Compte Rendu Challenge

Introduction

Dans le cadre de cette semaine intensive de challenge, notre groupe s'est vu confier la réalisation d'un projet ambitieux combinant développement web et technologies immersives : la création d'un environnement de réalité virtuelle inspiré de l'univers visuel emblématique du film "Tron", accompagné d'un système de gestion complet via un back-office.

Ce projet multidimensionnel s'articule autour de plusieurs axes techniques majeurs. D'une part, le développement d'un back-office robuste et structuré, respectant rigoureusement l'architecture MVC (Modèle-Vue-Contrôleur) et implémenté en PHP, permettant une gestion administrative efficace de l'environnement virtuel. D'autre part, la conception d'une expérience immersive en réalité virtuelle, plongeant l'utilisateur dans l'esthétique futuriste et néon-cybernétique caractéristique de l'univers Tron. L'ensemble repose sur une base de données MySQL soigneusement conçue pour assurer la persistance et l'intégrité des données.

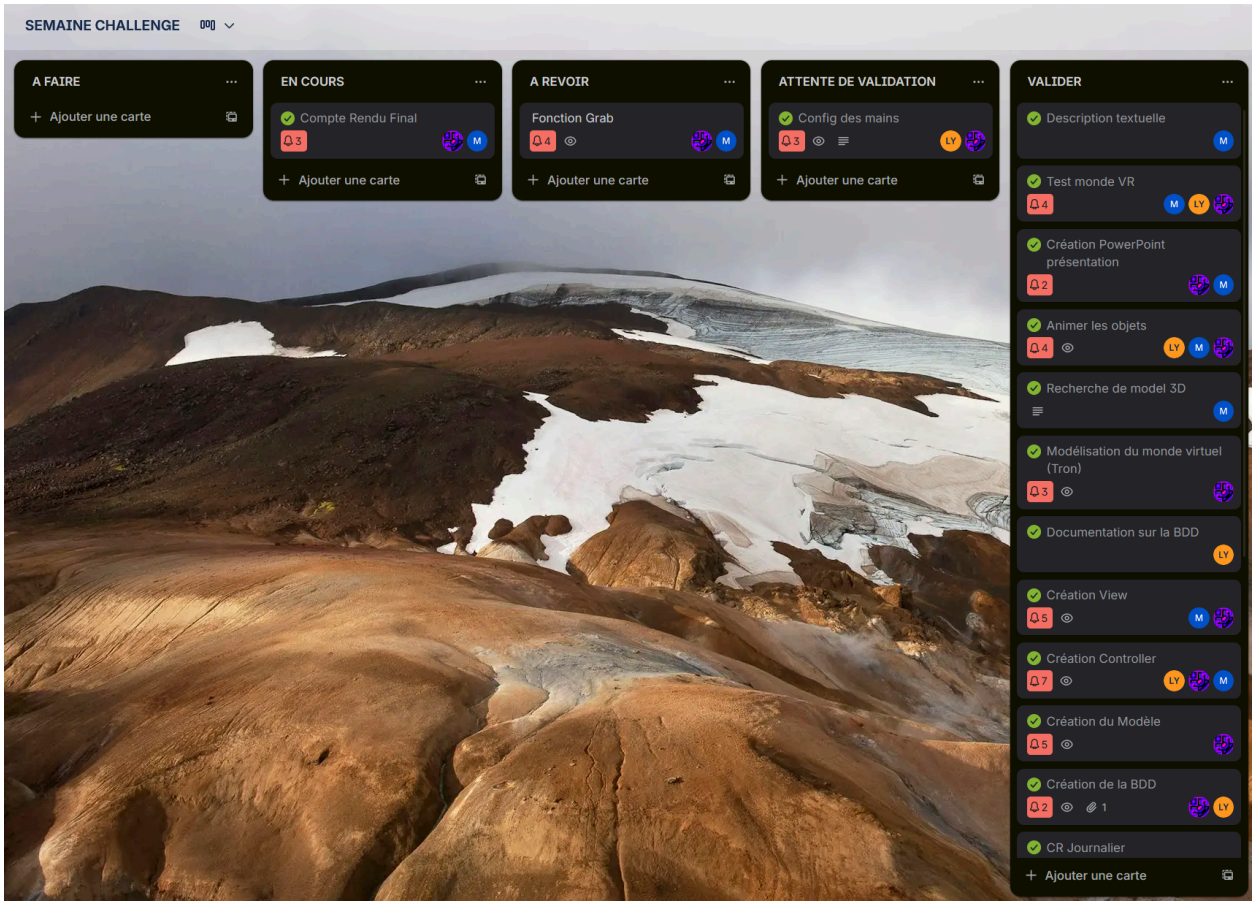
Au-delà de la simple réalisation technique, ce challenge nous a permis d'aborder des problématiques essentielles du développement logiciel professionnel : la modélisation UML et la conception préalable, la sécurisation des accès et des données sensibles, la mise en place de procédures de sauvegarde et de restauration, ainsi que le déploiement d'environnements de secours garantissant la continuité de service. La gestion de projet agile, matérialisée par un suivi quotidien via Trello et des reportings réguliers, a également constitué un aspect fondamental de notre démarche collaborative.

Sommaire

Organisation du groupe

Yenis a été désigné chef de projet pour la semaine, nous nous sommes répartis les tâches dès le début afin d'optimiser au mieux le temps mis à disposition pour le challenge et savoir à qui se référer en cas de besoin sur une tâche spécifique. Nous avons essentiellement utilisé les outils Trello, Github et Google Drive pour le partage de documents mais également leur sauvegarde ainsi que pour le suivi des tâches.

Tableau Trello à la fin de la semaine



Répartition des tâches à travers un tableau Excel

		coéquipier 1	Yenis							coéquipier 2	Marco									
		coéquipier 3	Marvin							coéquipier 4	Bob									
taches	resp.	lundi				mardi				mercredi				jeudi				vendredi		
dossier de conception																				
> diagramme UC	Marco																			
> description textuelle	Marco																			
> diagramme activité																				
> diagramme séquences	Marvin																			
> diagramme de classe	Yenis																			
> schema infra solution	Yenis																			
> ...																				
BACK OFFICE																				
> codage modele	Marvin																			
> codage controleur	Yenis																			
> codage vue	Marco																			
> rédaction documentation BO	Marco																			
> exercice de restauraton BO	Yenis																			
> deploiement envir. secours	Marco																			
> sécurisation des mdp en BDD	Marvin																			
> ...																				
Environnement de réalité virtuelle																				
> formation a-frame	Groupe																			
> raccordement casque VR au WIFI	Groupe + Partage connexion																			
> parametrage SSID WIFI	-																			
> sécurisation accès à l'envir virtuel	Marvin																			
> rédaction documentation VR	Yenis																			
> exercice de restauraton VR	Marco																			
> déploiement envir. secours	Marco																			
> ...																				
tâchtes transverses																				
> rédaction procédure de sauvegarde	Marco																			
> chef de projet	Yenis / Marvin																			
> création de la base de données	Groupe																			
> rédaction documentation BDD	Yenis																			
> sauvegarde hebdo	Marco																			
> reporting quotidien	Marco																			
> tenue Trello	Yenis / Marco																			
> support présentation soutenance	Marvin																			
> ...																				

Partie 1 - Dossier de conception

Description textuelle du Back-Office et de l’Architecture Système

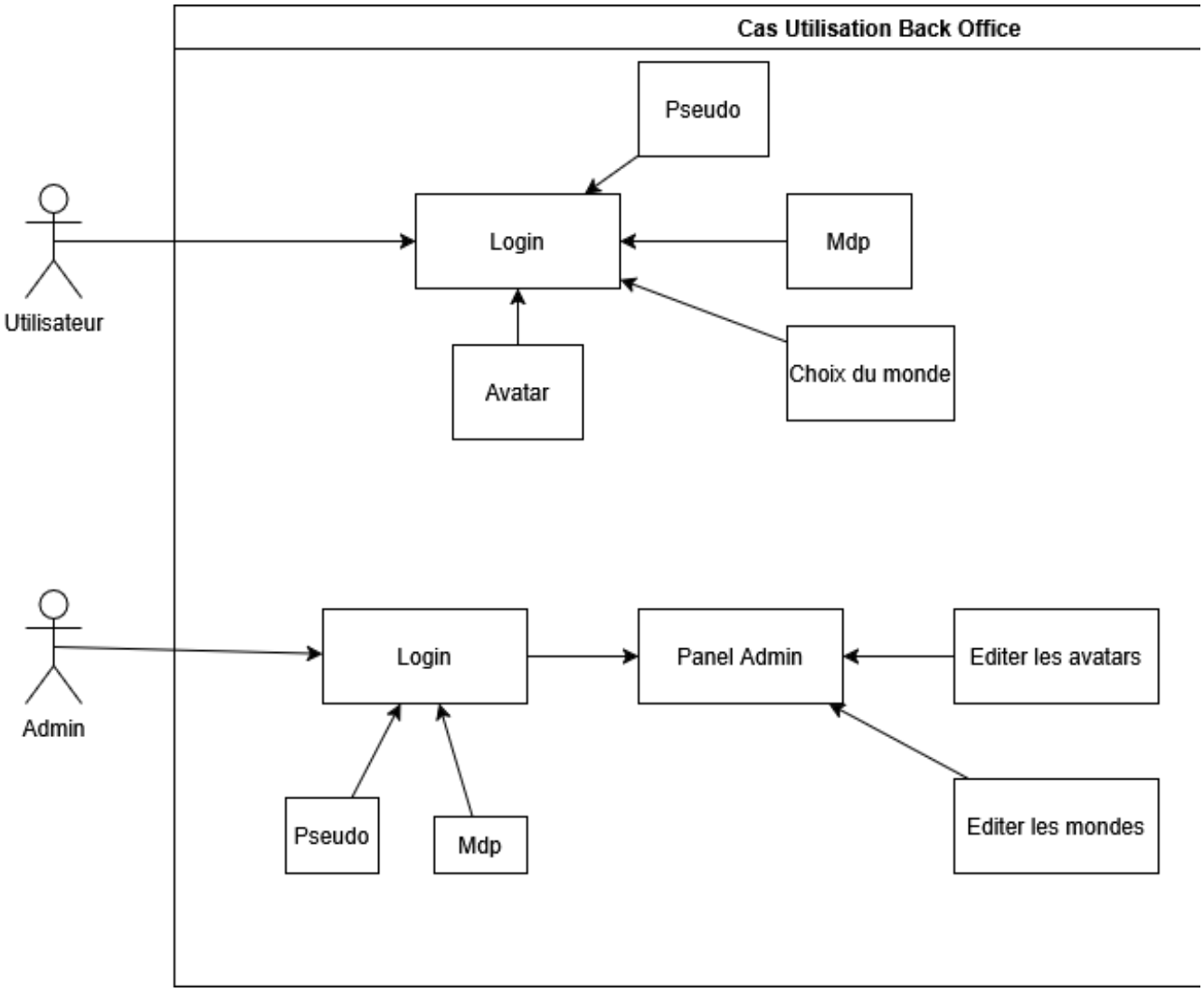
Le back-office constitue le point d’entrée central de l’écosystème, agissant comme une interface de gestion web permettant de configurer l'identité numérique de l'utilisateur avant toute immersion. Sa fonction première est d'orchestrer le parcours utilisateur en amont de l'expérience VR, en centralisant la création d'avatars et le paramétrage des préférences. Cette approche permet de dissocier les tâches administratives (inscription, choix) de l'environnement virtuel, garantissant ainsi une accessibilité multiplateforme (PC, smartphone) sans nécessiter l'usage immédiat d'un casque de réalité virtuelle.

Au cœur de cette architecture, une base de données MySQL assure la cohérence globale du système. Elle stocke de manière structurée les identifiants uniques, les modèles d’avatars sélectionnés et les environnements de destination choisis par les utilisateurs. Ce référentiel commun permet une synchronisation bidirectionnelle : les modifications effectuées sur l'interface web sont instantanément inscrites en base de données, puis lues par le controlleur de la page VR lors de la connexion. Ce mécanisme de "routage intelligent" garantit que l'utilisateur est automatiquement propulsé dans le bon univers VR avec l'apparence visuelle qu'il a choisi.

La sécurité est intégrée dès la conception via un système d'authentification robuste. Les mots de passe ne sont jamais stockés en clair, mais transformés en empreintes cryptographiques grâce à des algorithmes de hachage. Ce protocole, couplé à une gestion de sessions sécurisées, assure que seuls les profils configurés dans le back-office peuvent accéder aux portails immersifs. Cette barrière de sécurité protège non seulement les données utilisateurs, mais préserve également l'intégrité des mondes virtuels contre les accès non autorisés.

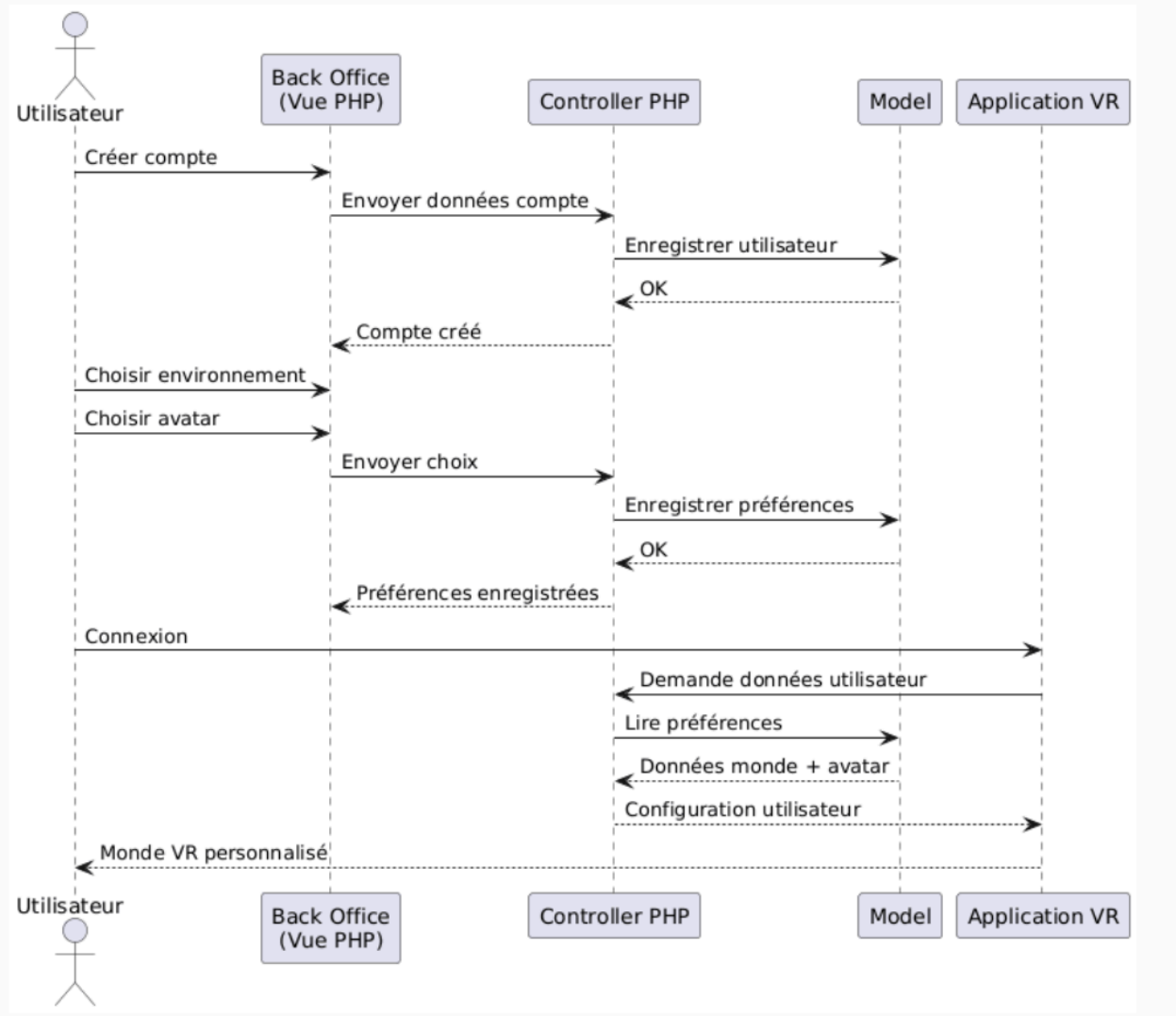
Enfin, cette architecture modulaire offre des avantages stratégiques en termes de performance et de scalabilité. En déchargeant les serveurs VR des processus de gestion de profil, le système préserve les ressources de calcul pour le rendu temps réel et la fluidité de l'expérience immersive. Cette séparation des responsabilités facilite également la maintenance et l'évolution du projet, permettant l'ajout de nouveaux mondes ou de nouvelles fonctionnalités de personnalisation sans impacter la stabilité des environnements déjà en production.

Diagramme de cas d'utilisation (back-office)



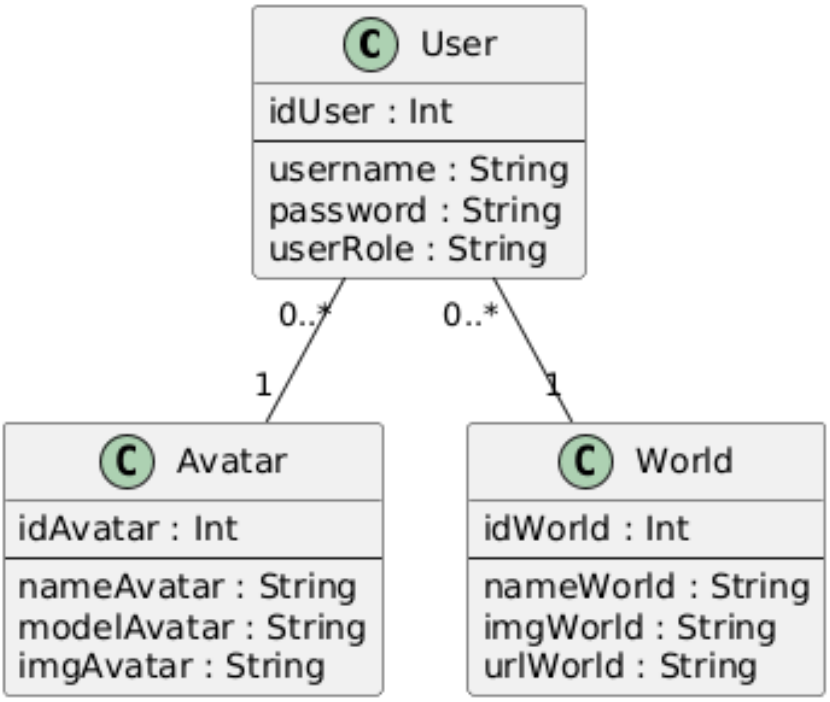
Avant la réalisation de ce cas d'utilisation il y a dû avoir une phase de réflexion pour la conception du schéma afin qu'il soit le plus fidèle possible au fonctionnement du Back Office. On y retrouve donc un utilisateur qui lors de son inscription choisit son avatar, son pseudo, son mot de passe et le monde VR dans lequel il souhaite aller. Il y a également l'admin, son login se compose d'un pseudo et mot de passe permettant d'accéder à un panel d'administration pour éditer les mondes et les avatars.

Diagramme de séquence



L'utilisateur crée un compte via le Back Office, qui envoie les données au contrôleur PHP. Le contrôleur enregistre l'utilisateur via le modèle et confirme la création du compte. Suite à ça, l'utilisateur choisit un environnement et un avatar, ces préférences sont transmises au contrôleur, puis enregistrées par le modèle. Lors de la connexion, l'application VR demande les données utilisateur au contrôleur qui lit les préférences dans le modèle et renvoie la configuration. L'utilisateur accède alors à un monde VR personnalisé.

Diagramme de classe



Ce diagramme de classes présente trois entités : User, Avatar et World. Un User possède des informations d'authentification et de rôle. Il est associé à un avatar et un monde, "Avatar" et "World" peuvent être partagés par plusieurs utilisateurs. La classe Avatar décrit les caractéristiques visuelles de l'avatar. La classe "World" décrit les données du monde virtuel.

Partie 2 - Back-office

Codage Modèle

1. Architecture Globale : Le Choix du MVC

Dans le cadre de ce projet, nous avons adopté l'architecture MVC (Modèle-Vue-Contrôleur). La couche "Modèle", que nous présentons ici, constitue le socle de l'application. Elle a pour rôle exclusif de gérer les données et d'interagir avec la base de données SQL.

En isolant cette logique, nous nous assurons que le reste de l'application (l'affichage ou la logique métier) n'a jamais besoin de connaître les détails des requêtes SQL. Cela rend le code plus propre, plus professionnel et surtout beaucoup plus facile à maintenir sur le long terme.

2. L'Initialisation et la Connexion via PDO

Chaque classe de modèle (UserModel, AvatarModel, WorldModel) commence par une propriété privée $db de type PDO. L'initialisation se fait dans le constructeur via un appel à Database::get().

Méthode de création de PDO:

```php
private static ?PDO $pdo = null;

3 references | 0 overrides
public static function get(): PDO
{
    if (self::$pdo === null) {
        self::$pdo = new PDO(
            dsn: 'mysql:host=127.0.0.1;dbname=chopin_vr;charset=utf8mb4',
            username: 'root',
            password: '',
            options: [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
        );
    }
    return self::$pdo;
```

Exemple d'utilisation :

```
6 references
private PDO $db;
5 references | 0 overrides
public function __construct()
{
    $this->db = Database::get();
}
```

Cette approche nous permet de centraliser la connexion. Plutôt que d'ouvrir une nouvelle connexion à chaque fois que nous avons besoin d'un utilisateur ou d'un avatar, nous réutilisons la même instance. C'est ce qu'on appelle une gestion optimisée des ressources, évitant de surcharger le serveur de base de données.

3. L'Implémentation du CRUD Standardisé

Pour garantir une cohérence parfaite entre les différentes entités du projet, nous avons appliqué une structure CRUD (Create, Read, Update, Delete) rigoureuse sur l'ensemble de nos modèles.

Exemple (WorldModel):

```
2 references | 0 overrides
17  public function getAll(): array ···
20  }
21
1 reference | 0 overrides
22  public function get(int $id): mixed ···
27  }
28
1 reference | 0 overrides
29  public function create(string $name, ?string $img, string $url): bool ···
33  }
34
0 references | 0 overrides
35  public function updateById(int $id, string $name, ?string $img, string $url): bool ···
39  }
40
1 reference | 0 overrides
41  public function deleteById(int $id): bool ···
45  }
```

Chaque modèle propose des méthodes claires : getAll() pour les listes, get() pour la récupération unitaire, ainsi que les méthodes de création, de mise à jour et de suppression. Cette standardisation permet à n'importe quel développeur de l'équipe de comprendre instantanément comment interagir avec une nouvelle table de la base de données.

4. La Sécurité au Cœur du Développement

La sécurité des données a été notre priorité lors du codage. Nous avons systématiquement banni les variables directement intégrées dans les chaînes SQL pour éviter les injections SQL.

Exemple d'update sécurisée :

```
$stmt = $this->db->prepare(query: 'UPDATE world SET nameWorld=?, imgWorld=?, urlWorld=? WHERE idWorld=?');
return $stmt->execute(params: [$name, $img, $url, $id]);
```

À la place, nous utilisons des requêtes préparées. Les données envoyées par l'utilisateur sont passées en paramètres dans la méthode execute(). De plus, pour le UserModel, nous avons intégré la fonction password_hash(). Cela garantit que même en cas d'accès non autorisé à la base de données, les mots de passe des utilisateurs restent cryptés et indéchiffrables.

5. Optimisation par les Jointures SQL

Au lieu de multiplier les appels à la base de données, nous avons optimisé nos modèles en utilisant des jointures SQL complexes, notamment dans le UserModel.

Exemple de jointure :

```
$sql = 'SELECT u.*, a.nameAvatar, w.nameWorld FROM user u
        JOIN avatar a ON u.idAvatar = a.idAvatar
        JOIN world w ON u.idWorld = w.idWorld
        WHERE u.username = ?';
```

Lorsqu'on récupère un utilisateur, le modèle effectue automatiquement un JOIN avec les tables avatar et world. Cela nous permet de récupérer en une seule fois toutes les informations liées (comme le nom de l'avatar ou le nom du monde), améliorant ainsi considérablement les performances d'affichage de l'application.

6. Typage Moderne et Robustesse PHP 8

Enfin, nous avons exploité les fonctionnalités modernes de PHP 8. Chaque méthode est strictement typée : nous définissons le type des arguments entrants (ex: int $id, string $username) et le type de retour attendu (ex: : bool, : array).

Exemple de méthode qui retourne un tableau :

```
2 references | 0 overrides
public function getAll(): array
{
```

Exemple de méthode qui retourne un booléen :

```
1 reference | 0 overrides
public function create(string $username, string $password, string $role, int $avatar, int $world): bool
{
```

Ce typage strict agit comme une première couche de test. Il permet de détecter les erreurs de logique dès la phase de développement et garantit que les données qui circulent entre nos modèles et nos contrôleurs sont toujours conformes à ce qui est attendu.

Codage Contrôleur

1. Le Rôle d'Orchestration

La couche "Contrôleur" fait le pont entre les demandes de l'utilisateur et les réponses de l'application. Dans notre architecture, chaque contrôleur a une mission précise : MainController gère l'accueil, CreationUtilisateurController s'occupe de l'inscription et la gestion du profil utilisateur, tandis que AdminController centralise la gestion du contenu (partie administrateur).

L'objectif principal ici est de maintenir des méthodes légères. En cas d'exécution d'un formulaire, le contrôleur reçoit les données de la superglobale $_POST, interroge les modèles nécessaires, traite les informations, puis délègue l'affichage au moteur de template.

Exemple d'action de AdminController (suppression d'un personnage) :

```php
if (isset($post['delete_perso'])) {
    return $this->supprimerPerso(id: $post['delete_perso']);
}
```

2. L'Intégration du Moteur de Template Twig

Pour la partie "Vue", nous avons fait le choix d'utiliser Twig. Dans chaque constructeur de nos contrôleurs, nous initialisons l'environnement Twig en lui indiquant le chemin vers nos fichiers .twig.

Initialisation de Twig :

```php
$loader = new FilesystemLoader(paths: "{$this->root}/templates");
$twig = new Environment(loader: $loader);
```

Utilisation dans AdminController :

```php
return $twig->render(name: 'admin/index.html.twig', context: [
    'path'          => $path
```

Cette séparation est fondamentale : le contrôleur prépare un tableau de données (comme la liste des avatars ou les messages d'erreur) et l'envoie à Twig. Cela permet d'avoir un code HTML propre, dépourvu de logique PHP complexe, facilitant ainsi le travail sur le design sans risquer de casser la logique métier.

3. Gestion des Flux et Logique de Décision

Nos contrôleurs intègrent une logique de décision poussée, notamment dans la méthode handlePost(), qui permet de diriger les différentes demandes selon les donnés trouvées dans le post.

Exemple de méthode handlePost() :

```php
private function handlePost(array $post): string
{

if (isset($post['nom_perso'])) {
    return $this->ajouterPerso(post: $post);
}
```

Suite à ça, une autre méthode plus spécifique prend le relai (ajouterPerso()) :

```php
private function ajouterPerso(array $post): string
{
    if (empty($post['label_perso']) || empty($post['nom_perso'])) {
        return "Champs manquants.";
    }

    if (!isset($_FILES['img_perso'], $_FILES['3d_perso'])) {
        return "Fichiers manquants.";
    }

    if (
        !$this->uploadImagePerso(file: $_FILES['img_perso'], nom: $post['nom_perso']) ||
        !$this->uploadModelPerso(file: $_FILES['3d_perso'], nom: $post['nom_perso'])
    ) {
        return "Erreur upload fichiers.";
    }

    $avatar = new AvatarModel();
    return $avatar->create(
        name: $post['nom_perso'],
        model: $post['label_perso'],
        img: "{$post['nom_perso']}.jpg"
    ) ? "Personnage ajouté." : "Erreur ajout personnage.";
}
```

Cette centralisation de la logique permet d'éviter les doublons de code, garantit un comportement cohérent de l'application face aux actions de l'utilisateur et permet par la suite de vérifier les données et la possibilité de mettre à jour la valeur.

4. Sécurité Avancée : Jetons CSRF et Authentification

La sécurité est renforcée au niveau des contrôleurs par la mise en place de tokens CSRF (Cross-Site Request Forgery). À chaque ouverture de formulaire, le contrôleur génère un jeton unique stocké en session. Lors de la soumission, le contrôleur compare le jeton reçu avec celui en session. Si les jetons ne correspondent pas, l'action est bloquée.

Vérification du jeton CSRF :

```php
if (empty($post['token_csrf']) || $post['token_csrf'] !== $_SESSION['token_csrf']) {
    return "Token CSRF invalide.";
}
```

De plus, l'AdminController vérifie systématiquement le rôle de l'utilisateur en session avant d'autoriser l'accès aux fonctions sensibles comme l'ajout ou la suppression de ressources, protégeant ainsi l'administration contre les accès non autorisés.

5. Traitement des Médias et Uploads Sécurisés

L'AdminController possède une responsabilité supplémentaire : la gestion des fichiers (images et modèles 3D au format .glb). Nous avons développé des méthodes privées spécialisées pour l'upload.

Exemple de méthode d'upload :

```php
private function uploadImagePerso(array $file, string $nom): bool
{
    $ext = strtolower(string: pathinfo(path: $file['name'], flags: PATHINFO_EXTENSION));
    if (!in_array(needle: $ext, haystack: ['jpg', 'jpeg', 'png'])) return false;

    $dest = "{$this->root}/public/assets/images/persos/{$nom}.{$ext}";
    return move_uploaded_file(from: $file['tmp_name'], to: $dest);
}
```

Le contrôleur ne se contente pas de déplacer les fichiers, il vérifie les extensions (JPG, PNG pour les images, GLB pour la 3D) et définit des chemins de destination structurés dans l'arborescence du projet. Cette rigueur permet de garantir que seuls des fichiers valides sont stockés sur le serveur, évitant ainsi les failles de sécurité liées à l'envoi de scripts malveillants.

6. Transformation et Formatage des Données

Enfin, nous avons mis en place une étape de transformation des données via array_map. Les données brutes sortant de la base de données (ex: idAvatar) sont renommées et formatées (ex: id, label avec une majuscule via ucfirst) avant d'être envoyées à la vue.

Exemple:

```php
$persos = array_map(callback: fn($p): array => [
    'id'    => $p['idAvatar'],
    'nom'   => $p['nameAvatar'],
    'img'   => $p['imgAvatar'],
    'model' => $p['modelAvatar'],
    'label' => ucfirst(string: $p['nameAvatar'])
], array: $avatars->getAll());
```

Cette étape de "mapping" permet de découpler totalement la structure de notre base de données de notre interface utilisateur. Si le nom d'une colonne change en base de données, nous n'avons qu'à modifier cette petite fonction de transformation dans le contrôleur, sans avoir à retoucher l'intégralité de nos templates Twig.

Codage Vue

1. Le Choix de Twig : Séparation et Clarté

Pour la couche "Vue", nous avons utilisé le moteur de template Twig. Ce choix technique permet de séparer strictement la structure HTML de la logique de programmation PHP.

L'intérêt majeur réside dans la lisibilité : au lieu d'avoir un code mélangé, nous utilisons des balises spécifiques : '{{ }}' pour l'affichage, '{% %}' pour la logique de contrôle et '{# #}' pour les commentaires. Cela rend les fichiers templates extrêmement clairs pour un développeur front-end. Cette approche garantit une maintenance simplifiée et une plus grande sécurité, Twig gérant nativement l'échappement des caractères pour prévenir les failles XSS.

2. Système d'Héritage et Template de Base

Nous avons mis en place un système d'héritage efficace grâce à un fichier base.html.twig. Ce fichier contient le "squelette" commun à toutes les pages de l'application (le head, les polices Google Fonts, FontAwesome, et les structures de base (block)).

base.html.twig :

```
<html lang="fr">
    <head>
        {#-Elements---------#}
        <meta charset="UTF-8">
        <title> ···
        </title>

        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <link rel="icon" type="image/x-icon" href="">

        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
        <link href="https://fonts.googleapis.com/css2?family=Ubuntu:ital,wght@0,300;0,400;0,500;0,700;1,300;1,400;1,500;1,700&display=swap" rel="stylesheet"

        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">

        {% block style %}
        {% endblock %}

    </head>

    <body>

        <header>
            {#-Header-------#}
            {% block header %}
            {% endblock %}
        </header>

        <main>
            {#-Main-------#}
            {% block main %}
            {% endblock %}
        </main>

        <footer>
            {#-Footer-------#}
            {% block footer %}
            {% endblock %}
        </footer>

        {#-Scripts-------#}
        {% block scripts %}
        {% endblock %}
```

Héritage :

```
{#-Récupère la base----------#}
{% extends '/base.html.twig' %}
```

Nous redéfinissons ensuite uniquement les blocs nécessaires (title, main, style). Cette architecture "DRY" (Don't Repeat Yourself) nous permet de modifier le design global de l'application en un seul endroit sans avoir à retoucher chaque fichier individuellement
.

3. Dynamisme et Affichage Conditionnel

L'un des points forts de nos vues est l'utilisation de l'affichage conditionnel. Grâce aux structures {% if %}, les pages s'adaptent en temps réel à l'état de l'utilisateur.

Condition qui vérifie si l'administrateur est connecté ou non :

```
<section>
    <h3> Espace administrateur</h3>
    <button>
        <a href="./admin">
        {% if not is_connected %}
            Se connecter
        {% else %}
            Espace adminstrateur
        {% endif %}
        </a>
    </button>
</section>
```

4. Expérience Utilisateur et Interactivité (3D)

Pour rendre l'application plus immersive dès la phase de configuration, nous avons intégré des technologies modernes directement dans les vues. Nous utilisons le composant <model-viewer> de Google pour afficher les avatars en 3D.

Affichage du personnage :

```
<model-viewer
    id="viewer"
    src="../assets/3d_model/persos/adventurer.glb"
    alt="Objet 3D"
    auto-rotate
    camera-controls>
</model-viewer>
```

Dans la vue creation_avatar, nous avons couplé Twig avec un script JavaScript personnalisé. Lorsqu'un utilisateur sélectionne un personnage dans la liste déroulante, le modèle 3D se met à jour instantanément sans recharger la page. Cette réactivité est cruciale pour l'expérience utilisateur (UX) dans un projet lié à la réalité virtuelle.

Script pour changer de personnage :

```
<script>
const select = document.getElementById("personnage");
const viewer = document.getElementById("viewer");

select.addEventListener("change", () => {

    const selectedOption = select.options[select.selectedIndex];
    const modelName = selectedOption.id;
    viewer.src = `../assets/3d_model/persos/${modelName}.glb`;
});
</script>
```

5. Gestion des Formulaires et Sécurité Visuelle

Chaque formulaire (création de compte, ajout de map, connexion) est conçu pour être à la fois sécurisé et explicite. Nous intégrons systématiquement des champs cachés pour les tokens CSRF afin de protéger les soumissions.

Champ du token csrf :

```
<form action="./" method="post">

    <input name="token_csrf" value="{{ token_csrf }}" type="hidden" hidden>
```

Nous avons également porté une attention particulière aux retours d'informations. Grâce à la variable a_afficher transmise par le contrôleur, la vue peut afficher des messages d'erreur ou de succès (<h3>{{ a_afficher }}</h3>) de manière dynamique. Cela permet de guider l'utilisateur tout au long de son parcours, que ce soit pour une erreur de mot de passe ou une confirmation de création d'environnement.

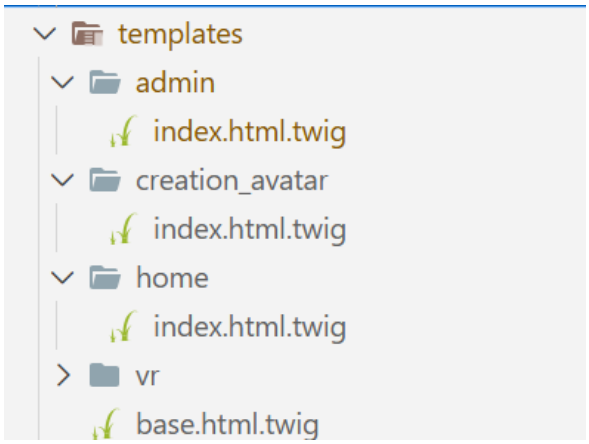
Retour d'information

```
{% if a_afficher is defined %}
    <h3>{{ a_afficher }}</h3>
{% endif %}
```

6. Structure de Fichiers et Organisation

Enfin, comme le montre l'arborescence de notre projet, les vues sont organisées de manière logique dans un dossier ‘templates’. Chaque fonctionnalité possède son propre sous-dossier (admin, home, creation_avatar).

Arborescence des templates

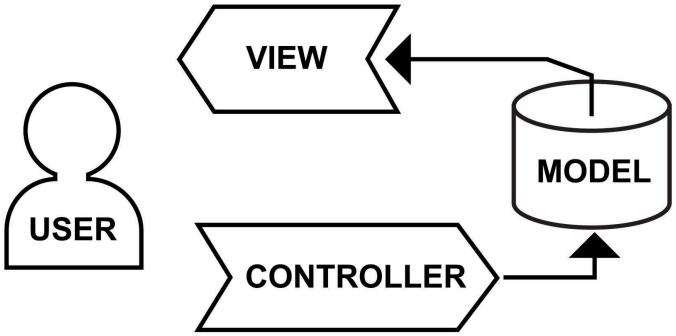


Cette organisation permet de retrouver instantanément le fichier index.html.twig correspondant à une route. Le fichier base.html.twig reste à la racine du dossier templates, agissant comme le pivot central de toute l'interface graphique du projet.

Documentation Technique et Fonctionnelle du Back-Office

Présentation Générale

Le Back-Office est le centre de contrôle du projet. Il permet de faire le pont entre la gestion administrative des données et l'expérience immersive. Développé en PHP 8 selon l'architecture MVC, il garantit une séparation stricte entre la logique métier, l'accès aux données et l'interface utilisateur.



MODEL - VIEW - CONTROLLER PATTERN

Architecture Technique

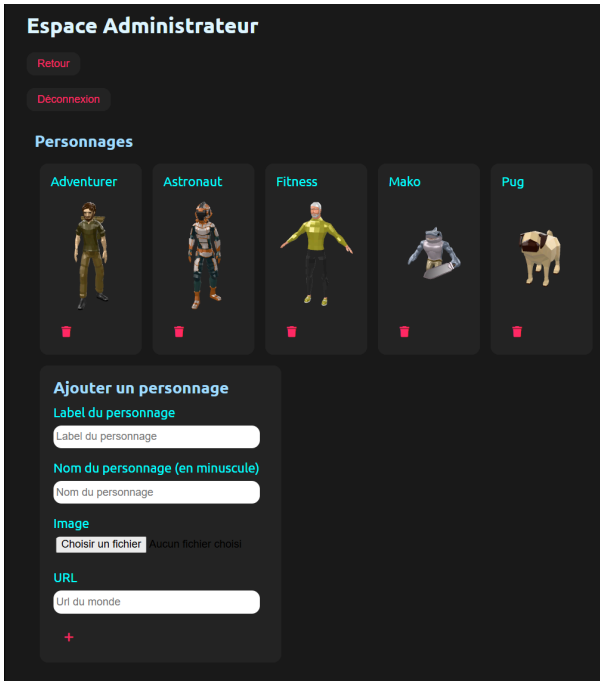
- Langage : PHP 8.x (Typage strict)
- Moteur de Template : Twig (Héritage de templates, sécurisation XSS)
- Base de Données : MySQL via l'interface PDO
- Frontend : HTML5/CSS3, JavaScript, <model-viewer> pour la prévisualisation 3D.

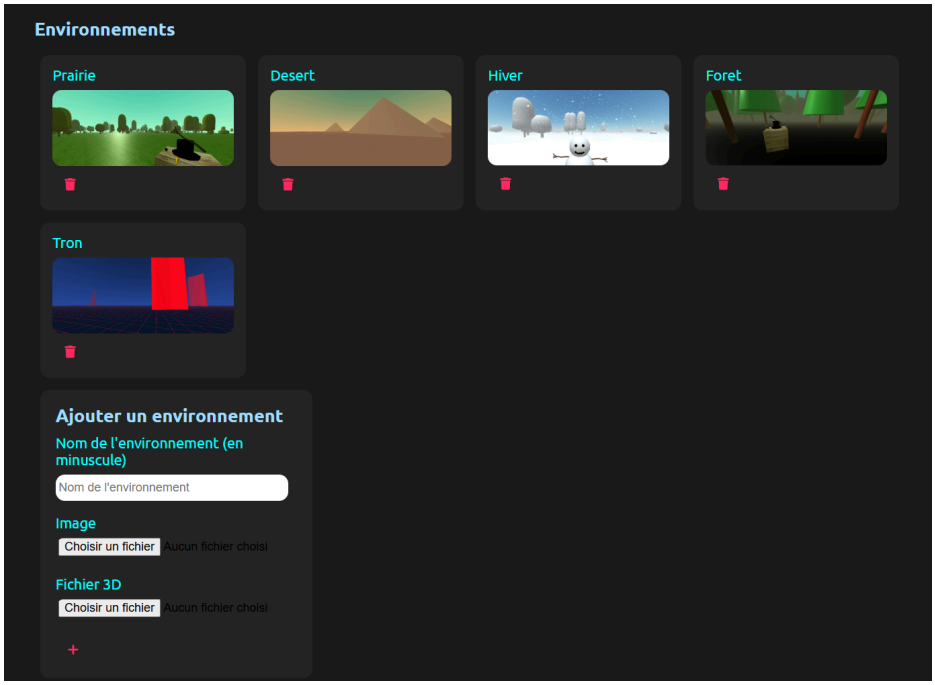
Guide de l'Administrateur (Fonctionnalités)

Le panneau d'administration (AdminController) permet de gérer les ressources critiques du système :

Fonctionnalité	Description
Gestion des Avatars	Ajouter, modifier ou supprimer les modèles 3D (fichiers .glb) et leurs textures.
Gestion des Mondes	Configurer les différents environnements VR disponibles pour les utilisateurs.
Upload Sécurisé	Système de vérification des extensions de fichiers pour les médias 3D et images.

Aperçu :





Sécurité et Intégrité des Données

La sécurité a été implémentée de manière multicouche pour protéger l'écosystème :

- Protection CSRF : Chaque formulaire génère un jeton unique en session. Toute soumission sans jeton valide est rejetée pour prévenir les attaques par falsification de requête.
- Hachage des Mots de Passe : Utilisation des fonctions password_hash() et password_verify(). Aucun mot de passe n'est stocké en clair.
- Prévention des Injections SQL : Utilisation systématique de requêtes préparées via PDO. Contrôle d'Accès (RBAC) : Vérification systématique du rôle "Admin" en session avant l'accès aux routes sensibles.

Exercice de restauration du back-office

Nous avons simulé une casse dans le back-office, le rendant totalement inopérationnel. Afin de le restaurer, nous avons simplement transféré le contenu du site stocké sur github vers l'hébergement principal du site. En écrasant le version cassé par la version fonctionnelle, le site est redevenu opérationnel. L'exercice de restauration fut donc un succès.

Déploiement environnement secours

Nous avons déployé l'environnement de secours sur l'hébergement web de Marco. Ainsi, si l'hébergement principale était hors d'état et qu'il était impossible à réparer, nous aurions quand même pu accéder au projet fonctionnel via l'hébergement de Marco.

Partie 3 : Environnement VR

L'environnement VR a également bénéficié d'une architecture MVC. Ainsi les tâches sont recoupées et la logique reste la même.

Sécurisation accès environnement

```php
//-Connexion
$user = new UserModel();
$user_list = $user->getAll();

$is_logged = false;

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = $_POST['username'] ?? '';
    $password = $_POST['password'] ?? '';

    if (!empty($username) && !empty($password)) {

        $userDb = $user->getByUsername(username: $username);
        if (password_verify(password: $_POST['password'], hash: $userDb['password'])) {

            $persoId = $userDb['idAvatar'];
            $mapId = $userDb['idWorld'];

            $avatar = new AvatarModel();
            $world = new WorldModel();

            $persoToLoad = $avatar->get(id: $persoId);
            $mapToLoad = $world->get(id: $mapId);

            $perso = $persoToLoad['nameAvatar'];

            if (strtolower(string: $mapToLoad['nameWorld']) !== "tron") {
                echo "<script>alert('Redirection vers le bon monde (" . $mapToLoad['nameWorld'] . ")');</script>";
                header(header: "Location: " . $mapToLoad['urlWorld']);
            } else {
                $is_logged = true;
            }
        } else {
            echo "<script>alert('Identifiant ou mot de passe incorrect.');</script>";
```

Ce code PHP, trouvable dans vrController, gère le processus d'authentification d'un utilisateur de manière sécurisée. Lorsqu'un formulaire est soumis en POST, le code récupère l'identifiant et le mot de passe, puis interroge la base de données via un modèle utilisateur. La sécurité est assurée par la fonction password_verify, qui compare le mot de passe saisi au "hash" stocké en base de données.

Si l'accès est validé, le script charge les informations associées au profil du joueur, notamment son avatar et son monde (map). Une logique spécifique est alors appliquée : si le monde de l'utilisateur n'est pas "tron", il est automatiquement redirigé vers l'URL de sa zone respective après l'affichage d'une alerte. Dans le cas où il doit aller dans le monde "tron", il est simplement marqué comme connecté via la variable $is_logged. En cas d'échec de l'authentification, une alerte JavaScript signale que les identifiants sont incorrects.

Documentation Technique : Expérience VR "TRON" (A-Frame)

Cette documentation détaille la structure technique, les interactions et les composants de l'environnement virtuel inspiré de l'univers TRON.

1. Stack Technique & Dépendances

Le projet repose sur le framework A-Frame et utilise plusieurs bibliothèques tierces pour les interactions et la physique. Voici les bibliothèques principales :

Bibliothèque	**Rôle**
A-Frame (v1.x)	Framework principal pour le rendu 3D/VR.
Super-Hands	Gestion des interactions complexes (saisir, survoler).
A-Frame Physics (Ammo)	Moteur physique pour la détection de collisions et la gravité.
Teleport Controls	Système de déplacement par téléportation pour le confort VR.

2. Architecture du Camera Rig (Le Joueur)

Le joueur est défini par un "Rig" qui sépare la position du corps de la vue de la caméra, permettant un mouvement fluide et des interactions manuelles.

- Main Gauche (Navigation) : Utilise teleport-controls. Le joueur peut viser les zones avec la classe ‘.teleportable’ pour se déplacer.
- Main Droite (Interaction) : Utilise super-hands et un sphere-collider. Elle permet d'interagir avec les objets ayant la classe .collidable (ex: la moto-2).
- Contrôles Joystick : Un script personnalisé écoute l'événement axismove pour mapper les axes du joystick sur les touches fléchées (arrowleft, arrowup, etc.), permettant de piloter des véhicules comme la Light Cycle (ne fonctionne pas).

3. Entités et Composants Personnalisés

Le Recognizer

L'entité ‘#recognizer’ utilise un composant spécialisé pour son comportement :

- Composant : patrouille-lointaine
- Paramètres : Gère la vitesse, l'altitude (hauteurMin/Max) et le rayon de patrouille.
- Modèle : recognizer.glb (scale 15).

Les Véhicules

Il existe deux types de voitures dans la scène :

- Voiture Statique/Auto : Utilise drive-cycle pour un mouvement piloté (fonctionne sur navigateur traditionnel, mais pas en VR).
- Voiture Interactive : La moto-2 est configurée avec dynamic-body (masse de 2) et peut être saisie ou déplacée grâce à super-hands.

Environnement et Arènes

La scène charge trois types d'arènes différentes (arena-model-1/2/3) positionnées à différentes coordonnées pour créer des zones de jeu distinctes.

4. Effets Visuels et Atmosphère

L'ambiance "Neon/Retro" est obtenue via :

- Environment Component : Preset tron avec des tours (towers) en arrière-plan.
- Fog : Un brouillard exponentiel bleu (#3840e6) pour donner de la profondeur.
- Bloom : Un effet de halo lumineux sur les objets émissifs (via bloom="strength: 2").
- Grid : la grille bleue iconique.

5. Gestion des Assets

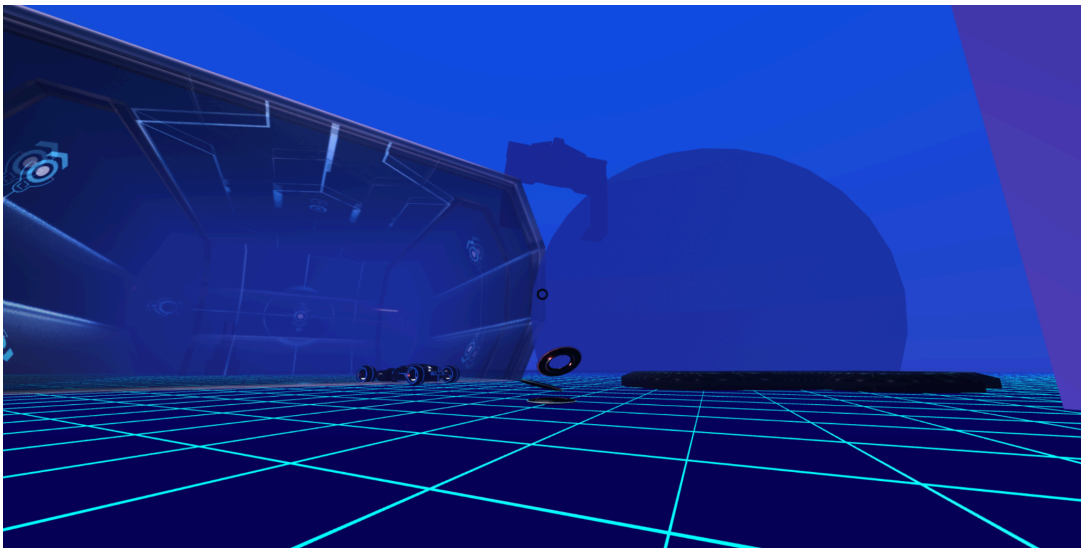
Tous les modèles sont chargés de manière asynchrone via <a-assets> pour optimiser les performances.

ID	Fichier	Usage
disk-model	tron_disk.glb	Disques d'identité (décor/armes).
moto-model	{{ perso }}.glb	Véhicules pilotables.
perso-model	{{ perso }}.glb	Avatar dynamique chargé selon la session.

6. Scripts

- script_recognizer.js : Gère la logique de vol de l'IA.
- script_moto.js : Contient la physique du mouvement du véhicule.
- script_grids.js : Probablement utilisé pour générer le sol quadrillé dynamique caractéristique de Tron.

Résultat :



Exercice restauration VR

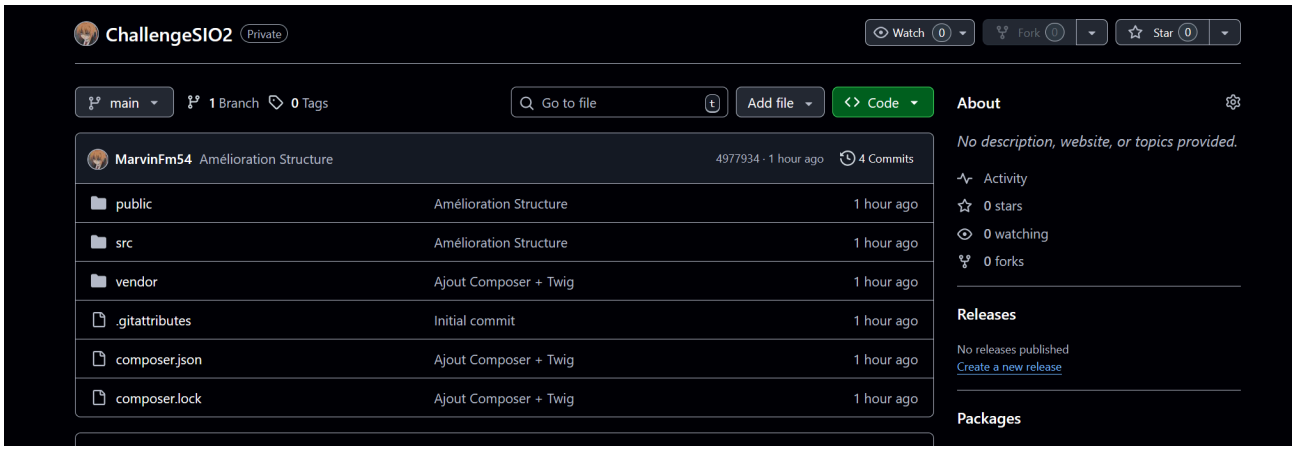
L’exercice de restauration ne fut pas d’une grande difficulté, car il était exactement le même que celui mené pour le back-office. Les fichiers vr étant stockés côte à côte avec les fichiers du back-office, ils peuvent être restaurés en même temps.
L’exercice de restauration fut donc également un succès.

Partie 4 : Tâches transverses

Procédure de sauvegarde

Application

Nous sauvegardons via commit / push sur GitHub. Le projet devient alors accessible et téléchargeable par tout le groupe.



Base de données

Nous sauvegardons la base de données en l'emportant (avec PhpMyAdmin) puis en l’emportant dans un dossier “Sauvegardes” dans Google Drive.

Documentation de la Base de Données : Chopin VR

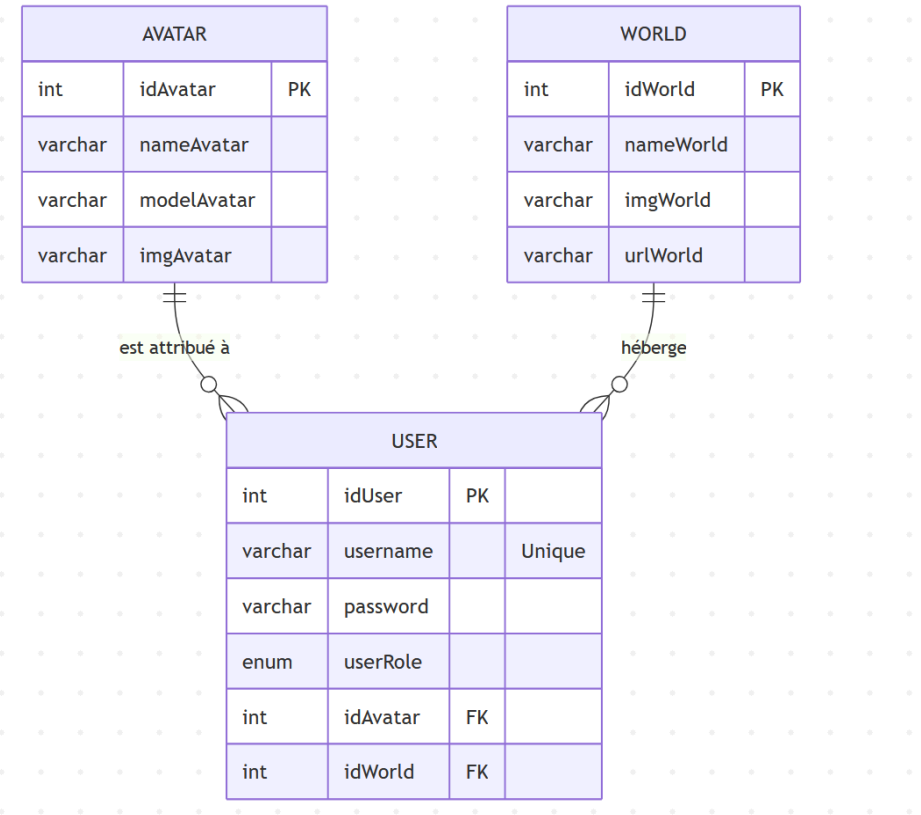
Présentation Générale

La base de données ‘chopin_vr’ est conçue pour gérer un environnement de réalité virtuelle multi-utilisateurs. Elle permet de stocker les informations relatives aux comptes utilisateurs, aux avatars personnalisables et aux différents mondes (environnements) accessibles dans l'application.

- Système de Gestion : MySQL
- Moteur de stockage : InnoDB (supporte les transactions et les clés étrangères)
- Encodage : utf8mb4_general_ci (support complet des caractères spéciaux)

Suite à des débats inter-groupes sur la structure de la base de données, nous avons finalement obtenu une base de données commune.

Diagramme de la base



Dictionnaire des Données

Table : **user**

Cette table stocke les informations d'authentification et l'état actuel des utilisateurs.

Colonne	Type	Attributs	Description
idUser	INT	PK, AI	Identifiant unique de l'utilisateur.
username	VARCHAR(50)	UNIQUE, NOT NULL	Nom de connexion (doit être unique).
password	VARCHAR(255)	NOT NULL	Mot de passe (stocké sous forme de hash).
userRole	ENUM	'ADMIN', 'JOUEUR'	Niveau de privilèges de l'utilisateur.
idAvatar	INT	FK, NOT NULL	Référence à l'avatar sélectionné par l'utilisateur.
idWorld	INT	FK, NOT NULL	Référence au monde où se trouve l'utilisateur.

Table : **avatar**

Répertorie les modèles 3D disponibles pour les utilisateurs.

Colonne	Type	Attributs	Description
idAvatar	INT	PK, AI	Identifiant unique de l'avatar.
nameAvatar	VARCHAR(100)	NOT NULL	Nom de l'avatar (ex: 'astronaut').
modelAvatar	VARCHAR(255)	NOT NULL	Chemin vers le fichier 3D (ex: .glb).
imgAvatar	VARCHAR(255)	NULL	Image d'aperçu de l'avatar.

Table : **world**

Liste les environnements VR disponibles et leurs adresses de déploiement.

Colonne	Type	Attributs	Description
idWorld	INT	PK, AI	Identifiant unique du monde.
nameWorld	VARCHAR(100)	NOT NULL	Nom de l'environnement (ex: 'desert').
imgWorld	VARCHAR(255)	NULL	Image de couverture du monde.
urlWorld	VARCHAR(255)	NOT NULL	Lien URL vers l'instance du monde VR.