

Gomez James

Dossier de Projet – CDA 2024

Sommaires

I. Listes des compétences du REAC

II. Résumé du Projet PlayForge (en Anglais)

III. Cahier des charges

- 1. Description du cahier des charges du Projet**
- 2. Public et utilisateurs du site**
- 3. Plateformes**

IV. Gestion du Projet

V. Spécification fonctionnelles du Projet PlayForge

- 1. PlayForge Back-end (Base de donnée)**
- 2. PlayForge Api-Upload**
- 3. PlayForge Front-End (Angular)**

VI. Spécification technique du Projet PlayForge

- 1. Technologie utilisés**
- 2. Contraintes**
- 3. principe du développement MVC et application en couches**

VII. Réalisation significatives & argumentations

- 1. Aspect Graphique**
- 2. Base de données**

VIII. Présentation des key features et jeu d'essai

- 1. Jouer à un jeu sur navigateur**
- 2. Ajouter un jeu à la BDD**

I.Liste des compétences du REAC

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles
1	Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	1	Maquetter une application
		2	Développer une interface utilisateur de type desktop
		3	Développer des composants d'accès aux données
		4	Développer la partie front-end d'une interface utilisateur web
		5	Développer la partie back-end d'une interface utilisateur web
2	Concevoir et développer la persistance des données en intégrant les recommandations de sécurité	6	Concevoir une base de données
		7	Mettre en place une base de données
		8	Développer des composants dans le langage d'une base de données
3	Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité	9	Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
		10	Concevoir une application
		11	Développer des composants métier
		12	Construire une application organisée en couches
		13	Développer une application mobile
		14	Préparer et exécuter les plans de tests d'une application
		15	Préparer et exécuter le déploiement d'une application

II. Résumé du projet en Anglais et en Français

Résumé du projet en Français :

PlayForge est une plateforme dédiée au partage et à l'achat de jeux indépendants, offrant une expérience utilisateur enrichissante.

Le système de bibliothèque permet aux utilisateurs de stocker et consulter tous leurs jeux achetés ou joués, avec un récapitulatif des achats et un suivi du statut de chaque jeu.

Les utilisateurs reçoivent des notifications pour les mises à jour de jeux et les nouveaux commentaires.

L'espace membre permet de gérer les profils avec inscription, connexion, et modification de données, ainsi que l'attribution des rôles d'Utilisateur ou de Développeur.

Tous les utilisateurs peuvent consulter et jouer à des jeux, tandis que les développeurs peuvent mettre en ligne leurs créations. Un système de commentaires et de notes favorise l'interaction et l'engagement au sein de la communauté.

Résumé du projet en Anglais :

PlayForge is a platform dedicated to sharing and purchasing indie games, offering an enriching user experience.

The library system allows users to store and view all their purchased or played games, with a purchase summary and a status tracker for each game. Users receive notifications for game updates and new comments.

The member area allows users to manage their profiles with registration, login, and data modification, as well as the assignment of roles as User or Developer.

All users can browse and play games, while developers can upload their creations. A comment and rating system encourages interaction and engagement within the community.

III.Cahier des charges

1. Description du Cahier des charges

- **Système de bibliothèque :**
 - La bibliothèque permet aux utilisateurs de stocker et de consulter tous leurs jeux achetés.

la bibliothèque comprend :

- **Récapitulatif des jeux achetés** : Les utilisateurs peuvent facilement retrouver leurs acquisitions.
 - **Statut des jeux** : Suivi du statut de chaque jeu, qu'il soit téléchargé ou non.
- **Système de notification**

Les utilisateurs sont informés des activités concernant leurs jeux grâce à des notifications :

- **Mise à jour des jeux** : Alerté sur les mises à jour disponibles.
- **Nouveaux commentaires** : Notification sur les nouveaux commentaires.

- **Espace membres**

L'espace membre permet la gestion des profils :

- **Inscription et connexion** : Création et gestion de comptes.
- **Modification du profil** : Changement de photo, nom, ou description.
- **Rôles** : Attribution des rôles d'Utilisateur ou de Développeur.

- **Rôles des utilisateurs**

Tous les utilisateurs ont accès aux fonctionnalités suivantes :

- Consulter et jouer aux jeux sur navigateur.
- Créer un compte.

- **Utilisateurs**

Les Utilisateurs peuvent :

- Télécharger et commenter des jeux.
- Consulter leur bibliothèque et recevoir des notifications.

- **Développeurs**

Les Développeurs peuvent :

- Mettre en ligne des jeux.

- **Administrateurs**

Les administrateurs ont tous les droits.

- **Système de commentaires et de notes**

Chaque jeu dispose d'un espace de commentaires où les utilisateurs et développeurs peuvent laisser des avis et noter les jeux, favorisant ainsi une communauté active.

2. Public et Utilisateurs du sites

- **Public visé : PlayForge s'adresse principalement à deux groupes de personnes :**

- **Joueurs indépendants** : Ceux qui sont passionnés par les jeux vidéo, en particulier les titres indépendants. Ils recherchent des expériences uniques et souhaitent soutenir les développeurs indépendants.
- **Développeurs de jeux** : Les créateurs de jeux indépendants qui cherchent à partager et vendre leurs créations, ainsi qu'à interagir avec la communauté de joueurs.

- **Utilisateurs : Les utilisateurs de PlayForge peuvent être classés en plusieurs catégories :**

- **Utilisateurs réguliers :**

Ils peuvent créer un compte pour accéder à la plateforme.

Ils ont la possibilité de télécharger des jeux, commenter et évaluer les titres, ainsi que consulter leur bibliothèque personnelle.

- **Développeurs :**

Ils peuvent mettre en ligne leurs jeux après vérification d'identité.

Ils interagissent avec les joueurs en répondant aux commentaires et en recevant des retours sur leurs créations.

- **Administrateurs :**

Ils ont des droits complets pour gérer la plateforme, superviser les interactions et s'assurer du bon fonctionnement du site.

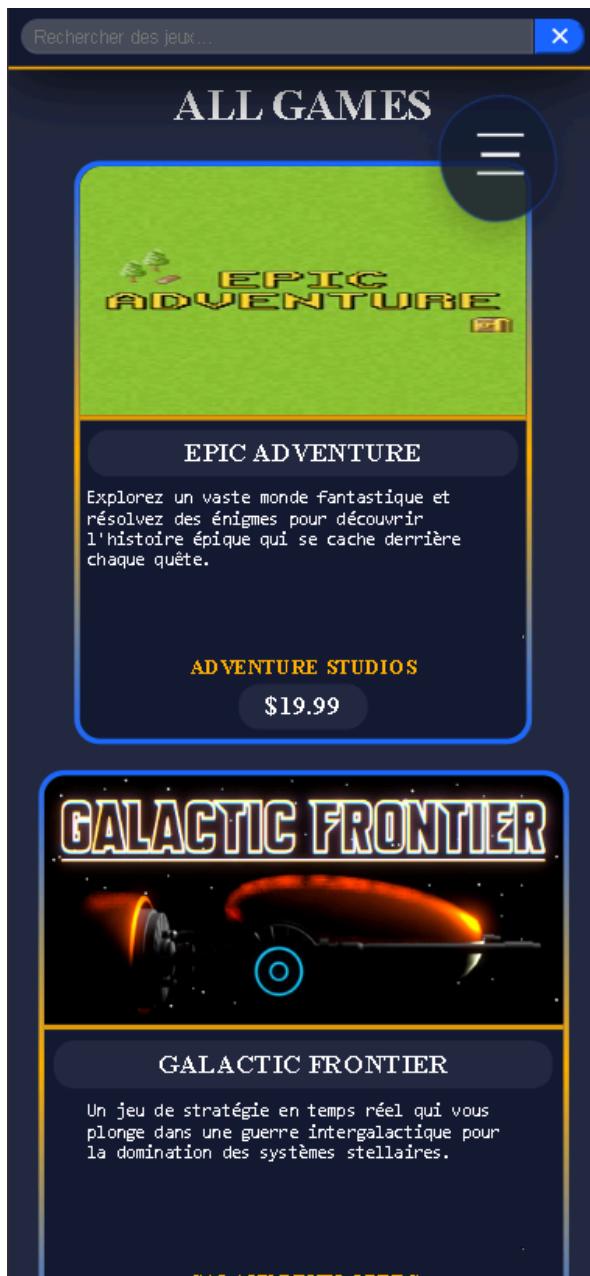
Cette diversité d'utilisateurs contribue à créer une communauté active et engagée autour des jeux indépendants.

3. Plateformes

PlayForge est une plateforme accessible sur tous les appareils, y compris les ordinateurs, téléphones et tablettes. Cette flexibilité permet aux utilisateurs de profiter de l'expérience de partage et d'achat de jeux indépendants, quel que soit leur appareil. Que ce soit via un navigateur web ou une application

dédiée, PlayForge garantit une interface conviviale et intuitive, offrant un accès facile à la bibliothèque de jeux, aux fonctionnalités de téléchargement, et aux interactions au sein de la communauté.

- **Version téléphone :**



La version mobile de PlayForge est conçue pour offrir aux utilisateurs et aux développeurs une expérience optimale sur leurs smartphones.

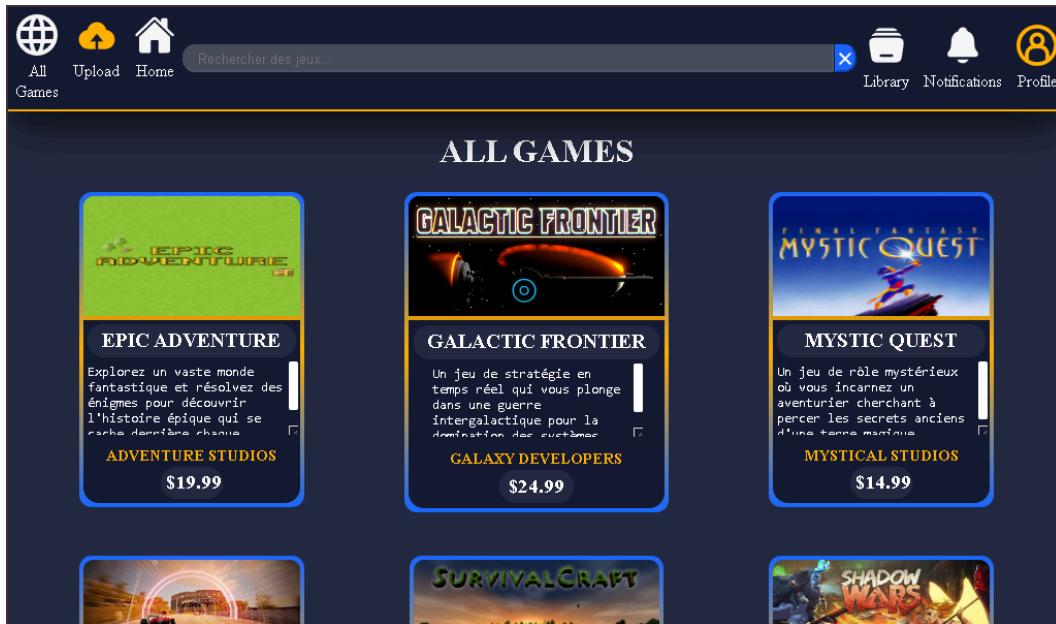
- Répondre rapidement aux commentaires : Les développeurs peuvent gérer les retours des joueurs en temps réel, favorisant ainsi une interaction directe avec la communauté. Cela leur permet de répondre rapidement aux questions, de prendre en compte les retours et d'améliorer leurs jeux en fonction des suggestions des utilisateurs.

- Consulter les derniers jeux sortis : Les utilisateurs ont un accès facile aux nouvelles sorties et aux jeux populaires directement depuis leur téléphone. Ils peuvent parcourir les jeux récemment ajoutés, ce qui leur permet de rester informés des dernières tendances et de découvrir de nouvelles expériences de jeu.

- Jouer à des jeux sur navigateur : La version mobile de PlayForge permet aux utilisateurs de jouer à des jeux directement depuis leur navigateur. Grâce à l'optimisation pour les écrans tactiles, les jeux sont adaptés pour une jouabilité fluide et intuitive sur mobile, offrant ainsi une expérience agréable sans nécessiter de téléchargement supplémentaire.

Cette accessibilité sur mobile fait de PlayForge une plateforme conviviale, permettant aux utilisateurs et aux développeurs de rester connectés et de profiter des fonctionnalités essentielles à tout moment et en tout lieu.

- Version Tablette :



La version tablette de PlayForge est spécialement conçue pour offrir une expérience utilisateur enrichissante et adaptée aux écrans plus grands.

-Gestion des commentaires :

Les développeurs peuvent facilement répondre aux commentaires des utilisateurs grâce à une interface optimisée pour les tablettes. La taille d'écran permet une visualisation claire des retours, facilitant ainsi des échanges interactifs et constructifs avec la communauté.

-Navigation fluide pour découvrir les nouveaux jeux :

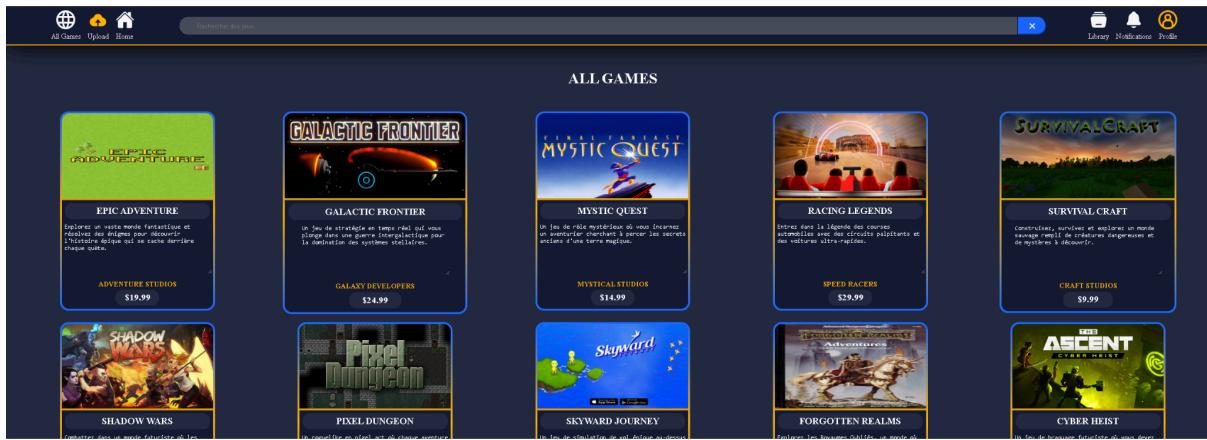
Les utilisateurs peuvent explorer les dernières sorties et les jeux populaires avec une interface tactile conviviale. L'expérience de navigation est améliorée grâce à un affichage plus large, permettant une consultation rapide des détails des jeux, des évaluations et des critiques.

-Jouer à des jeux sur navigateur : La version tablette permet aux utilisateurs de jouer à des jeux directement dans leur navigateur avec un confort accru. Les contrôles tactiles sont parfaitement adaptés aux jeux, offrant une jouabilité intuitive et agréable, que ce soit pour des jeux d'action, d'aventure ou de réflexion.

-Visualisation améliorée de la bibliothèque : Les utilisateurs peuvent consulter leur bibliothèque de jeux achetés ou joués avec une interface claire et organisée, facilitant la gestion de leurs collections et l'accès rapide à leurs titres préférés.

Cette accessibilité sur tablette permettent aux utilisateurs et aux développeurs de profiter d'une expérience riche et immersive, que ce soit pour jouer, commenter ou découvrir de nouveaux jeux, tout en étant confortablement installés.

- Version Ordinateurs / Grand écrans :



La version ordinateur de PlayForge est optimisée pour offrir une expérience utilisateur complète et immersive sur les écrans de bureau.

- Gestion avancée des commentaires** : Les développeurs peuvent facilement suivre et répondre aux commentaires des utilisateurs grâce à une interface riche en fonctionnalités. La mise en page permet une visualisation détaillée des retours, facilitant les interactions et la collecte de feedback pour améliorer les jeux.
- Exploration approfondie des nouveaux jeux** : Les utilisateurs bénéficient d'une interface plus large qui permet une navigation fluide à travers les dernières sorties et les jeux populaires. Avec des images de haute qualité et des descriptions détaillées, les utilisateurs peuvent explorer en profondeur les titres, consulter les évaluations et lire des critiques.
- Jouer à des jeux sur navigateur** : La version ordinateur permet aux utilisateurs de jouer à des jeux directement dans leur navigateur avec des performances optimales. Les jeux sont souvent plus riches en contenu et offrent des graphismes améliorés, garantissant une expérience de jeu fluide et immersive.
- Support des contrôleurs** : Les utilisateurs peuvent profiter d'une expérience de jeu améliorée grâce à la compatibilité avec divers contrôleurs, y compris les manettes de jeux et les claviers/souris. Cela permet aux joueurs de choisir leurs méthodes de contrôle préférées, que ce soit pour des jeux de plateforme, d'action ou de simulation, offrant ainsi une jouabilité intuitive et réactive.
- Bibliothèque bien organisée** : Les utilisateurs peuvent gérer leur bibliothèque de jeux avec une interface claire, qui affiche tous les jeux achetés. Les fonctionnalités de tri et de filtrage facilitent la recherche de titres spécifiques, permettant un accès rapide à leurs jeux préférés.

Cette accessibilité sur ordinateur fait de PlayForge une plateforme robuste, offrant aux utilisateurs et aux développeurs une expérience enrichissante pour jouer, interagir et découvrir de nouveaux jeux dans un environnement complet et fonctionnel.

IV.Gestion du Projet

Pour la réalisation de ce projet, j'ai travaillé de manière autonome durant ma formation, tout en recevant parfois l'aide de certains apprenants et formateurs. J'ai utilisé une méthode agile, mettant en place des sprints de différentes durées.

Le projet a commencé par une première étape : trouver une idée générale à adapter selon mes besoins. Une fois cette idée trouvée, j'ai commencé à élaborer mon diagramme de cas d'utilisation (Use Case) pour définir le parcours des utilisateurs sur mon projet. À partir de ce travail, j'ai ensuite développé le diagramme d'entités relationnelles, en définissant les entités (Objet, Image/Vidéos, Utilisateur).

Ce diagramme m'a permis d'avoir les bases de mon architecture logicielle, en sélectionnant la stack technologique appropriée. J'ai pu finaliser mon diagramme MCD UML en intégrant les relations entre les différentes tables.

**Diagramme
Entités /
Relation**

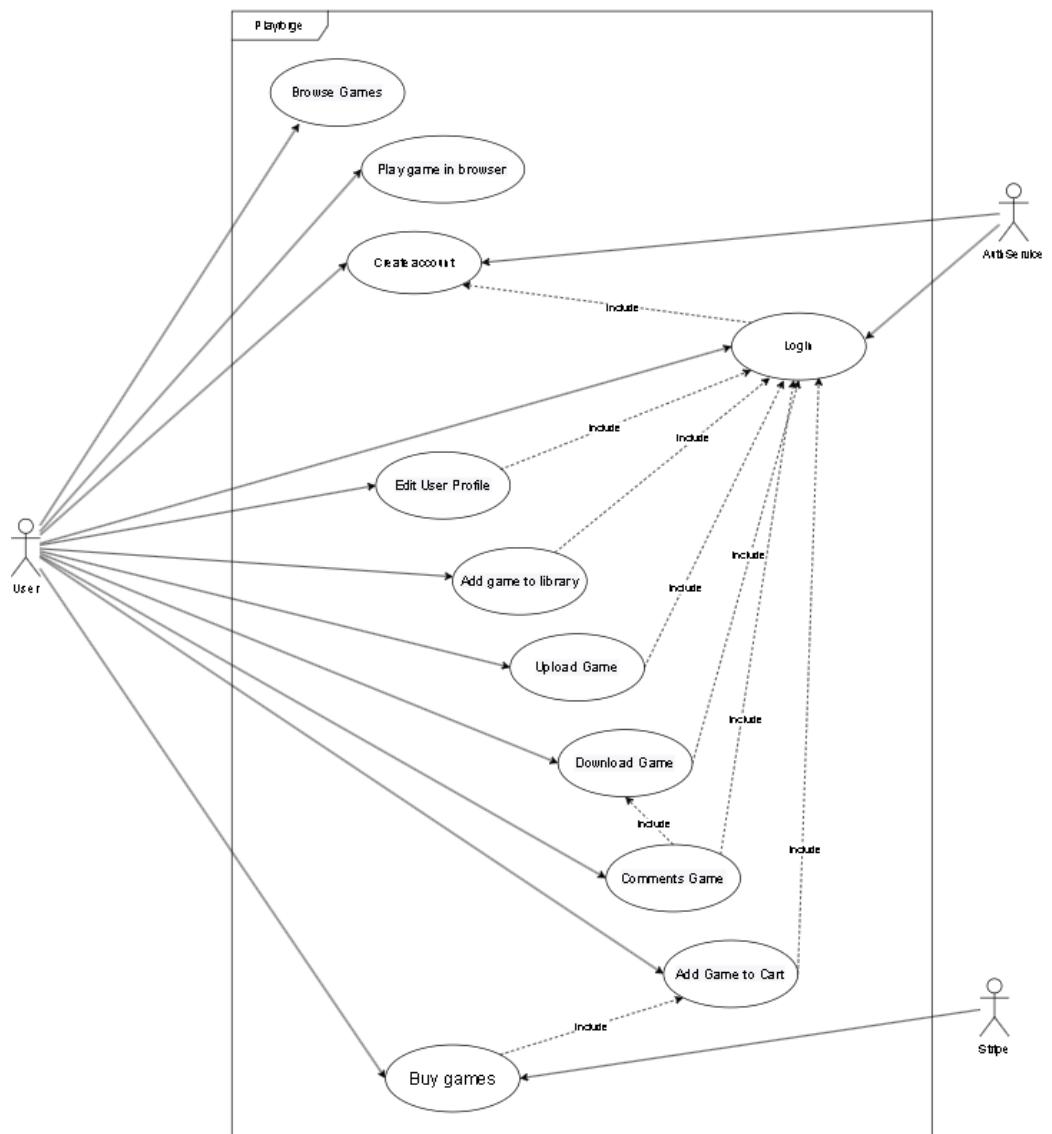


Diagramme UML

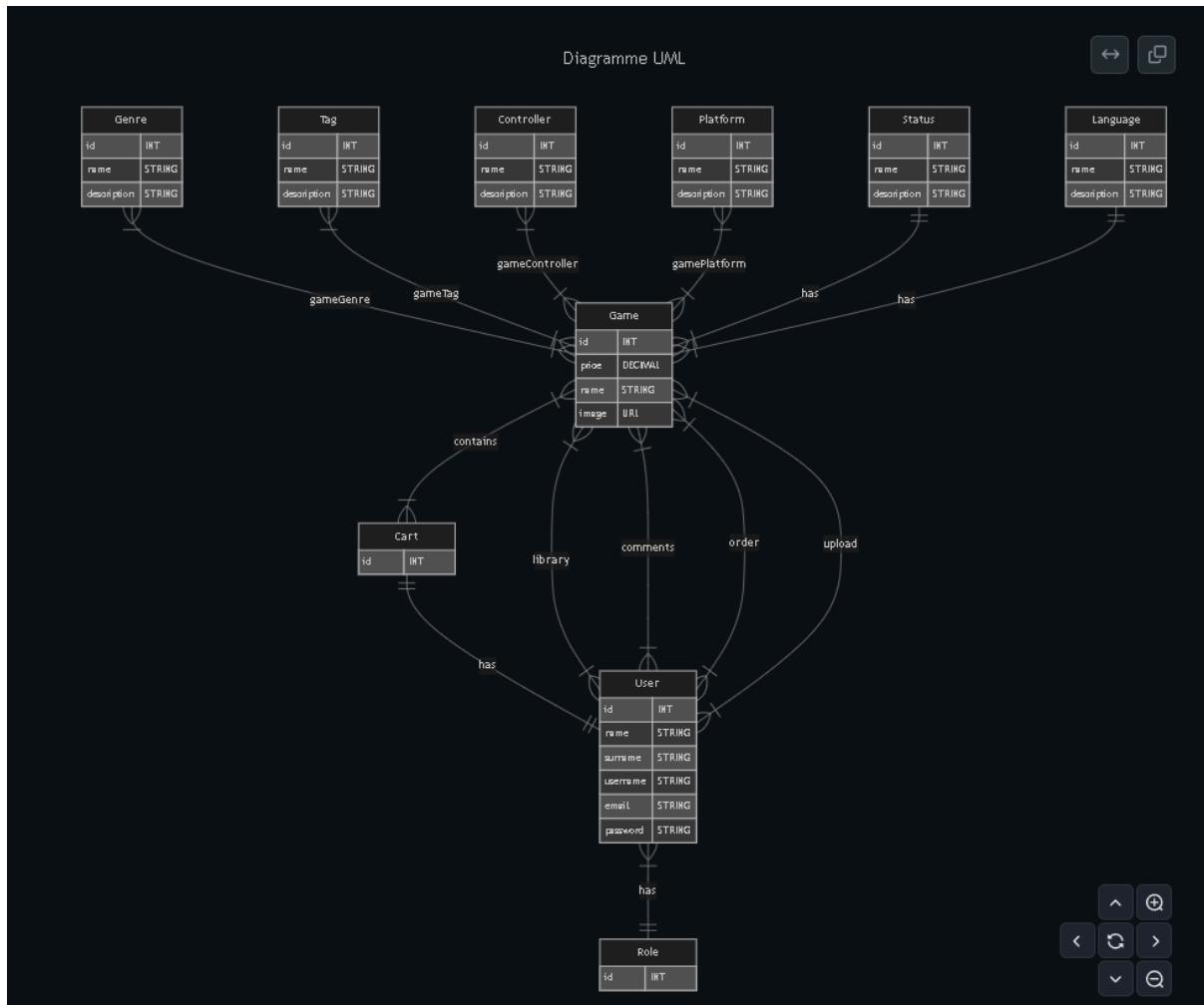
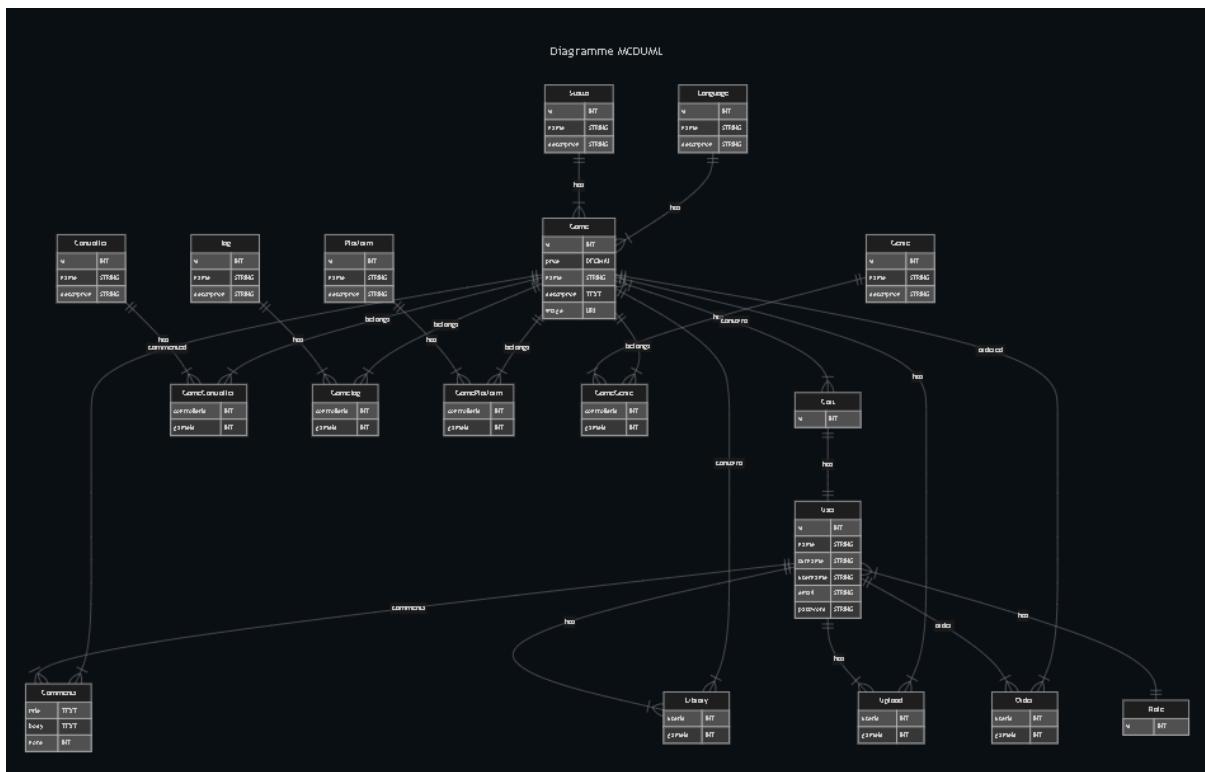
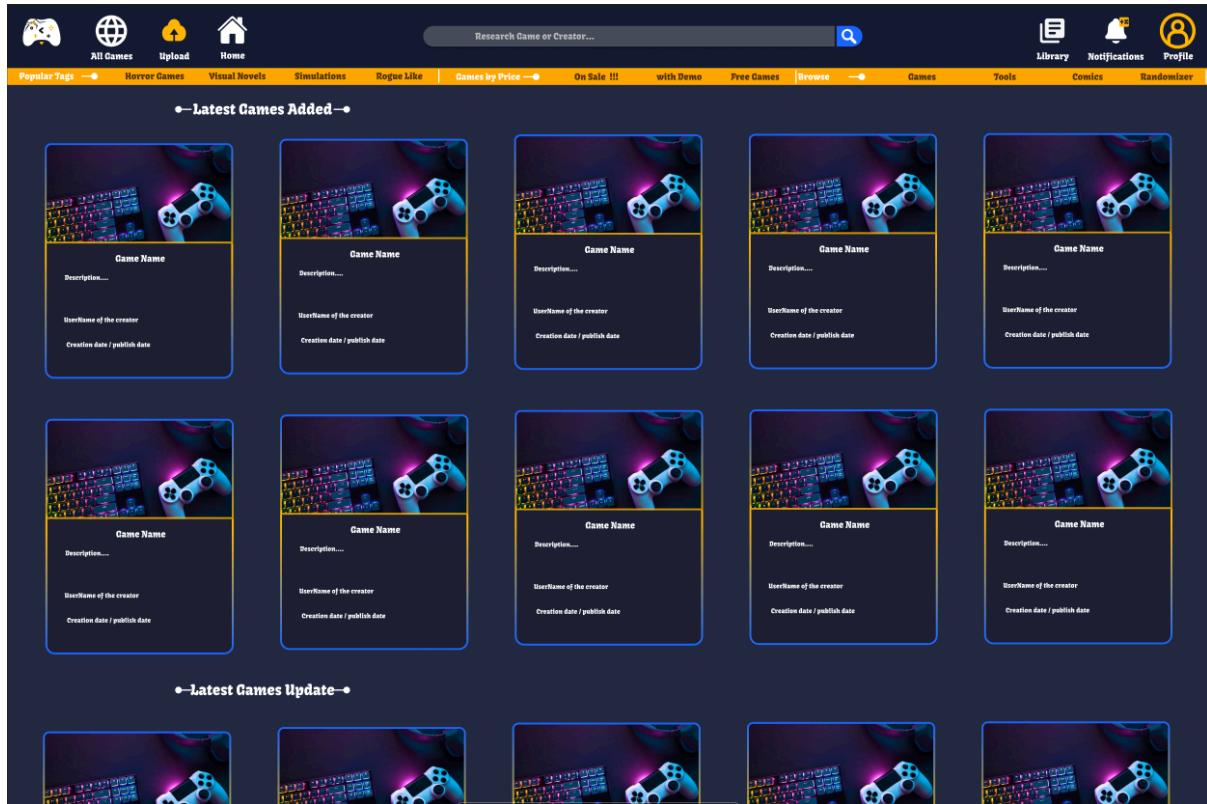


Diagramme MCD(UML)

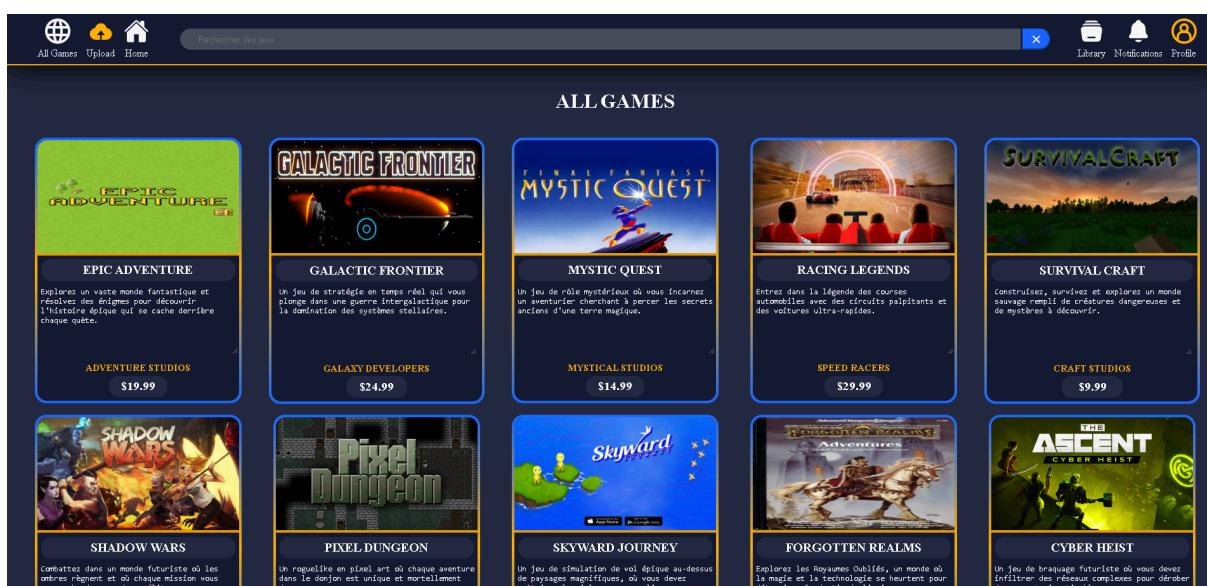


mon front-end Angular, car il existe une relation directe entre les deux. Cette approche a facilité la transition entre la conception et le développement, garantissant une cohérence visuelle et fonctionnelle dans l'application.

- Version Figma :



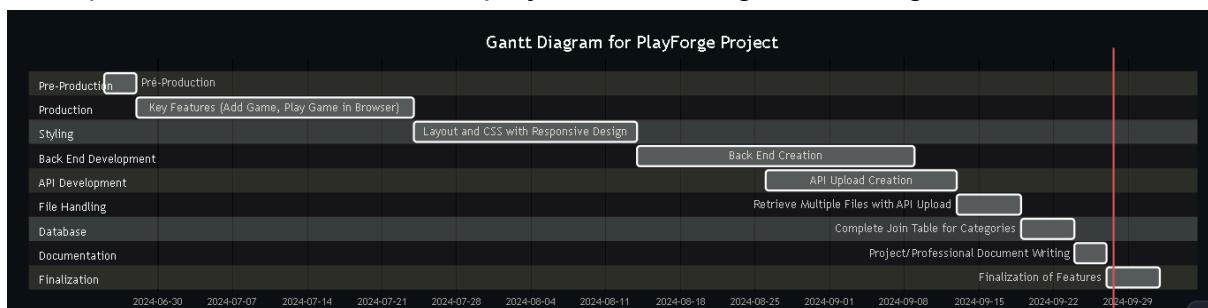
- Version Angular :



Concernant la charte graphique, j'ai utilisé mes connaissances en gaming pour attirer principalement ce public. J'ai intégré des éléments visuels et des designs qui résonnent avec les passionnés de jeux vidéo. Parallèlement, j'ai veillé à ajouter des fonctionnalités simples et intuitives, permettant ainsi à tout le monde d'accéder facilement au site. Cette approche garantit une expérience utilisateur agréable, tant pour les joueurs chevronnés que pour les nouveaux venus.

Au cours de plusieurs semaines, j'ai travaillé par sprints, respectés ou non, et j'ai réussi à développer une application fonctionnelle. Cette application permet aux utilisateurs d'ajouter des jeux, d'accéder aux jeux créés, de donner de la visibilité aux utilisateurs et de jouer à des jeux directement dans le navigateur. Cette approche itérative a permis d'améliorer continuellement les fonctionnalités tout en répondant aux besoins des utilisateurs. Beaucoup de fonctionnalités sont manquantes mais les key features sont respectés

J'ai répertorié toute la timeline du projet avec un diagramme de gantt :



V. Spécification fonctionnels du Projet

1. Back-End PlayForge

1. Gestión des utilisateurs

- Création de compte** : Le système doit permettre aux utilisateurs de créer un compte avec des informations telles que nom, prénom, nom d'utilisateur, adresse e-mail et mot de passe.
- Connexion** : Authentification des utilisateurs via un système de connexion sécurisé.
- Gestion des profils** : Permettre aux utilisateurs de mettre à jour leurs informations de profil (photo, nom, description).

2. API de bibliothèque

- Stockage des jeux** : Fournir des endpoints pour que les utilisateurs puissent ajouter, consulter, et supprimer des jeux dans leur bibliothèque.

- **Récapitulatif des achats** : Une fonction permettant aux utilisateurs de récupérer la liste de tous les jeux achetés.

3. Gestion des jeux

- **Ajout de jeux** : Permettre aux développeurs de soumettre de nouveaux jeux via un endpoint sécurisé.
- **Consultation des jeux** : Offrir des endpoints pour récupérer des informations sur les jeux disponibles, incluant détails, images, et évaluations.

```
GameRoute.post('/new', async (req, res) => {
  const { title, description, price, authorStudio, madewith, StatusId, LanguageId } = req.body;

  try {
    const game = await Game.create([ title, description, price, authorStudio, madewith, StatusId, LanguageId ]);

    res.status(201).json(game);
  } catch (error) {
    console.error('Erreur lors de la création du jeu :', error);
    res.status(500).json({ error: 'Erreur lors de la création du jeu' });
  }
});
```

Route Post qui permet de créer un jeu

```
GameRoute.get("/id/:id", async (req, res) => {
  try {
    const game_id = req.params.id
    const game = await Game.findById(game_id)

    res.status(200).json(game);
  } catch (error) {
    console.log(error);
  }
});
```

Route Get qui récupère un jeu selon son Id (Primary Key)

4. Système de commentaires

- **Ajouter des commentaires** : Permettre aux utilisateurs de laisser des commentaires sur les jeux.
- **Récupération des commentaires** : Fournir un endpoint pour récupérer tous les commentaires associés à un jeu.

5. Système de notifications

- **Notifications d'actualisation** : Informer les utilisateurs des mises à jour de leurs jeux ou des nouveaux commentaires via des notifications push ou par e-mail.

6. Gestion des rôles

- **Rôles utilisateur** : Le back-end doit gérer différents rôles (Utilisateur, Développeur, Administrateur) avec des permissions spécifiques pour chaque rôle.
 - **Utilisateurs** : Accès aux fonctionnalités de téléchargement et de commentaires.
 - **Développeurs** : Accès à l'API pour soumettre et gérer leurs jeux.
 - **Administrateurs** : Contrôle total sur la plateforme, y compris la gestion des utilisateurs et des contenus.

7. Sécurité

- **Authentification et autorisation** : Implémenter un système d'authentification sécurisé (comme JWT) pour protéger les routes et les données des utilisateurs.
- **CORS** : Configurer CORS pour permettre des requêtes sécurisées entre le front-end et le back-end.

La configuration CORS est essentielle pour sécuriser les échanges entre le front-end et le back-end, en s'assurant que seules les requêtes provenant de l'origine spécifiée sont acceptées et que les bonnes méthodes et en-têtes sont utilisés

```
app.use(cors({
  origin: 'http://localhost:4200', // Remplace par l'URL de ton front-end
  methods: ['POST', 'GET', 'PUT', 'DELETE'],
  credentials: true
}));
```

2. Api-Upload PlayForge

1. Gestion des utilisateurs

- **Authentification** : L'API doit permettre aux utilisateurs de s'authentifier afin de garantir que seuls les utilisateurs autorisés peuvent télécharger des fichiers.
- **Gestion des rôles** : L'API doit vérifier les rôles des utilisateurs pour s'assurer que seuls les développeurs peuvent télécharger des jeux.

2. Upload de fichiers

- **Endpoint d'upload** : Fournir un endpoint (`/upload`) permettant aux utilisateurs de télécharger des fichiers (jeux, images, etc.).

```
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
```

- **Types de fichiers autorisés** : L'API doit spécifier et valider les types de fichiers autorisés pour le téléchargement (ex. : `.exe`, `.zip`, `.png`, `.jpg`).
- **Taille maximale des fichiers** : L'API doit gérer une limite de taille pour les fichiers téléchargés afin de prévenir les abus (ex. : 100 Mo).

```
const multer = require('multer');
const path = require('path');

// Configuration du stockage avec Multer
export const storage = multer.diskStorage({
  destination: (req: Request, file: any, cb: any) => {
    const uploadPath = path.join(__dirname, '../uploads'); // Chemin vers le répertoire uploads
    cb(null, uploadPath);
  },
  filename: (req: Request, file: any, cb: any) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    cb(null, uniqueSuffix + path.extname(file.originalname)); // Génère un nom unique
  }
});

// Filtrer les types de fichiers autorisés
export const fileFilter = (req: Request, file: any, cb: any) => {
  const allowedFileTypes = ['.jpeg', '.jpg', '.png', '.gif', '.zip', '.exe'];
  const fileExt = path.extname(file.originalname).toLowerCase();
  if (allowedFileTypes.includes(fileExt)) {
    cb(null, true);
  } else {
    cb(new Error(`Type de fichier non autorisé : ${fileExt}`), false);
  }
};

// Initialiser Multer avec le filtre et la limite de taille (ici 100 Mo)
export const upload = multer({
  storage: storage,
  limits: { fileSize: 1024 * 1024 * 100 }, // 100 Mo de limite de taille
  fileFilter: fileFilter
});
```

Configuration du stockage

Multer est configuré pour stocker les fichiers dans un répertoire spécifié (`../uploads`). La fonction `destination` définit ce chemin, tandis que la fonction `filename` génère un nom de fichier unique en ajoutant un suffixe basé sur l'horodatage et un nombre aléatoire, tout en conservant l'extension d'origine du fichier.

Filtrage des fichiers

Un filtre de fichiers est également défini pour n'autoriser que certains types de fichiers, tels que JPEG, PNG, GIF, ZIP, et EXE. Si le type de fichier est autorisé, il est accepté ; sinon, une erreur est retournée.

Limites de taille

Enfin, la configuration inclut une limite de taille de 100 Mo pour les fichiers téléchargés, ce qui aide à éviter les abus et à gérer l'espace de stockage.

Cette configuration permet de s'assurer que seuls des fichiers valides et de taille acceptable sont téléchargés, tout en les organisant de manière efficace dans le répertoire désigné.

3. Stockage des fichiers

- **Gestion du stockage** : Les fichiers téléchargés doivent être stockés de manière sécurisée sur le serveur ou dans un service de cloud.
- **Organisation des fichiers** : Les fichiers doivent être organisés de manière à faciliter leur récupération (par exemple, en créant des sous-dossiers pour chaque utilisateur ou catégorie).

4. Récupération des fichiers

- **Endpoint de récupération** : Fournir un endpoint permettant aux utilisateurs de récupérer les fichiers téléchargés. Par exemple, un endpoint `/api/upload/:id` pour récupérer un fichier spécifique.

5.

```
FileRoute.get('/image/:gameId', async (req, res) => {
  const { gameId } = req.params;

  try {
    const parsedGameId = parseInt(gameId, 10);
    if (isNaN(parsedGameId)) {
      return res.status(400).json({ error: 'gameId invalide' });
    }

    // Rechercher le fichier dans la base de données
    const file = await FileUpload.findOne({ where: { gameId: parsedGameId } });

    if (!file) {
      return res.status(404).json({ error: 'Aucune image trouvée pour ce jeu' });
    }

    // Générez l'URL du fichier en fonction du chemin du fichier dans la base de données
    const fileUrl = `http://localhost:9091/uploads/${path.basename(file.dataValues.pathname)}`;
    console.log('URL générée :', fileUrl); // Vérifie l'URL dans les logs

    res.json({ fileUrl });
  } catch (error) {
    console.error('Erreur lors de la récupération de l\'image', error);
    res.status(500).json({ error: 'Erreur lors de la récupération de l\'image' });
  }
});
```

Vérification et validation

- **Vérification de l'intégrité** : L'API doit vérifier l'intégrité des fichiers téléchargés (ex. : vérification de la taille et du type de fichier).
- **Validation des données** : Avant d'enregistrer un fichier, l'API doit s'assurer que toutes les données nécessaires (comme l'ID de l'utilisateur et le nom du fichier) sont fournies et valides.

6. Notifications et Retours

- **Notifications** : Informer les utilisateurs sur le succès ou l'échec de l'opération de téléchargement.

- **Retour d'erreur :** Fournir des messages d'erreur clairs et détaillés en cas de problème lors du téléchargement ou de la récupération des fichiers.
-

7. Sécurité

- **Protection contre les attaques :** Mettre en place des mesures de sécurité, comme la protection CSRF, pour sécuriser les endpoints de l'API.
- **Contrôle d'accès :** L'API doit s'assurer que seuls les utilisateurs ayant les droits appropriés peuvent accéder aux fonctionnalités d'upload et de récupération.

3. Front-end PlayForge

1. Gestion des utilisateurs

- **Inscription et connexion :**

Les utilisateurs doivent pouvoir créer un compte en fournissant un nom, un prénom, un nom d'utilisateur, une adresse e-mail et un mot de passe.

Authentification des utilisateurs via un système de connexion sécurisé.

- **Gestion des profils :**

Les utilisateurs doivent pouvoir modifier leurs informations de profil, y compris leur photo, nom et description.

2. Système de bibliothèque

- **Affichage des jeux :**

Les utilisateurs doivent pouvoir consulter tous les jeux qu'ils ont achetés ou joués dans leur bibliothèque.

- **Statut des jeux :**

Afficher le statut de chaque jeu (téléchargé ou non) dans la bibliothèque.

- **Ajout de jeux :**

Permettre aux utilisateurs d'ajouter des jeux à leur bibliothèque.

3. Consultation et jeu

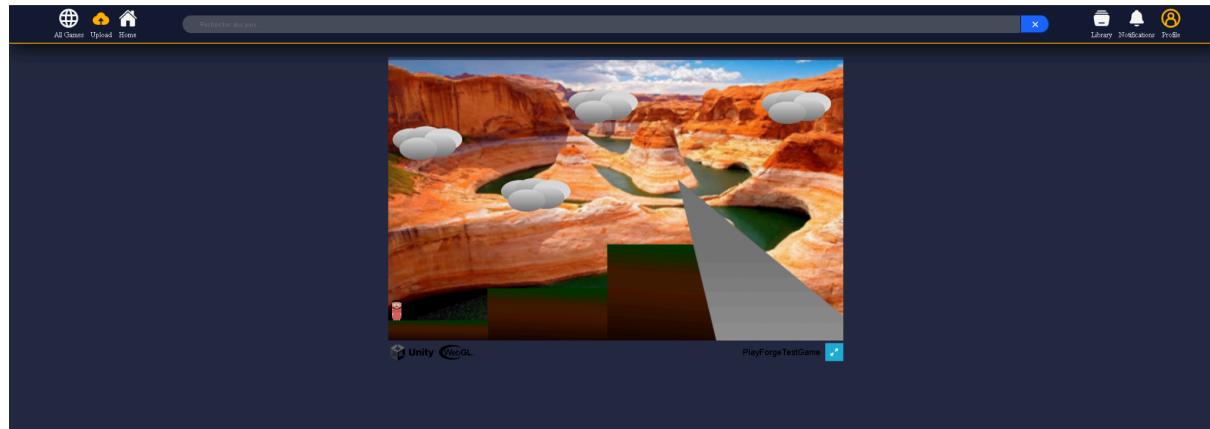
- **Navigation intuitive :**

Interface utilisateur claire et intuitive pour naviguer entre les différents jeux.



- Jouer dans le navigateur :

Les utilisateurs doivent pouvoir jouer à des jeux directement dans leur navigateur, avec une expérience fluide et réactive.



4. Système de notifications

- Alertes :

Les utilisateurs doivent recevoir des notifications pour les mises à jour des jeux, les nouveaux commentaires et autres activités pertinentes.

5. Système de commentaires et de notes

- Ajouter des commentaires :

Les utilisateurs doivent pouvoir laisser des commentaires sur les jeux.

- Notation des jeux :

Permettre aux utilisateurs de noter les jeux, contribuant ainsi à l'engagement de la communauté.

6. Gestion des rôles

- Affichage des fonctionnalités selon le rôle :

Différencier les fonctionnalités accessibles aux Utilisateurs et aux Développeurs dans l'interface, en adaptant l'affichage en fonction du rôle de l'utilisateur connecté.

7. Intégration avec le back-end

- **Appels API :**

Utiliser des services Angular pour interagir avec l'API backend pour récupérer, ajouter, et gérer les données (jeux, commentaires, utilisateurs).

```
import { ReturnStatement } from '@angular/compiler';
import { Injectable, OnInit } from '@angular/core';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class GameService {

  private apiUrl = 'http://localhost:9090/game/new';
  private apiUrlAllGames = 'http://localhost:9090/game/AllGames'
  constructor() { }

  > sendGameData(gameData: any): Promise<any> { ... }
  >
  > async getAllGames(): Promise<any> { ... }
  >
  > async getGameById(gameId: string): Promise<any> { ... }
  >
  > async getGamesByDate(): Promise<any> { ... }
  >
  > async searchGames(query: string): Promise<any> { ... }
  >
  > async associateGameWithCategories(GameId: any, ControllerId: any, PlatformId: any, LanguageId :any, StatusId :any) { ... }
}
```

GameService qui interagit avec le Backend pour ajouter / récupérer

- **Gestion des erreurs :**

Afficher des messages d'erreur clairs et utiles en cas de problèmes lors des appels API.

```
async getAllGames(): Promise<any> {
  try {

    const response = await fetch(this.apiUrlAllGames, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      throw new Error('Erreur lors de la récupération des jeux : ' + response.statusText);
    };

    const data = await response.json();
    return data;
  } catch (error) {
    console.log('erreur getAllGames()')
    console.error('Erreur:', error);
    throw error;
  }
}
```

Promesse
getAllGames() qui
 renvoie une erreur
 ('Erreur lors de la
 récupération des

jeux : ‘ + response.statusText) si la réponse de l’Api est pas bonne

8. Accessibilité et réactivité

- **Responsive Design :**

L’interface doit être optimisée pour s’afficher correctement sur différents appareils (ordinateurs, tablettes, téléphones).

```
@media (max-width:1600px){  
  
    .card-game{  
        width: 100vw;  
        padding: 2%;  
        display: grid;  
        grid-template-columns: 1fr 1fr 1fr;  
        gap: 2%;  
    }  
  
}  
  
@media (max-width:768px){  
  
    .card-game{  
        width: 100vw;  
        height: 100%;  
        padding: 5%;  
        display: flex;  
        flex-direction: column;  
    }  
  
}
```

9. Performances

- **Chargement rapide :**

Optimiser le chargement des jeux et des données pour garantir une expérience utilisateur fluide et rapide.

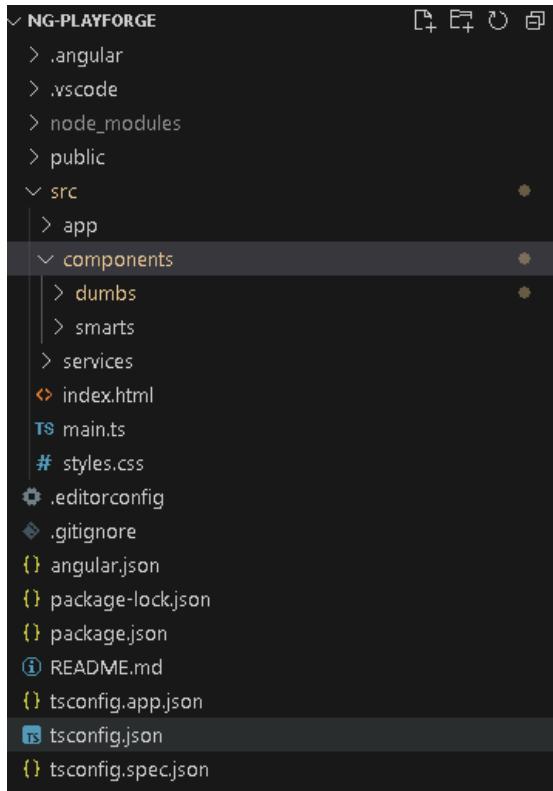
10. Sécurité

- **Protection des données :**

S'assurer que les données des utilisateurs sont sécurisées lors des interactions avec l'API et que les pratiques de sécurité appropriées sont mises en œuvre.

VI. Spécification techniques du projet

1. Technologie Utilisés



Front-end :

Framework : Angular (version 11 ou supérieure).

Langage : TypeScript.

CSS : Utilisation de CSS ou de préprocesseurs comme SASS pour le style.



Back-end :

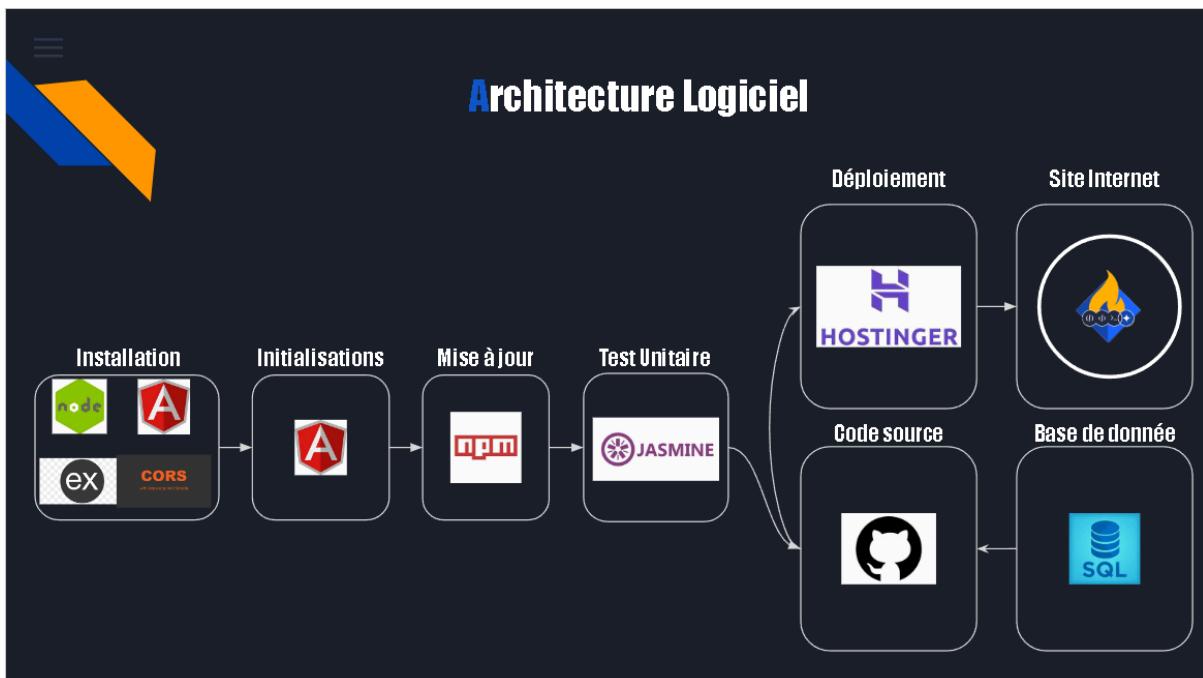
- **Serveur : Node.js avec Express.**
- **Langage : TypeScript.**
- **Base de données : MySQL ou une autre base de données relationnelle.**
- **Middleware : CORS pour la gestion des requêtes cross-origin.**

```
└─ APIUPLOAD
    └─ build
    └─ node_modules
    └─ src
        └─ config
            └─ multer.ts
        └─ Models
            └─ File.ts
            └─ Image.ts
            └─ app.ts
            └─ database.ts
        └─ package-lock.json
        └─ package.json
        └─ README.md
        └─ tsconfig.json
```

Stockage de fichiers :

- Utilisation de Multer pour gérer les uploads de fichiers.

• Architecture Logiciel:



2.Les Contraintes

Contraintes Techniques

- **Compatibilité Navigateur :**
 - L'application a été développée en utilisant Angular, qui est conçu pour être compatible avec les navigateurs modernes. Des tests ont été effectués sur les principaux navigateurs (Chrome, Firefox, Safari, Edge) pour garantir une expérience utilisateur cohérente.
 -
- **Sécurité :**
 - Utilisation de JSON Web Tokens (JWT) pour sécuriser les sessions utilisateurs.
 - Protection contre les attaques courantes (XSS, CSRF, injections SQL).
 -
- **Accessibilité :**
 - Conformité aux normes WCAG (Web Content Accessibility Guidelines) pour garantir que l'application soit utilisable par tous, y compris les personnes ayant des handicaps.
- **Responsive Design :**
 - L'application est entièrement responsive, s'adaptant aux différentes tailles d'écran, y compris les ordinateurs de bureau, les tablettes et les smartphones.

Contraintes de Développement

- **Méthodologie :**
 - Utilisation de méthodes agiles avec des sprints pour le développement itératif et incrémental.
- **Environnement de Développement :**
 - Utilisation de TypeScript avec un compilateur configuré pour assurer la conformité et la qualité du code.
 - Utilisation de Git pour la gestion de version.
- **Tests :**
 - Mise en place de tests unitaires et d'intégration avec des frameworks comme Jasmine et Karma pour Angular.
 - Tests fonctionnels pour l'API avec des outils comme Postman.

3. Principe du développement MVC et application en couches

Le modèle MVC (Modèle-Vue-Contrôleur) est une architecture de développement logiciel qui divise une application en trois composants principaux :

- **Modèle (Model) :**
 - Représente les données de l'application et la logique métier. Il gère l'accès aux données, la validation et la manipulation des données. Le modèle est responsable de l'état de l'application et est indépendant de la présentation.
- **Vue (View) :**
 - C'est l'interface utilisateur de l'application. La vue est chargée de l'affichage des données provenant du modèle et de la présentation des informations à l'utilisateur. Elle écoute les interactions de l'utilisateur et envoie les actions au contrôleur.
- **Contrôleur (Controller) :**
 - Agit comme un intermédiaire entre le modèle et la vue. Le contrôleur reçoit les entrées de l'utilisateur via la vue, interagit avec le modèle pour traiter les données, puis met à jour la vue en conséquence. Il orchestre les actions de l'application.

PlayForge est structurée en couche pour plusieurs raison :

Séparation des responsabilités :

- Chaque couche a une responsabilité distincte (données, logique métier, présentation), ce qui rend le code plus organisé et facile à maintenir. Cela permet également de réduire les dépendances entre les différentes parties de l'application.

PlayForge :

- Front-End (Angular)
- Back-End (Node.js)
- API-Upload (Multer)

Facilité de maintenance et d'évolutivité :

- En séparant les préoccupations, il est plus facile de modifier ou d'améliorer une couche sans affecter les autres. Par exemple, vous pouvez changer la logique de l'application ou la présentation sans devoir toucher à la couche des données.

Réutilisation de code :

- Les couches permettent de réutiliser du code. Par exemple, des composants de la couche de présentation peuvent être réutilisés avec différents modèles de données ou logiques métiers.

Réutilisation du composant <app-nav-item> à plusieurs reprise (ci-dessous)

```
<nav>
    <ul class="nav">

        <li><app-nav-item text="All Games" img="/ion_network.svg"
            routerLink="/all-game"></app-nav-item></li>
        <li><app-nav-item text="Upload" img="/ion_upload.svg"
            routerLink="/game"></app-nav-item></li>
        <li><app-nav-item text="Home" img="/ion_home.svg"
            routerLink="/"></app-nav-item></li>

    </ul>

    <div class="app-searchbar">
        <app-searchbar></app-searchbar>
    </div>

    <div class="mobile-menu">
        <app-mobile-menu></app-mobile-menu>
    </div>

    <ul class="nav">

        <li><app-nav-item text="Library" img="/ion_library.svg"
            routerLink="/library"></app-nav-item></li>
        <li><app-nav-item text="Notifications"
            img="ion_notification.svg"></app-nav-item></li>
        <li><app-nav-item text="Profile" img="ion_profile_color.svg"
            routerLink="/profile"></app-nav-item></li>

    </ul>

</nav>
```

Testabilité :

- Les tests peuvent être effectués de manière isolée sur chaque couche. Cela signifie que les développeurs peuvent tester la logique métier indépendamment de l'interface utilisateur, facilitant ainsi le processus de débogage et de validation.

```
it('devrait uploader un fichier avec succès', async () => {
  const mockFile = new File(['dummy content'], 'example.txt', { type: 'text/plain' });
  const mockResponse = { fileUrl: 'http://localhost:9091/files/example.txt' };
  const gameId = 123;

  service.uploadFile(mockFile, gameId).then((response) => {
    expect(response.fileUrl).toBe('http://localhost:9091/files/example.txt');
  });

  const req = httpMock.expectOne('http://localhost:9091/game/upload/file');
  expect(req.request.method).toBe('POST');
  req.flush(mockResponse); // Simule la réponse de l'API
});
```

Flexibilité dans les technologies :

- La structure en couches permet de remplacer ou de mettre à jour une couche sans avoir à modifier l'ensemble de l'application. Par exemple, vous pourriez décider de changer de technologie pour le stockage des données sans affecter la présentation ou la logique. Ce qui est au contraire se produit lors d'une réalisation d'un projet MONOLITHIQUE où aucune couche n'est isolée.

Clarté et lisibilité du code :

- Une architecture en couches rend le code plus clair et plus lisible. Les développeurs peuvent comprendre rapidement la structure de l'application et comment les différentes parties interagissent.

VII. Réalisation significative et argumentation

1. Aspect Graphique

Concernant l'Aspect Graphique, j'ai choisi un fond sombre pour le projet PlayForge, en ajoutant deux couleurs vives ( : #ffb000 et  #1a68ff) pour mettre en valeur les éléments importants. Ce choix reflète plusieurs considérations personnelles et esthétiques.

D'abord, un fond sombre ( : #252B43) offre un confort visuel appréciable, surtout lors de longues sessions de jeu. Je voulais créer une ambiance qui réduise la fatigue oculaire, permettant aux utilisateurs de se concentrer sans se sentir gênés.

Ensuite, l'aspect moderne et élégant du design sombre s'inscrit bien dans l'univers du gaming, attirant ainsi des passionnés qui cherchent une expérience visuelle engageante. Les couleurs vives ajoutent une touche dynamique, attirant l'attention sur des éléments clés comme les boutons d'action et les notifications. Cela facilite la navigation et rend l'expérience plus intuitive.

Enfin, le contraste entre le texte blanc et le fond sombre garantit une excellente lisibilité. Je voulais que chaque utilisateur puisse lire facilement les informations sans effort.

En conclusion, ce choix de couleurs vise à offrir une expérience agréable et immersive, tout en rendant l'application attrayante et facile à utiliser.

2. Base de donnée / Api-Upload

J'ai choisi de séparer ma base de données de celle de mon API pour plusieurs raisons stratégiques et techniques. Tout d'abord, cette séparation améliore la **sécurité** des données. En isolant les bases de données, je peux appliquer des mesures de sécurité spécifiques à chaque système, réduisant ainsi les risques d'accès non autorisé. Cela permet de mieux contrôler qui a accès à quelles données, renforçant ainsi la protection des informations sensibles.

Ensuite, cela facilite la **gestion et la maintenance**. En ayant des bases de données distinctes, je peux effectuer des mises à jour, des sauvegardes et des optimisations sans affecter l'autre système.

Cela permet également de réduire les temps d'arrêt potentiels, assurant une meilleure disponibilité des services pour les utilisateurs.

De plus, cette approche permet une **scalabilité** accrue. En séparant les données, je peux adapter chaque base de données en fonction de ses besoins spécifiques. Par exemple, si l'API nécessite plus de ressources, je peux la faire évoluer indépendamment de la base de données principale.

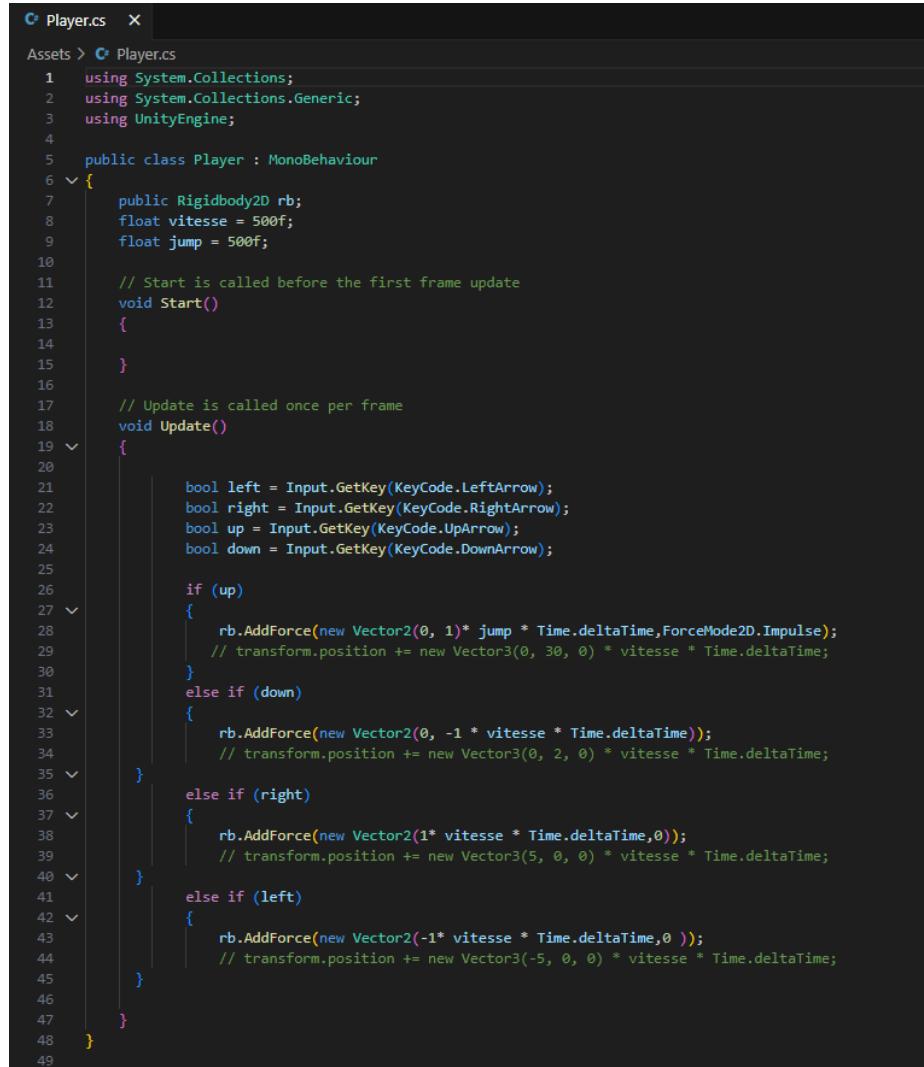
Enfin, cette séparation favorise une meilleure **organisation** des données. Chaque base de données peut être conçue et optimisée en fonction des besoins particuliers de l'application ou de l'API, ce qui simplifie le développement et améliore l'efficacité des requêtes.

IX. Présentation des key features et jeu d'essaie

1. Jouer à un jeu sur navigateur

Pour pouvoir jouer à un jeu, j'ai décidé de le créer moi-même afin d'explorer de nouveaux horizons. J'ai choisi de tester Unity, ce qui impliquait l'apprentissage d'un nouveau langage : le C#. Mon objectif initial était de réaliser un jeu test. J'ai donc créé un nouveau projet dans Unity et commencé par suivre le tutoriel "Get Started". Après quelques heures de pratique, j'ai réussi à développer un petit jeu très simple, sans véritable début ni fin, mais comprenant des scripts attachés à des objets du décor ainsi qu'au personnage.

Script Player :



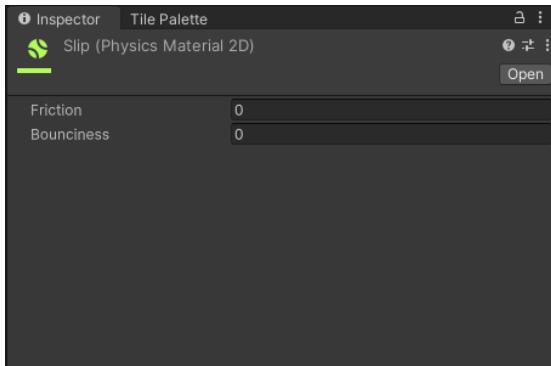
The screenshot shows a code editor window with the file 'Player.cs' open. The code is written in C# and defines a MonoBehaviour script for a player character. It includes declarations for a Rigidbody2D component and variables for speed and jump height. The Start() method is empty. The Update() method handles movement based on key inputs (up, down, left, right) by applying forces to the rigidbody. The code uses Vector2 and Vector3 for force calculations and Time.deltaTime for frame-based timing.

```
 1  using System.Collections;
 2  using System.Collections.Generic;
 3  using UnityEngine;
 4
 5  public class Player : MonoBehaviour
 6  {
 7      public Rigidbody2D rb;
 8      float vitesse = 500f;
 9      float jump = 500f;
10
11     // Start is called before the first frame update
12     void Start()
13     {
14     }
15
16
17     // Update is called once per frame
18     void Update()
19     {
20
21         bool left = Input.GetKey(KeyCode.LeftArrow);
22         bool right = Input.GetKey(KeyCode.RightArrow);
23         bool up = Input.GetKey(KeyCode.UpArrow);
24         bool down = Input.GetKey(KeyCode.DownArrow);
25
26         if (up)
27         {
28             rb.AddForce(new Vector2(0, 1)* jump * Time.deltaTime,ForceMode2D.Impulse);
29             // transform.position += new Vector3(0, 30, 0) * vitesse * Time.deltaTime;
30         }
31         else if (down)
32         {
33             rb.AddForce(new Vector2(0, -1 * vitesse * Time.deltaTime));
34             // transform.position += new Vector3(0, 2, 0) * vitesse * Time.deltaTime;
35         }
36         else if (right)
37         {
38             rb.AddForce(new Vector2(1* vitesse * Time.deltaTime,0));
39             // transform.position += new Vector3(5, 0, 0) * vitesse * Time.deltaTime;
40         }
41         else if (left)
42         {
43             rb.AddForce(new Vector2(-1* vitesse * Time.deltaTime,0 ));
44             // transform.position += new Vector3(-5, 0, 0) * vitesse * Time.deltaTime;
45         }
46
47     }
48 }
```

Le script du joueur récupère les entrées des flèches du clavier à l'aide de `Input.GetKeyDown(KeyCode.Arrow)`.

Ensuite, en utilisant des conditions simples telles que `if` et `else if`, j'applique une force sur ces entrées pour faire avancer le personnage dans la direction souhaitée.

Physics Décors :



Slip (Physics Materials 2D) est un type de matériau physique dans Unity où la friction et la bounciness sont réglées à 0, permettant ainsi aux objets de glisser facilement sur les surfaces. Cela est utile pour créer des interactions fluides, comme des personnages ou des objets qui glissent sans résistance sur le sol, offrant ainsi une expérience de jeu dynamique.



Bounce (Physics Materials 2D) est un type de matériau physique dans Unity où la friction est fixée à 0 et la bounciness est réglée à 1,13. Cela permet aux objets de rebondir de manière dynamique lors des collisions, tout en glissant sans résistance sur les surfaces. Ce matériau est idéal pour créer des interactions amusantes et réactives, comme des balles qui rebondissent avec énergie.

Les **Materials Physics** de Unity permettent de gérer la friction et le rebond des objets. La **friction** contrôle la résistance au glissement sur une surface, tandis que la **bounciness** détermine le rebond lors des collisions. Ces paramètres permettent de créer des interactions physiques réalistes entre les objets du jeu.

Suite à ça, je me suis amusé à réalisé des décors sur figma au format .jpeg :



Player



Grass



rock



Cloud

Le décor et les scripts ont été réalisés et sont fonctionnels. Il ne reste plus qu'à exporter le jeu. Pour ce faire, il suffit d'aller dans **File**, puis **Build Settings**. Ensuite, il faut régler la **max texture size** à Max512 et choisir **Force Fast Compression** pour la compression des textures.

Une fois le fichier .zip téléchargé, il fallait le transférer sur un serveur pour pouvoir y jouer. J'ai rencontré un problème en essayant de le mettre sur un serveur Python, car Unity nécessite un serveur HTTPS, c'est-à-dire sécurisé, et le serveur Python ne supporte pas le HTTPS. Pour contourner ce problème, j'ai mis en place un serveur capable de gérer les connexions sécurisées avec **Apache**.

Cependant, j'ai également rencontré des problèmes de compression et de validation avec Unity. Une solution m'a alors été proposée : obtenir un sous-domaine d'un domaine via Hostinger. Mon formateur m'a accordé l'accès via FileZilla, ce qui m'a permis d'effectuer mes tests et de publier mon jeu sur le site <https://mediumslateblue-cod-630349.hostingersite.com/>.

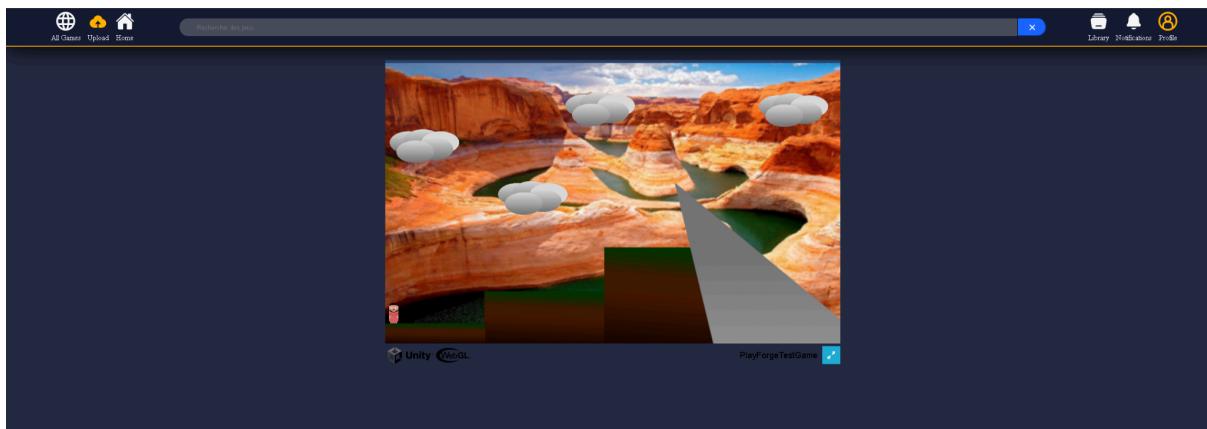
Ainsi, avec mon site en ligne opérationnel, il ne me reste plus qu'à l'intégrer sur ma page en utilisant une balise `<iframe>`, qui prend en source l'URL du site où le jeu est accessible. Cela permettra aux utilisateurs de jouer directement depuis mon site sans avoir à naviguer ailleurs.

```
src > components > dumbs > play-in-browser > play-in-browser.component.html > div.browser
    Go to component
1   <div class="browser">
2     <h1>{{titleGame}}</h1>
3     <iframe [src]="'urlSafe'" frameborder="0" width="100%" height="700vh"></iframe>
4   </div>
```

```
src > components > dumbs > play-in-browser > play-in-browser.component.ts > ...
1  import { Component, Input } from '@angular/core';
2  import { DomSanitizer, SafeResourceUrl } from '@angular/platform-browser';
3
4  @Component({
5    selector: 'app-play-in-browser',
6    standalone: true,
7    imports: [],
8    templateUrl: './play-in-browser.component.html',
9    styleUrls: ['./play-in-browser.component.css']
10 })
11 export class PlayInBrowserComponent {
12   @Input() url: string = 'https://mediumslateblue-cod-630349.hostingersite.com/';
13   urlSafe: SafeResourceUrl | undefined;
14   @Input() titleGame = "";
15
16   constructor(public sanitizer: DomSanitizer) {}
17
18   ngOnInit() {
19     this.urlSafe = this.sanitizer.bypassSecurityTrustResourceUrl(this.url);
20   }
21
22 }
```

Grâce à toutes les mesures prises et au code établi, il ne me reste plus qu'à appeler mon application **app-play-in-browser**. Cela marquera l'achèvement du processus et permettra aux utilisateurs de profiter du jeu directement depuis leur navigateur.

Sur le site PlayForge, les utilisateurs peuvent jouer en utilisant uniquement les entrées du clavier, ce qui assure une expérience de jeu fluide et accessible :



Annexe :

Unity : est un moteur de développement de jeux multiplateforme qui permet la création de jeux 2D et 3D, offrant des outils avancés pour le rendu graphique, l'animation, et le scripting, ainsi qu'un accès à une vaste bibliothèque d'actifs.

Hostinger: Hostinger est un fournisseur d'hébergement web offrant des solutions abordables, des performances optimales et un support technique 24/7 pour les utilisateurs

2. Ajouter un jeu à la BDD

Pour ajouter un jeu à ma base de données, j'ai conçu deux formulaires dans mon front-end Angular : **PostForm Game** et **Category CreateForm**. Ces formulaires récupèrent les valeurs saisies et, grâce à la fonction **emit**, les transmettent au composant parent **GameComponent**.

Dans ce composant parent, nous récupérons les informations des deux formulaires. Un bouton dans le HTML déclenche une fonction **onSubmit**, qui permet de récupérer ces valeurs et de les envoyer via un service dédié.

```
async onSubmit() {
  if (this.postGameForm && this.categoryCreateForm) {
    const postGameFormValues = this.postGameForm.getFormValue();
    const categoryCreateFormValues = this.categoryCreateForm.getFormValue();

    const gameData = {
      ...postGameFormValues,
      ...categoryCreateFormValues,
    };

    try {
      // Envoyer les données du jeu
      const result = await this.gameService.sendGameData(gameData);
      console.log('Jeu créé avec succès:', result);

      const gameId = result.id;
      const controllerId = gameData.ControllerId;
    }
  }
}
```

Cette capture d'écran montre une partie de la fonction **onSubmit()** où nous récupérons correctement les informations dans **gameData**. On peut observer que les valeurs saisies dans les formulaires **PostForm Game** et **Category CreateForm**

sont bien extraites et stockées dans l'objet **gameData**. Cela garantit que toutes les données nécessaires sont prêtes à être envoyées au service dédié pour l'ajout à la base de données.

Une fois les informations récupérées, nous écoutons le service dédié pour envoyer ces données, tout en passant d'autres informations en parallèle, telles que **StatusId** et **LanguageId**, dans le modèle Game. De plus, nous envoyons l'URL d'une image à un autre service qui écoute une route POST pour ajouter cette image à l'API d'upload. Cette API récupérera également le **GameId** lors de la création, assurant ainsi une association correcte entre le jeu et son image.

```

async onSubmit() {
  if (this.postGameForm && this.categoryCreateForm) {
    const postGameFormValues = this.postGameForm.getFormValues();
    const categoryCreateFormValues = this.categoryCreateForm.getFormValues();

    const gameData = {
      ...postGameFormValues,
      ...categoryCreateFormValues,
    };

    try {
      // Envoyer les données du jeu
      const result = await this.gameService.sendGameData(gameData);
      console.log('Jeu créé avec succès', result);

      const GameId = result.id;
      const ControllerId = gameData.ControllerId;
      const PlatformId = gameData.PlatformId;
      const LanguageId = gameData.LanguageId;
      const StatusId = gameData.StatusId

      console.log(GameId)
      console.log(ControllerId)
      console.log(PlatformId)
      console.log(LanguageId)
      console.log(StatusId)

      // Associer l'élément sélectionné au jeu
      if (GameId || ControllerId || PlatformId || StatusId || LanguageId) {
        await this.gameserver.associateGameWithCategories(GameId, ControllerId, PlatformId, StatusId, LanguageId);
        console.log('Élément associé avec succès au jeu');
      }

      // Si un fichier est sélectionné, upload le fichier
      if (this.postGameForm.selectedFile) {
        await this.fileservice.uploadFile(this.postGameForm.selectedFile, GameId);
        console.log('Image uploadée avec succès');
      }

      // Réinitialiser les formulaires
      this.postGameForm.resetForm();
      this.categoryCreateForm.resetForm();
    } catch (error) {
      console.error('Erreur lors de la création du jeu:', error);
    }
  }
}

```

Nous vérifions les informations envoyées dans le back-end en utilisant des **console.log**, ce qui nous permet de les afficher dans la console du navigateur. Cela nous aide à nous assurer que les données sont correctement transmises et à identifier d'éventuels problèmes lors du processus de communication entre le front-end et le back-end.

```
ID sélectionné dans Controllers: 2                                         category-create.component.ts:68
ID sélectionné dans Status: 3                                         category-create.component.ts:72
ID sélectionné dans Platforms: 3                                         category-create.component.ts:76
ID sélectionné dans Language: 13                                         category-create.component.ts:80
ID sélectionné dans Language: Languages                                         category-create.component.ts:80
ID sélectionné dans Language: 3                                         category-create.component.ts:80
Jeu créé avec succès:                                                 game.component.ts:49
▶ {createdAt: '2024-09-27T18:22:13.578Z', updatedAt: '2024-09-27T18:22:13.578Z', id: 20, title: 'zfezfze', description: 'fzefzfefz
f', ...}                                         game.component.ts:59
2                                         game.component.ts:60
3                                         game.component.ts:61
3                                         game.component.ts:62
3                                         game.component.ts:63
Élément associé avec succès au jeu                                         game.component.ts:69
Image uploadée avec succès                                         game.component.ts:75
```

Maintenant, il ne reste plus qu'à vérifier les informations demandées par le back-end. La route POST de **Game** nécessite les éléments suivants :

```
FileRoute.post('/upload/file', upload.single('file'), async (req, res) => {
  let { gameId } = req.body;
  gameId = parseInt(gameId, 10);
  req.file as Express.Multer.File
  try {
    if (!req.file || !gameId) {
      return res.status(400).json({ error: 'No File or No GameId' });
    }

    const gameResponse = await fetch(`http://localhost:9090/game/id/${gameId}`);
    if (!gameResponse.ok) {
      return res.status(404).json({ error: 'GameId not found in the database.' });
    }

    const createdfile = await FileUpload.create({
      filename: req.file.filename,
      filepath: req.file.path,
      fileType: path.extname(req.file.originalname),
      fileSize: req.file.size,
      gameId: gameId
    });

    res.status(201).json({
      message: 'Fichier uploadé avec succès',
      file: createdfile,
      fileUrl: `http://localhost:9091/uploads/${req.file.filename}`
    });
  } catch (error) {
    console.error('Erreur lors du téléchargement', error);
    res.status(500).json({ error: 'Erreur lors du téléchargement du fichier' });
  }
});
```

La route POST de mon **API d'upload** pour le téléchargement de fichiers nécessite les éléments suivants :

```
GameRoute.post('/new', async (req, res) => {
  const { title, description, price, authorStudio, madewith, StatusId, LanguageId } = req.body;

  try {
    const game = await Game.create({ title, description, price, authorStudio, madewith, StatusId, LanguageId });

    res.status(201).json(game);
  } catch (error) {
    console.error('Erreur lors de la création du jeu :', error);
    res.status(500).json({ error: 'Erreur lors de la création du jeu' });
  }
});
```

Ainsi, si aucune erreur ne se produit, les données devraient apparaître dans la vue de ma base de données via phpMyAdmin, qui est lancée via Docker Compose. Cette configuration permet également de démarrer la base de données MySQL en parallèle, assurant un environnement de développement cohérent et efficace.

Nous pouvons apercevoir le jeu créé contenant les informations nécessaires:

<input type="checkbox"/>	Éditer	Copier	Supprimer	19	zlezelzefzez	17	zlezelzefzez	NULL	zlezelzefzez	2024-09-27 18:09:11	2024-09-27 18:09:11	1	4
--------------------------	--------	--------	-----------	----	--------------	----	--------------	------	--------------	---------------------	---------------------	---	---

La table de jointure de Game complete (GameController) :

<input type="checkbox"/>	Éditer	Copier	Supprimer	2024-09-27 18:09:11	2024-09-27 18:09:11	19	1	2024-09-27 18:22:13	2024-09-27 18:22:13	20	2
<input type="checkbox"/>	Éditer	Copier	Supprimer	2024-09-27 18:22:13	2024-09-27 18:22:13	20	2				

La table de jointures de Game complete (GamePlatforms) :

<input type="checkbox"/>	Éditer	Copier	Supprimer	2024-09-27 18:09:12	2024-09-27 18:09:12	19	2	2024-09-27 18:22:13	2024-09-27 18:22:13	20	3
<input type="checkbox"/>	Éditer	Copier	Supprimer	2024-09-27 18:22:13	2024-09-27 18:22:13	20	3				

Le jeu est donc bel et bien créé et associé à ses tables de jointure. De plus, le système d'upload de fichiers récupère également le **GameId** de manière correcte, assurant ainsi que toutes les informations liées au jeu sont bien intégrées dans la base de données.

<input type="checkbox"/>	Éditer	Copier	Supprimer	1	1727461333671-483005310.png	C:\Users\Admin\OneDrive\Bureau\PlayForge\api-uploa...	.png	10131	2024-09-27 18:22:13	2024-09-27 18:22:13	20	2024-09-27 18:22:13	2024-09-27 18:22:13	
<input type="checkbox"/>	Éditer	Copier	Supprimer											