SAE : Développement d'une Application Vélib « Groupe Paris »

Réalisé par :

- Oumaima EL-KHADRAOUI : ***Stymphale***
- Tassadit OUZIA : ***Crete***
- Kenza HALIL : ***Stymphale***
- Yannel AISSANI (leader) : ***Dyomède***
- Youssouf REZZAG-MAHCENE : ***Dyomède***

Encadré par :

- Mme Hanane AZZAG
- M. Bouchaib LEMAIRE

Nom de l'établissement :

- IUT de Villetaneuse, Université Sorbonne Paris Nord

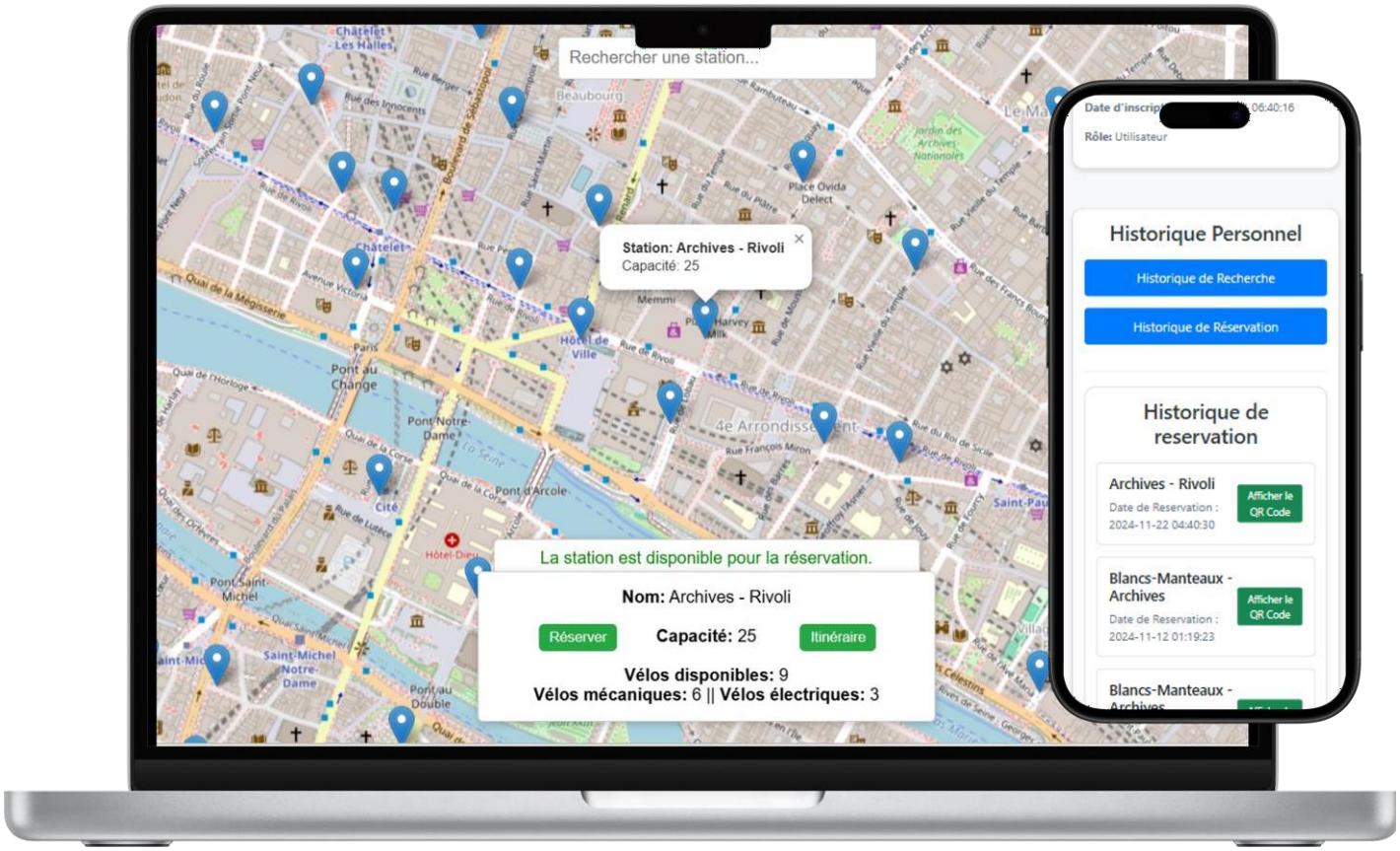
Année Académique :

- 2024/2025

Date de Remise : 03/01/2025

Livrable 2 : Développement d'un référentiel

« Conception de la base de données »



« Application web en cours de développement par le groupe PARIS »

Table des Matières :

1. Introduction :

Dans ce livrable, il nous a été demandé de concevoir une base de données pour le projet Vélib'. Cette base de données joue un rôle central dans la gestion et l'organisation des informations nécessaires au fonctionnement du projet. Elle est conçue pour stocker et structurer divers types de données, notamment celles liées aux utilisateurs, aux vélos, aux stations, et à leurs interactions.

La conception de cette base de données repose sur plusieurs éléments essentiels :

- **Tables statiques :** Les tables des vélos et des stations, remplies une seule fois, contiennent des données fixes. La table des stations est alimentée à partir des données publiques fournies par le service Vélib', grâce à un script Python développé pour récupérer et insérer ces informations automatiquement.
- **Tables dynamiques :** Les tables des utilisateurs, des recherches, et des réservations, conçues pour évoluer au fil du temps, permettent de gérer les actions et les données générées par les utilisateurs du service.
- **Gestion des droits :** Un utilisateur spécifique a été créé dans la base de données avec des droits bien définis sur chaque table, afin de garantir une gestion sécurisée et adaptée des données.

L'objectif principal de cette base de données est de fournir une structure solide et optimisée pour centraliser toutes les informations nécessaires. Elle assure la cohérence et l'intégrité des données tout en permettant un traitement rapide et efficace. Ce travail constitue une base essentielle pour le bon fonctionnement du projet Vélib' et la mise en place des futures fonctionnalités.

2. Interaction de l'utilisateur avec la base de données :

Dans le projet Vélib', l'utilisateur interagit directement avec la base de données à travers plusieurs actions clés. Ces interactions permettent de gérer son compte, de stocker l'historique des recherches et des réservations, ainsi que de réserver des vélos.

Voici un aperçu des principales actions que l'utilisateur peut effectuer dans la base de données :

2.1 Création de compte :

Lorsqu'un utilisateur souhaite accéder à l'application, il doit créer un compte. Ce processus inclut :

- Renseignement des informations personnelles (nom, email, mot de passe sécurisé).
- Transmission des données au serveur pour validation et insertion dans la table **users**.
- Hachage du mot de passe avant son stockage pour garantir la sécurité.

2.2 Réservation de vélos :

Après avoir identifié une station, l'utilisateur peut réserver un vélo. Ce processus inclut :

- Sélection d'un type de vélo disponible.
- Création d'une entrée dans la table ***reservations*** avec les informations sur l'utilisateur, la station et le vélo réservé.

2.3 Gestion de l'historique :

L'utilisateur peut visualiser son historique de recherches et réservations :

- L'utilisateur peut supprimer les recherches effectuées (nom de station ou bien une adresse) si nécessaire, mais ne peut pas supprimer l'historique des réservations.

3. Technologies Utilisées

Dans ce projet, nous avons utilisé plusieurs technologies afin de concevoir et gérer efficacement la base de données pour le système Vélib'. Voici les technologies principales que nous avons utilisées :

1. **MySQL :** Nous avons choisi MySQL comme système de gestion de base de données. MySQL nous permet de créer des bases de données, des tables et de gérer les données de manière efficace et sécurisée.

2. **PHPMyAdmin :** est un outil web populaire pour la gestion de bases de données. Nous avons utilisé PHPMyAdmin pour interagir facilement avec la base de données sans avoir à écrire manuellement des commandes SQL à chaque fois. Cet outil nous a permis de créer, modifier et visualiser les tables, d'insérer des données, et d'exécuter des requêtes SQL pour gérer la base de données de manière intuitive et rapide. Il fournit une interface graphique conviviale pour manipuler la base de données.

3. **Python :** Python a été utilisé principalement pour automatiser certains processus, notamment pour remplir la base de données. Par exemple, nous avons écrit un script Python pour récupérer automatiquement les données des stations Vélib' et les insérer dans la table ***stations*** de la base de données.

4. **Git et GitHub :** Pour gérer les versions du code et faciliter la collaboration entre les membres de l'équipe, nous avons utilisé Git et GitHub. Git nous a permis de suivre les modifications apportées au code, tandis que GitHub a servi de plateforme pour héberger le code, stocker les versions et faciliter le travail en équipe. Cela a permis à chaque membre de travailler sur des fonctionnalités spécifiques et de fusionner les changements sans conflits.

Ces technologies ont été choisies pour leur fiabilité, leur efficacité et leur capacité à répondre aux besoins spécifiques du projet Vélib'. Grâce à MySQL et PHPMyAdmin, nous avons pu concevoir une base de données solide et facile à gérer, tandis que Python a automatisé les processus pour faciliter la gestion des données dans le projet.

4. Présentation des Tables et Vues de la Base de Données

Dans cette section, nous allons détailler les tables créées pour la base de données du projet Vélib'. Nous expliquerons chaque table, les attributs qu'elles contiennent, les types de données utilisés, ainsi que les relations entre les différentes tables. Chaque table est essentielle au bon fonctionnement de la base de données et joue un rôle clé dans l'organisation des informations.

4.1 Tables principales :

1. Table *users*

Cette table est utilisée pour stocker les informations des utilisateurs inscrits sur le site Vélib'. Chaque utilisateur aura un identifiant unique, ainsi que des informations telles que son nom, son email, son mot de passe et la date de création de son compte.

Attributs :

- **id** (INT, AUTO_INCREMENT, PRIMARY KEY) : L'identifiant unique de chaque utilisateur, généré automatiquement à chaque insertion.
- **username** (VARCHAR(255), NOT NULL) : Le nom d'utilisateur.
- **email** (VARCHAR(100), NOT NULL) : L'email de l'utilisateur, unique.
- **password** (VARCHAR(255), NOT NULL) : Le mot de passe de l'utilisateur, utilisé pour l'authentification.
- **created_at** (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP) : La date et l'heure de la création du compte.

Relations :

- **Clé primaire** : **id** permet de garantir l'unicité de chaque utilisateur dans la base de données.
- Cette table est référencée par la table ***reservations*** et la table ***recherches*** via **client_id**.

2. Table *velo*

La table ***velo*** contient des informations sur les types de vélos. Chaque vélo a un identifiant unique.

Attributs :

- **id_velo** (INT, PRIMARY KEY) : Identifiant unique du vélo.
- **type** (VARCHAR(50)) : Le type de vélo, par exemple "mecanique" ou "électrique".

Relations :

- **Clé primaire** : **id_velo** identifie chaque vélo de manière unique.
- Cette table est référencée par la table ***reservations*** via **id_velo**, pour lier un type de vélo à une réservation.

3. Table *stations*

La table stations stocke les informations des stations, telles que la localisation (latitude et longitude) et le nom de la station.

Attributs :

- **station_id** (BIGINT, PRIMARY KEY) : L'identifiant unique de chaque station.
- **lat** (DOUBLE PRECISION) : La latitude de la station pour la géolocalisation.
- **lon** (DOUBLE PRECISION) : La longitude de la station pour la géolocalisation.
- **station** (VARCHAR(255)) : Le nom de la station.

Relations :

- **Clé primaire** : **station_id** permet d'identifier chaque station de manière unique.
- Cette table est référencée par les tables ***reservations*** et ***recherches*** via ***station_id***, pour lier une station à une réservation ou une recherche.

4. Table *reservations*

La table réservations enregistre les réservations effectuées par les utilisateurs pour des vélos spécifiques.

Attributs :

- **id** (INT, AUTO_INCREMENT, PRIMARY KEY) : L'identifiant unique de chaque réservation.
- **confirmationID** (VARCHAR(255), NOT NULL) : Un identifiant pour confirmer chaque réservation.
- **id_velo** (INT, NOT NULL) : L'identifiant du vélo réservé, lié à la table ***velo***.
- **client_id** (INT, NOT NULL) : L'identifiant de l'utilisateur ayant effectué la réservation, lié à la table ***users***.
- **create_time** (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP) : La date et l'heure de la réservation.
- **station_id** (BIGINT) : L'identifiant de la station où le vélo est réservé, lié à la table ***stations***.

Relations :

- **Clé primaire** : **id** pour identifier de manière unique chaque réservation.
- **Clés étrangères** :
    - **id_velo** fait référence à **id_velo** dans la table ***velo***, liant une réservation à un vélo.

- **client_id** fait référence à id dans la table ***users***, liant une réservation à un utilisateur.
- **station_id** fait référence à **station_id** dans la table ***stations***, liant la réservation à une station spécifique.

5. Table *recherches*

La table recherches enregistre les recherches effectuées par les utilisateurs sur le site Vélib'. Chaque recherche est liée à un utilisateur et à une station spécifique (si elle existe).

Attributs :

- **id** (INT, AUTO_INCREMENT, PRIMARY KEY) : Identifiant unique de la recherche.
- **client_id** (INT, NOT NULL) : L'identifiant de l'utilisateur qui a effectué la recherche, lié à la table users.
- **recherche** (VARCHAR(255)) : Le texte de la recherche (elle est `NULL` si le terme rcherché est une station existante dans le tabalux ***stations***).
- **created_at** (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP) : La date et l'heure de la recherche.
- **resultat** (TINYINT(1)) : Un indicateur pour savoir si la recherche a donné un résultat (1 = résultat trouvé, 0 = aucun résultat).
- **station_id** (BIGINT) : L'identifiant de la station correspondant à la recherche, lié à la table stations. (Elle est `NULL` si le terme rcherché est une station NON existante dans le tabalux ***stations***)

« Consultez le fichier README pour une explication détaillée du fonctionnement »

Relations :

- **Clé primaire** : id pour identifier de manière unique chaque recherche.
- **Clés étrangères** :
  - **client_id** fait référence à id dans la table ***users***, liant une recherche à un utilisateur.
  - **station_id** fait référence à **station_id** dans la table ***stations***, liant une recherche à une station.

4.2 Les Vues

1. Vue *recherches_vue*

Elle affiche l'historique des recherches des utilisateurs, en ajoutant des détails sur les stations. La vue utilise un « LEFT JOIN » entre les tables ***recherches*** et ***stations*** pour afficher toutes les recherches, même celles qui ne correspondent pas à une station valide (avec **station_id** à NULL)

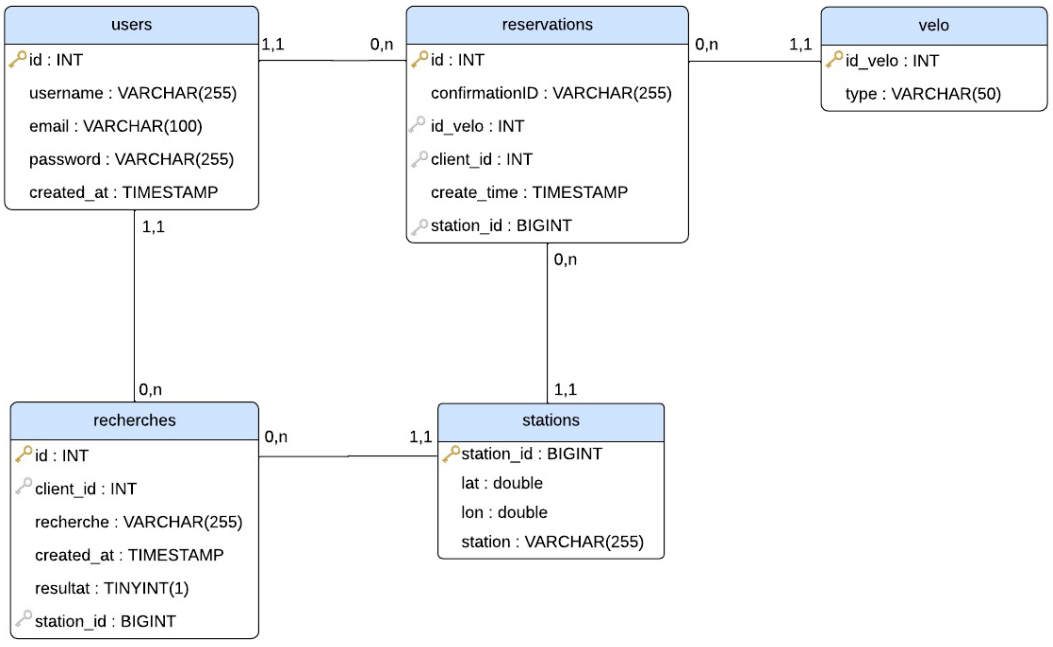
« Consultez le fichier README pour une explication détaillée du fonctionnement »

2. Vue *reservations_vue*

Elle fournit un récapitulatif des réservations, avec les détails des vélos et des stations associés. Cette vue combine les données des tables ***reservations***, ***stations*** et ***velo*** pour donner un aperçu complet de chaque réservation.

Les vues dans une base de données simplifient l'accès aux données en masquant la complexité des tables sous-jacentes. Elles facilitent également la gestion des requêtes complexes en regroupant des jointures et des filtres.

Diagramme UML Représentant la Structure de Notre Base de Données



5. Conception et normalisation de la base de données :

La base de données a été structurée pour garantir une gestion claire et efficace des informations tout en évitant les duplications inutiles. Chaque table a un rôle bien défini et est conçue pour être facilement reliée aux autres via des clés primaires et étrangères. Plus précisément, les tables ***stations*** et ***velo*** jouent un rôle clé dans cette organisation :

1. **Table *stations* :**
   La table ***stations*** stocke les informations géographiques et descriptives des stations, cela centralise les données liées aux stations et permet une réutilisation efficace.

   Avantages :

   - Évite la duplication des informations liées aux stations dans d'autres tables comme ***recherches*** et ***reservations***.
   - Simplifie la mise à jour des données (par exemple, correction des coordonnées GPS ou modification du nom d'une station).

2. **Table *velo* :**
   La table ***velo*** stocke les informations sur les types de vélo. Elle sépare les données des vélos des informations sur les stations et les réservations pour une gestion plus claire et modulaire.

   Avantages :

   - Centralise les informations sur les vélos, ce qui facilite leur gestion.
   - La séparation des données des vélos améliore la flexibilité pour intégrer de nouveaux types ou caractéristiques de vélos.
   - Réduit les redondances et assure que chaque vélo est identifié de manière unique dans le système.

5.1 Relations et cohérence

Les relations définies avec des clés étrangères assurent une parfaite cohérence des données :

- La colonne **id_velo** dans ***reservations*** référence les vélos dans la table ***velo***.
- La colonne **station_id** dans ***reservations*** et ***recherches*** référence les stations dans la table ***stations***.

Ces relations permettent de :

- Éviter les incohérences, comme une réservation liée à un vélo inexistant ou une recherche associée à une station supprimée ou inexistante.
- Faciliter les requêtes complexes grâce à des jointures efficaces pour extraire des données consolidées.

Imaginons maintenant que ces tables n'aient pas été créées. Les noms des stations recherchées et les types de vélos réservés seraient directement stockés dans des tables comme ***recherches*** et ***reservations***. Cela entraînerait une redondance importante des données sur plusieurs lignes. Par exemple :

- Chaque fois qu'un utilisateur effectue une recherche ou réserve un vélo, le nom de la station ou le type de vélo serait répété, même si ces informations restent identiques.
- Si nous avons des milliers ou des millions d'utilisateurs, cette duplication de données aurait un coût considérable en termes de mémoire et de stockage.

De plus, en cas de modification d'un élément (par exemple, corriger le nom d'une station), il serait nécessaire de mettre à jour toutes les occurrences dans la base de données, augmentant ainsi la charge de travail et le risque d'incohérences.

En créant des tables séparées pour les stations et les vélos, nous centralisons ces informations :

- Chaque station et chaque type de vélo est défini une seule fois.
- Les autres tables n'ont besoin que de référencer ces données via des clés étrangères, éliminant ainsi les duplications et optimisant l'utilisation de la mémoire.

IMPORTANT :

Le fait de ne pas stocker le nombre de vélos disponibles et d'autres informations liées est dû à leur nature dynamique. En effet, ces données changent fréquemment en raison des réservations effectuées par les utilisateurs. Par conséquent, il est plus efficace de récupérer ces informations en temps réel directement depuis l'API, afin d'assurer une précision optimale et d'éviter la gestion complexe de mises à jour continues dans la base de données.

6. Création d’un utilisateur avec privilèges limités :

Pour renforcer la sécurité, un utilisateur spécifique, **velib_user** a été créé avec des privilèges limités :

- **Privilèges accordés :**
  - **SELECT** sur toutes les tables pour permettre la lecture.
  - **INSERT** sur ***users***, ***recherches***, et ***reservations*** pour autoriser la création de comptes, recherches et réservations.
  - **UPDATE** uniquement sur ***users*** pour modifier les informations personnelles.
  - **DELETE** uniquement sur la table ***recherches***, pour supprimer ces recherches.
- **Privilèges refusés :**
  - Aucun droit de création, modification ou suppression des tables.

« Consultez le fichier README pour une explication détaillée du fonctionnement »

Cette approche limite les actions possibles à celles strictement nécessaires, réduisant ainsi les risques d’abus ou d’erreurs. De plus, du côté back-end de notre application web, des mesures de sécurité seront mises en place pour empêcher toute injection de requêtes malveillantes, garantissant ainsi la protection des données.

7.Restauration des tables :

Dans cette étape, nous avons procédé à l'alimentation de certaines tables essentielles de notre base de données, à savoir les tables ***stations*** et ***velo***. Ces étapes sont cruciales pour garantir le bon fonctionnement du système, en assurant que les tables contiennent les données nécessaires pour les fonctionnalités de recherche et de réservation.

7.1 Remplissage de la table *stations*

La table ***stations*** stocke les informations liées aux stations Vélib', notamment l'identifiant de la station, sa latitude, sa longitude et son nom. Pour remplir cette

table, nous avons récupéré les données à partir de notre **API FLASK** qui fournit des informations sur les stations Vélib' en temps réel.

Le processus de remplissage des stations s'effectue de la manière suivante :

- Pour chaque station, nous vérifions si elle est déjà présente dans la base de données en vérifiant la valeur de **station_id**.
- Si la station n'existe pas, nous procédons à son insertion.
- Si la station existe déjà, nous comparons les données actuelles avec celles présentes dans la base de données. Si des différences sont détectées, nous mettons à jour les informations correspondantes (latitude, longitude, nom de la station).

Ainsi, cette opération permet de garantir que la table ***stations*** contient les informations les plus récentes et les plus cohérentes, en tenant compte des mises à jour possibles des stations Vélib' au fil du temps.

7.2 Remplissage de la table *velo*

La table ***velo*** est utilisée pour stocker les informations sur les types vélos disponibles (électrique, mécanique, ou NULL si non précisé). Pour cette étape, nous avons inséré manuellement trois types de vélos dans la table afin de disposer d'un échantillon suffisant pour tester les fonctionnalités de réservation et de recherche.

Les trois vélos insérés dans cette table représentent les types suivants :

1. Un vélo de type « **électrique »**.
2. Un vélo de type « **mécanique »**.
3. Un vélo dont le type est « **NULL »** (par exemple, pour des vélos dont le type n’est pas spécifié ou pour une future extension des types de vélos).

Cela permet de préparer la base de données pour des opérations de réservation en utilisant ces vélos, garantissant ainsi que les processus de réservation fonctionnent correctement.

7.3 Importance de l'alimentation des tables

Le remplissage des tables ***stations*** et ***velo*** est une étape indispensable car ces données sont directement liées aux autres tables, comme ***recherche*****s** et ***reservations***. Sans données dans ces tables, les fonctionnalités de recherche par station et de réservation de vélos ne pourraient pas être exécutées correctement, ce qui compromettrait le bon fonctionnement du système.

En conclusion, cette étape de restauration des tables permet d'assurer l'intégrité et la cohérence des données dans la base de données, en alimentant les tables nécessaires à la gestion des stations et des vélos dans le cadre des réservations et des recherches. Elle garantit également que les données restent à jour et cohérentes avec les informations externes (API Flask).

8. Conclusion et Perspectives :

Ce livrable a permis de concevoir une base de données fonctionnelle pour le projet Vélib', en mettant en place les tables nécessaires pour gérer les utilisateurs, les réservations, les recherches, les stations et les vélos. Nous avons également ajouté des vues pour faciliter la consultation des données et créé un utilisateur avec des privilèges limités pour assurer la sécurité de la base.

Bilan : La mise en place de la base de données garantit une gestion cohérente et fluide des informations, permettant aux utilisateurs d'interagir efficacement avec le système.

Perspectives :

- Optimisation des performances de la base de données.
- Amélioration de la validation des données pour éviter les erreurs.
- Développement de nouvelles fonctionnalités, comme la gestion des paiements ou l'ajout de nouveaux types de vélos.

En somme, ce livrable constitue une base solide pour le projet, avec des perspectives d'amélioration continue pour enrichir l'expérience utilisateur et la performance du système.