Système de Gestion d'un Parking

Introduction

**1-Contexte et Justification**

* Présentation du besoin : La gestion d'un parking est cruciale pour optimiser l'utilisation de l'espace et faciliter l'expérience d’utilisateur. Les gestionnaires ont besoin d'une application pour suivre efficacement l'entrée et la sortie des véhicules.
* Importance du développement de l’application : Une telle application permet de réduire l'attente, d'améliorer la satisfaction client, et d'automatiser le processus de facturation.

**2. Objectifs du Projet**

* Objectif principal : Développer une application autonome pour gérer le stationnement et la facturation des véhicules.
* Objectifs spécifiques:
* Gérer l’entrée et la sortie des véhicules.
* Calculer le montant à payer en fonction de la durée de stationnement.

3. **Méthodologie Adoptée**

* Choix des outils et technologies : Java pour le développement, Springs Boot pour la structure, et une éventuelle base de données SQL pour stocker les données.
* Organisation du développement : Suivre le cycle en V,
* Organisation du développement : Suivre le cycle en V, qui est idéal pour les projets bien définis. Chaque phase inclut des tests et des validations avant de passer à la suivante

***ii. Analyse des besoins***

1. Identification des parties prenantes

* + Utilisateurs finaux : Clients utilisant le parking.
  + Administrateurs : Gestionnaires du parking, en charge de la supervision des opérations.
* Développeurs : Équipe de développement responsable de la création et de la maintenance de l'application.

**2.Cahier des charge**

* **Spécifications fonctionnelles :**

- Enregistrement des véhicules entrants et sortants.

- Calcul automatique des frais en fonction de la durée de stationnement.

- Génération de reçus pour les paiements.

* **Spécifications non fonctionnelles** :

- Performance : Temps de réponse inférieur à 2 secondes pour les opérations critiques.

- Sécurité : Protection des données sensibles et contrôle d'accès sécurisé.

- Ergonomie : Interface intuitive et facile à utiliser.

**ii. Analyse des besoins**

3. **Modélisation du système**

* **Diagramme des cas d’utilisation (UML) :**

- Utilisateurs peuvent : enregistrer un véhicule, sortir un véhicule, consulter des frais.

- Administrateurs peuvent : voir la liste des véhicules, gérer les tarifs, générer des rapports.

* ***Diagramme de classes :***

- Classes principales : Véhicule Parking Utilisateur Facture.

- Relations entre classes :

- Un Parking contient plusieurs Véhicule.

- Un Utilisateur peut générer des Facture.

**Modèle de données (si base de données utilisée) :**

- Tables : Véhicules (id, plaque, temps Entrée), Factures (id, montant, temps Sortie).

**iii. Conception de l’application**

**1. Architecture logicielle**

- Architecture choisie : MVC (Modèle-Vue-Contrôleur).

- Justification du choix : Permet une séparation claire des préoccupations, facilitant l’évolution et la maintenance

**2. Choix des technologies et outils**

* **Langages :**

- Java pour la logique métier.

- SQL pour la gestion de la base de données.

* **Frameworks** :

- Spring Boot pour le développement backend.

- JavaFX pour l'interface utilisateur desktop (ou Angular pour une application web).

- IDE et gestion des versions :

- Intellij IDEA ou Eclipse pour le développement.

- Git pour le contrôle de version.

- Maven pour la gestion des dépendances.

**3.Schéma de la base de données**

- Tables :

- Véhicules :

- `ID` (int, clé primaire)

- `Plaque` (varchar)

- `TempsEntree` (datetime)

**Factures** :

- `ID` (int, clé primaire)

- `Montant` (double)

- `TempsSortie` (datetime)

- `VehiculeID` (int, clé étrangère)

iv. **Développement de l’application**

1. **Implémentation des fonctionnalités principale**

-Authentification et gestion des utilisateurs :

- Implémentation de rôles pour les utilisateurs et administrateurs.

Interface utilisateur :

- Conception d'une interface simple pour l'enregistrement, la sortie, et la consultation des frais.

Services métiers et logique applicative :

- Méthodes pour gérer l'entrée et la sortie des véhicules.

- Calcul automatique des frais sur la base du temps enregistré.

**2. Gestion des erreurs et exceptions**

- Stratégie de gestion des erreurs :

- Utilisation d'exceptions personnalisées pour gérer les erreurs de saisie ou d'affichage.

- Sécurité et validation des entrées :

- Validation des plaques d'immatriculation et gestion des accès aux données sensibles.

**3. Tests et validation**

- Tests unitaires :

- Utilisation de JUnit et Mockito pour tester les méthodes clés, comme le calcul des frais et la gestion des véhicules.

- Tests d’intégration et fonctionnels:

- Vérification du bon fonctionnement des interactions entre les différentes parties de l’application, comme l’enregistrement et la facturation.

- Débogage et optimisation du code :

- Analyse des performances et correction des bugs détectés lors des tests.

**v. Déploiement et maintenance**

**1. Stratégie de déploiement**

- Serveur d’hébergement:

- Déploiement sur un serveur cloud, comme AWS ou Azure, pour garantir la scalabilité et la disponibilité.

- Conteneurisation :

- Utilisation de Docker pour isoler l’application et faciliter sa gestion.

**2.Plan de maintenance et mises à jour**

- Surveillance des performances :

- Mise en place d’outils pour surveiller l’utilisation des ressources et réaliser des diagnostics.

- Gestion des mises à jour :

- Établir un calendrier de mises à jour régulières pour intégrer de nouvelles fonctionnalités et patcher les bugs.

- Corrections de bugs :

- Réponses rapides aux retours des utilisateurs et corrections basées sur les feedbacks récoltés après le déploiement.

**vi. Conclusion et perspectives**

**1. Bilan du projet**

- Résumé des réalisations :

- Développement complet d'une application de gestion de parking fonctionnelle, intégrant l'enregistrement, le calcul des frais et la gestion des utilisateurs.

- Difficultés rencontrées et solutions adoptées:

- Problèmes de scalabilité résolus par la mise en place d’une architecture adaptée et l'utilisation de technologies robustes.

2. **Améliorations et évolutions possibles**

- Fonctionnalités futures envisagées :

- Intégration de paiements en ligne et d'analyses avancées des données d'utilisation.

-Adaptabilité

**2. Améliorations et évolutions possibles**

- Fonctionnalités futures:

- Intégration de paiements en ligne pour faciliter les transactions.

- Ajout d'analyses avancées pour mieux comprendre les comportements des utilisateurs.

- Adaptabilité et évolutivité :

- Préparation de l’architecture pour accueillir de nouvelles fonctionnalités selon les besoins des utilisateurs.

**3.Retours d’expérience**

- Leçons apprises :

- Importance d'une bonne communication avec les parties prenantes.

- Nécessité d’un bon système de gestion des tests pour assurer la qualité du produit final.

Impact sur l'équipe:

- Renforcement des compétences techniques et amélioration du travail collaboratif.

**4. Remerciements**

- Reconnaissance envers l'équipe de développement, les parties prenantes et tous ceux qui ont contribué au projet

Conclusion finale

- Synthèse des résultats :

- Création d'une application efficace répondant aux besoins de gestion de parking, avec une interface intuitive et une logique robuste.

- Vision future :

- Engagement à continuer d'améliorer l’application avec des mises à jour basées sur les retours utilisateurs, garantissant une expérience optimale et durable.

**6. Annexes**

Documentation technique :

- Manuels d’utilisation et guides de développement.

- Diagrams UML et Schémas de base de données :

- Illustrations des structures et interactions.