Projet : Système de Parking Partagé

Hetic 3 eme année – 2025

Contexte

Avec le dérèglement climatique, la réduction des émissions de gaz à effet de serre devient un enjeu mondial et les transports représentent une part significative de ces émissions

Selon certaines études, la recherche d'une place de stationnement représente environ 30 % du trafic en ville ce qui entraîne une augmentation des embouteillages et des émissions, ainsi qu'une perte de temps pour les conducteurs. Réduire ce trafic lié au stationnement constitue un moyen efficace d'améliorer la circulation et de limiter l'impact environnemental.

Paradoxalement, de nombreuses places de stationnement existent en ville mais restent inexploitées à certains moments de la journée. Par exemple de nombreuses places deviennent libres en journée dans les immeubles d'habitations et les hôtels et d'autres restent vident la nuit et le week-end dans les entreprises, créant un décalage entre l'offre réelle et la demande.

Pour répondre à cette problématique, nous proposons de développer une solution de parking partagé qui permet à des propriétaires de places de stationnement (particuliers, entreprises ou collectivités) de proposer leurs places inoccupées à la location horaire. Un utilisateur cherchant une place peut consulter, réserver et payer une place via une application ou une interface web pour un créneau horaire défini.

Objectif

L'objectif du projet est de développer une application web en PHP qui implémente un système de parking partagé. Ce projet doit être réalisé par un groupe de 4 personnes. Votre application devra implémenter toutes les fonctionnalités écrites dans ce document et devra respecter la clean architecture.

Il faudra également mettre en place des tests sur une partie de l'implication

Enfin le projet devra posséder un système d'authentification pour restreindre l'accès à certaines parties de l'application.

Ce genre de projet nécessite normalement une partie hardware et IOT pour gérer l'ouverture des portes de parking. Pour simplifier le projet, nous ferons abstraction de la partie matérielle et admettrons que les portes de parkings s'ouvrent automatiquement lors de l'enregistrement d'une entrée dans un parking.

Technologies à utiliser

Le développement backend doit être réalisé en PHP « pur », sans framework (comme Laravel ou Symfony), afin de mettre en pratique les principes de la clean architecture

Voici les autres contraintes

- PHP 8.x
- Test avec PHPUnit
- Librairies PHP : L'utilisation de librairies externes est autorisée afin de faciliter certaines tâches spécifiques. Utilisez des librairies compatible avec composer.
- L'utilisation d'un framework JavaScript côté client (tel que React, Vue ou Angular) est autorisée pour améliorer l'interactivité et l'expérience utilisateur de l'application.

Données du système

Le projet devra contenir les données suivantes :

Parking :

Un parking contient les données suivantes :

- Une coordonnées GPS
- Un nombre de places de parkings
- Un tarif horaire qui peut varier avec le temps
- Des horaires d'ouverture
- une liste de réservations
- une liste de stationnements

Chaque parking possède son propre tarif. Les utilisateurs sont facturés par tranches de 15 minutes, et le tarif peut augmenter à partir d'une certaine durée de stationnement.

Utilisateur:

Un Utilisateur possède les propriétés suivantes :

- email
- password
- nom
- prénom
- une liste de réservations
- une liste de stationnements
-

Propriétaire de parking:

Un propriétaire de parking possède les propriétés suivantes :

- email
- password
- nom
- prénom
- Une liste de parkings qu'il possède

Stationnement:

Un stationnement correspond au temps entre une entrée et sortie d'un parking. Un stationnement est contient les données suivantes :

- un utilisateur
- le début du stationnement (timestamp)
- la fin du stationnement (timestamp)
- un parking

Réservation:

Une réservation correspond à la réservation préalable d'une place de parking pour un créneau défini. Elle contient les informations suivantes :

- un utilisateur
- un parking
- le début de la réservation (timestamp)
- la fin de la réservation (timestamp)

Abonnement:

Les utilisateurs ont la possibilité de payer des abonnements qui leur garantie une place de parking sur certains créneaux de la semaine de facon mensuel. La durée minimale d'un abonnement est de 1 mois et peut durer jusqu'a un an

Les abonnements doivent être flexibles et il est possible d'avoir un abonnement total sur le mois ou seulement sur certains créneaux horaires.

Exemple :

Abonnement total : Accès illimité à une place de parking à n'importe quelle heure
Abonnement week end : du Vendredi 18h au lundi 10h
Abonnement spécifique : Le jeudi de 10h au vendredi 10h
Abonnement soir : Tout les soirs de 18h a 8h du matin le lendemain

Les créneaux horaires sont géré par semaine et ne peuvent pas varier pendant la durée de l'abonnement

Un abonnement est composé des données suivantes :

- Un utilisateur
- Un parking
- Le ou les créneaux horaires ou la place est reservée.

Remarque :

Les données listées ci-dessus représentent le minimum obligatoire à implémenter.
Vous êtes libres d'ajouter d'autres attributs ou entités si besoin

Fonctionnalités

Le projet devra être développé en respectant les principes de la Clean Architecture.
Le code doit être organisé de manière à séparer les responsabilités entre les différentes couches.

La logique métier devra être implémenté avec des use cases indépendants.
Aucune règle métier ne doit dépendre directement d'un contrôleur, d'une base de données ou d'une technologie externes

Votre projet devra implémenter les use case suivant :

Propriétaires de parkings

Les propriétaires de parkings disposent d'un espace dédié leur permettant de gérer leurs parkings et de suivre leur activité. Ils doivent pouvoir effectuer les actions suivantes :

- Créer un compte propriétaire et s'authentifier
- Ajouter un parking
- Modifier les tarifs d'un parking
- Modifier les horaires du parking
- Voir la liste des réservations d'un parking
- Voir la liste des stationnements d'un parking
- Voir le nombre de place disponible dans un parking à une date précise (timestamp)
- Obtenir le chiffre d'affaire mensuel d'un parking (somme de toutes les réservations terminées du mois + les abonnements)
- Ajouter un type d'abonnement sur un parking
- Voir la liste des conducteurs qui sont garés hors des créneaux de réservation ou d'abonnement pour un parking donné

Utilisateurs

Les utilisateurs représentent les conducteurs souhaitant réserver ou utiliser une place de parking. Ils doivent pouvoir interagir avec le système à travers les fonctionnalités suivantes :

- Créer un compte et s'authentifier
- Rechercher des parkings avec des places disponibles autour d'une coordonnées GPS
- Voir les informations d'un parking
- Réserver une place de parking
- Consulter la liste des abonnements d'un parking
- Souscrire à un abonnement
- Entrer dans un parking
- Sortir d'un parking
- Voir la liste de ses stationnements
- Voir la liste de ses réservations
- Obtenir la facture d'une réservation

Gestion du comptage des places d'un parking

Le système doit maintenir en permanence le nombre de places disponibles dans chaque parking. Lorsqu'un utilisateur fait une réservation dans un parking une place est automatiquement décomptée du nombre de places libres pendant le créneau de la réservation

Inversement, lorsque la réservation prend fin cette place est libérée et redevient disponible pour d'autres utilisateurs.

Les conducteurs possédant un abonnement peuvent entrer et sortir librement pendant la plage horaire de celle ci. Un abonnement actif compte comme une place occupé pendant sa plage horaire même si l'utilisateur n'est pas dans le parking.

Entrée et sortie d'un parking

Un utilisateur ne peut entrer dans un parking que s'il dispose d'une réservation active correspondant au créneau horaire en cours. Sans réservation, l'entrée doit être refusée par le système hors de son créneau.

Lorsqu'un utilisateur effectue une demande de réservation, le système doit vérifier la disponibilité réelle du parking sur la plage horaire souhaitée.
Si le parking est plein à un moment quelconque de cette plage (en raison d'autres réservations ou de stationnements actifs), la réservation doit être refusée. Cela est également valable pour souscrire à un abonnement.

Un utilisateur qui effectue une réservation mais qui ne se rend pas dans le parking ou ne se gare pas pendant la totalité du créneau réservé se voit quand même facturé sur la totalité de la réservation

Une fois la réservation commencée, l'utilisateur peut alors entrer dans le parking durant le créneaux de réservation, le système enregistre l'heure d'entrée et décompte la place correspondante.

Lors de la sortie, l'heure de départ est enregistrée et la place est automatiquement libérée, redevenant disponible pour d'autres utilisateurs ou réservations ultérieures.
Pour simplifier l'exercice, cette action se fera par un appel à un endpoint web pour les entrées/sorties plutôt qu'une détection automatique via un dispositif physique

Horaires d'ouverture

Chaque parking à des horaires d'ouverture spécifiques
Un parking peut être

- disponible en permanence
- ou disponible uniquement sur une ou plusieurs plages horaires (Par exemple, uniquement le week end du vendredi 18h à Lundi 8h, le Mardi de 8h à 18h, tout les soirs de 18h à 8h du matin le lendemain)

Les réservations actives doivent également être prises en compte dans le calcul : dès qu'une réservation débute, la place correspondante est considérée comme occupée, même si l'utilisateur n'est pas entré dans le parking

Cas particulier et pénalités de stationnement

Un conducteur n'est censé se garer dans un parking que pendant un créneau d'une réservation ou d'un abonnement.
Une pénalité de 20 € est appliqué au prix total de la réservation si utilisateur reste au-delà d'un créneau d'une réservation ou d'un abonnement et le temps de stationnement additionnel est également facturé

Exemple :

Un conducteur qui possède une réservation de 3h mais qui est garé 4h doit payer comme une réservation de 4h + 20€ de pénalité

Pour éviter les blocages, le système doit pouvoir détecter les gens resté trop longtemps dans un parking. Les conducteurs qui ne sont pas sorti d'un parking compte comme occupant une place et il ne doit pas être possible de réserver une place si un parking est plein a cause des conducteurs qui sont garés hors créneaux.

Prix d'une réservation

Le prix d'une réservation dépend de la grille tarifaire du parking et de la durée de réservation. Chaque parking possède sa propre grille tarifaire et peut être dégressive avec le temps. **Le tarif peut évoluer par tranche de 15 mins**

Une fois l'utilisateur sorti d'un parking après une réservation, l'application doit correctement calculer le prix du stationnement dans le parking en incluant les éventuelles pénalités. L'application doit pouvoir générer une facture avec le détail de la **facturation au format PDF ou affiché en HTML**

Clean Architecture

Votre programme devra être implémenté en suivant les principes de la clean architecture

Vous devrez notamment découper les partie de votre projet en différentes couches tout en séparant les responsabilités correctement et en respectant les règles de dépendance

Des points vous seront retiré si vous faites de mauvais choix de conception ou si vous ne respectez pas les contraintes de la Clean Architecture

Il vous sera demandé d'utiliser au moins l**es 3 couches suivantes** : Domaine, Use Case, External interface/ View

Vous pouvez implémenter plus de couches si vous en ressentez le besoin.

Stockage des données

Le projet devra intégrer au moins deux systèmes de stockage distincts pour stocker les données telles que les utilisateurs, les parkings, les réservations et les abonnements:

- Base de données relationnelle (MySQL, PostgreSQL ou SQLite)
- Une base de données NoSQL ou un stockage en fichier

Les systèmes de stockage doivent être interchangeable et ne doit nécessiter **aucune modification dans les entités ni dans les use case**

Frontend

Le projet devra proposer deux modes d'accès aux fonctionnalités de l'application :

- Une interface HTML
  L'application doit comporter une interface web classique permettant à un utilisateur d'interagir avec le système depuis un navigateur.
  Cet affichage servira à visualiser les parkings disponibles, effectuer des réservations, consulter ses abonnements et gérer son compte.
- Une Web API
  En parallèle, les mêmes fonctionnalités devront être accessibles via une API REST exposée en JSON.
  Cette API devra permettre l'interaction avec le système à travers des requêtes HTTP (GET, POST, PUT, DELETE) et pourra être utilisée par un client JavaScript ou toute autre application externe.

Les modifications dans l'interface ne doivent avoir aucun impact sur le code des use case et des entités

Authentification

Le projet devra inclure un système d'authentification pour sécuriser l'accès de certaines parties de l'application.

Vous devrez mettre en place un mécanisme d'authentification basé sur des JSON Web Tokens (JWT), afin de gérer la connexion des utilisateurs et la protection des routes nécessitant une identification.

Le projet devra également respecter les bonnes pratiques de sécurité, notamment :

- Hashage des mots de passe (en PHP vanilla)
- des protections contre les injections SQL et les failles XSS
- une gestion correcte du cycle de vie des tokens JWT

Tests

Le projet devra contenir des tests automatisés avec PHP Unit. Il vous sera demandé de mettre en place des **tests unitaires pour vérifier le bon fonctionnement des entités et des cas d'usage** ainsi que des **tests fonctionnels pour tester toute la chaîne applicative pour certains scénario.**

Il faudra que les tests unitaires **permettent d'avoir un taux de couverture d'au moins 60 % du domaine et des entités.** Concernant les tests fonctionnels, ils faudra en créer **au moins 4 dont 2 qui concernent les utilisateurs et 2 concernant les propriétaires de parking**

Les tests devront permettre de garantir la qualité et la fiabilité du systeme. Ils devront être pertinents et couvrir les règles métiers et parties importante du code. Votre note dépendra de l'exhaustivité des tests ainsi que des choix que vous ferez.

Critère d'évaluation

Vous serez tout d'abord évalué sur votre capacité à utiliser correctement les concepts de POO dans un projet composé de nombreuses classes.
Prenez le temps de faire les bons choix de modélisation. Respectez les bonnes pratiques de programmation orientée objet :

DRY (Don't Repeat Yourself)
Single responsability principle

Vous devez essayer de créer une architecture flexible et évolutive en utilisant la Clean Architecture

Le projet est composé de nombreuses fonctionnalités indépendantes et il est possible de réaliser beaucoup de taches en parallèle. Un bon développeur sait travailler en équipe, organisez vous bien avec votre binôme et découpez astucieusement le travail.

Enfin, le projet est composé de nombreuses fonctionnalités, vous devez essayer d'aller au bout des choses et de rendre un projet abouti.

Il n’y a pas une solution unique à ce projet et des architectures très différentes peuvent être tout à fait valide. Vous obtiendrez une bonne note si vous faites preuve de jugement

Critère d’évaluation

Votre note dépendra des points suivants

- Le nombre de fonctionnalités implémentées
- La qualité de votre architecture
- Le fonctionnement de votre système authentification
- L’exhaustivité de vos tests

Vous devrez suivre très rigoureusement les consignes et votre programme devra contenir les entités attendues et implémenter les fonctionnalités demandées

Vous serez évalué sur l’architecture de votre projet qui devra respecter la clean architecture et une partie des points dépendent de vos choix de conception, votre architecture doit avoir du sens et utiliser les concepts de POO au bon moment: Héritage, Encapsulation, Polymorphisme, Design patterns etc... Vous devez essayer d’avoir un programme extensible et évolutif.

Une mauvaise compréhension de la Clean Architecture ou des utilisations manqués des concepts en POO vous feront perdre des points.
Une conception trop rigide, mal pensée ou un programme peu évolutif influencera négativement votre note

Vous serez également évalué sur la qualité de votre code et devrez mettre en place les bonnes pratiques vues lors de ce module. A savoir :

- Des noms de variables et de fonctions claires
- Utiliser la convention CamelCase
- Des fonctions de moins de 20 lignes
- Une indentation correcte
- Un code correctement organisé
- Mise en forme du code uniforme dans le projet (accolades, saut de ligne, conventions de codage)
- Des commentaires si besoin

Des petits écarts peuvent être toléré, cependant si aucun effort n’est réalisé ou si certaines pratiques ne sont pas mises en place, **vous pourrez perdre jusqu’à 4 points sur votre note finale** même si votre projet fonctionne parfaitement.

Barème

- Implémentation de toutes les fonctionnalités : 12 points
- Mise en place de tests avec PHPUnit : 4 points
- Mise en place d'un système d'authentification JWT : 2 points
- Architecture du projet : 2 points

Rendu

Vous devez rendre votre projet sous la forme d'un fichier compressé .zip contenant l'ensemble du code source et un readme qui explique comment installer et exécuter votre projet. Vous devrez envoyer le zip à l'adresse email suivante :

dany.siriphol@gmail.com

Vous avez jusqu'au Lundi **22 Décembre 2025 23h59** pour rendre votre projet. 2 points seront retirés à votre note par jour de retard et tout projet rendu après le dimanche 24 Décembre 23h59 ne sera pas évalué.