PROJET D'ANNEE POUR LE DIPLÔME DE LICENCE PROFESSIONNELLE
METIERS DE L'INFORMATIQUE : APPLICATIONS WEB

PARCOURS : SIG ORIENTE WEB

MISE EN PLACE D'UNE APPLICATION DE GESTION DES TOMBES D'UN CIMETIERE

Période : 3 octobre au 23 juin 2023

Rédigé et soutenu par :

ATCHEAKOU Kossivi Florent

NKONGO SAME Jules

Année : 2022-2023

AVANT-PROPOS

La licence professionnelle "Métiers de l'informatique : Applications Web" avec spécialisation en Système d'Information Géographique orienté Web (SIGWEB), dispensée à l'IUT de Perpignan-Carcassonne et rattachée à l'UPVD est une formation axée sur les technologies cartographiques en ligne. Elle permet d'apprendre à créer des cartes interactives, à programmer dans les langages de programmation SIG, ainsi qu'à manipuler les données géographiques en différents formats. Cette licence pluridisciplinaire nous forme également en analyse spatiale et statistique, en géographie et en économie. Grâce à des périodes d'alternance et des projets professionnels, elle offre une insertion dans le milieu du travail, avec des opportunités dans le développement informatique, l'aménagement du territoire, la sécurité, le géomarketing et le développement économique. En bref, cette licence professionnelle permet d'acquérir des compétences solides pour travailler dans le domaine des applications cartographiques en ligne et d'accéder à diverses opportunités professionnelles.

L'objectif de ce projet est de permettre aux étudiants de mettre en pratique les connaissances acquises au cours de la formation et de l'expérience en entreprise. Dans le but de devenir des développeurs web SIG efficaces et qualifiés sur le marché de l'emploi. Notre projet a été effectué au cours de notre année de formation en licence professionnelle en vue de réaliser notre application web dont le sujet est : « Mise en place d'une APPLICATION DE GESTION DES TOMBES D'UNE CIMETIERE ».

INTRODUCTION

Dans le cadre de notre formation en licence professionnelle « SIG WEB », nous avons eu à travailler sur un projet de mise en place d'une application WEB au cours de l'année. Au moment de choisir les sujets, c'est un des membres de notre binôme qui a eu l'idée de faire une appli web sur la gestion des tombes d'un cimetière car il avait déjà travaillé sur un sujet similaire par le passé. Le projet consiste donc à créer une application web pour gérer les tombes d'un cimetière. Le but de l'application est d'offrir aux gestionnaires de cimetières une vue globale de leur espace, de suivre les emplacements des tombes, qu'elles soient vides ou occupées, et de gérer les informations relatives aux défunts et aux propriétaires des tombes. L'application permettra également la réservation d'emplacements pour des individus ou des familles.Pour la réalisation de ce projet, nous avons utilisé les langages et les outils suivants tels que PHP, HTML, CSS et Javascript pour la programmation. Bootstrap pour l'interface de navigation web. Les Systèmes de Gestion de Bases de Données MySQL et PostgreSQL nous ont permis de développer notre application avec ses outils utilisateur PHPMyAdmin et PG Admin avec lequel nous avons créé, rempli et utilisé notre propre base de données. Nous avons

également mis à profit nos connaissances acquises en cours d'année sur la librairie Javascript de carte interactive **Leaflet** ainsi que sur les différents logiciels de cartographie et de gestion de données tels que **ArcGIS Pro, QGIS et Excel** pour effectuer des traitements divers. Tous ces outils ont été utilisés avec pour objectif pour les utilisateurs de permettre une meilleure prise en main de l'application.

CAHIER DES CHARGES

1. Introduction

L'objectif de ce projet est de développer une application web permettant de gérer les tombes d'un cimetière.

L'application doit pouvoir permettre de répondre à un cahier des charges bien précis sur le côté utilisateur. Elle est destinée à un agent/gestionnaire de cimetière qui cherche à gérer les tombes du cimetière de la commune où il travaille.

L'application offrira des fonctionnalités permettant de visualiser les emplacements, de créer et modifier des propriétaires, de gérer les concessions, de suivre les défunts et de gérer les documents liés aux propriétaires.

2. Fonctionnalités de l'application

2.1 Attribution des tombes :

L'application permet d'attribuer facilement une tombe à un propriétaire en concession. Cela facilite le suivi de la disponibilité des emplacements et de leur affectation.

2.2 Ajout de bénéficiaires

Il est possible d'ajouter des bénéficiaires pour chaque concession, favorisant ainsi la traçabilité et la gestion des droits.

2.3 Cartes interactives

Les gestionnaires peuvent consulter les tombes en fonction de leur disponibilité, type, et zone grâce à des cartes interactives intégrées. Ces cartes offrent des informations détaillées sur chaque tombe au clic.

2.4 Recherche de défunts

La fonction de recherche intégrée facilite la localisation précise des défunts dans le cimetière. Elle offre aussi un itinéraire vers la tombe correspondante, rendant ainsi la navigation dans le cimetière plus facile.

2.5 Aperçu de l'état du cimetière

L'application donne un aperçu complet de l'état du cimetière, facilitant ainsi la planification et la gestion des ressources.

2.6 Tableau de bord

Cette fonctionnalité offre des informations en temps réel sur les tombes, les places disponibles, les propriétaires, les bénéficiaires, les défunts et les types de tombes. Ce tableau de bord facilite la prise de décision et aide à la gestion stratégique du cimetière.

3. Technologies requises

- L'application sera développée en utilisant les technologies web courantes, telles que HTML, CSS, PHP et JavaScript et les collections d'outils et de framework que l'on peut retrouver sur Bootstrap notamment pour la gestion de l'interface utilisateur.

- Pour le stockage des données, une base de données relationnelle (BDDR) sera utilisée. Il est recommandé d'utiliser un système de gestion de bases de données (SGBD) tel que MySQL ou PostgreSQL.

- Pour l'interface cartographique, la bibliothèque open-source de cartographie Leaflet, telle qu'étudiée durant la formation, pourra servir pour l'affichage d'une carte dynamique de visualisation des tombes selon les différents statuts.

4. Interface administrateur

L'interface administrateur de l'application est conçue de manière intuitive et conviviale, permettant aux gestionnaires de cimetière d'exécuter efficacement leurs tâches quotidiennes.

- Tableau de bord : L'interface administrateur est dotée d'un tableau de bord robuste qui affiche des informations pertinentes en temps réel. Il fournit des statistiques détaillées sur le nombre de tombes, les places disponibles, les propriétaires, les bénéficiaires, les défunts et les types de tombes. Ce tableau de bord facilite la prise de décision en offrant une vue d'ensemble claire et concise de la gestion du cimetière.

- Gestion des tombes : Les gestionnaires peuvent attribuer des tombes aux propriétaires en concession en quelques clics. Ils peuvent également ajouter et gérer les bénéficiaires pour chaque concession, le tout dans une seule et même interface.

- Carte interactive : L'interface comprend une carte interactive qui offre une visualisation en temps réel des emplacements des tombes dans le cimetière. Les gestionnaires peuvent filtrer et chercher des tombes par disponibilité, type, et zone, rendant ainsi la gestion des emplacements plus aisée.

- Recherche de défunts : L'interface dispose d'une fonction de recherche intégrée qui permet aux gestionnaires de trouver rapidement des informations précises sur les défunts et leurs tombes respectives.

Cette interface administrateur vise à faciliter la tâche des gestionnaires de cimetière en centralisant toutes les fonctionnalités nécessaires à une gestion efficace et précise du

cimetière. Elle est non seulement facile à utiliser, mais elle a également été conçue pour optimiser les processus de gestion de cimetière, rendant ainsi les opérations quotidiennes plus fluides et plus efficientes.

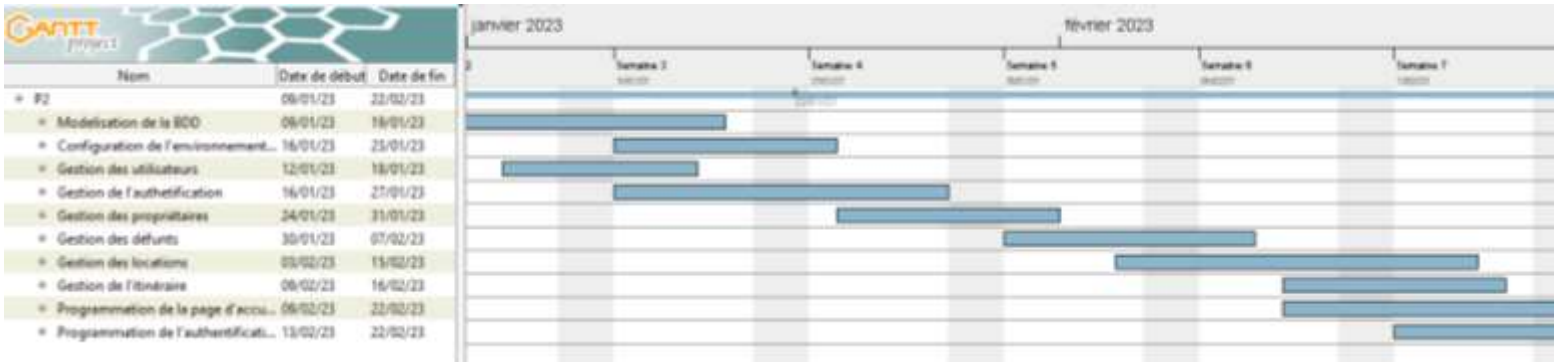
5. Conclusion

Ce cahier des charges définit les principales fonctionnalités de l'application de gestion des tombes d'un cimetière. Il est recommandé de suivre les meilleures pratiques en matière de développement web pour assurer la qualité, la sécurité et la convivialité de l'application.

PLANNING PRÉVISIONNEL

1. Diagramme de Gantt P1

2. Diagramme de Gantt P2



3. Diagramme de Gantt P3

ANALYSE ET CONCEPTION

1. Présentation de l'outil et de la méthode de modélisation

Looping est un outil puissant et polyvalent de modélisation de bases de données relationnelles. Il offre une interface conviviale et des fonctionnalités avancées pour concevoir et gérer des modèles de données.

La méthode de modélisation utilisée dans Looping repose sur deux concepts principaux : le Modèle Conceptuel de Données (MCD) et le Modèle Logique de Données Relationnelles (MLDR).

2. Modèle Conceptuel de Données (MCD) :

Le MCD est une représentation abstraite et conceptuelle des entités, de leurs attributs et des relations entre elles. Il permet de décrire les principales entités du système et leurs caractéristiques, indépendamment des considérations techniques de mise en œuvre. Dans Looping, le MCD est créé en utilisant des diagrammes entité-association (E-A) ou des notations similaires.

Dans l'exemple du projet de gestion des tombes d'un cimetière, le MCD définit les entités telles que "utilisateur", "type_tombeau", "statut_place", "zone", "duree", "tombeau", "adresse", "personne", "place", "beneficiaire", "defunt", "proprietaire", etc. Chaque entité est

associée à ses attributs, tels que le nom, la date de naissance, l'adresse, etc., et aux relations entre elles, comme les relations "possède", "bénéficie", "affecte", etc.

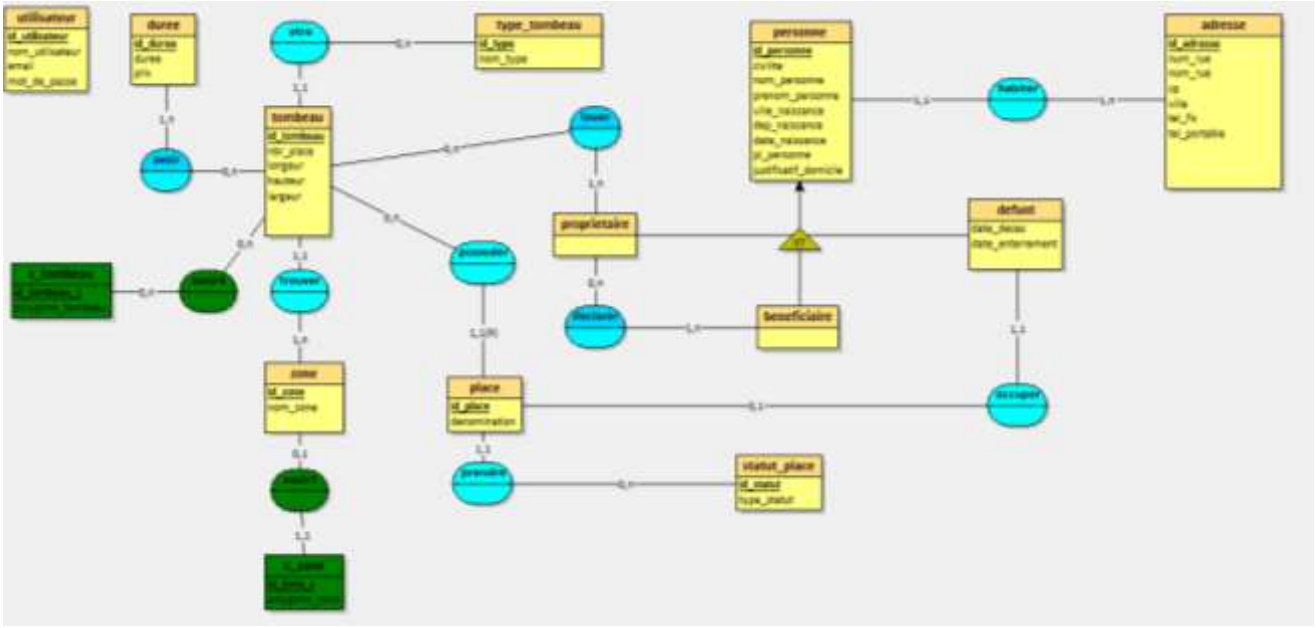


Figure 1 : Schéma du MCD de notre projet (Looping)

3. Modèle Logique de Données Relationnelles (MLDR) :

Le MLDR est une représentation plus concrète et technique du modèle de données, spécifique à la mise en œuvre d'une base de données relationnelle. Il traduit le MCD en tables et en relations entre les tables. Dans Looping, le MLDR est exprimé à l'aide d'un script SQL (Structured Query Language) qui définit la structure des tables et les contraintes d'intégrité.

Dans l'exemple fourni, le script SQL décrit les différentes tables (utilisateur, type_tombeau, statut_place, zone, duree, tombeau, adresse, personne, place, beneficiaire, defunt, proprietaire, avoir, louer, declarer, avoir4) et leurs colonnes respectives. Les contraintes de clé primaire, de clé étrangère et d'unicité sont également définies pour maintenir l'intégrité des données.

L'utilisation de l'outil Looping facilite la création et la gestion des modèles de données en fournissant une interface visuelle pour concevoir le MCD et en générant automatiquement le script SQL correspondant, comme illustré dans l'exemple fourni. Ainsi, les développeurs peuvent rapidement transformer le modèle conceptuel en une structure logique de base de données relationnelle prête à être implémentée dans un SGDB.

En résumé, l'outil Looping associé à la méthode de modélisation MCD et MLDR offre une approche pratique et efficace pour concevoir et implémenter des bases de données relationnelles, en transformant les concepts abstraits du MCD en une structure logique exploitable grâce au script SQL généré.

RÉALISATION ET MISE EN ŒUVRE

1. Conception de la charte graphique :

Pour commencer, le binôme a utilisé l'outil de design **Canva** pour créer une charte graphique, définissant ainsi un premier aspect visuel et les couleurs de l'interface de l'application. Tout en sachant que ce serait l'aperçu provisoire, avant de passer à la partie programmation.

2. Répartition des tâches avec Trello :

Une fois la charte graphique établie, il a fallu se répartir les tâches en fonction des compétences et des intérêts de chacun. L'équipe a donc utilisé l'outil de gestion de projets Trello. Cela a permis de définir clairement les responsabilités de chacun, de créer des listes de tâches et de suivre leur progression tout au long du projet.

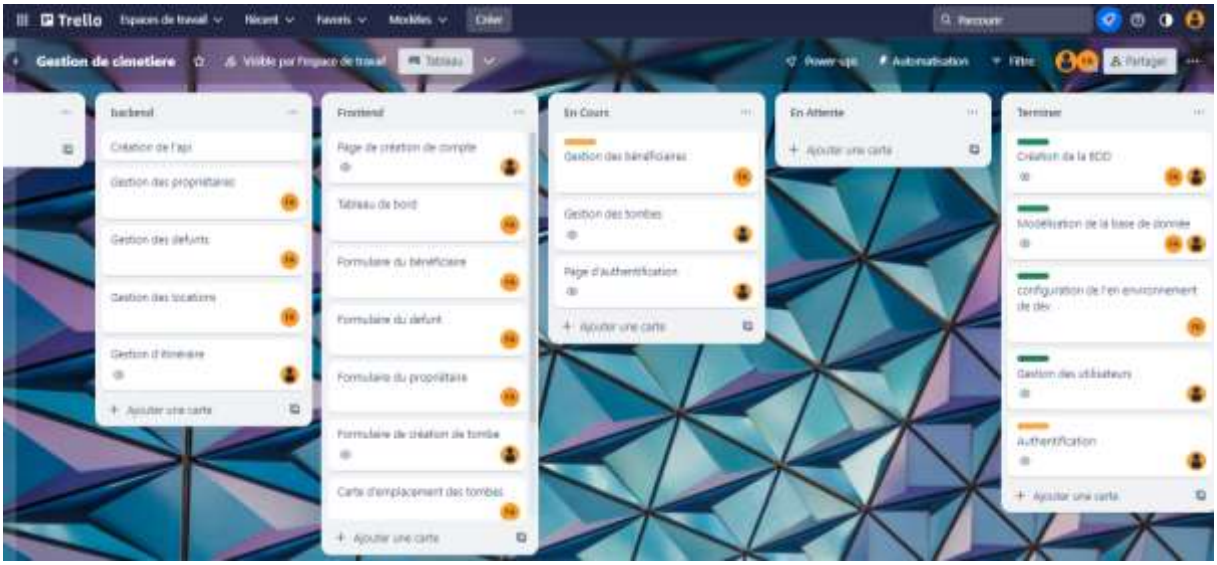


Figure 2 : Répartition des tâches entre le binôme sur la page du Trello

3. Apprentissage des langages de programmation :

Étant donné que les membres du binôme avaient peu d'expérience dans les langages de programmation requis, ils ont consacré un certain nombre d'heures à l'apprentissage de ces

langages et des outils, notamment JavaScript, PHP, Bootstrap, et SQL pour la modélisation de la base de données avec PHP My Admin.

4. Configuration de l'environnement de développement :

Le binôme a choisi l'éditeur de texte **Visual Studio Code** comme environnement de développement, en le configurant pour qu'il soit adapté au projet. Cela inclut l'installation d'extensions et de plugins pertinents pour faciliter la programmation et améliorer la productivité.

5. Collaboration avec GitHub :

Pour faciliter la collaboration et le partage du code source, l'équipe a utilisé le service web GitHub. Chaque membre pouvait ainsi partager ses modifications de code en créant des branches, proposer des modifications via des pull requests et fusionner les changements une fois validés.

6. Programmation des fonctionnalités principales :

À partir de là, l'équipe a commencé à programmer les fonctionnalités principales de l'application. Cela comprend la gestion des utilisateurs, l'authentification, la gestion des propriétaires de tombes, des défunts, des locations, ainsi que l'affichage des pages d'accueil et d'authentification.

7. Utilisation de Bootstrap pour l'interface web :

Après avoir réalisé la charte graphique avec Canva, l'équipe a décidé d'utiliser le framework **Bootstrap** pour le développement de l'interface web. Bootstrap a facilité la création de composants réactifs et esthétiquement agréables, en permettant de créer une interface cohérente et conviviale pour les utilisateurs de l'application.

8. Création de la base de données :

Pour créer et administrer la base de données, il était préférable, dans un premier temps d'utiliser un serveur local grâce au logiciel **Xampp Server** avec PHP My Admin et My SQL, plutôt que de travailler sur PG Admin avec les ordinateurs de l'IUT. Ceci afin d'éviter d'avoir des problèmes de serveurs et pour travailler facilement en dehors de l'établissement sur les ordinateurs personnels.

Tout au long de ces étapes, l'équipe a fait preuve d'adaptabilité et d'apprentissage continu, en surmontant les défis liés à la maîtrise et aux requis de la formation concernant les langages de programmation utilisés. Ainsi qu'en s'appuyant sur des ressources en ligne, des tutoriels et des forums pour résoudre les problèmes rencontrés.

Grâce à ces efforts, l'application de gestion des tombes du cimetière a été développée et mise en œuvre, offrant des fonctionnalités clés telles que la création, la modification et la suppression des données des propriétaires, des défunts, des locations, ainsi que l'authentification sécurisée des utilisateurs.

PRESENTATION ET EXPLOITATION DE L'APPLICATION

1. Backend avec Node.js et transition vers PHP :

L'application a été initialement développée en utilisant le langage Node.js pour le backend. Cependant, après évaluation et prise en compte des besoins spécifiques du projet par rapport aux attentes de la formation, une décision a été prise de passer à **PHP**, offrant une plus grande flexibilité et une meilleure correspondance avec les technologies déjà utilisées.

2. Acquisition des données SIG du cimetière :

Les données nécessaires à l'application ont été obtenues grâce à l'un des membres de l'équipe qui travaille à l'agglomération de Bourg en Bresse. Ce membre disposait des jeux de données **SHP** du cimetière de la ville, qui ont été utilisés pour alimenter notre application. Les données géographiques sur l'emplacement des tombes étaient particulièrement importantes pour la fonctionnalité de visualisation de la carte.

3. Conversion des données en format GeoJSON :

- Les données SIG du cimetière ont été traitées et converties en format **GeoJSON**, qui est un format standard pour représenter des données géospatiales. Cette conversion a été réalisée à l'aide d'outils de **géotraitement** sur ArcGIS Pro et QGIS, ainsi que des traitements supplémentaires effectués dans Excel. Les coordonnées géographiques de chaque tombe ont donc été générées puis intégrées au format GeoJSON en WGS 84.

4. Utilisation de Leaflet pour la création de la carte :

- L'outil de cartographie **Leaflet** a été utilisé pour créer la fonctionnalité de visualisation de la carte dans l'application. Grâce aux données GeoJSON obtenues et converties, l'équipe a pu afficher les emplacements des tombes sur une carte interactive, offrant aux utilisateurs une

visualisation claire et pratique. L'objectif était de pouvoir visualiser chaque tombe identifiée par un numéro, une zone

L'application développée présente donc une transition du backend de Node.js vers PHP, offrant une plus grande flexibilité. L'interface utilisateur a été conçue à l'aide de **Bootstrap**, garantissant un design réactif et esthétiquement agréable. Les données nécessaires à l'application ont été fournies par un membre de l'équipe ayant accès aux jeux de données SIG du cimetière de Bourg en Bresse. Ces données ont ensuite été converties en format GeoJSON à l'aide d'ArcGIS Pro, QGIS et Excel. Enfin, la fonctionnalité de visualisation de la carte a été mise en œuvre avec l'outil Leaflet, permettant aux utilisateurs de visualiser les emplacements des tombes de manière interactive.

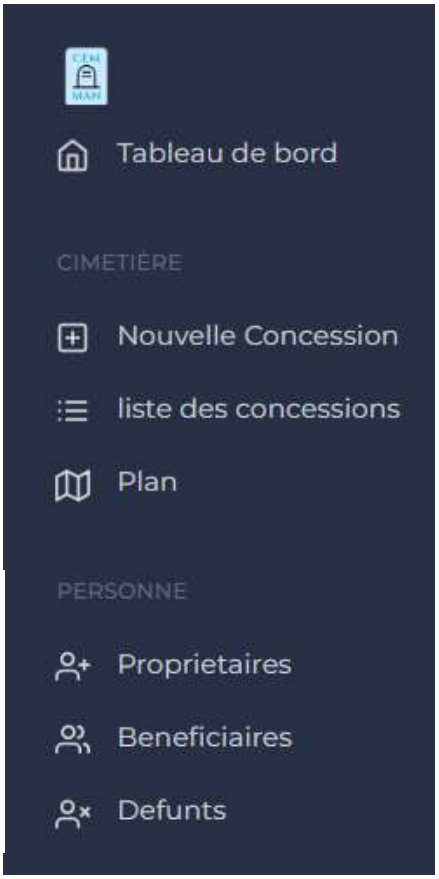


Figure 3 : Affichage de la barre de menu sur le site WEB, avec les icônes générées par Bootstrap

INTERFACE CARTOGRAPHIQUE

1. Création de la carte à partir des données SIG en GeoJSON :

La partie cartographique de l'application a permis la visualisation des **emplacements des tombes** à l'aide des données GeoJSON stockées dans notre base de données MySQL sur PG Admin. Nous avons utilisé ces coordonnées géographiques pour afficher les tombes sur une carte interactive. Pour cela, nous avons utilisé la bibliothèque JavaScript Leaflet, connue pour sa facilité d'utilisation et ses fonctionnalités de cartographie avancées. En utilisant **Leaflet**, nous avons pu intégrer facilement la carte à notre application et marquer les emplacements des tombes avec des marqueurs appropriés.

2. Intégration de la carte dans l'application :

Pour lier la carte à notre application, nous avons établi une connexion entre notre application et la base de données. Nous avons utilisé des requêtes SQL pour extraire les données géographiques des tombes de la base de données et les avons ensuite affichées sur la carte à l'aide de Leaflet. Cette intégration permet aux utilisateurs de visualiser les emplacements des tombes de manière interactive et conviviale, en leur offrant une représentation visuelle claire de la disposition des tombes dans le cimetière.

Figure 4 : Affichage de la carte des emplacements avec les tombes du fichier geoJSON provenant de la base de données (carte.php)

3. Utilisation de l'API d'itinéraire :

En plus de la visualisation des tombes sur la carte, nous avons intégré une fonctionnalité d'itinéraire pour faciliter la navigation des utilisateurs vers une tombe spécifique. Pour cela, nous avons utilisé une API de cartographie qui nous a permis de générer des itinéraires en fonction de l'adresse saisie par l'utilisateur. Lorsqu'un utilisateur sélectionne une tombe, notre application utilise cette API pour calculer et afficher l'itinéraire depuis l'adresse de l'utilisateur jusqu'à la tombe choisie. Cela permet aux utilisateurs de planifier facilement leur visite et de suivre un itinéraire précis pour se rendre à la tombe souhaitée.

4. Impact de la partie cartographique :

L'intégration de la partie cartographique a considérablement amélioré l'expérience utilisateur de notre application de gestion des tombes du cimetière. En fournissant une visualisation claire et interactive des emplacements des tombes sur une carte, l'utilisateur peut facilement naviguer et localiser les tombes qui l'intéresse. De plus, la fonctionnalité d'itinéraire permet aux utilisateurs de planifier leurs déplacements et de se rendre directement aux tombes choisies en suivant des indications précises. Cela facilite la gestion des visites au cimetière et offre aux utilisateurs une expérience pratique et conviviale.

En résumé, la partie cartographique de notre application, réalisée avec Leaflet et l'utilisation de données GeoJSON, a permis la visualisation des emplacements des tombes sur une carte interactive. L'intégration de l'API d'itinéraire a renforcé la fonctionnalité de l'application en offrant aux utilisateurs la possibilité de planifier et de suivre des itinéraires précis vers les tombes choisies. Cette partie a considérablement amélioré l'expérience utilisateur et facilité la gestion des visites au cimetière.

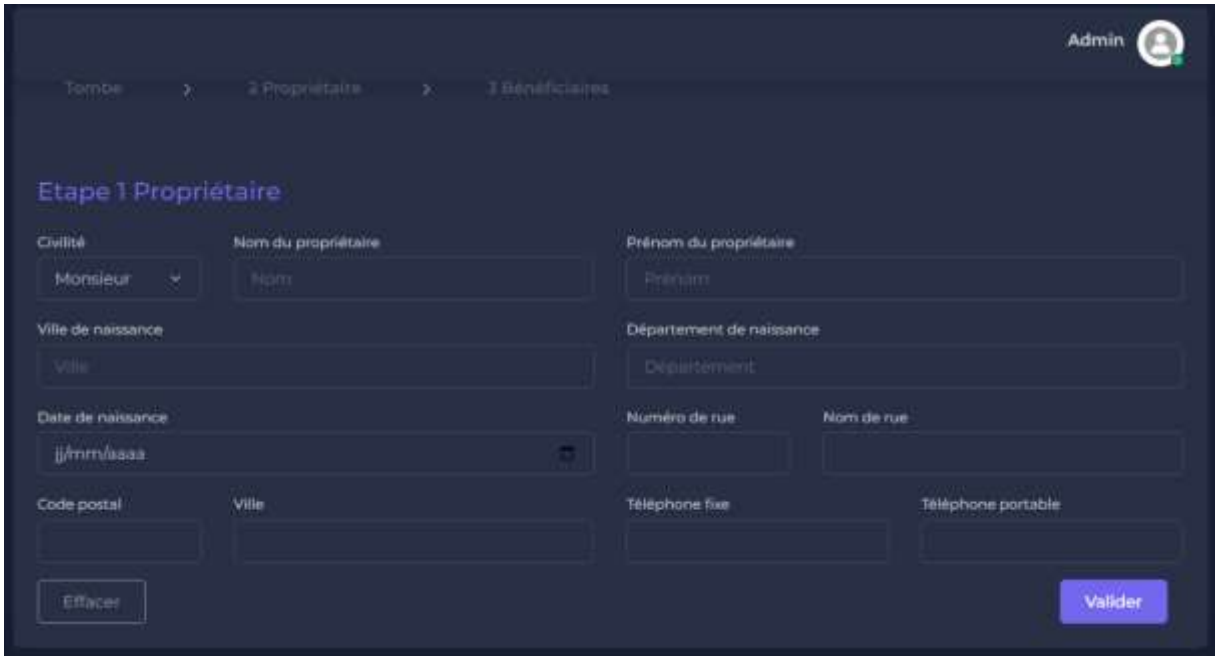


Figure 5 : Affichage du formulaire de création de propriétaire (ConcessionAdd.php)

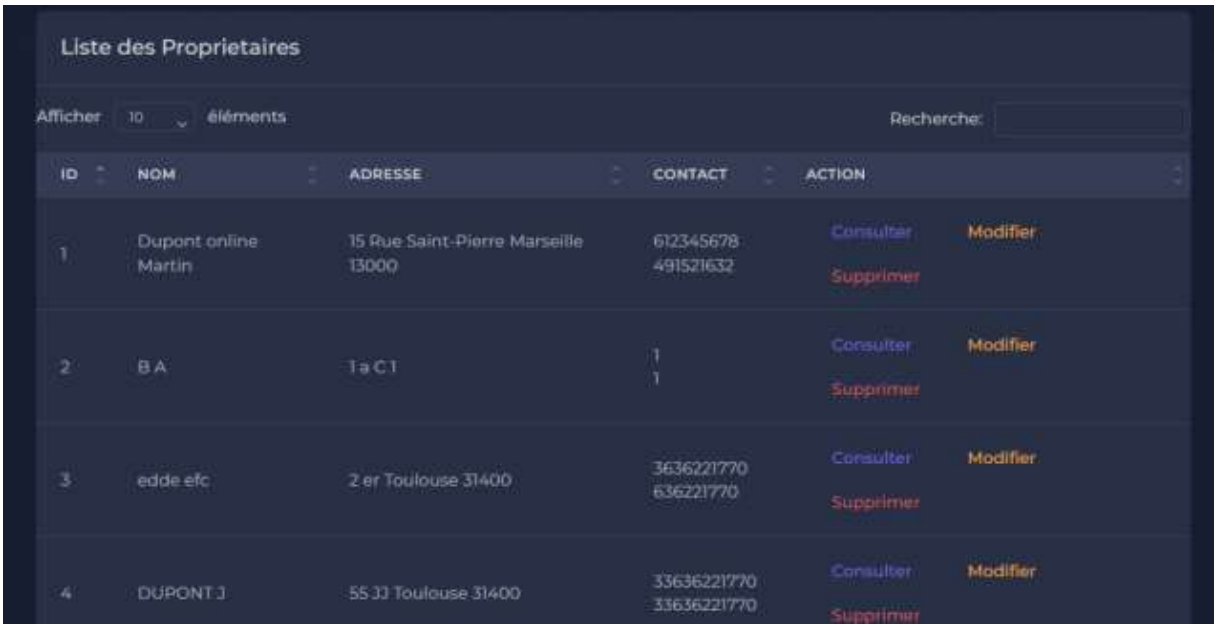


Figure 6 : Affichage de la liste des propriétaires pour l'utilisateur (liée à la BDD)

ARCHITECTURE MATERIELLE ET LOGICIELLE DE L'APPLICATION

1. Architecture matérielle

L'architecture en informatique, désigne la structure générale (inhérente à un système informatique), l'organisation des différents éléments du système (logiciels, matériels, humains, informations) et les relations entre ceux-ci. Il existe quatre (4) types d'architecture : architectures un-tiers, deux–tiers, trois–tiers et n–tiers (n>3). La mise en œuvre de notre système a nécessité une architecture de type « trois- tiers (3-tiers) ». L'architecture 3-tiers est une architecture basée sur l'environnement client-serveur qui vise à modéliser une application comme un empilement de trois couches logicielles. La séparation entre ces trois couches d'une même application ou système aide à avoir une idée globale en ce qui concerne la décomposition du système. Cette architecture nous permet donc de subdiviser l'application en trois couches :

• Couche de présentation (Client) Cette couche, représentée en HTML et exploitée par le navigateur dans notre cas de figure, est la partie visible pour l'utilisateur. Elle prend en charge tout ce qui concerne l'interface graphique et l'interaction homme machine. La couche de présentation relaie les requêtes de l'utilisateur à destination de la couche métier et en retour, elle présente les informations renvoyées par cette dernière. • Couche métier (Serveur de traitements) C'est le cœur de l'application. Elle correspond à la partie fonctionnelle de l'application et décrit les opérations que l'application opère sur les données en fonction des requêtes des utilisateurs effectués au travers de la couche de présentation. C'est elle qui communique avec la couche d'accès aux données. Elle implémente la logique métier de l'application.

• Couche d'accès aux données (Serveur de données) Elle gère la permanence des données et a pour but de conserver une quantité plus ou moins importante de données de façon structurée. Elle fournit au serveur d'application les données dont il a besoin pour mettre en œuvre sa logique métier

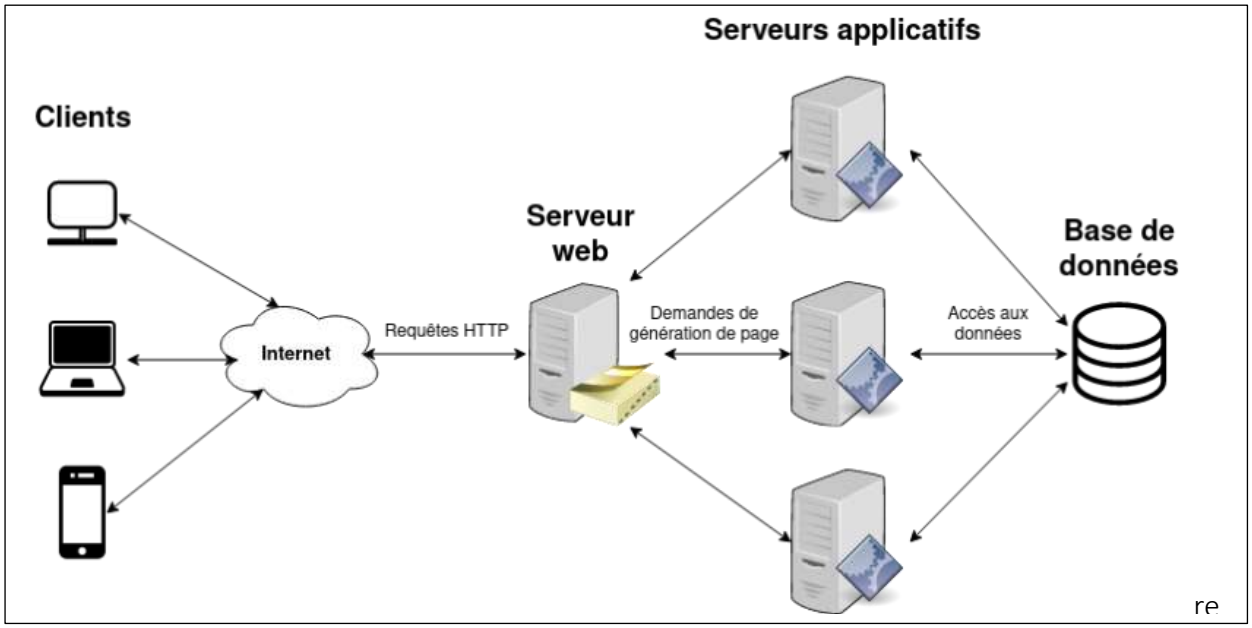


Figure 7 : Schéma de l'architecture matérielle de l'application

2. Architecture logicielle

L'architecture logicielle que nous avons utilisée pour la mise en œuvre de notre projet est l'architecture Modèle-Vue-Contrôleur (MVC). Dans cette architecture chacun des composants joue un rôle spécifique : ⎪ Le modèle est chargé d'interagir avec la base de données. ⎪ La vue est chargée de la mise en forme des pages affichée à l'utilisateur. ⎪ Le contrôleur est essentiellement chargé de lier la vue au modèle et gérer les processus métier.

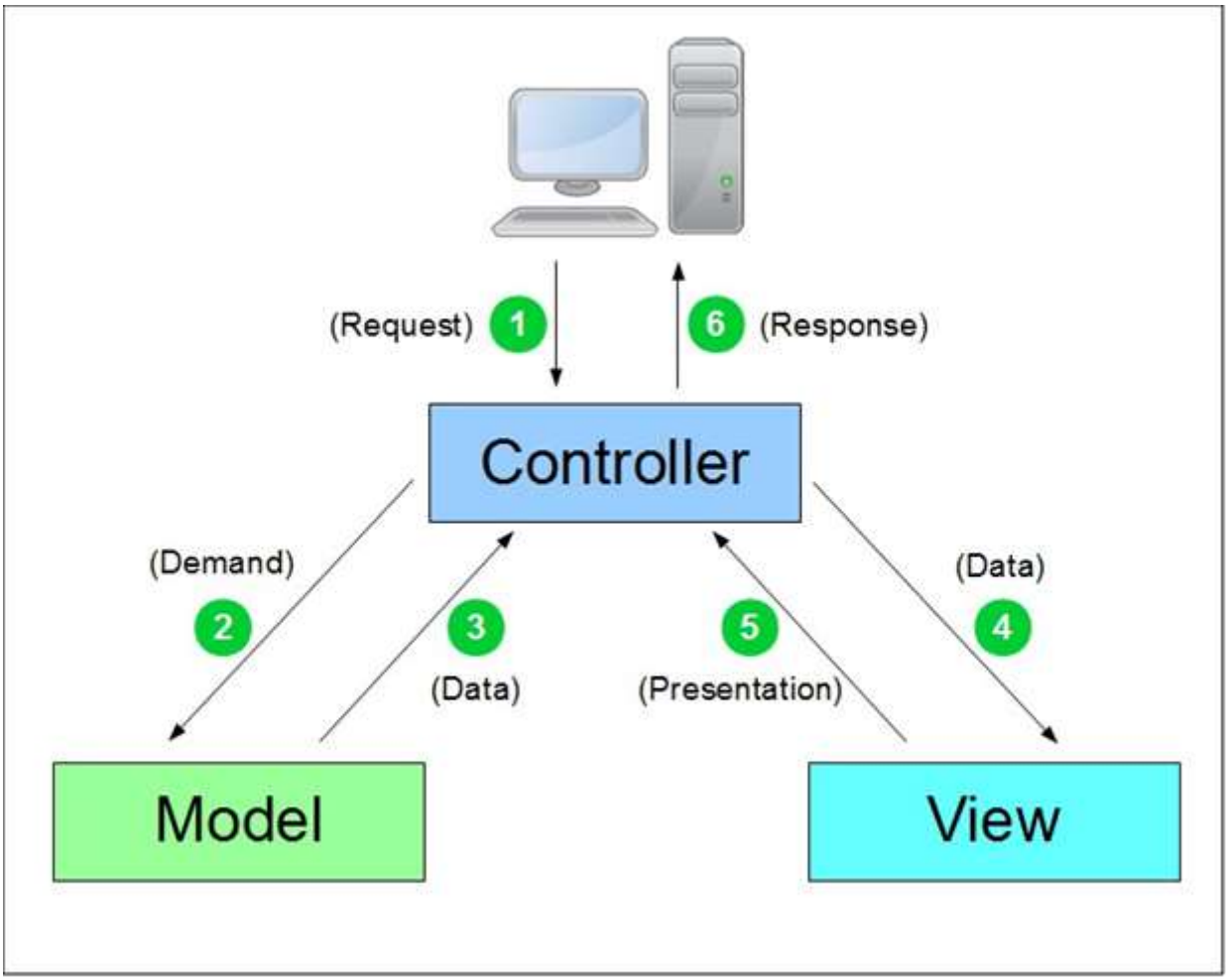


Figure 8 : Schéma de l'architecture logicielle de l'application

CODES SOURCES DE L'APPLICATION

Quelques fonctions, authentification, création d'utilisateur, mise à jour du mot de passe et gestion du propriétaire (ajout, modification, suppression)

```php
function authenticate($email, $password) {
    $db = DB::getConnection();

    $stmt = $db->prepare("SELECT * FROM users WHERE email = :email");
    $stmt->execute(array(':email' => $email));
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    if ($user && password_verify($password, $user['password'])) {
        return $user;
    } else {
        return null;
```

```
    }
}

function create_user($nom_agent, $prenom_agent, $email, $password) {
    $db = DB::getConnection();

    $stmt = $db->prepare("INSERT INTO users (nom_agent, prenom_agent, email,
password, createdAt, updatedAt) VALUES (:nom_agent, :prenom_agent, :email,
:password, NOW(), NOW())");
    $stmt->execute(array(':nom_agent' => $nom_agent, ':prenom_agent' =>
$prenom_agent, ':email' => $email, ':password' => password_hash($password,
PASSWORD_DEFAULT)));

    return $db->lastInsertId();
}


function update_password_user($email, $password) {
    $db = DB::getConnection();

    $stmt = $db->prepare("UPDATE users SET password = :password, updatedAt =
NOW() WHERE email = :email");
    $stmt->execute(array(':email' => $email, ':password' => pass-
word_hash($password, PASSWORD_DEFAULT)));

    return $stmt->rowCount() > 0;
}


    //Gestion du proprietaire:

    //Création du proprietaire
function create_proprietaire($civilite, $nom_personne, $prenom_personne,
$ville_naissance, $dep_naissance, $date_naissance,$num_rue, $nom_rue, $cp,
$ville, $tel_fix, $tel_portable) {
    $db = DB::getConnection();

    // Validation des entrées
    if (empty($civilite) || empty($nom_personne) || empty($prenom_personne) ||
empty($ville_naissance) || empty($dep_naissance) || empty($date_naissance) ||
empty($num_rue) || empty($nom_rue) || empty($cp) || empty($ville)  ||
empty($tel_portable)) {
        throw new InvalidArgumentException('Tous les champs sont obliga-
toires');
    }

    // Insertion dans la table adresse
```

```
    $stmt = $db->prepare("INSERT INTO adresse (num_rue, nom_rue, cp, ville,
tel_fix, tel_portable) VALUES (:num_rue, :nom_rue, :cp, :ville, :tel_fix,
:tel_portable)");
    $stmt->execute(array(':num_rue' => $num_rue, ':nom_rue' => $nom_rue, ':cp'
=> $cp, ':ville' => $ville, ':tel_fix' => $tel_fix, ':tel_portable' =>
$tel_portable));

    $id_adresse = $db->lastInsertId();

    // Insertion dans la table personne
    $stmt = $db->prepare("INSERT INTO personne (civilite, nom_personne, pre-
nom_personne, ville_naissance, dep_naissance, date_naissance,  id_adresse) VA-
LUES (:civilite, :nom_personne, :prenom_personne, :ville_naissance, :dep_nais-
sance, :date_naissance, :id_adresse)");
    $stmt->execute(array(':civilite' => $civilite, ':nom_personne' =>
$nom_personne, ':prenom_personne' => $prenom_personne, ':ville_naissance' =>
$ville_naissance, ':dep_naissance' => $dep_naissance, ':date_naissance' =>
$date_naissance, ':id_adresse' => $id_adresse));

    $id_personne = $db->lastInsertId();

    // Insertion dans la table proprietaire
    $stmt = $db->prepare("INSERT INTO proprietaire (id_personne) VALUES
(:id_personne)");
    $stmt->execute(array(':id_personne' => $id_personne));

    $id_proprietaire = $db->lastInsertId();

    return $id_proprietaire;
}


    //Modification du proprietaire
function update_proprietaire($id_proprietaire, $civilite, $nom_personne, $pre-
nom_personne, $ville_naissance, $dep_naissance, $date_naissance, $num_rue,
$nom_rue, $cp, $ville, $tel_fix, $tel_portable) {
    $db = DB::getConnection();

    // Validation des entrées
    if (empty($civilite) || empty($nom_personne) || empty($prenom_personne) ||
empty($ville_naissance) || empty($dep_naissance) || empty($date_naissance)
||  empty($num_rue) || empty($nom_rue) || empty($cp) || empty($ville) ||
empty($tel_fix) || empty($tel_portable)) {
        throw new InvalidArgumentException('Tous les champs sont obliga-
toires');
    }

    // Mise à jour de la table adresse
```

```
    $stmt = $db->prepare("UPDATE adresse, personne, proprietaire SET num_rue =
:num_rue, nom_rue = :nom_rue, cp = :cp, ville = :ville, tel_fix = :tel_fix,
tel_portable = :tel_portable WHERE proprietaire.id_proprietaire = :id_proprie-
taire AND personne.id_personne = proprietaire.id_personne AND
adresse.id_adresse = personne.id_adresse");
    $stmt->execute(array(':num_rue' => $num_rue, ':nom_rue' => $nom_rue, ':cp'
=> $cp, ':ville' => $ville, ':tel_fix' => $tel_fix, ':tel_portable' =>
$tel_portable, ':id_proprietaire' => $id_proprietaire));

    // Mise à jour de la table personne
    $stmt = $db->prepare("UPDATE personne, proprietaire SET civilite = :civi-
lite, nom_personne = :nom_personne, prenom_personne = :prenom_personne,
ville_naissance = :ville_naissance, dep_naissance = :dep_naissance, date_nais-
sance = :date_naissance WHERE proprietaire.id_proprietaire = :id_proprietaire
AND personne.id_personne = proprietaire.id_personne");
    $stmt->execute(array(':civilite' => $civilite, ':nom_personne' =>
$nom_personne, ':prenom_personne' => $prenom_personne, ':ville_naissance' =>
$ville_naissance, ':dep_naissance' => $dep_naissance, ':date_naissance' =>
$date_naissance, ':id_proprietaire' => $id_proprietaire));

    return true;
}


    //Suppression d'un proprietaire
function delete_proprietaire($id_proprietaire) {
    $db = DB::getConnection();

    // Suppression de l'enregistrement proprietaire
    $stmt = $db->prepare("DELETE proprietaire, personne, adresse FROM proprie-
taire JOIN personne ON proprietaire.id_personne = personne.id_personne JOIN
adresse ON personne.id_adresse = adresse.id_adresse WHERE proprietaire.id_pro-
prietaire = :id_proprietaire");
    $stmt->execute(array(':id_proprietaire' => $id_proprietaire));

    return true;
}

    //liste de tous les propritaires
function get_all_proprietaires() {
    $db = DB::getConnection();

    // Requête pour obtenir tous les propriétaires
    $stmt = $db->prepare("SELECT proprietaire.id_proprietaire, personne.*,
adresse.* FROM proprietaire JOIN personne ON proprietaire.id_personne = per-
sonne.id_personne JOIN adresse ON personne.id_adresse = adresse.id_adresse");
    $stmt->execute();

    // Récupération de tous les propriétaires
```

```
    $proprietaires = $stmt->fetchAll(PDO::FETCH_ASSOC);

    return $proprietaires;
}

    //avoir un proprietaire par son id
function get_proprietaire_by_id($id_proprietaire) {
    $db = DB::getConnection();

    // Requête pour obtenir un propriétaire spécifique
    $stmt = $db->prepare("SELECT proprietaire.id_proprietaire, personne.*,
adresse.* FROM proprietaire JOIN personne ON proprietaire.id_personne = per-
sonne.id_personne JOIN adresse ON personne.id_adresse = adresse.id_adresse
WHERE proprietaire.id_proprietaire = :id_proprietaire");
    $stmt->execute(array(':id_proprietaire' => $id_proprietaire));

    // Récupération des informations du propriétaire    $proprietaire = $stmt-
>fetch(PDO::FETCH_ASSOC);

    return $proprietaire;
}
```

CONCLUSION

L'élaboration de notre application web de gestion des tombes d'un cimetière a été un processus rigoureux, nécessitant une planification minutieuse et l'application de différentes compétences techniques. En combinant notre expertise en développement web, l'utilisation d'outils spécialisés et la manipulation de données géospatiales, nous avons pu créer une application fonctionnelle et conviviale.

Nous avons commencé par définir les fonctionnalités essentielles de l'application, en accordant une attention particulière à la gestion des propriétaires, des défunts, des locations et à la visualisation des emplacements des tombes. La conception d'une charte graphique et l'utilisation de l'outil Canva nous ont permis de créer une interface esthétiquement attrayante, tandis que l'adoption du Bootstrap a facilité le développement de l'interface utilisateur réactive et cohérente.

La répartition des tâches via l'outil Trello nous a aidés à organiser le travail et à optimiser notre productivité. En parallèle, l'apprentissage des langages de programmation et l'utilisation de l'éditeur de texte Visual Studio Code ont renforcé notre compréhension des technologies utilisées et nous ont permis de coder les fonctionnalités principales de l'application.

Grâce à l'intégration des données géospatiales fournies de l'agglomération de Bourg en Bresse, nous avons pu manipuler et convertir ces données en format GeoJSON, essentiel pour la création de la carte interactive. L'utilisation de Leaflet, conjointement avec notre base de

données MySQL et PG Admin, a permis d'afficher les emplacements des tombes sur la carte et d'offrir aux utilisateurs une expérience de visualisation précise et conviviale.

Nous avons également incorporé une fonctionnalité d'itinéraire grâce à l'utilisation d'une API de cartographie. Cette fonctionnalité permet aux utilisateurs de planifier et de suivre des itinéraires précis depuis leur adresse actuelle jusqu'à la tombe choisie, améliorant ainsi leur expérience et leur facilitant la visite du cimetière.

En résumé, notre application web de gestion des tombes d'un cimetière a été développée en suivant une méthodologie structurée et en utilisant des outils appropriés. La combinaison des compétences en développement web, la gestion des données géospatiales, l'utilisation de Leaflet et l'intégration d'une API d'itinéraire ont abouti à une application fonctionnelle, offrant une expérience utilisateur conviviale et une gestion optimisée des emplacements des tombes.

Ce projet nous a permis de mettre en pratique nos connaissances techniques, d'apprendre de nouvelles compétences et de relever les défis inhérents au développement d'une application web complète. Nous sommes fiers du résultat obtenu et convaincus que notre application peut apporter une réelle valeur ajoutée dans la gestion des tombes d'un cimetière.

Dans une perspective future, des améliorations supplémentaires pourraient être apportées à l'application, telles que l'ajout de fonctionnalités supplémentaires, l'optimisation des performances et l'élargissement de l'accès aux données géospatiales provenant de différentes sources.

En conclusion, notre projet d'application web de gestion des tombes d'un cimetière a été une expérience enrichissante, démontrant notre capacité à développer une application fonctionnelle répondant aux besoins spécifiques du domaine funéraire.

Figure 9 : Icône et logo du site WEB, crée avec Canva

TABLE DES MATIERES