Les fonctionnalités livrées

# LE FRONT

Les fonctionnalités livrées sont les suivantes :

## La maquette

La maquette blablabla….

## L’écran d’accueil

La maquette blablabla

## Xxxx

jjjj

# LE BACK

Les fonctionnalités livrées sont les suivantes :

## L’ACCES A L’API

L’API est accessible via le chemin : /wattswap/api/v1

## L’AUTHENTIFICATION

Mise en place de l’authentification.

## LE CRUD USER

Le CRUD user est accessible via l’URI : /users.

3 profils sont gérés par l’application :

**ADMIN** = administrateur. Il a accès à tout.

**USER** = utilisateur simple. Il a accès à tout sauf la gestion/suivi des bornes.

**UOWNER** = utilisateur propriétaire qui met à disposition sa borne. Il a accès à tout.

**Création d’un utilisateur** : URI/add.

Lorsque l’utilisateur crée son compte, l’application crée son adresse et géolocalise l’adresse (latitude et longitude) pour stocker l’information dans la table.

2 profils d’utilisateur sont disponibles.

L’utilisateur qui met à disposition sa borne de chargement dans l’application aura le profil UOWNER qui lui ouvre la fonctionnalité gestion des bornes.

L’utilisateur qui utilise notre application uniquement pour se géolocaliser aura le profil USER. Il n’aura donc pas accès à la fonctionnalité gestion des bornes.

Contraintes :

* l’email est unique dans la base.
* Le mot de passe contient à minima 8 caractères dont un chiffre, une majuscule, un caractère spécial

**Modification d’un utilisateur** : URI/upd/{email}

L’utilisateur peut modifier toutes ses informations. Il peut décider de partager une borne ou au contraire d’arrêter de la partager. Son profil sera adapté en fonction de son choix. Lorsqu’il choisit de ne plus partager sa borne, cette dernière est désactivée.

Il peut modifier son adresse. Si une ou plusieurs bornes sont liée(s) à l’ancienne adresse l’application modifie le lien adresse – borne pour attacher le(s) borne(s) sur la nouvelle adresse, puis l’ancienne adresse est supprimée si plus personne n’est lié à cette adresse.

La modification applique les mêmes contraintes que la création.

**Suppression d’un utilisateur** : URI/del/{email}

Lorsque l’utilisateur supprime son compte, ses données sont anonymisées. S’il partageait une borne, celle-ci est désactivée.

**Recherche d’un utilisateur** : URI/get/{email}

## LE CRUD DES ADRESSES

Le CRUD des adresses est accessible via l’URI : /users.

**Création d’une adresse** : URI/add. (création du user)

Lorsque l’utilisateur crée son compte il renseigne son adresse. Lorsqu’il modifie son profil, il peut être amené à modifier son adresse ou à ajouter une adresse secondaire. Dans tous les cas, l’adresse est créée dans la table adresse. Le front envoie un DTO UserDTO au back pour lui transmettre les informations utilisateur. Le lien utilisateur – adresse est chargé via l’entitée Habiter.

Contraintes :

* L’ adresse doit être complète (rue , code postal et ville)
* L’adresse doit être correctement orthographiée pour être retrouvée par l’API de géolocalisation
* L’adresse ne doit pas contenir de virgule après la numérotation dans la rue

**Création d’une ou des adresse(s) secondaire(s)** : URI/adrs/add/{email}.

Lorsque l’utilisateur ajoute une (ou plusieurs) adresse secondaire, le front envoie au back une liste d’AddressDTO et l’email de l’utilisateur à lier avec les adresses. Les adresses sont créées dans la table adresse ainsi que le lien avec l’utilisateur

Contraintes :

* Chaque adresse doit être complète (rue , code postal et ville)
* Chaque adresse doit être correctement orthographiée pour être retrouvée par l’API de géolocalisation
* L’adresse ne doit pas contenir de virgule après la numérotation dans la rue

**Changement d’adresse de l’utilisateur**: URI/update/{UUID du user} (modification de l’utilisateur)

Les informations devront transiter du front vers back avec une AddressDTO. L’application crée une liste de 2 AddressDTO. Le premier AddressDTO contient l’adresse actuelle de l’utilisateur et le second contient la nouvelle adresse. Une vérification dans l’entité Habiter est faite pour contrôler que l’utilisateur habite bien à l’ancienne adresse. Ce lien est supprimé. La nouvelle adresse est créée et l’utilisateur est rattaché à cette nouvelle adresse. Si une borne est liée à l’ancienne adresse, elle reste attachée à l’ancienne adresse et est désactivée. L’utilisateur doit créer une nouvelle borne pour la nouvelle adresse. Le changement n’est effectif que si une seule adresse est associée à l’utilisateur.

NB : Si le changement intervient au cours d’une modification de profil, le front envoie un userDTO au back qui prendra en charge le paramétrage de la méthode.

La modification applique les mêmes contraintes que la création.

**Suppression d’une adresse** :

La méthode est créée, mais n’est pas activée dans l’application. Les informations devront transiter du front vers back avec un AddressDTO.

Contraintes :

* Il n’y a pas de borne liée à cette adresse.
* Il n’y a pas d’utilisateur lié à cette adresse (table habiter)
* Il n’y a pas de garage lié à cette adresse

**Recherche de toutes les adresse de l’utilisateur** : URI2/get/{email}

Le front envoie l’email de l’utilisateur au back et l’application renvoie une liste de AddressDTO(s).

## GEOLOCALISATION DE L’ADRESSE

Lorsque l’utilisateur crée son compte on lui demande son adresse. Lorsque l’adresse est chargée dans la base, l’application envoie une requête à l’API https://api-adresse.data.gouv.fr/search/?q=xxx&postcode=yyyyy&city=zzzz pour récupérer les coordonnées de l’adresse.

‘xxx, yyyyy et zzz correspondent à la partie paramétrable. ‘xxx est l’adresse à chercher, yyyyy le code postal et zzz le nom de la ville. L’API renvoie une liste d’adresses dont le nom de la rue contient tout ou partie des éléments envoyés à l’API. La première de la liste est celle qui est la plus proche (normalement l’adresse recherchée). L’application récupère donc les coordonnées gps de la première réponse de l’API.

La géolocalisation de l’adresse est sollicitée à chaque création d’adresse (par le biais de la création d’utilisateur, la modification d’utilisateur et l’ajout d’adresses secondaires).

NB : Au cours des test l’API n’a pas fonctionné tout le temps. Voir s’il existe une contrainte de nombre de sollicitations par espace temps.

## LE CRUD DES BORNES

Le CRUD des bornes est accessible via l’URI : /bornes.

**Création d’une borne** : URI/add.

Lorsque l’utilisateur crée une borne, il doit d’abord choisir l’adresse à laquelle la rattacher. La création de la borne est complète lorsqu’il a choisi l’adresse puis décrit la borne et défini les périodes de validité avec leur prix respectif. Les informations sont transmises au back via le DTO BorneDTO.

Contraintes :

* La puissance de la borne est obligatoire.
* Les périodes de disponibilité et leur prix respectif sont obligatoires.

**Modification d’une borne** : URI/upd/{num}

L’utilisateur peut modifier la puissance et le type de la borne. Il peut indiquer si la borne est en panne. Il peut aussi modifier les périodes de disponibilité ainsi que leur prix. L’adresse n’est pas modifiable. Les informations sont transmises au back via le DTO BorneDTO. L’application renvoie un BorneDTO.

La modification d’une borne gère la modification des périodes de disponibilité en annule et remplace.

**Suppression d’une borne** : URI/del/{num}

Lorsque l’utilisateur supprime sa borne ou ne veut plus la partager, celle-ci est désactivée. La borne est identifiée par son numéro dans la adresse ({num}) et son adresse fournie au back par le DTO AddressDTO.

*Pour information : La suppression physique est prête pour le même URI mais pas activée.*

**Recherche d’une borne** : URI/get/{num}

La borne est identifiée par son numéro dans l’adresse ({num}) et son adresse fournie au back par le DTO AddressDTO. L’application renvoie un BorneDTO.

**Recherche de toutes les bornes de l’adresse** : URI/get/all

L’adresse fournie au back par le DTO AddressDTO. L’application renvoie une liste de BorneDTO(s).

## LA GEOLOCALISATION DES BORNES

La géolocalisation des bornes est accessible par deux URI possibles. Il faudra choisir celle qui conviendra le mieux au fonctionnement. On peut déclencher la géolocalisation par l’URI /geo ou par l’URI /bornes .

**Recherche de toutes les bornes à géolocaliser** : URI/get/geo La position de l’utilisateur et le nombre de kilomètres de distance souhaité sont transmis au back par le biais du GeoFilterDTO. L’application renvoie une liste de GeoBorneDTO(s).

L’application recherche d’abord les bornes publiques correspondant aux attributs du GeoFilterDTO en requêtant l’API

<https://odre.opendatasoft.com/api/explore/v2.1/catalog/datasets/bornes-irve/records?xxx&limit=-1>

xxx correspond à la partie paramétrée par l’application. Il existe une fonctionnalité (within\_distance) de l’application qui n’a besoin que de la position centrale et de la distance en kilomètres pour extraire toutes les bornes publiques autour de ce point.

L’API fonctionne bien mais il existe beaucoup de doublons dans le JSon généré. Nous avons donc fait un test sur l’existence des coordonnées dans les éléments déjà chargés pour éviter au maximum les doublons.

Les adresses liées aux bornes publiques n’ont pas toutes le format standard rue code postal ville. Dans ce cas, dans l’attribut Street on charge l’adresse complète trouvée dans le JSon puis nous faisons appel à une autre API pour récupérer la ville et le code postal à partir du code INSEE : <https://geo.api.gouv.fr/communes?code=xxx>. Xxx étant le code INSEE de la ville. L’API nous renvoie le nom de la ville et une liste de codes postaux liés à ce code INSEE. Nous avons pris le parti de prendre le premier de la liste.

## LE CRUD DES RESERVATIONS

Le CRUD des réservations est accessible via l’URI : /api/reservations.

**Création d’une réservation** : URI/reserve.

Lorsque l’utilisateur réserve une borne, il doit la sélectionner sur la carte et noter dans le calendrier la période à réserver.

Les informations sont transmises au back via le DTO ReservationDTO. Le back renvoie au front un ReservationSummaryDTO.

La méthode contrôle si la période réservée est toujours disponible, si c’est le cas elle enregistre la réservation, sinon elle renvoie une exception.

Contraintes :

* Tous les attributs du DTO sont renseignés.

**Suppression d’une réservation** : URI/{reservationId}

L’utilisateur peut annuler sa réservation. Elle est supprimée de la physiquement de la base

**Recherche de toutes les réservations d’un utilisateur** : URI/user/{userEmail}

L’utilisateur est identifié par son email. L’application renvoie une liste de ReservationSummaryDTO.

**Recherche de toutes les réservations d’une borne**: URI/ borne/{numBorne}

La borne est identifiée par son numéro dans l’adresse ({numBorne}) et son adresse fournie au back par le DTO AddressDTO. L’application renvoie une liste de ReservationSummaryDTO.

## LE PAIEMENT DE LA RESERVATION

Le paiement est déclenché lors de la création d’une réservation.

paiements via Stripe.

**1. Création de paiement (createPayment)**

* Récupère les informations de l'utilisateur et de la réservation.
* Crée un PaymentMethod Stripe avec un **token de test** (tok\_visa).
* Initialise un PaymentIntent avec confirmation immédiate.
* Enregistre les détails du paiement dans la base de données.

**2. Validation du paiement (validatePayment)**

* Récupère le PaymentIntent via Stripe.
* Met à jour le statut du paiement en fonction de la réponse de Stripe.

**3. Points d'accès API (PaymentController.java)**

* POST /api/payments/create : Crée un nouveau paiement.
* GET /api/payments/validate/{paymentIntentId} : Valide le statut d’un paiement.
* GET /api/payments/{paymentId} : Récupère les détails d'un paiement.

## SUIVI DE LA CONSOMMATION

Le suivi de la consommation est accessible par deux URI. Il faudra choisir celle qui conviendra le mieux au fonctionnement. On peut extraire le suivi de la consommation par l’URI1 /consos ou par l’URI2 /bornes .

**Suivi des consommations** : URI\*/get/conso

La borne à suivre (numéro et adresse), la période de consommation à extraire (début et fin) et le niveau d’aggrégat (h=heure, d=jour, w=semaine, m=mois) sont envoyés au back par un FIlterDTO. L’application renvoie une liste de ConsumptionResponseDTO.

Chaque élément de la liste contient la consommation aggrégée , le début et la fin de la période, le numéro de tranche de période. Seules les périodes avec consommation sont renvoyées. S’il y a des temps sans consommation entre deux , il faudra prévoir des tranches d’aggrégat à zéro.

**Création d’une consommation** : URI1/add

Le front envoie un ConsumptionDTO au back pour lui transmettre l’email de l’utilisateur qui a consommé, la borne qui a été utilisée (numéro et adresse), la période de consommation (début-fin) et quantité consommée (en kwH).

1. Les EVOLUTIONS

# LES FONCTIONNALITES DU CAHIER DES CHARGES N’AYANT PU ETRE DEVELOPPEES

## LES TESTS UNITAIRES

Les tests unitaires n’ont pas été développées.

## LES NOTIFICATIONS

Les notifications n’ont pas été développées.

Pour rappel, il était prévu une notification auprès du propriétaire pour chaque réservation de borne et une notification auprès de l’utilisateur indiquant si la réservation est confirmée ou annulée.

## LA GEOLOCALISATION DE L’UTILISATEUR

La géolocalisation n’a pas été développée, ni dans le front, ni dans le back.

## LE CRUD ET LE SUIVI DES CONNEXIONS

La gestion des connexions reste à développer

## LE PAIEMENT DE LA CONSOMMATION

Le paiement est aujourd’hui développé au niveau de la réservation. Lorsqu’il sera possible de charger la consommation réelle dans l’application, pour une meilleur équité , il faudra déplacer le paiement au niveau de la consommation.

## ASSISTANCE ET RECHERCHE DE GARAGISTE

Les entités (et les tables) sont créées. Aucune analyse ou développement n’a été commencé pour cette partie.

# NOS IDEES COMPLEMENTAIRES

## AIDE A LA SAISIE DE LA BORNE (DUPLICATION DE BORNE)

Pour le cas où un utilisateur propriétaire déménage, et pour le cas où un privé veut faire commerce de plusieurs bornes sur une même adresse, la duplication de borne pourrait faciliter la saisie de borne. L’utilisateur saisit et enregistre une borne.

S’il veut en créer une nouvelle, on lui propose de préremplir le formulaire avec ce qu’il a préalablement saisi.

S’il fait un changement d’adresse et qu’il est propriétaire d’une ou plusieurs bornes, on lui demande si toutes ses bornes changent d’adresse aussi si oui, on lui propose de les transférer sur la nouvelle adresse (sans saisie supplémentaire). Et dans ce cas , l’application crée autant de nouvelles bornes sur la nouvelle adresse qu’il y en avait sur l’ancienne (bonus : elles sont triées sur le numborne pour s’afficher dans le même ordre sur la nouvelle adresse).

## LES AVIS

Il pourrait être intéressant de laisser s’exprimer les utilisateurs de l’application.

* **Avis sur l’accessibilité et/ou fonctionnement d’une borne :**

L’utilisateur pourrait noter la borne et laisser un message visible par tous. Cet avis pourrait être envoyé au propriétaire sous forme de notification.

* **Avis sur l’application (vitesse/ pertinence des réponses/ facilité d’utilisation/ facilité de gestion des bornes) :**

L’utilisateur pourrait noter l’application et laisser un message visible par tous. Cet avis pourrait être envoyé dans une adresse mail dédiée afin d’améliorer le fonctionnement de l’application dans les versions ultérieures.

* **Contact :**

L’utilisateur pourrait contacter l’entreprise 1CIVIL’IT sur une adresse mail dédiée. Il faudra alors prévoir une personne pour répondre aux messages.

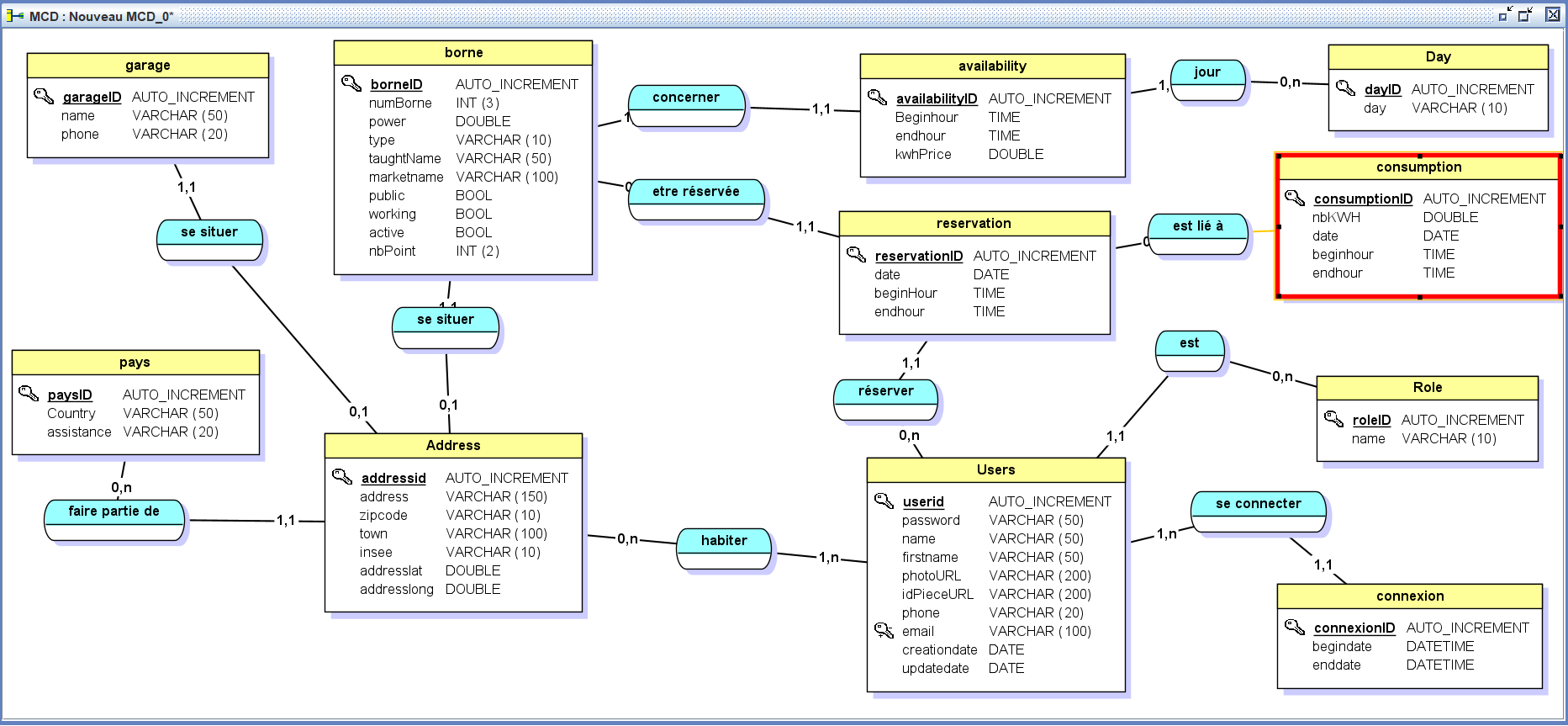
* **Questionnaire:**

Un questionnaire pourrait être accessible en permanence pour permettre à l’utilisateur de s’exprimer et dans la perspectives d’amélioration de l’application pour les prochaines versions. Les questions porteraient sur

* + la vitesse de réponse de l’application
  + la portabilité sur les appareils
  + la vitesse et pertinence des réponses
  + la facilité d’utilisation
  + les fonctionnalités que l’utilisateur aimerait trouver
  + les améliorations des fonctionnalités existantes

1. ANNEXE

# LE MCD



# LES ENTITYs

## BORNE

@Getter  
@Setter  
@Entity  
@NoArgsConstructor  
@Table(name = "borne")  
public class Borne {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "borneid", updatable = false, nullable = false)  
 private UUID id;  
  
 @NotNull  
 @Column(name = "numborne", nullable = false,columnDefinition = "INTEGER DEFAULT 1")  
 private Integer numborne = 1;  
  
 @NotNull  
 @Column(name = "power")  
 private Double power;  
  
 @Size(max = 10)  
 @Column(name = "type", length = 10)  
 private String type;  
  
 @Size(max = 100)  
 @Column(name = "taughtname", length = 100)  
 private String taughtname;  
  
 @Size(max = 100)  
 @Column(name = "marketname", length = 100)  
 private String marketname;  
  
 @NotNull  
 @Column(name = "public", nullable = false,columnDefinition = "BOOLEAN DEFAULT FALSE")  
 private Boolean publicField = false;  
  
 @NotNull  
 @Column(name = "working", nullable = false,columnDefinition = "BOOLEAN DEFAULT TRUE")  
 private Boolean working = true;  
  
 @NotNull  
 @Column(name = "active", nullable = false,columnDefinition = "BOOLEAN DEFAULT TRUE")  
 private Boolean active = true;  
  
 @NotNull  
 @Column(name = "nbpoint", nullable = false,columnDefinition = "INTEGER DEFAULT 1")  
 private Integer nbpoint = 1;  
  
 @NotNull  
 @ManyToOne(fetch = FetchType.*EAGER*, optional = false)  
 //@ManyToOne(fetch = FetchType.LAZY, optional = false)  
 @JoinColumn(name = "addressid", nullable = false)  
 private Address address;  
}

## AVAILABILITY

@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
@Entity  
@Table(name = "availability")  
public class Availability {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "availabilityid", updatable = false, nullable = false)  
 private UUID id;  
  
 @NotNull  
 @Column(name = "beginhour", nullable = false)  
 private LocalTime beginhour;  
  
 @NotNull  
 @Column(name = "endhour", nullable = false)  
 private LocalTime endhour;  
  
 @NotNull  
 @Column(name = "kwhprice", nullable = false)  
 private Double kwhprice;  
  
 @NotNull  
 @ManyToOne(fetch = FetchType.*EAGER*, optional = false)  
 //@ManyToOne(fetch = FetchType.LAZY, optional = false)  
 @JoinColumn(name = "borneid", nullable = false)  
 private Borne borne;  
  
 @NotNull  
 @ManyToOne(fetch = FetchType.*EAGER*, optional = false)  
 //@ManyToOne(fetch = FetchType.LAZY, optional = false)  
 @JoinColumn(name = "dayid", nullable = false)  
 private Day day;  
}

## ADDRESS

@NoArgsConstructor  
@Getter  
@Setter  
@Entity  
@Table(name = "address")  
public class Address {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "addressid", updatable = false, nullable = false)  
 private UUID id;  
  
 @Size(max = 150)  
 @NotNull  
 @Column(name = "address", nullable = false, length = 150)  
 private String address;  
  
 @Size(max = 10)  
 @NotNull  
 @Column(name = "zipcode", nullable = false, length = 10)  
 private String zipcode;  
  
 @Size(max = 100)  
 @NotNull  
 @Column(name = "town", nullable = false, length = 100)  
 private String town;  
  
 @Size(max = 10)  
 @Column(name = "insee", length = 10)  
 private String insee;  
  
 @Column(name = "addresslat")  
 private Double addresslat;  
  
 @Column(name = "addresslong")  
 private Double addresslong;  
  
 @ManyToOne(fetch = FetchType.*LAZY*, optional = true)  
 @JoinColumn(name = "paysid")  
 private Pays pay;  
}

## USER

package com.stage.wattswap.model.entity;  
  
import jakarta.persistence.\*;  
import jakarta.validation.constraints.NotNull;  
import jakarta.validation.constraints.Size;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
import org.hibernate.annotations.GenericGenerator;  
  
import java.time.LocalDate;  
import java.util.UUID;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "users")  
public class User {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "userid", updatable = false, nullable = false)  
 private UUID id;  
  
 @Size(max = 50)  
 @NotNull  
 @Column(name = "password", nullable = false, length = 50)  
 private String password;  
  
 @Size(max = 50)  
 @NotNull  
 @Column(name = "name", nullable = false, length = 50)  
 private String name;  
  
 @Size(max = 50)  
 @NotNull  
 @Column(name = "firstname", nullable = false, length = 50)  
 private String firstname;  
  
 @Size(max = 200)  
 @Column(name = "photourl", length = 200)  
 private String photourl;  
  
 @Size(max = 200)  
 @NotNull  
 @Column(name = "idpieceurl", nullable = false, length = 200)  
 private String idpieceurl;  
  
 @Size(max = 20)  
 @NotNull  
 @Column(name = "phone", nullable = false, length = 20)  
 private String phone;  
  
 @NotNull  
 @Column(name = "creationdate", nullable = false)  
 private LocalDate creationdate;  
  
 @NotNull  
 @Column(name = "updatedate", nullable = false)  
 private LocalDate updatedate;  
  
 @Size(max = 100)  
 @NotNull  
 @Column(name = "email", nullable = false, length = 100)  
 private String email;  
  
 @NotNull  
 @ManyToOne(fetch = FetchType.*LAZY*, optional = false)  
 @JoinColumn(name = "roleid", nullable = false)  
 private Role role;  
}

## HABITER

package com.stage.wattswap.model.entity;  
  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
import java.util.UUID;  
  
@Getter  
@Setter  
@Entity  
@NoArgsConstructor  
@Table(name = "habiter")  
public class Habiter {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*UUID*)  
 private UUID id;  
  
 // @MapsId("addressid")  
 @ManyToOne(fetch = FetchType.*EAGER*, optional = false)  
 @JoinColumn(name = "addressid", nullable = false)  
 private Address address;  
  
 // @MapsId("userid")  
 @ManyToOne(fetch = FetchType.*LAZY*, optional = false)  
 @JoinColumn(name = "userid", nullable = false)  
 private User user;  
}

## RESERVATION

@Getter  
@Setter  
@Entity  
@NoArgsConstructor  
@Table(name = "reservation")  
public class Reservation {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "reservationid", updatable = false, nullable = false)  
 private UUID id;  
  
 @NotNull  
 @Column(name = "date", nullable = false)  
 private LocalDate date;  
  
 @NotNull  
 @Column(name = "beginhour", nullable = false)  
 private LocalTime beginhour;  
  
 @NotNull  
 @Column(name = "endhour", nullable = false)  
 private LocalTime endhour;  
  
 @NotNull  
 @ManyToOne(fetch = FetchType.*EAGER*, optional = false)  
 @JoinColumn(name = "userid", nullable = false)  
 private User user;  
  
 @NotNull  
 @ManyToOne(fetch = FetchType.*EAGER*, optional = false)  
 @JoinColumn(name = "borneid", nullable = false)  
 private Borne borne;  
}

## CONSUMPTION

@Getter  
@Setter  
@Entity  
@NoArgsConstructor  
@Table(name = "consumption")  
public class Consumption {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "consumptionid", updatable = false, nullable = false)  
 private UUID id;  
  
 @NotNull  
 @Column(name = "nbkwh", nullable = false)  
 private Double nbkwh;  
  
 @NotNull  
 @Column(name = "date", nullable = false)  
 private LocalDate date;  
  
 @NotNull  
 @Column(name = "beginhour", nullable = false)  
 private LocalTime beginhour;  
  
 @NotNull  
 @Column(name = "endhour", nullable = false)  
 private LocalTime endhour;  
  
 @NotNull  
 @OneToOne(fetch = FetchType.*EAGER*, optional = false)  
 @JoinColumn(name = "reservationid", nullable = false)  
 private Reservation reservation;  
}

## ROLE

@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@Entity  
@Table(name = "role")  
public class Role {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "roleid", updatable = false, nullable = false)  
 private UUID id;  
  
 @Size(max = 10)  
 @NotNull  
 @Column(name = "name", nullable = false, length = 10)  
 private String name;  
}

## DAY

@Getter  
@Setter  
@Entity  
@NoArgsConstructor  
@AllArgsConstructor  
@Table(name = "day")  
public class Day {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "dayid", updatable = false, nullable = false)  
 private UUID id;  
  
 @Size(max = 10)  
 @NotNull  
 @Column(name = "day", nullable = false, length = 10)  
 private String day;  
}

## Payment

@Getter  
@Setter  
@NoArgsConstructor  
@ToString  
@Entity  
@Table(name = "payments")  
public class Payment {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "paymentid", updatable = false, nullable = false)  
 private UUID id;  
  
 @Column(nullable = false, unique = true)  
 private String paymentIntentId;  
  
 @Column(nullable = false)  
 private BigDecimal amount;  
  
 @Column(nullable = false)  
 private String paymentMethod;  
  
 @Column(nullable = false)  
 private String currency;  
  
  
 @Enumerated(EnumType.*STRING*)  
 @Column(nullable = false)  
 private PaymentStatus status;  
  
 @ManyToOne  
 @JoinColumn(name = "reservation\_id", nullable = false)  
 private Reservation reservation;  
  
 @ManyToOne  
 @JoinColumn(name = "user\_id", nullable = false)  
 private User user;  
  
 @CreationTimestamp  
 @Column(nullable = false, updatable = false)  
 private LocalDateTime createdAt;  
}

## CONNEXION

@Getter  
@Setter  
@Entity  
@NoArgsConstructor  
@Table(name = "connexion")  
public class Connexion {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "connexionid", updatable = false, nullable = false)  
 private UUID id;  
  
 @NotNull  
 @Column(name = "begindate", nullable = false)  
 private LocalDate begindate;  
  
 @NotNull  
 @Column(name = "enddate", nullable = false)  
 private LocalDate enddate;  
  
 @NotNull  
 @ManyToOne(fetch = FetchType.*LAZY*, optional = false)  
 @JoinColumn(name = "userid", nullable = false)  
 private User user;  
}

## GARAGE

@Getter  
@Setter  
@Entity  
@NoArgsConstructor  
@Table(name = "garage")  
public class Garage {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "garageid", updatable = false, nullable = false)  
 private UUID id;  
  
 @Size(max = 50)  
 @NotNull  
 @Column(name = "name", nullable = false, length = 50)  
 private String name;  
  
 @Size(max = 20)  
 @NotNull  
 @Column(name = "phone", nullable = false, length = 20)  
 private String phone;  
  
 @NotNull  
 @OneToOne(fetch = FetchType.*LAZY*, optional = false)  
 @JoinColumn(name = "addressid", nullable = false)  
 private Address address;  
}

## PAYS

@Getter  
@Setter  
@Entity  
@NoArgsConstructor  
@Table(name = "pays")  
public class Pays {  
 @Id  
 @GeneratedValue(generator = "UUID")  
 @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")  
 @Column(name = "paysid", updatable = false, nullable = false)  
 private UUID id;  
  
 @Size(max = 50)  
 @NotNull  
 @Column(name = "country", nullable = false, length = 50)  
 private String country;  
  
 @Size(max = 20)  
 @NotNull  
 @Column(name = "assistance", nullable = false, length = 20)  
 private String assistance;  
}

# LES DTOS

## ADDRESSDTO

@Setter  
@Getter  
@AllArgsConstructor  
@NoArgsConstructor  
public class AddressDTO {  
 private String street;  
 private String zipcode;  
 private String town;  
}

## AVAILABILITYDTO

@Setter  
@Getter  
@AllArgsConstructor  
@NoArgsConstructor  
public class AvailabilityDTO {  
 private int day;  
 private LocalTime beginhour;  
 private LocalTime endhour;  
 private Double price;  
  
 @Override  
 public String toString(){  
 return this.day+" "+this.beginhour+" "+this.endhour+" "+ this.price.toString();  
 }  
}

## BORNEDTO

@Setter  
@Getter  
@AllArgsConstructor  
@NoArgsConstructor  
public class BorneDTO {  
 private String taughtname;  
 private String marketname;  
 private AddressDTO address;  
 private int numborne;  
 private double power;  
 private String type;  
 private boolean publicField;  
 private int nbpoint;  
 private Boolean working;  
 private ArrayList<AvailabilityDTO> lstAvailability;  
  
}

## CONSUMPTIONDTO

@Setter  
@Getter  
@AllArgsConstructor  
@NoArgsConstructor  
public class ConsumptionDTO {  
 private String email;  
 private Double nbkwh;  
 private LocalDate date;  
 private LocalTime beginhour;  
 private LocalTime endhour;  
 private int numborne;  
 private AddressDTO address;  
}

## CONSUMPTIONRESPONSEDTO

@Setter  
@Getter  
@AllArgsConstructor  
@NoArgsConstructor  
//utilisé pour le suivi des consommations  
public class ConsumptionResponseDTO {  
 private Double nbkwh;  
 private LocalDateTime begindate;  
 private LocalDateTime enddate;  
 private int numSlice;  
}

## FILTERDTO

@Setter  
@Getter  
@AllArgsConstructor  
@NoArgsConstructor  
public class FilterDTO {  
 private int numborne;  
 private AddressDTO address;  
 private LocalDate begindate;  
 private LocalDate enddate;  
 private String type;  
  
}

## GEOBORNEDTO

@Setter  
@Getter  
@AllArgsConstructor  
@NoArgsConstructor  
public class GeoBorneDTO {  
 private String taughtname;  
 private String marketname;  
 private AddressDTO address;  
 private int numborne;  
 private double latitude;  
 private double longitude;  
 private double power;  
 private String type;  
 private boolean payant;  
 private boolean publicField;  
 private int nbpoint;  
 private Boolean working;  
 private String accessibility;  
 private ArrayList<AvailabilityDTO> lstAvailability;  
  
}

## GEOFILTERDTO

@Setter  
@Getter  
@AllArgsConstructor  
@NoArgsConstructor  
public class GeoFilterDTO {  
 private double userLat; //latitude de l'utilisateur géolocalisé  
 private double userlong; //longitude de l'utilisateur géolocalisé  
 private double userKm; //distance en kilomètres choisie par l'utilisateur  
}

## USERDTO

package com.stage.wattswap.model.dto;  
  
import jakarta.validation.constraints.Email;  
import jakarta.validation.constraints.NotBlank;  
import jakarta.validation.constraints.Size;  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
public class UserDTO {  
  
 @NotBlank(message = "Le nom est obligatoire")  
 @Size(max = 50)  
 private String name;  
  
 @NotBlank(message = "Le prénom est obligatoire")  
 @Size(max = 50)  
 private String firstname;  
  
 @NotBlank(message = "L'email est obligatoire")  
 @Email(message = "Email invalide")  
 @Size(max = 100)  
 private String email;  
  
 @NotBlank(message = "Le mot de passe est obligatoire")  
 @Size(max = 50)  
 private String password;  
  
 private AddressDTO address;  
  
 @NotBlank(message = "Le téléphone est obligatoire")  
 @Size(max = 20)  
 private String phone;  
  
 @Size(max = 200)  
 private String photourl;  
  
 @Size(max = 200)  
 private String idpieceurl;  
}

## RESERVATIONDTO

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

public class ReservationDTO {

@NotNull(message = "User email is required")

@Size(min = 5, message = "User email must be at least 5 characters")

private String userEmail;

@NotNull(message = "Address is required")

private AddressDTO address;

@NotNull(message = "Reservation date is required")

private LocalDate date;

@NotNull(message = "Start time is required")

private LocalTime beginHour;

@NotNull(message = "End time is required")

private LocalTime endHour;

private int numBorne;

}

## RESERVATIONSUMMARYDTO

@Data

@NoArgsConstructor

@AllArgsConstructor

public class ReservationSummaryDTO {

private String email;

private String terminalAddress;

private LocalDate date;

private LocalTime beginHour;

private LocalTime endHour;

}

## PaymentRequestDTO

@Getter  
@Setter  
public class PaymentRequestDTO {  
 @NotNull(message = "Amount cannot be null")  
 @DecimalMin(value = "0.01", message = "Amount must be greater than zero")  
 private BigDecimal amount;  
  
 @NotBlank(message = "Payment method cannot be blank")  
 private String paymentMethod;  
  
 @NotBlank(message = "Currency cannot be blank")  
 private String currency;  
  
 @NotNull(message = "User ID cannot be null")  
 private UUID userId;  
  
 @NotNull(message = "Reservation ID cannot be null")  
 private UUID reservationId; // Added to link payment to reservation  
  
}

## PaymentResponseDTO

public class PaymentResponseDTO {  
 private UUID id;  
 private String paymentIntentId;  
 private BigDecimal amount;  
 private String paymentMethod;  
 private String currency;  
 private UUID reservationId; //   
 private UUID userId; //   
 private LocalDateTime createdAt; //   
}