



GAMESUP



Nom et prénom du stagiaire : Jordy AKRA MESCHEBA



Contexte

Il s'agit d'un projet de refonte du site **GameUp**, une plateforme de vente de jeux de société en ligne. La plateforme GameUp permet de gérer un catalogue de jeux de sociétés, d'enregistrer les utilisateurs et leurs achats et enfin de faire des recommandations des jeux aux clients en fonction des articles visités par chaque client sur le site.

La problématique est que la plateforme connaît quelques bugs depuis quelques temps, empêchant les clients de l'utiliser et ainsi de bénéficier de l'entièreté des fonctionnalités et cela fait baisser les ventes sur le site.

Objectif

Le backend du site est devenu obsolète et il va falloir le refondre pour faire fonctionner toutes les fonctionnalités **CRUD**.

Par ailleurs, le projet a été confié précédemment à un stagiaire qui a commencé la correction des bugs sur le backend. L'objectif serait de reprendre le code et l'améliorer pour faire fonctionner toutes les fonctionnalités nécessaires.

Pour cela, voici les points qui seront abordés:

- Faire un diagnostic complet des fonctionnalités de la plateforme en abordant les points essentiels des mauvaises et bonnes pratiques du code et ainsi identifier les faiblesses;
- Refondre le backend et partir sur une architecture REST modulable en appliquant les bonnes pratiques de la méthode SOLID;
- Mettre en place la sécurité de l'API en utilisant Hibernate pour la gestion de la création des tables de la base des données;
- Réaliser des tests unitaires et d'intégrations de l'API permettant de réduire les risques de crash de la plateforme.



- Rendre plus efficace les recommandations en optimisant la qualité du code de la Machine Learning;

1. Fonctionnalités attendues

Ces corrections devaient permettre l'accès à des comptes avec deux rôles différents sur le site:

Compte client

- Les utilisateurs doivent pouvoir s'inscrire et se connecter à leurs comptes utilisateurs,
- Une fois connecté, un utilisateur devrait pouvoir accéder à la page d'accueil avec la liste de jeux disponible.
- L'utilisateur peut rechercher un jeu dans la barre de recherche en saisissant le nom du jeu;
- L'utilisateur peut rechercher les jeux par catégorie et sélectionnant la catégorie souhaitée;
- L'utilisateur peut accéder aux détails d'un jeu en cliquant sur le jeu souhaité;
- L'utilisateur peut ajouter ou retirer ses jeux favoris dans une wishlist;
- L'utilisateur peut commander un jeu en cliquant sur le bouton commander accessible dans la description de chaque jeu et dans l'onglet wishlist;
- L'utilisateur peut donner son avis sur la page avec une note allant de 1 à 10 et un commentaire dans un formulaire dédié.

Compte administrateur



- Un administrateur peut accéder à la liste des utilisateurs, des jeux, des catégories, des authors et des avis des clients,
- Un administrateur peut ajouter, mettre à jour et supprimer les informations d'un utilisateur, des jeux, des authors et catégories;
- En ce qui concerne les avis des clients, l'administrateur peut uniquement accéder à la liste, lire et supprimer les avis de la liste.

Diagnostic des problème généraux dans le projet

Le code actuel présente des problèmes dans sa configuration et son architecture. Ces problèmes peuvent entraîner des problèmes liés à la sécurité des données clients, un problème de performance et de maintenabilité lié à la structure du code et un problème de scalabilité .

En effet l'architecture du backend est basée sur l'architecture mvc, ce qui est plutôt une bonne approche permettant d'avoir un projet structuré avec des composants tels que les **modèles** définissant les attributs d'une classe comme les informations nécessaire telles que un id, un nom, un email, un mot de passe qui définissent la classe user, et également pour les classes game, catégorie et bien d'autres, des **Controllers** spécifiques à chaque modèles, permettant la communication entre modèle et instruction afin de conjuguer les différents verbes du **CRUD** à savoir le **GET**, **POST**, **PUT**, **PATCH** et **DELETE**, les **services** et leurs implémentations permettant la communication avec la base des données en passant par les repositories et ainsi permettre d'avoir un code clair et automatisé.

En revanche, cette architecture ainsi que le backend dans son ensemble restent tout de même inachevés.

L'architecture actuelle se limite au modèle et à quelques controllers, ce qui rend les fonctionnalités de création , mise à jour ou suppression des jeux, des utilisateurs ou d'autres modèles impossibles, les services et leurs implémentations ne sont pas définis d'une part. D'autre part, la configuration de hibernate n'est pas complète ou non définie, les propriétés du dialect d'hibernate n'étant pas définies, cela empêche au



server de reconnaître l'url de la base des données et entraîne à la suite un crache du serveur qui empêche le serveur démarrer.

Par ailleurs, les méthodes crud ne sont pas fonctionnelles et restent inachevées dans l'ensemble et également une structure de code qui intègre directement les requêtes SQL dans le controller, cela pose un problème de sécurité et rend difficile la maintenabilité du code en générale et de la manipulation des tables de la base des données en particulier.

Enfin, les attributs de la classe de certains modèles sont incomplets et certains définis en public, ce qui les empêche de communiquer avec le reste de la logique du code et les rend également moins sécurisés.

Choix de l'architecture du projet

Afin de faire fonctionner le site et préserver la sécurité des données des utilisateurs, tout en respectant les bonnes pratiques d'écriture du code, il est essentielle de définir une architecture robuste permettant de garantir la modularité c'est à dire la séparation du projet en modules distincts ayant des rôles distincts, pouvant être développer, tester et maintenu indépendamment des autres, la scalabilité permettant au projet d'évoluer dans le temps et de s'adapter à l'évolution des technologies et la maintenabilité permettant de rendre plus facile la maintenabilité du site en automatisant certains processus.

Pour toutes ces raisons, j'ai choisi une architecture **mvc** native à java et reposant sur le principe SOLID .

Diagramme d'architecture

Ci dessous le diagramme d'architecture

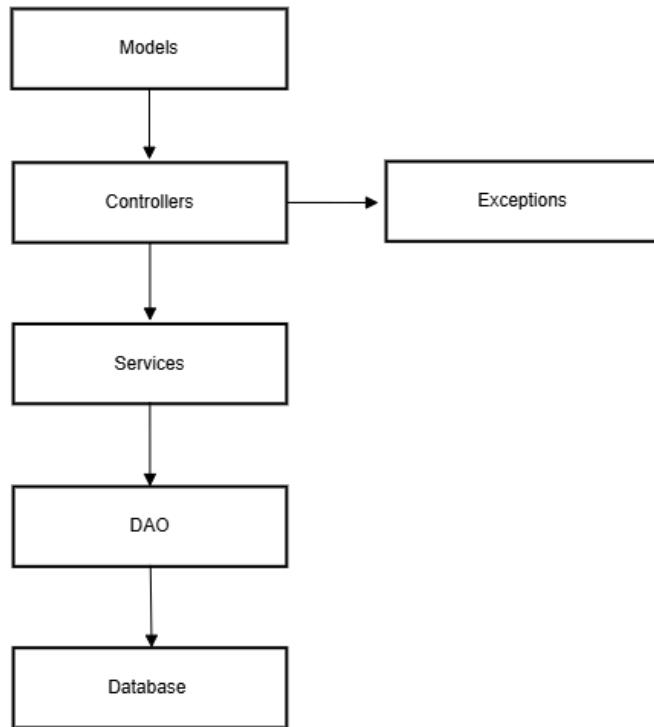


Diagramme de classes

Ci dessous le diagramme des classes descriptif de toutes les classes du projet ainsi que les relations internes **ManyToOne** ou **OneToMany** entre certaines classes. Cependant, certaines classes n'ont pas visiblement de relations entre elles, je tiens donc à préciser que certaines relations sont établies à travers l'API via les Endpoints directement dans le frontend.

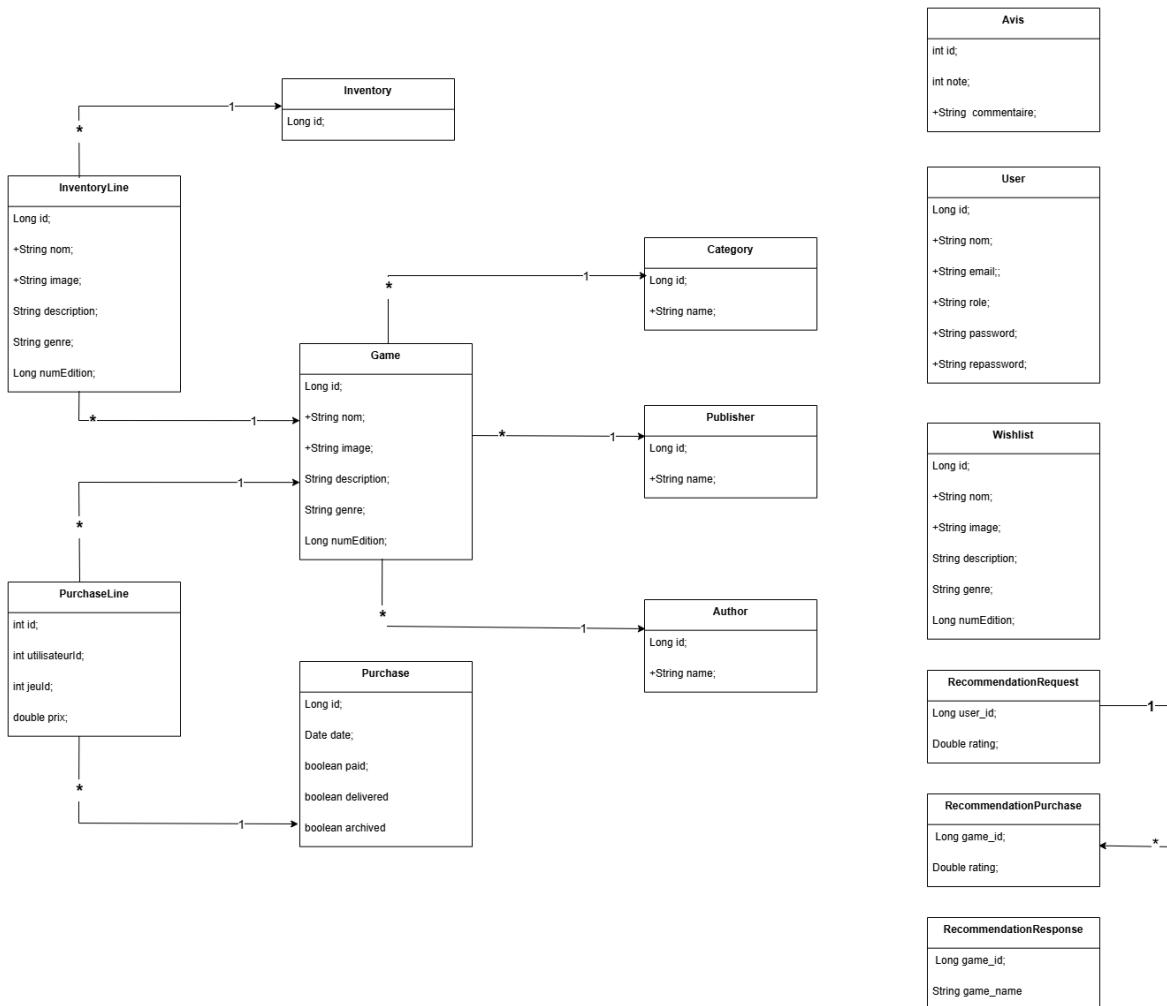




Diagramme des composants

Le diagramme ci dessous est le diagramme de composant descriptif des relations et les dépendances entre les composants

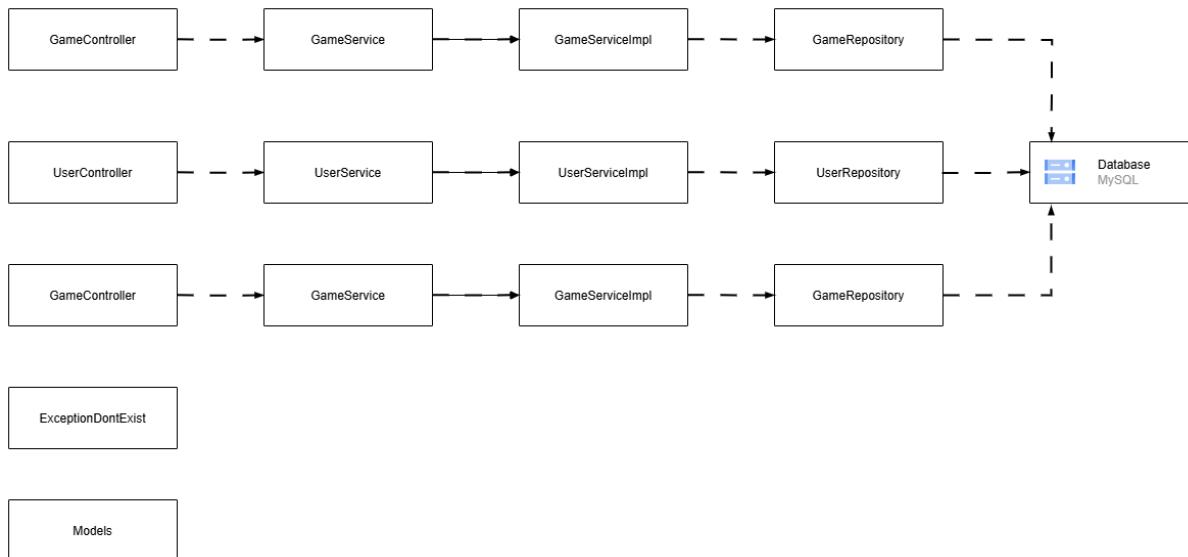
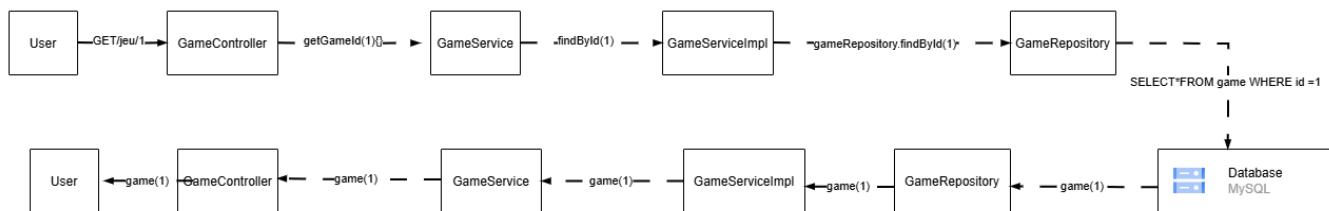


Diagramme de séquence

Le diagramme de séquence ci-dessous représente le parcours d'une requête du jeu dont l'identifiant est égal à 1 faite par un utilisateur .

Ce diagramme décrit le parcours de la requête depuis la barre de recherche en passant par le composants controller et tous les composants intermédiaires pour atteindre la base des données et exécuter la requête SQL générée par Hibernate. Une fois que la requête atteint sa cible à savoir le Jeu dont l'ID = 1, s'en suit l'envoie de la réponse vers l'utilisateur.





Reprise de l'API SPRING

Les bonnes pratiques et le respect des principes SOLID

Concernant l'architecture du projet, et afin de respecter les bonnes pratiques des principes SOLID, j'ai adopté pour une séparation des responsabilités entre les différents composants.

Ainsi, j'ai suivi la continuité du projet qui contenait déjà une architecture MVC avec quelques modèles et controllers existants. Pour cela, j'ai défini tous les attributs nécessaires dans les modèles, par exemple pour le modèle user, j'ai défini une adresse email, un mot de passe permettant d'avoir les informations utilisateurs nécessaires pour une authentification.

Afin de séparer les responsabilités, j'ai créé des controllers, des services, et des repositories spécifiques à chaque modèle et déplacé les logiques des controllers existants ainsi que de tous les controllers, dans les services, permettant ainsi aux controllers de communiquer avec les méthodes CRUD à savoir les méthodes GET, POST, PATCH, PUT et DELETE et les logiques métiers définies dans les services de manière indépendante.

Cependant, les services importent à leur tour, les repository qui permettent une communication avec la base des données. Afin de faciliter le nommage de chaque composant tout en gardant une cohérence entre eux permettant de les identifier, chaque composant : Controller, Service, ServiceImpl et Repository sont nommés en ajoutant un préfixe portant le nom des modèles correspondants.

Cette pratique m'a permis de définir toutes les fonctionnalités CRUD de tous les modèles en respectant les principes SOLID.

Ci après, des captures illustrant les composants de la classe Game:

Modèle

Le modèle est composé d'une classe des ses attributs et de ses getters et setters . Certaines classes ont des relations avec d'autres comme le cas de la classe **Game** que nous allons prendre pour exemple tout au long de ce rapport pour décrire tout le travail effectué. Concernant les



autres modèles, ils suivent la même logique de construction du modèle Game.

```
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;

@Entity
@Table(name = "game")

public class Game {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)

    private Long id;
    private String nom;
    private String image;
    private String description;
    private String genre;
    private Integer numEdition;

    public Game(){};

    @ManyToOne
    @JsonIgnoreProperties("games")
    private Category category;

    @ManyToOne
    @JsonIgnoreProperties("games")
    private Publisher publisher;

    @ManyToOne
    @JsonIgnoreProperties("games")
    private Author author;

    @ManyToOne
    @JsonIgnoreProperties("games")
    private Wishlist wishlist;
```

Les setters et getters d'une classe permettent l'accessibilité aux attributs de la classe.



```
/*AJOUT DES GETTERS ET SETTERS*/
public Author getAuthor() {
    return author;
}
public void setAuthor(Author author) {
    this.author = author;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getNom() {
    return nom;
}

public void setNom(String nom) {
    this.nom = nom;
}
public String getImage() {
    return image;
}

public void setImage(String image) {
    this.image = image;
}

public String getGenre() {
    return genre;
}
```



Controllers

Le controller est le composant où sont définies toutes les méthodes CRUD à savoir le GET permettant de récupérer une donnée d'utilisateur depuis la base des données.

```
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin(origins={"http://localhost:4200"}, allowedHeaders = "*")

public class GameController {

    @Autowired
    private GameService gameService;

    //AJOUT DU VERBE GET - POUR AFFICHER LES TOUS JEUX
    @GetMapping("/jeux")
    @ResponseStatus(code = HttpStatus.OK)
    public List<Game> getAllGame() {
        return gameService.findAll();
    }

    //AJOUT DU VERBE GET - POUR AFFICHER UN JEU PAR ID
    @GetMapping("/jeu/{id}")

    @ResponseStatus(code = HttpStatus.OK)

    public Game getGameById(@PathVariable("id") Long id) {
        Game game = gameService.findById(id);

        if (game == null) {
            throw new EntityNotExist();
        }
        game.getCategory();
        game.getPublisher();
        game.getAuthor();
        return game;
    }
}
```

Le POST permet d'envoyer et créer des données dans la base des données, le PUT et le PATCH sont des méthodes de mise à jour des données. La différence est que PUT permet de faire une mise à jour globale, c'est-à-dire que lorsque vous modifiez les informations d'un formulaire, toutes les données sont mises à jour lors de la soumission du



formulaire. Le PATCH quant à lui permet de mettre à jour, des champs spécifiques, la méthode permet de détecter uniquement les champs qui ont été modifiés et les mettre à jour lors de la soumission du formulaire.

Enfin, le DELETE est la méthode qui permet de supprimer une donnée dans la base des données.

```
//AJOUT DU VERBE POST - AJOUTER UN JEU
@PostMapping("/jeu")
@ResponseStatus(code = HttpStatus.CREATED)
public Game createGame(@RequestBody GameDTO gameDTO) {
    return gameService.ajouterJeu(gameDTO);
}

//AJOUT DU VERBE PUT - MODIFIER UN JEU PAR ID
@PutMapping("/jeu/{id}")
@ResponseStatus(code = HttpStatus.OK)
public void updateGame(@PathVariable Long id, @RequestBody Game games) {
    if (gameService.findById(id) == null) {
        throw new ExceptionEntityDontExist();
    }
    gameService.update(id, games);
}

/*AJOUT DU VERBE PUT - MODIFIER UN CHAMP PRECIS DU JEU PAR ID*/
@PatchMapping("/jeu/{id}")
@ResponseStatus(code = HttpStatus.OK)
public void updatePartialGame(@PathVariable Long id, @RequestBody GameDTO gameDTO) {
    gameService.updatePartial(id, gameDTO);
}

//AJOUT DU VERBE DELETE - SUPPRIMER UN JEU
@DeleteMapping("/jeu/{id}")
@ResponseStatus(code=HttpStatus.OK)
public void deleteGame(@PathVariable Long id) {
    if (gameService.findById(id) == null) {
        throw new ExceptionEntityDontExist();
    }
    gameService.delete(id);
}
```

Service

La notion de service ici est scindée en deux, à savoir le service et le serviceImplé.

Il s'agit tout simplement d'une stratégie d'organisation afin de respecter les bonnes pratiques SOLID, permettant de diviser le gestionnaire des



logiques métiers (SERVICE, proprement dit), en une interface contenant tous les constructeurs des méthodes du CRUD (GET, POST,PUT et PATCH) qui seront implémenté dans le serviceImpl. Par la suite, le serviceImpl importe les repositories grâce aux annotation @Autowired, à travers lesquelles, l'ensemble arrive à communiquer avec la base des données.

```
package com.gamesUP.gamesUP.services;

import java.util.List;

public interface GameService {

    public List<Game> findAll();

    public Game findById(Long id);

    public void update(Long id, Game game);

    public void delete(Long id);

    public Game ajouterJeu(GameDTO gameDTO);

    public void updatePartial(Long id, GameDTO gameDTO);
```

Service Implementation

Les servicesImpl utilise l'annotation @Service pour indiquer à spring qu'il s'agit d'une class Service, afin que cette classe soit détectée automatiquement par spring lors des démarrages comme une classe métier pour y injecter les dépendances nécessaires. Ci après, un exemple du composant **GameServiceImpl**.



```
package com.gamesUP.gamesUP.services.impl;
import com.gamesUP.gamesUP.dao.AuthorRepository;□

@Service
public class GameServiceImpl implements GameService {

    @Autowired
    private GameRepository gameRepository;
    @Autowired
    private AuthorRepository authorRepository;
    @Autowired
    private CategoryRepository categoryRepository;
    @Autowired
    private PublisherRepository publisherRepository;

    @Override
    public List<Game> findALL() {
        List<Game> games = new ArrayList<Game>();
        gameRepository.findAll().forEach(games::add);
        return games;
    }

    @Override
    public Game findById(Long id) {
        Optional<Game> game = gameRepository.findById(id);
        if (game.isPresent()) {
            return game.get();
        }
        throw new ExceptionEntityDontExist();
    }
}
```



```
*@Override
/*METHODE D'AJOUT DE JEU*/
    public Game ajouterJeu(GameDTO gameDTO) {
        Game game = new Game();
        game.setNom(gameDTO.getNom());
        game.setImage(gameDTO.getImage());
        game.setGenre(gameDTO.getGenre());
        game.setDescription(gameDTO.getDescription());
        game.setNumEdition(gameDTO.getNumEdition());
        mapRelations(game, gameDTO);

        return gameRepository.save(game);
    }

    private void mapRelations(Game game, GameDTO gameDTO) {
        if (gameDTO.getCategoryId() != null) {
            Category category = categoryRepository.findById(gameDTO.getCategoryId())
                .orElseThrow(() -> new EntityNotFoundException("La Category avec l'id " + gameDTO.getCategoryId() + " n'existe pas"));
            game.setCategory(category);
        }

        if (gameDTO.getPublisherId() != null) {
            Publisher publisher = publisherRepository.findById(gameDTO.getPublisherId())
                .orElseThrow(() -> new EntityNotFoundException("Le Publisher avec l'id " + gameDTO.getPublisherId() + " n'existe pas"));
            game.setPublisher(publisher);
        }

        if (gameDTO.getAuthorId() != null) {
            Author author = authorRepository.findById(gameDTO.getAuthorId())
                .orElseThrow(() -> new EntityNotFoundException("L'Author avec l'id " + gameDTO.getAuthorId() + " n'existe pas"));
            game.setAuthor(author);
        }
    }
}
```

```
@Override
public void updatePartial(Long id, GameDTO gameDTO) {
    Game gameExistant = gameRepository.findById(id)
        .orElseThrow(() -> new ExceptionEntityDontExist());

    if (gameDTO.getNom() != null) {
        gameExistant.setNom(gameDTO.getNom());
    }

    if (gameDTO.getImage() != null) {
        gameExistant.setImage(gameDTO.getImage());
    }

    if (gameDTO.getGenre() != null) {
        gameExistant.setGenre(gameDTO.getGenre());
    }

    if (gameDTO.getDescription() != null) {
        gameExistant.setDescription(gameDTO.getDescription());
    }

    if (gameDTO.getNumEdition() != null) {
        gameExistant.setNumEdition(gameDTO.getNumEdition());
    }
    mapRelations(gameExistant, gameDTO);

    gameRepository.save(gameExistant);
}
```



```
*@Override
public void update(Long id, Game games) {
    Game gameExistant = gameRepository.findById(id)
        .orElseThrow(() -> new ExceptionEntityDontExist());
    gameExistant.setNom(games.getNom());
    gameExistant.setImage(games.getImage());
    gameExistant.setGenre(games.getGenre());
    gameExistant.setDescription(games.getDescription());
    gameExistant.setNumEdition(games.getNumEdition());
    gameExistant.setCategory(games.getCategory());
    gameExistant.setPublisher(games.getPublisher());
    gameRepository.save(gameExistant);
}

*@Override
public void delete(Long id) {
    gameRepository.deleteById(id);
}
```

Réporstory

Les repositories sont des interfaces qui s'étendent aux **CrudRepository** ou à **JpaReporstory** qui sont des interface spécifiques à **Spring Data JPA** permettant de simplifier l'accès à la base des données en gérant les requêtes SQL, et permettant d'éviter d'écrire toutes les requêtes du projet directement dans le controller. Cela permet de faciliter la gestion de la base des données par un processus automatisé.

```
package com.gamesUP.gamesUP.dao;

import org.springframework.data.repository.CrudRepository;

import com.gamesUP.gamesUP.model.Game;

public interface GameRepository extends CrudRepository<Game, Long>{}
```

Le relations entre les modèles



Les relations les plus utilisées dans ce projet sont les relations ManyToOne, OneToMany et OneToOne. Ces relations permettent de relier plusieurs classes entre elles afin qu'elles communiquent ensemble et permettre une relation fluide avec la base des données pour la sécurité et la sauvegarde des données utilisateurs

La relation ManyToOne

La relation ManyToOne est une relation entre deux modèles de classes, définie par une annotation @ManyToOne grâce au système de décorateurs de spring-boot. Cette relation confère à plusieurs modèles de classe Game d'être reliés à une seule classe Catégorie par exemple. C'est une relation qui pourrait exister entre une liste de plusieurs jeux appartenant à la même catégorie. Ci dessous une illustration de la classe Game partageant la relation ManyToOne avec les classe Category, Author, Publisher et Wishlist.

```
@Entity
@Table(name = "game")

public class Game {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)

    private Long id;
    private String nom;
    private String image;
    private String description;
    private String genre;
    private Integer numEdition;

    public Game(){};

    @ManyToOne
    @JsonIgnoreProperties("games")
    private Category category;
```



Illustration du schéma JSON des données

The screenshot shows a Postman interface with the following details:

- Method: GET
- URL: http://localhost:8082/jeux
- Status: 200 OK
- Time: 204 ms
- Size: 1993 B
- Last updated: 1 Minute Ago
- Preview tab selected
- Body tab selected (green dot)
- Headers tab (6 items)
- Cookies tab
- Tests tab (0 / 0)
- Mock
- Console

The Preview section displays the JSON response:

```
1 [  
2 {  
3   "id": 2,  
4   "nom": "Final Fantasy VII",  
5   "image": "https://www.lamontagne.fr/photoSRC/Gw--/jeu-de-societe-the-gang-iello_7297287.jpeg",  
6   "description": "Un RPG culte de Square ",  
7   "genre": "RPG",  
8   "numEdition": 1,  
9   "category": {  
10     "id": 7,  
11     "type": " category 3"  
12   },  
13   "publisher": {  
14     "id": 1,  
15     "name": " publisher "  
16   },  
17   "author": {  
18     "id": 2,  
19     "name": " author "  
20   },  
21   "wishlist": null  
22 }]
```



La relation OneToMany

La relation OneToMany est une relation qui relie une classe donnée avec plusieurs autres modèles d'une même classe. C'est une relation qui agit comme un nœud, dans l'exemple précédente, la catégorie peut être vue comme un nœud de plusieurs reliant plusieurs jeux de même nature. Un autre exemple palpable est l'exemple d'un panier qui pourrait contenir plusieurs articles. Ici, le panier agit comme un nœud commun à tous les articles qu'il comporte. La définition de la relation OneToMany renvoie également à la notion de la relation ManyToOne

```
package com.gamesUP.gamesUP.model;
import java.util.List;

@Entity
@Table(name = "category")
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String type;

    @OneToMany(mappedBy = "category", cascade= CascadeType.ALL)
    @JsonIgnoreProperties("category")
    private List<Game> games ;

    public Category(){}
}
```

Illustration du schéma JSON des données



GET ▼ http://localhost:8082/categories

Send ▾

Params Body (1) Auth Headers (4) Scripts Docs

200 OK 24 ms 1826 B 1 Minute Ago ▾

Preview Headers (6) Cookies Tests 0 / 0 → Mock Console

Preview ▾

```
1 [  
2 {  
3   "id": 7,  
4   "type": "category 3",  
5   "games": [  
6     {  
7       "id": 2,  
8       "nom": "Final Fantasy VII",  
9       "image": "https://www.lamontagne.fr/photoSRC/Gw--/jeu-de-societe-the-gang-iello\_7297287.jpeg",  
10      "description": "Un RPG culte de Square ",  
11      "genre": "RPG",  
12      "numEdition": 1,  
13      "publisher": {  
14        "id": 1,  
15        "name": "publisher "  
16      },  
17      "author": {  
18        "id": 2,  
19        "name": "author "  
20      },  
21      "wishlist": null  
22    },  
23    {  
24      "id": 3,  
25      "nom": "Final Fantasy VII",  
26      "image": "https://www.lamontagne.fr/photoSRC/Gw--/jeu-de-societe-the-gang-iello\_7297287.jpeg",  
27      "description": "Un RPG culte de Square ",  
28      "genre": "RPG"  
29    }
```

La relation OneToOne

La relation OneToOne est une relation qui lie deux classes différentes. Dans le cas de ce projet, c'est la relation qui existe dans les deux sens entre la classe User et la classe Wishlist. Cette relation permet à la classe Wishlist de lier son contenu à un utilisateur, c'est-à-dire une liste de jeux, à un utilisateur. Cela permet d'avoir des wishlist (liste de jeux mise en favoris) spécifique à chaque utilisateur.



```
@Entity
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
    private String role;
    private String password;
    private String repassword;

    @OneToOne
    private Wishlist wishlist;
```

Sérialisation des données

GET ▾ http://localhost:8082/wishlist/22	Send ▾	200 OK	28 ms	176 B
Params	Body	Auth	Headers (3)	Scripts
Inherit from parent ▾	Preview ▾			
	<pre>1 ▶ { 2 "id": 22, 3 "name": "wishlist", 4 "user": { 5 "id": 4, 6 "nom": "Cient", 7 "email": "meschebajordy@yahoo.fr", 8 "role": "client", 9 "password": "client", 10 "repassword": "client", 11 "wishlist": null 12 }, 13 "games": [] 14 }</pre>			



Afin de résoudre ce problème, Spring met à disposition les annotations `JsonIgnoreProperties`. Dans le cas de ce projet, j'ai utilisé l'annotation `JsonIgnoreProperties` comme indiquées sur les images suivantes pour les classe Game et Category, permettant d'ignorer les champs non nécessaires.

```
@Entity
@Table(name = "game")

public class Game {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)

    private Long id;
    private String nom;
    private String image;
    private String description;
    private String genre;
    private Integer numEdition;

    public Game(){};

    @ManyToOne
    @JsonIgnoreProperties("games")
    private Category category;
```

L'annotation dans la classe Game elle même permet d'ignorer et d'éviter l'affichage répétitif de la la table, de même que pour category, permettant d'éviter la boucle infinie.



```
package com.gamesUP.gamesUP.model;
import java.util.List;

@Entity
@Table(name = "category")
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String type;

    @OneToMany(mappedBy = "category", cascade= CascadeType.ALL )
    @JsonIgnoreProperties("category")
    private List<Game> games ;

    public Category(){}
}
```

Cela à permis d'avoir des bonnes structure JSON comme les suivantes :

Affichage des jeux en format JSON

The screenshot shows a Postman interface with the following details:

- Method: GET
- URL: http://localhost:8082/jeux
- Status: 200 OK (highlighted in green)
- Time: 334 ms
- Size: 508 B
- Last Updated: Just Now
- Preview tab selected
- JSON response content:

```
1 [ 
2 {
3     "id": 1,
4     "nom": "CODENAMES",
5     "image": "https://m.media-amazon.com/images/I/81YDYNrClZL.AC_SV879_.jpg",
6     "description": "Le jeu culte de déduction ✌. Plébiscité par des milliers de joueurs ! Trouvez les bons mots et évitez l'assassin - Jeu de société addictif en équipe pour des soirées inoubliables - VF",
7     "genre": "Stratégie",
8     "numEdition": 2000,
9     "category": {
10         "id": 1,
11         "type": "Category"
12     },
13     "publisher": {
14         "id": 1,
15         "name": " publisher"
16     },
17     "author": {
18         "id": 1,
19         "name": " auteur"
20     },
21     "wishlist": null
22 }
23 ]
```



Affichage des catégories en format JSON

GET ▼ http://localhost:8082/categories

Send ▾

Params Body (1) Auth Headers (4) Scripts Docs

200 OK 42 ms 507 B Just Now ▾

Preview Headers (6) Cookies Tests 0 / 0 → Mock Console

Preview ▾

```
1 [ 
2 { 
3   "id": 1,
4   "type": "Category",
5   "games": [
6     {
7       "id": 1,
8       "nom": "CODENAMES",
9       "image": "https://m.media-amazon.com/images/I/81YDYNrC1ZL..AC_SX879_.jpg",
10      "description": " Le jeu culte de déduction 🎲. Plébiscité par des milliers de joueurs ! Trouvez les bons mots et évitez l'assassin - Jeu de société addictif en équipe pour des soirées inoubliables - VF ",
11      "genre": "Stratégie",
12      "numEdition": 2000,
13      "publisher": {
14        "id": 1,
15        "name": " publisher"
16      },
17      "author": {
18        "id": 1,
19        "name": " auteur"
20      },
21      "wishlist": null
22    }
23  ]
24 }
25 ]
```



GET ▼ http://localhost:8082/wishlist/22 Send ▼ 200 OK 353 ms 160 B

Params Body Auth Headers 3 Scripts Preview Headers 6 Cookies Tests 0 / 0

Inherit from parent ▼ Preview ▼

Select an auth type from above

```
1 var {  
2   "id": 22,  
3   "name": " wishlist",  
4   "user": {  
5     "id": 4,  
6     "nom": " Cient",  
7     "email": " meschebajordy@yahoo.fr",  
8     "role": " client",  
9     "password": " client",  
10    "repassword": " client"  
11  },  
12  "games": []  
13}
```

GET ▼ http://localhost:8082/jeu/3 Send ▼ 200 OK 10.7 s 309 B Just Now ▼

Params Body Auth Headers 4 Scripts Docs Preview Headers 6 Cookies Tests 0 / 0 → Mock Console

URL PREVIEW http://localhost:8082/jeu/3

QUERY PARAMETERS Import from URL Bulk Edit

+ Add Delete all Description

name	value
name	value

PATH PARAMETERS

```
1 var {  
2   "id": 3,  
3   "nom": " Final Fantasy VII",  
4   "image": " https://www.lamontagne.fr/photoSRC/GW--/jeu-de-societe-the-gang-iello_7297287.jpeg ",  
5   "description": " Un RPG culte de Square ",  
6   "genre": " RPG ",  
7   "numEdition": 1,  
8   "category": {  
9     "id": 1,  
10    "type": " sandbox "  
11  },  
12  "publisher": {  
13    "id": 1,  
14    "name": " Jory AKRA "  
15  },  
16  "author": {  
17    "id": 2,  
18    "name": " Jason "  
19  }  
20}
```



GET ▾ http://localhost:8082/category/2 Send ▾ 200 OK 68 ms 233 B Just Now ▾

Params Body (1) Auth Headers (4) Scripts Docs

Preview Headers (6) Cookies Tests 0 / 0 → Mock Console

URL PREVIEW
http://localhost:8082/category/2

QUERY PARAMETERS Import from URL ⚡ Bulk Edit

+ Add Delete all Description

name	value

Preview

```
1 + {
2   "id": 2,
3   "type": "Guerres",
4   "games": [
5     {
6       "id": 6,
7       "nom": "Final Fantasy VII",
8       "image": "https://www.lamontagne.fr/photoSRC/Gw--/jeu-de-societe-the-gang-iello_7297287.jpeg",
9       "description": "Un RPG culte de Square",
10      "genre": "RPG",
11      "numEdition": 1
12    }
13  ]
14 }
```

GET ▾ http://localhost:8082/author/2 Send ▾ 200 OK 2.42 s 231 B Just Now ▾

Params Body (1) Auth Headers (4) Scripts Docs

Preview Headers (6) Cookies Tests 0 / 0 → Mock Console

URL PREVIEW
http://localhost:8082/author/2

QUERY PARAMETERS Import from URL ⚡ Bulk Edit

+ Add Delete all Description

name	value

Preview

```
1 + {
2   "id": 2,
3   "name": "Jason",
4   "games": [
5     {
6       "id": 3,
7       "nom": "Final Fantasy VII",
8       "image": "https://www.lamontagne.fr/photoSRC/Gw--/jeu-de-societe-the-gang-iello_7297287.jpeg",
9       "description": "Un RPG culte de Square",
10      "genre": "RPG",
11      "numEdition": 1
12    }
13  ]
14 }
```



GET ▾ http://localhost:8082/publisher/1 Send ▾ 200 OK 512 ms 432 B Just Now ▾

Params Body (1) Auth Headers (4) Scripts Docs

Preview Headers (6) Cookies Tests 0 / 0 → Mock Console

URL PREVIEW
http://localhost:8082/publisher/1

QUERY PARAMETERS Import from URL ⚡ Bulk Edit

+ Add Delete all Description

#	name	value
1		

PATH PARAMETERS

```
1 {  
2   "id": 1,  
3   "name": "Jory AKRA ",  
4   "games": [  
5     {  
6       "id": 3,  
7       "nom": "Final Fantasy VII",  
8       "image": "https://www.lamontagne.fr/photosSRC/Gw--/jeu-de-  
societe-the-gang-iello_7297287.jpeg",  
9       "description": "Un RPG culte de Square ",  
10      "genre": "RPG",  
11      "numEdition": 1  
12    },  
13    {  
14      "id": 6,  
15      "nom": "Final Fantasy VII",  
16      "image": "https://www.lamontagne.fr/photosSRC/Gw--/jeu-de-  
societe-the-gang-iello_7297287.jpeg",  
17      "description": "Un RPG culte de Square ",  
18      "genre": "RPG",  
19      "numEdition": 1  
20    }  
21  ]  
22 }
```

Liste des endpoints

Class / Méthodes	GET Listes	POST, PUT, PATCH et GET/{id}
User	http://localhost:8082/users	http://localhost:8082/user
Category	http://localhost:8082/categories	http://localhost:8082/category
Game	http://localhost:8082/jeux	http://localhost:8082/jeu
Author	http://localhost:8082/authors	http://localhost:8082/author
Publisher	http://localhost:8082/publishers	http://localhost:8082/publisher
Wishlist	http://localhost:8082/wishlists	http://localhost:8082/wishlist
Avis	http://localhost:8082/avis	http://localhost:8082/avis
Purchase	http://localhost:8082/purchases	http://localhost:8082/purchase
Inventory	http://localhost:8082/inventories	http://localhost:8082/inventory
Recommendation	http://127.0.0.1:8000/docs	http://127.0.0.1:8000/recommendations



Mise en place d'hibernate

Afin de faire fonctionner la base des données, tout en automatisant certains processus et faciliter la maintenabilité du site permettant de respecter les principes de la méthode SOLID, il est nécessaire de mettre en place Hibernate. Cela permettrait au serveur de démarrer correctement et d'automatiser certains processus comme la création automatique des table de la base des données, facilitant ainsi les maintenance du site.

Pour cela, j'ai rajouté quelques lignes de code dans le fichier application.properties qui est le fichier de configuration du serveur dans spring-boot.

dans ce fichier voici quelques lignes de code que j'ai ajouté pour configurer hibernate:

- **spring.datasource.url=jdbc:mysql://\${MYSQL_HOST:localhost}:3306/game_up**

Sur cette ligne, j'ai ajouté le nom **game_up** de la base de données en respectant les caractères de nommage pour éviter la casse.

- **spring.datasource.username=root**

Sur cette ligne, j'ai défini l'utilisateur root par travailler en développement.

- **spring.datasource.password=admin**

Sur cette ligne, j'ai défini le mot de passe permettant d'accéder au serveur MySQL.

- **spring.jpa.hibernate.ddl-auto = update.**

Cette ligne gère la création ainsi que la mise à jour automatique des tables dans la base de données.



- **spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect**

Cette ligne indique à hibernate quel dialecte SQL utiliser pour générer les requêtes adaptées à la base de données.

Ci joint une image d'illustration.

```
spring.application.name=gamesUP

spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/game_up
spring.datasource.username=root
spring.datasource.password=admin
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=true
server.port=8082
```

Dans cette configuration, la bonne pratique consiste à définir le nom de la base des données ainsi que des tables en minuscule ou en snake case, cela permet une bonne lisibilité rendant les nom de la base des tables explicite pour MySQL, permettant ainsi de respecter le principe de maintenabilité.

En effet, une écriture en CamelCase pourrait fonctionner dans les requêtes SQL, sous windows, cependant, cela pourrait être problématique pour d'autres systèmes comme linux.

Tests unitaire avec junit

En ce qui concerne les tests unitaires, j'ai testé tous les endpoints et toutes les méthodes du crud à savoir le GET, POST, PUT PATCH et DELETE.



Ci après, une capture des tests unitaires de la classe Game. Le test des autres class suivent également la même méthode.

```
package com.gamesUP.gamesUP.dao;
import static org.junit.jupiter.api.Assertions.assertEquals;
import dto.GameDTO;
import java.util.List;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.test.context.ActiveProfiles;
import com.gamesUP.gamesUP.model.Author;
import com.gamesUP.gamesUP.model.Category;
import com.gamesUP.gamesUP.model.Game;
import com.gamesUP.gamesUP.model.Publisher;
import com.gamesUP.gamesUP.services.impl.GameServiceImpl;

@DataJpaTest
@ActiveProfiles("test")

//TEST DES METHODES GETs, POST, PATCH, DELETE;
public class GameRepositoryTest {
    @Autowired
    private GameServiceImpl gameServiceImpl;

    //TEST GET ALL
    @Test
    void shouldGetAllGame() {
        Game game = new Game();
        game.setId(1L);
        List<Game> games = gameServiceImpl.findAll();
        assertEquals(1, games.size());
    }
    //TEST GET BY ID
    @Test
    void shouldGetGameById() {
        Game games = new Game();
        games.setId(1L);
        games = gameServiceImpl.findById((long) 1);
        assertEquals("Everspace", games.getNom());
    }
}
```



```
//TEST POST
@Test
void shouldCreateGame() {
    Game newGame = new Game();

    newGame.setNom("Everspace");
    newGame.setImage("https://gaming-cdn.com/images/products/9495/616x353/everspac");
    newGame.setDescription("Description");
    newGame.setNumEdition(2002);

    Category category = new Category();
    category.setType("Action");
    newGame.setCategory(category);

    Publisher publisher = new Publisher();
    publisher.setName("Jordy AKRA MESCHEBA");
    newGame.setPublisher(publisher);

    Author author = new Author();
    author.setName("Jordy AKRA MESCHEBA");
    newGame.setAuthor(author);

    assertEquals("Everspace", newGame.getNom());
    assertEquals("Jordy AKRA MESCHEBA", newGame.getAuthor().getName());
    assertEquals("Jordy AKRA MESCHEBA", newGame.getPublisher().getName());
    assertEquals("Action", newGame.getCategory().getType());
}
```

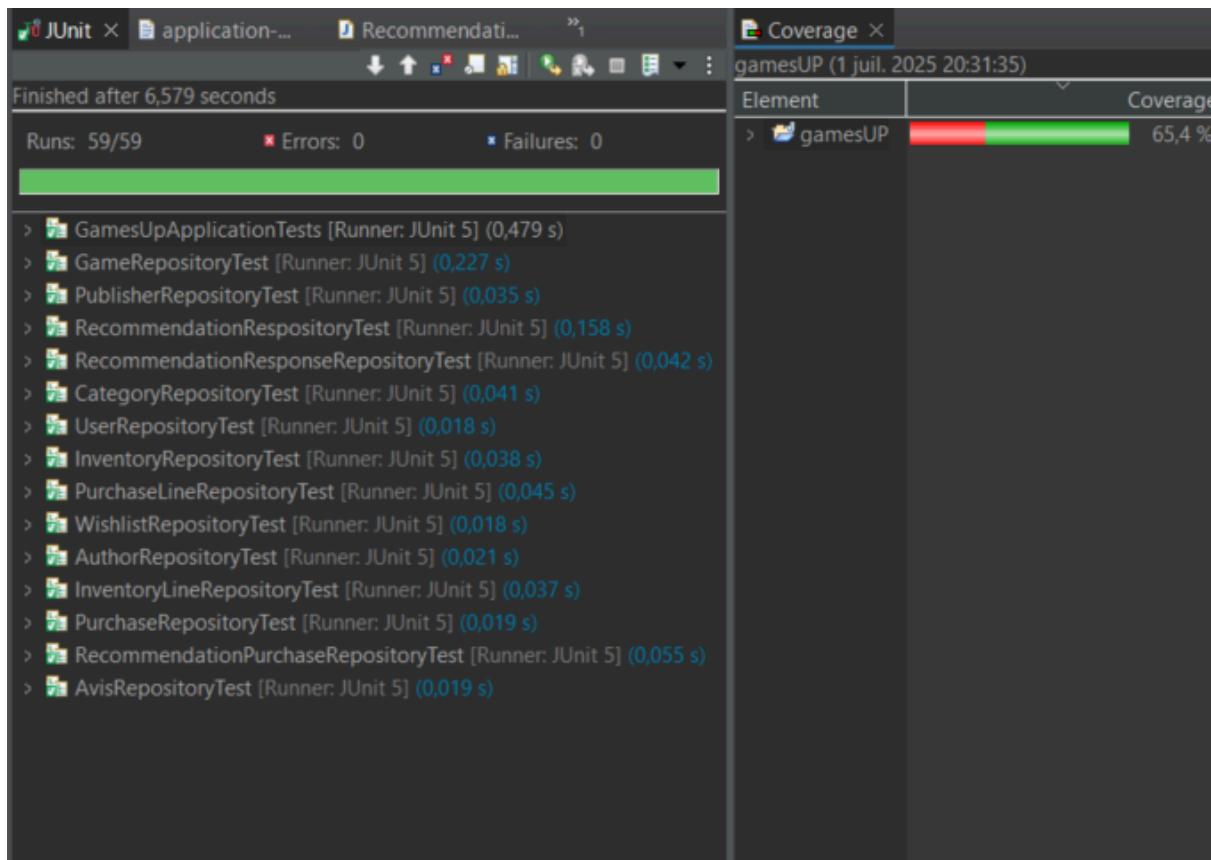


```
//TEST UPDATE PARTIAL
@Test
void shouldUpdatePartialGame() {
    GameDTO newGame = new GameDTO();
    newGame.setNom("Everspace");
    gameServiceImpl.updatePartial(1L, newGame);
    newGame.setId(1L);
    assertEquals(1L, newGame.getId());

}
//TEST UPDATE
@Test
void shouldUpdateGame() {
    Game gameExistant = gameServiceImpl.findById(1L);
    Game newGame = new Game();
    newGame.setNom("Everspace");
    newGame.setId(1L);
    gameServiceImpl.update(1L, gameExistant);
    assertEquals(1L, newGame.getId());

}

//TEST DELETE
void shouldDeleteGame() {
    gameServiceImpl.delete((long) 1);
};
```



Algorithme du Machine Learning

La partie du code python est globalement la partie responsable de l'implémentation de la machine learning permettant de faire des recommandations de jeux sur la plateforme en tenant compte des jeux précédemment visités par les utilisateurs pour leurs proposer des jeux similaires. Cette partie est principalement divisée en quatre.

data_loader.py

Dans ce fichier existe une fonction dans la principal rôle est d'importer le fichier data.csv d'entraînement de l'algorithme du machine learning.



```
import pandas as pd\n\n\ndef load_training_data(file_path="data/data.csv"):\n    # Ex: Charger un fichier CSV contenant les données d'utilisateur, jeux, etc.\n    return pd.read_csv(file_path, usecols=['user_id', 'game_id', 'rating', 'game_name'])
```

Recommendations.py

Ce fichier comporte la fonction principale generate_recommendations, qui recharge dans un premier temps le fichier data.csv en l'important depuis le fichier data_loader. Par la suite, ces données sont transformées en une matrice constituée des utilisateurs sur chaque ligne, des jeux sur chaque colonne et les notations des jeux par les utilisateurs .

La boucle for permet de mettre à jour la notation des jeux. Ces informations seront par la suite insérées dans l'algorithme de machine learning KNN qui fait des recherches basées sur les voisins proches pour pouvoir faire la prédiction des jeux similaires. Cet algorithme est entraîné sur la matrice précédemment calculée pour pouvoir faire des recommandations pertinentes.

Un calcul de la moyenne des notes des jeux attribuées par les utilisateurs et un ti des jeux est effectué afin d'avoir les jeux recommandés.

Enfin, le résultat est renvoyé sous forme d'un dictionnaire contenant l'identifiant, le nom et la notation des jeux recommandés.



```
from models import UserData
from data_loader import load_training_data
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
from sklearn.neighbors import NearestNeighbors

def generate_recommendations(user_data: UserData):
    # Définition du chemin d'accès et chargement du fichier d'entraînement du ML
    df = load_training_data("data/data.csv")
    tableau = df.pivot_table(
        index='user_id',
        columns='game_id',
        values='rating'
    ).fillna(0)

    for p in user_data.purchases:
        tableau.loc[user_data.user_id, p.game_id] = p.rating
    # Entraînement du ML
    n_neighbors = min(3, len(tableau))
    model_knn = NearestNeighbors(n_neighbors=n_neighbors, metric='cosine')
    model_knn.fit(tableau.values)

    # Recherche des voisins
    distances, indices = model_knn.kneighbors(
        tableau.loc[[user_data.user_id]],
        n_neighbors=n_neighbors
    )
    similar_users = tableau.index[indices[0][1:]].tolist()
    # Moyenne des ratings des voisins
    recs_series = (
        df[df['user_id'].isin(similar_users)]
        .groupby('game_id')['rating']
        .mean()
        .sort_values(ascending=False)
    )
    # Recupération des tops 5 des jeux
    already = df[df['user_id'] == user_data.user_id]['game_id'].tolist()
    top_ids = recs_series.drop(already, errors='ignore').head(5).index.tolist()
    # Récupération des game_name à partir des game_id
    recommendations = []
    for gid in top_ids:
        matches = df[df['game_id'] == gid]
        if not matches.empty:
            # Prend le premier game_name trouvé
            game_name = matches.iloc[0]['game_name']
        else:
            game_name = "Inconnu"
        recommendations.append({
            "game_id": gid,
            "game_name": game_name,
            "rating": float(recs_series[gid])
        })
    return recommendations
```

You, now • Uncommitted changes



main.py

Ce fichier permet de mettre en place l'api à l'aide de FastAPI et expose deux endpoints.

La méthode GET contient le endpoint racine permettant de vérifier que l'api fonctionne en renvoyant le message “API de recommandation en ligne”.

La méthode POST appelle la fonction generate_recommendations pour générer la liste des jeux recommandés

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List
from recommendation import generate_recommendations
from models import UserData

app = FastAPI()
    You, 2 months ago • Ajout du projet existant ...
# Endpoint de base pour tester que l'API est en ligne
@app.get("/")
async def root():
    return {"message": "API de recommandation en ligne"}

# Endpoint pour envoyer les données d'utilisateur et récupérer des recommandations
@app.post("/recommendations")
async def get_recommendations(data: UserData):
    try:
        recommendations = generate_recommendations(data)
        return {"user_id":data.user_id, "recommendations": recommendations}
    except Exception as e:
        print("Erreur lors de la génération des recommandations :", e)
        raise HTTPException(status_code=500, detail=str(e))
```



models.py

Le modèle définit le schéma des données d'utilisateur et des jeux attendues par l'API.

```
from pydantic import BaseModel
from typing import List

You, 2 months ago | 1 author (You)
class UserPurchase(BaseModel):
    game_id: int
    rating: float

You, 2 months ago | 1 author (You)
class UserData(BaseModel):
    user_id: int
    purchases: List[UserPurchase]
```

RecommendationDTO

Afin de mettre en relation l'API Python et le backend spring-boot permettant ainsi de tester l'api , le sécuriser et l'exposer en http sur le port 8082 pour être utilisable dans angular, le modèle DTO permet de définir les attributs qui seront exposés dans le fichier JSON utilisable par l'application Angular.



```
package dto;          You, 3 days ago • Ajout du modèle ML KNN ...
You, 4 minutes ago | 1 author (You)
public class RecommendationDTO {

    private Integer game_id;
    private String game_name;
    private Double rating;

    public String getGame_name() {
        return game_name;
    }

    public void setGame_name(String game_name) {
        this.game_name = game_name;
    }

    public Integer getGame_id() {
        return game_id;
    }

    public void setGame_id(Integer game_id) {
        this.game_id = game_id;
    }

    public Double getRating() {
        return rating;
    }

    public void setRating(Double rating) {
        this.rating = rating;
    }
}
```

RecommendationController

Le controller définit les méthodes GET permettant d'exposer l'API python sur les endpoints :

- **http://localhost:8082/recommendations/{id}** et
- **http://localhost:8082/recommendations**



```
package com.gamesUP.gamesUP.controller;
import java.util.List;          You, 4 days ago • Ajout des tests unitaire avec JUNIT avec un cov...
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import com.gamesUP.gamesUP.services.RecommendationService;
import dto.RecommendationDTO;

You, 16 seconds ago | 1 author (You)
@RestController
@CrossOrigin(origins={"http://localhost:4200"}, allowedHeaders = "*")
public class RecommendationController {
    @Autowired
    private RecommendationService recommendationService;
    @GetMapping("/recommendations")
    public List<RecommendationDTO> getAllRecommendation() {
        return recommendationService.getAllRecommendation();
    }

    @GetMapping("/recommendations/{userId}")
    public ResponseEntity<List<RecommendationDTO>> getRecommendations(@PathVariable Long userId) {
        List<RecommendationDTO> recommendations = recommendationService.getRecommendations(userId);
        return ResponseEntity.ok(recommendations);
    }
}
```

Service

C'est dans le service qu'une requête est envoyée à l'Api python via la méthode POST.



```
@Service
public class RecommendationServiceImpl implements RecommendationService {

    @Autowired
    private RestTemplate restTemplate;
    @Autowired
    RecommendationRepository recommendationRepository;

    @SuppressWarnings("null")
    @Override
    public List<RecommendationDTO> getRecommendations(Long userId) {
        String pythonApiUrl = "http://localhost:8000/recommendations";

        Map<String, Object> requestBody = new HashMap<>();
        requestBody.put(key:"user_id", userId);
        requestBody.put(key:"purchases", new ArrayList<>());

        ResponseEntity<RecommendationResponse> response = restTemplate.postForEntity(
            pythonApiUrl,
            requestBody,
            responseType:RecommendationResponse.class
        );
        return response.getBody().getRecommendations();
    }

    @Override
    public List<RecommendationDTO> getAllRecommendation() {
        List<RecommendationDTO> recommendationsList = new ArrayList<RecommendationDTO>();
        RecommendationDTO recommendationsDTO = new RecommendationDTO();
        recommendationsList.add(recommendationsDTO);
        return recommendationsList;
    }
}
```

La réponse attendue est du type RecommendationResponse



```
package com.gamesUP.gamesUP.model;
import java.util.List;
import dto.RecommendationDTO;
...
public class RecommendationResponse {
    private Long user_id;
    private List<RecommendationDTO> recommendations;
    public Long getUser_id() {
        return user_id;
    }
    public void setUser_id(Long user_id) {
        this.user_id = user_id;
    }

    public List<RecommendationDTO> getRecommendations() {
        return recommendations;
    }
    public void setRecommendations(List<RecommendationDTO> recommendations) {
        this.recommendations = recommendations;
    }
}
```

Cette réponse sera convertie sous forme d'une liste respectant le type d'attributs définis dans la RecommendationDTO.

Test avec insomnia

Le test du endpoint suivant permet d'avoir la réponse envoyée par l'API Python comme observée sur l'image ci dessous :

- <http://localhost:8082/recommendations/{id}>



The screenshot shows a Postman collection interface. The top bar indicates a GET request to 'ajouterJeu'. Below the URL 'http://localhost:8082/recommendations/3' are tabs for Params, Body (green dot), Auth, Headers (6), Scripts, and Docs. The status bar shows '200 OK', '33 ms', '104 B', and '1 Minute Ago'. Below the status bar are tabs for Preview, Headers (6), Cookies, Tests (0 / 0), Mock, and Console. The Preview section displays a JSON array of two game objects:

```
1 [  
2 {  
3   "game_id": 101,  
4   "game_name": "Pup Go",  
5   "rating": 5.0  
6 },  
7 {  
8   "game_id": 102,  
9   "game_name": "CODENAMES",  
10  "rating": 4.0  
11 }  
12 ]
```

Frontend

Le frontend de ce projet n'est pas explicitement demandé, cependant afin de rendre l'expérience plus agréable en testant directement le site sur une interface utilisateur agréable, j'ai décidé de réaliser l'interface utilisateur en utilisant Angular.

J'ai fait ce choix pour plusieurs raisons. Premièrement parce que Angular est le framework que je maîtrise le mieux actuellement, également c'est un choix qui permet d'avoir en même temps une interface utilisateur agréable, responsive mais également robuste car on peut y appliquer les principes SOLID par la séparation des tâches en utilisant le concepte de



composants angular, permettant de ce fait la séparation des responsabilitées.

En ce qui concerne cette partie dédiée au côté frontend, je vais faire une description brève des fonctionnalités suivie des captures d'écran contenant des indications de quelques pages nécessaires,

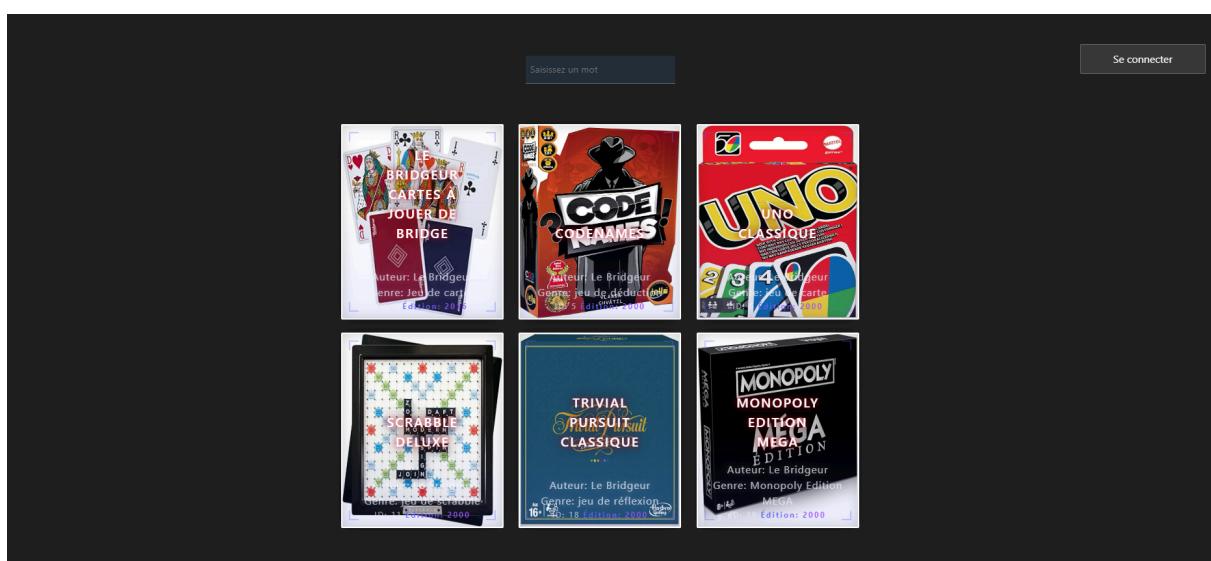
Page d'accueil

L'accès à la page d'accueil ne nécessite pas d'avoir un compte utilisateur.

Sur la page d'accueil , les fonctionnalités sont restreintes, on peut accéder à l'affichage de la liste des jeux. Il est également possible de naviguer dans les catégories afin d'accéder à une liste de jeux classée par catégorie, on peut aussi naviguer dans l'onglet auteur permettant d'accéder uniquement à la liste des jeux d'un auteur spécifique. Vous avez également la possibilité de chercher des jeux via une barre de recherche et accéder à leurs détails. Enfin, il est possible d'accéder aux commentaires des clients sur leurs expériences sur le site.

Cependant, si vous souhaitez avoir plus de fonctionnalités, vous devez créer un compte utilisateur.

Ci jointe, une image d'illustration de la page d'accueil.





The screenshot shows a dark-themed website interface. At the top right is a "Se connecter" button. Below it are three game cards: "LE BRIDGEUR CARTES À JOUER DE BRIDGE" (Author: Le Bridgeur, Genre: Jeu de cartes, Edition: 2017), "CODE! CODEAMES!" (Author: Le Bridgeur, Genre: jeu dg déduction), and "SCRABBLE DELUXE" (Genre: jeu de scrabble, édition: 2000). Below the cards is a horizontal scrollable area containing four user reviews:

- ISCOD** Note : 5
Posté le 05-07-25 à 00:30
Une expérience exceptionnelle
- PAUL DOUMER** Note : 8
Posté le 05-04-25 à 14:30
Une expérience satisfaisante
- PAUL DOUMER** Note : 8
Posté le 05-04-25 à 14:30
Une expérience satisfaisante
- ISCOD** Note : 10
Posté le 06-07-25 à 10:34
Commentaire de test

The screenshot shows a dark-themed website interface, identical to the one above but with different review data. It features three game cards: "LE BRIDGEUR CARTES À JOUER DE BRIDGE", "CODE! CODEAMES!", and "SCRABBLE DELUXE". Below the cards is a horizontal scrollable area containing four user reviews:

- ISCOD** Note : 5
Posté le 05-07-25 à 00:30
Une expérience exceptionnelle
- PAUL DOUMER** Note : 8
Posté le 05-04-25 à 14:30
Une expérience satisfaisante
- PAUL DOUMER** Note : 8
Posté le 05-04-25 à 14:30
Une expérience satisfaisante
- ISCOD** Note : 10
Posté le 06-07-25 à 10:34
Commentaire de test

Page d'authentification

Lors de l'accès au site **game_up**, on apparaît sur la page d'authentification.

Pour accéder à un compte client il est obligatoire de renseigner les identifications d'utilisateur en saisissant les informations dans le formulaire suivant.



ACCUEIL

The home page features a dark blue background with a central login form. The form is titled "Connexion" and includes fields for "nom d'utilisateur" (visiplus@email.com) and "Mot de passe". A purple "Se connecter" button is at the bottom. Below the form, links for "Vous n'avez pas de compte?" and "S'inscrire" are visible. To the left of the form are four playing cards (King of Hearts, Queen of Clubs, King of Spades, King of Diamonds). To the right is a large graphic for "GAMES UP BOARD GAME" featuring a stylized dice and the text "GAMES UP BOARD GAME". At the bottom, the word "GAME UP" is written in large red letters.

Page d'inscription

La page d'inscription permet de créer un compte. Cependant, sur cette page, il est uniquement possible de créer un compte utilisateur.

ACCUEIL

The registration page has a similar layout to the home page. It features a central "Créer un compte" form with fields for "Name", "Email", "Rôle" (Role), "Définir votre mot de passe" (Define your password), and "Confirmer votre mot de passe" (Confirm your password). A purple "Submit" button is at the bottom. Below the form, links for "Avez vous déjà un compte?" and "Connectez-vous" are visible. The left side shows playing cards, and the right side shows the "GAMES UP BOARD GAME" graphic. At the bottom, the word "GAME UP" is written in large red letters.

Compte client

La page d'accueil d'un compte client affiche globalement un navbar avec les onglets author, categories et le bouton de déconnexion, une liste de jeux, une barre de recherche, un bouton d'ajout d'avis et un footer avec un carrousel défilant les avis récemment ajoutés par les utilisateurs.



En dessous de l'affichage de la liste des jeux, s'affichent les recommandations. Lorsque le serveur python n'est pas démarré ou lorsqu'il n'y a pas de recommandations, un message s'affiche, informant qu'**Aucune recommandation n'est disponible pour le moment**, comme indiqué sur l'image avec la flèche.

Page d'accueil

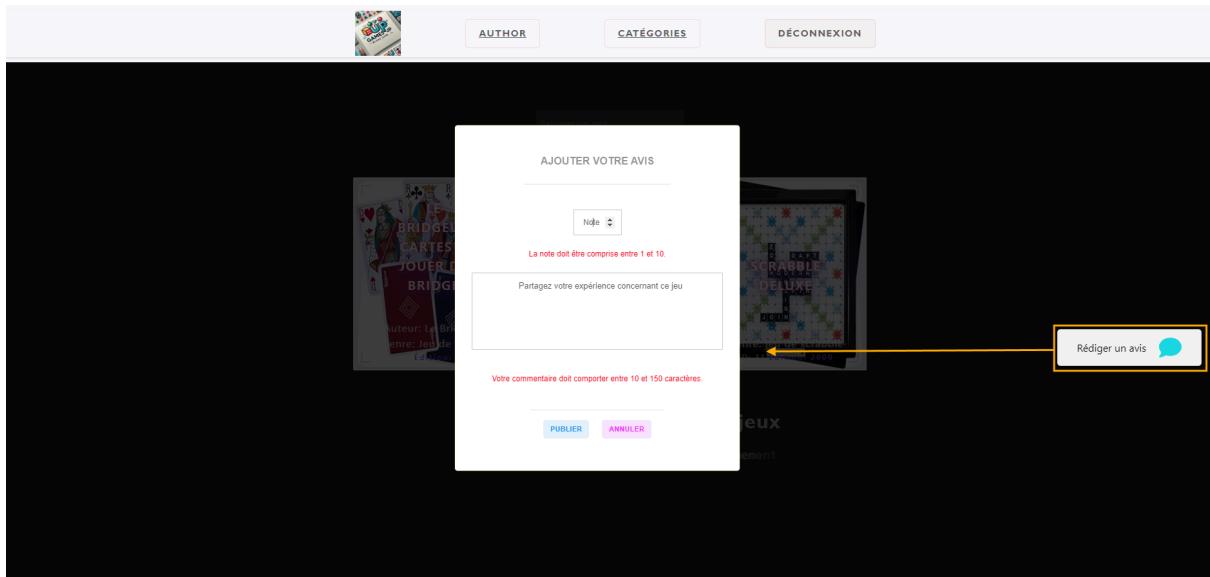
The screenshot shows a dark-themed web application interface. At the top, there's a navigation bar with icons for user profile, search, categories, and logout. A search bar is present above the main content area. Below the content, there's a button labeled "Rédiger un avis" (Write a review) with a speech bubble icon. The main content area displays three game boxes: "LE BRIDGEUR CARTES À JOUER DE BRIDGE", "CODE WORD NAMES", and "SCRABBLE DELUXE". Below these, a section titled "Nos recommandations des jeux" contains a yellow-bordered box with the text "Aucune recommandation pour le moment".

The second screenshot is identical to the first, showing the same dark-themed interface, game boxes, and the "No recommendations" message in the "Nos recommandations des jeux" section.

At the bottom, there's a section for user reviews. It shows four reviews from users "ISCOD" (Note: 5), "Paul DOUMER" (Note: 8), "ISCOD" (Note: 10), and "Paul DOUMER" (Note: 8). Each review includes a timestamp and a short comment. Navigation arrows are visible on either side of the reviews.

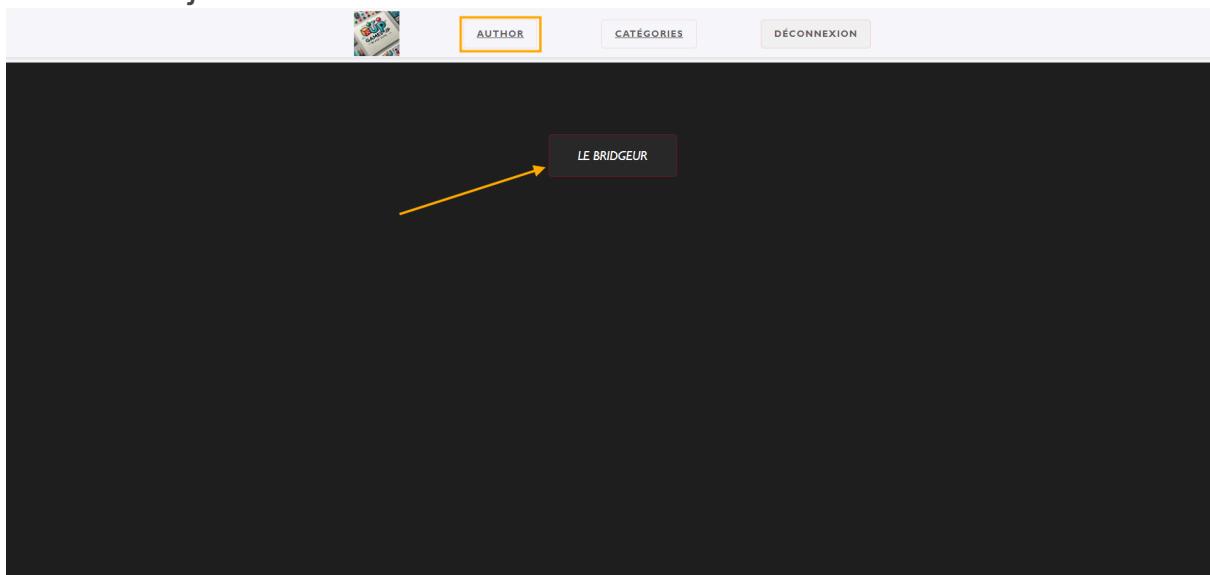


Bouton rédiger un avis

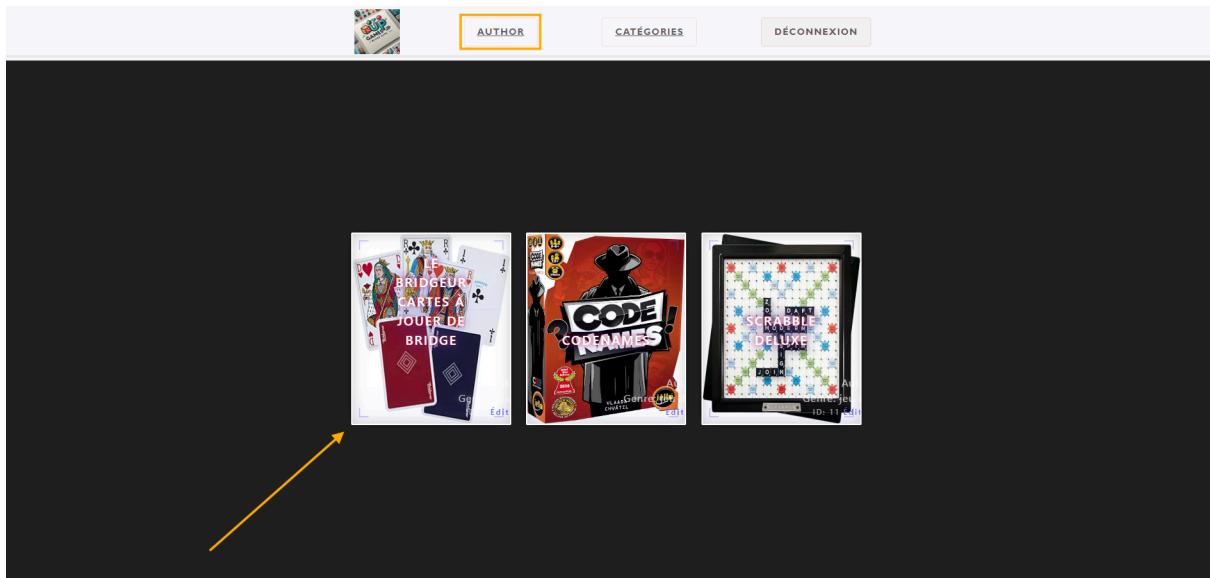


Onglet Author

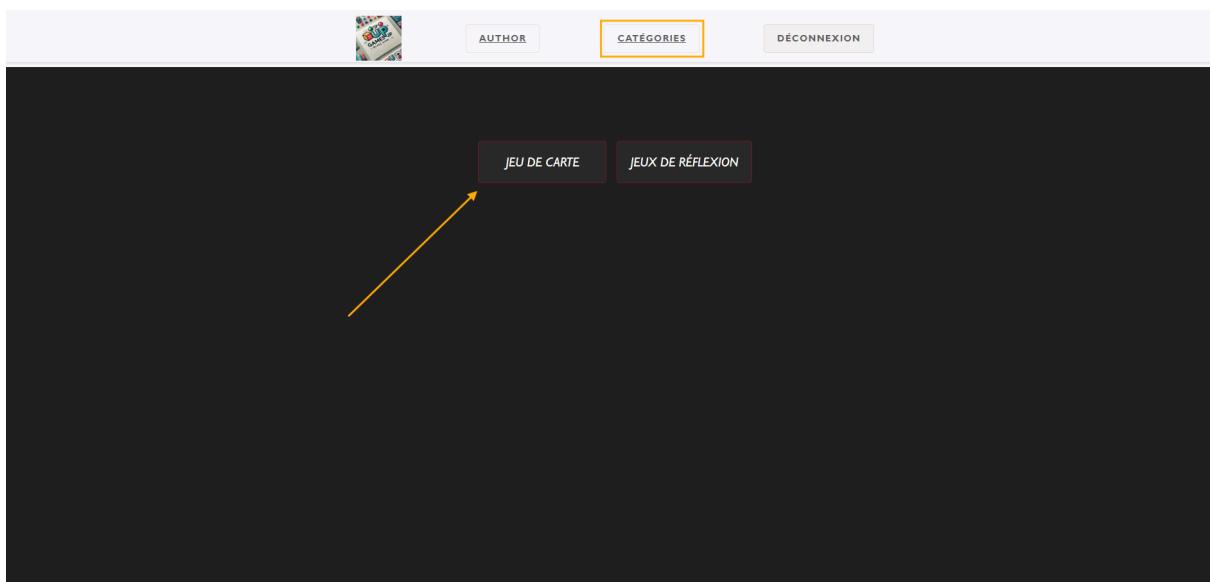
L'onglet author permet d'afficher la liste des jeux classée par auteur . Sur l'image suivante, un clique sur l'auteur LE BRIDGEUR permet d'afficher la liste des jeux dont LE BRIDGER est auteur.



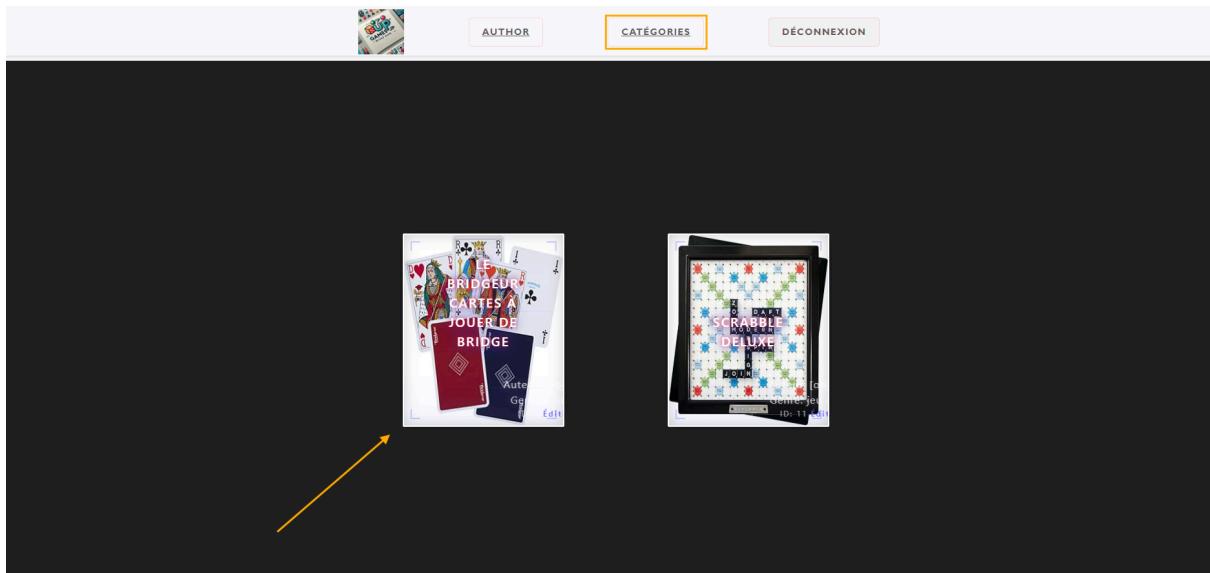
Ci après une image illustrant l'affichage des jeux dont LE BRIDGEUR est auteur



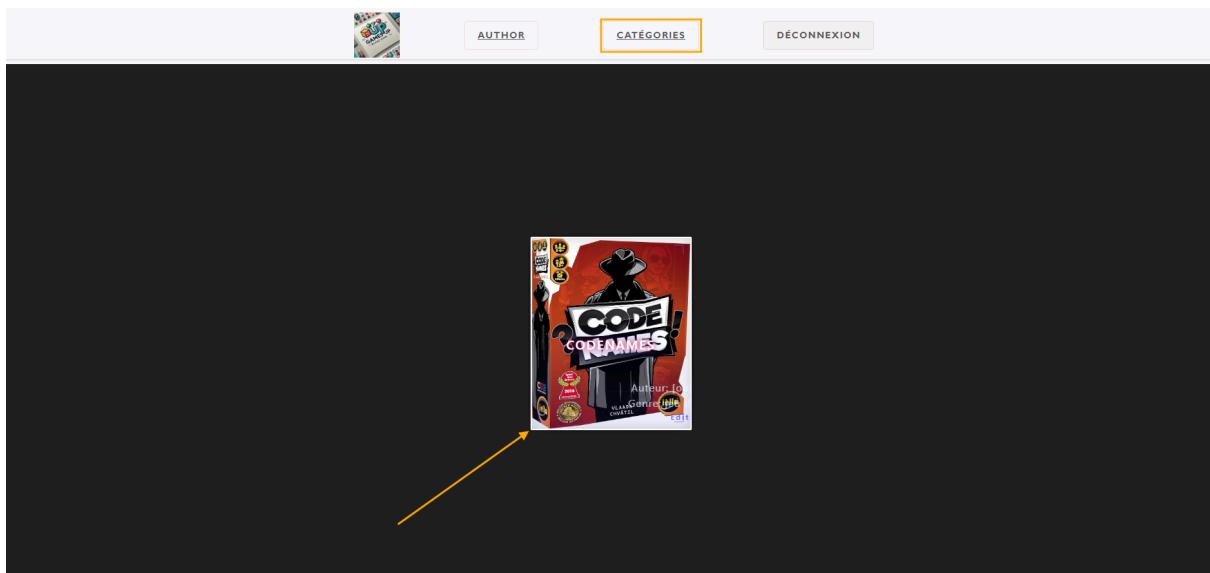
Onglet Catégories

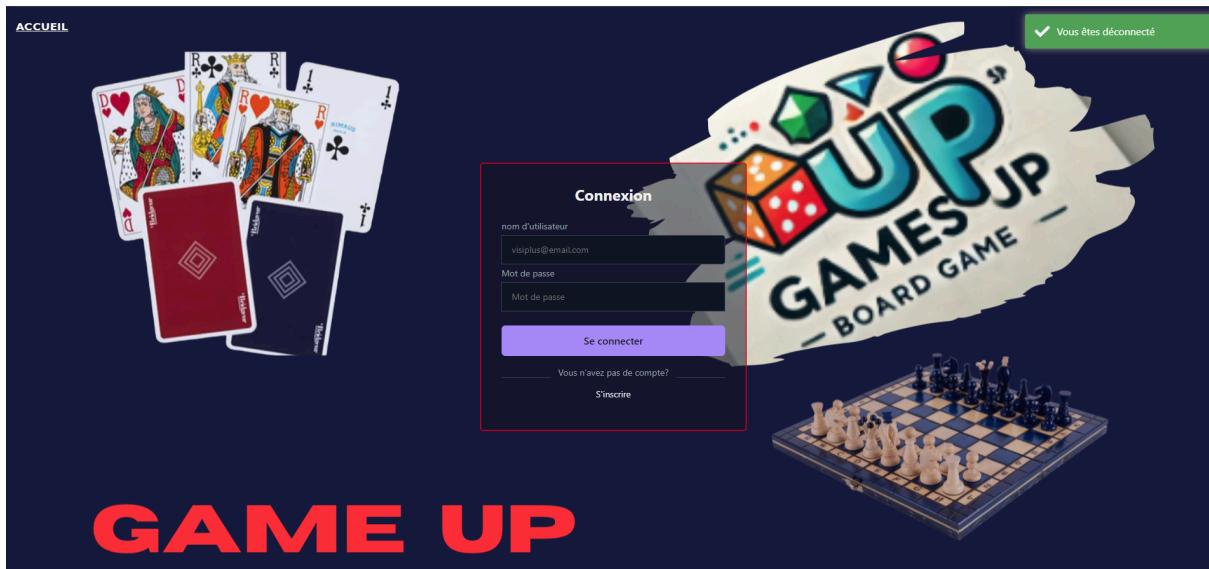


Ci après une image illustrant l'affichage des jeux de la catégorie JEU DE CARTE.



Ci après une image illustrant l'affichage des jeux de la catégorie JEU DE RÉFLEXION.





Compte administrateur

Sur le compte administrateur, il y a toutes les possibilités de gérer les utilisateurs, les jeux, les auteurs, les catégories, les publisher.

Un compte administrateur ne peut être créé que par un autre administrateur. Par ailleurs, le premier compte utilisateur doit être créé en backend via Postman, Insomnia ou directement par requête SQL en accédant au server du projet.

Page de gestion des utilisateurs

Liste des utilisateurs



DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

Authors

Catégories

Publishers

★ Liste des avis

★ Purchase

Déconnexion

Créer un utilisateur

ISCOD
Identifiant: 2
Email : iscod@email.com
Rôle : client
MODIFIER SUPPRIMER

VISIPLUS
Identifiant: 5
Email : visiplus@email.com
Rôle : administrateur
MODIFIER SUPPRIMER

Boutons d'ajout, de modification et de suppression d'utilisateur

DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

Authors

Catégories

Publishers

★ Liste des avis

★ Purchase

Déconnexion

Créer un utilisateur

ISCOD
Identifiant: 2
Email : iscod@email.com
Rôle : client
MODIFIER SUPPRIMER

VISIPLUS
Identifiant: 5
Email : visiplus@email.com
Rôle : administrateur
MODIFIER SUPPRIMER

Formulaire de modification d'utilisateur



The screenshot shows the VISIPLUS digital learning platform's dashboard. On the left, there is a sidebar with various menu items: Utilisateurs (highlighted with a yellow border), Jeux, Authors, Catégories, Publishers, Liste des avis, Purchase, and Déconnection. Below the sidebar is a button labeled "Créer un utilisateur". On the right, a modal window titled "MODIFIER DE L'UTILISATEUR" is open, showing fields for Nom (ISCOD), email (iscod@email.com), Mot de passe (*****), and Rôle (client). At the bottom of the modal are "Valider" and "Annuler" buttons.

Formulaire de création d'utilisateur

The screenshot shows the VISIPLUS digital learning platform's dashboard. On the left, there is a sidebar with various menu items. In the center, a modal window titled "AJOUTER UN UTILISATEUR" is open, containing fields for Nom (placeholder: Renseignez votre nom), Email (placeholder: Renseignez votre email), Mot de passe (placeholder: Renseignez votre mot de passe), Confirmez le mot de passe (placeholder: Renseignez à nouveau votre mot de passe), and Rôle (dropdown menu). At the bottom of the modal is an "Ajouter" button.



Page de gestion des jeux

The screenshot shows a dark-themed dashboard interface. On the left, a sidebar menu includes 'Utilisateurs', 'Jeux' (highlighted with a yellow box), 'Auteurs', 'Catégories', 'Publishers', 'Liste des avis', 'Purchase', and 'Déconnexion'. A central search bar has the placeholder 'Saisissez un mot'. Below it are three cards representing different games: 'BRIDGEUR CARTES A JOUER DE BRIDGE', 'CODE JEUX !', and 'SCRABBLE DELUXE'. At the bottom of the main area is a 'Créer un jeu' button.

Affichage des détails d'un jeu et boutons de modification et suppression de jeu

This screenshot shows the detailed view of a game from the previous page. The sidebar remains the same. The central area displays a large image of the 'SCRABBLE DELUXE' board. Below the image, the game title 'SCRABBLE DELUXE' is shown, along with its identifier (11), author ('Le Bridgeur'), genre ('Jeu de scrabble'), edition ('2000'), category ('Jeu de carte'), and a description in French. At the bottom of this panel are 'Modifier' and 'Supprimer' buttons.

Formulaire de modification d'un jeu



The screenshot shows the Visiplus digital learning platform's dashboard on the left and an 'Edit Game' modal on the right.

Dashboard:

- User icon: VISIPLUS
- Utilisateurs
- Jeux** (highlighted with a yellow border)
- Auteurs
- Catégories
- Publishers
- Liste des avis
- Purchase
- Déconnexion

Modifier un jeu (Edit Game) Modal:

- Saisissez un mot search bar
- Lien de l'image: <https://m.media-amazon.com/images/I/71uogzrbz2>
- Nom: Scrabble Deluxe
- Auteur: (dropdown menu)
- Genre: jeu de scrabble
- Catégorie: (dropdown menu)
- Edition: 2000
- Ajoutez une description:

Mattel Games Scrabble Deluxe. Jeu De Société De Mots Croisés Familière Pour Adultes Et Enfants. Version Française. Jeu De Lettres Pour 2 À 4 Joueurs. Jouet pour Enfant de 10 ans et Plus. Y9585
- Valider (blue button)
- Annuler (red button)

Formulaire de création de jeu

The screenshot shows the Visiplus digital learning platform's dashboard on the left and a 'Create Game' modal on the right.

Dashboard:

- User icon: VISIPLUS
- Utilisateurs
- Jeux
- Auteurs
- Catégories
- Publishers
- Liste des avis
- Purchase
- Déconnexion

Crée un jeu (Create Game) Modal:

- Lien de l'image: Insertez un lien vers l'image du jeu
- Nom: Renseignez le nom du jeu
- Auteur: (dropdown menu)
- Genre: Renseignez le genre
- Edition: Renseignez le numéro d'édition
- Catégorie: (dropdown menu)
- Ajoutez une description
- Créer (blue button)



Page de gestion et modification des auteurs

DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

Authors

Catégories

Publishers

★ Liste des avis

★ Purchase

Déconnexion

LE BRIDGEUR

MODIFIER SUPPRIMER

Créer un auteur

Affichage des jeux liés à un auteur

DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

Authors

Catégories

Publishers

★ Liste des avis

★ Purchase

Déconnexion

LE BRIDGEUR

MODIFIER SUPPRIMER

Créer un auteur



DASHBOARD

VISIPLUS

- 8 Utilisateurs
- + Jeux
- Authors**
- Catégories
- Publishers
- ★ Liste des avis
- ★ Purchase
- Déconnexion

Créer un auteur

The dashboard shows a sidebar with navigation links for users, games, authors (highlighted with a yellow border), categories, publishers, reviews, purchases, and logout. At the bottom is a button to 'Create a author'. The main area displays three game covers: 'Le Bridgeur' (Bridge cards), 'CODE NAMES!' (a deduction game), and 'SCRABBLE DELUXE'.

Affichage des détails d'un jeu lié à un auteur

DASHBOARD

VISIPLUS

- 8 Utilisateurs
- + Jeux
- Authors**
- Catégories
- Publishers
- ★ Liste des avis
- ★ Purchase
- Déconnexion

Créer un auteur

The dashboard is identical to the previous one, but the central area is dominated by a large image of the 'CODE NAMES!' board game box. Below the image, detailed product information is displayed:

Identifiant : 5
Auteur :
Genre : jeu de déduction
Edition : 2000
Catégorie : Jeux de réflexion
Description : Le jeu culte de déduction Plébiscité par des milliers de joueurs ! Trouvez les bons mots et évitez l'assassin – Jeu de société addictif en équipe pour des soirées inoubliables
Publié par : VISIPLUS

Modification d'un author



DASHBOARD

VISIPLUS

- 8 Utilisateurs
- + Jeux
- Authors**
- Catégories
- Publishers
- ★ Liste des avis
- ★ Purchase
- Déconnexion

Créer un auteur

LE BRIDGEUR
MODIFIER **SUPPRIMER**

DASHBOARD

VISIPLUS

- 8 Utilisateurs
- + Jeux
- Authors**
- Catégories
- Publishers
- ★ Liste des avis
- ★ Purchase
- Déconnexion

Créer un auteur

MODIFIER UN AUTEUR
Nom de l'auteur
Le Bridgeur
Valider **Annuler**

Suppression d'un author



DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

Authors

Catégories

Publishers

★ Liste des avis

★ Purchase

Déconnexion

Créer un auteur

LE BRIDGEUR

MODIFIER SUPPRIMER

Warning de confirmation de suppression d'un auteur

DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

Authors

Catégories

Publishers

★ Liste des avis

★ Purchase

Déconnexion

Créer un auteur

ATTENTION !

Vous êtes sur le point de supprimer la catégorie Le Bridgeur!

Cette action est irréversible et supprimera tous les jeux de cette catégorie.

Confirmer Annuler



Formulaire de création d'auteur

DASHBOARD

Utilisateurs

Jeux

Catégories

Authors

Publishers

Liste des avis

Purchase

Déconnexion

Créer

Onglet de gestion des CRUD des catégories

DASHBOARD

VISIPLUS

8 Utilisateurs

Jeux

Authors

Catégories

Publishers

Liste des avis

Purchase

Déconnexion

JEU DE CARTE

JEUX DE RÉFLEXION

MODIFIER SUPPRIMER MODIFIER SUPPRIMER

Créer une catégorie

Affichages de la liste des jeux liés à une catégorie données



DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

Authors

Catégories

Publishers

★ Liste des avis

★ Purchase

Déconnexion

Créer une catégorie

JEU DE CARTE

JEUX DE RÉFLEXION

MODIFIER SUPPRIMER MODIFIER SUPPRIMER

Affichages de la liste des jeux liés à la catégorie JEU DE CARTE

DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

Authors

Catégories

Publishers

★ Liste des avis

★ Purchase

Déconnexion

Créer une catégorie

Affichages de la liste des jeux liés à la catégorie JEU DE RÉFLEXION



DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

♥ Authors

✓ Catégories

Publishers

★ Liste des avis

★ Purchase

Déconnexion

Créer une catégorie



Affichage des détails d'un jeu lié à une catégorie

DASHBOARD

VISIPLUS

8 Utilisateurs

+ Jeux

♥ Authors

✓ Catégories

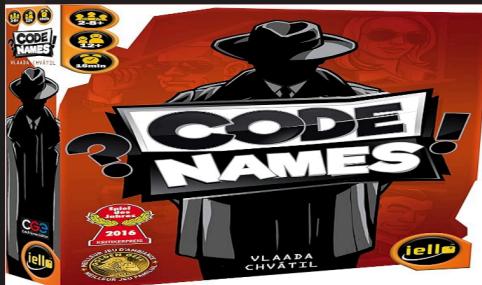
Publishers

★ Liste des avis

★ Purchase

Déconnexion

Créer une catégorie



Identifiant : 5

Auteur : Le Bridger

Genre : jeu de déduction

Édition : 2000

Catégorie :

Description : Le jeu culte de déduction Plébiscité par des milliers de joueurs ! Trouvez les bons mots et évitez l'assassin – Jeu de société addictif en équipe pour des soirées inoubliables.

Publié par : VISIPLUS



Formulaire de modification de catégorie

DASHBOARD

VISIPLUS

- Utilisateurs
- Jeux
- Authors
- Catégories
- Publishers
- Liste des avis
- Purchase
- Déconnection

Créer une catégorie

JEU DE CARTE	JEUX DE RÉFLEXION
MODIFIER	SUPPRIMER
MODIFIER	SUPPRIMER

DASHBOARD

VISIPLUS

- Utilisateurs
- Jeux
- Authors
- Catégories
- Publishers
- Liste des avis
- Purchase
- Déconnection

Créer une catégorie

MODIFIER LA CATÉGORIE

Type de votre catégorie

Valider **Annuler**



Suppression de catégorie

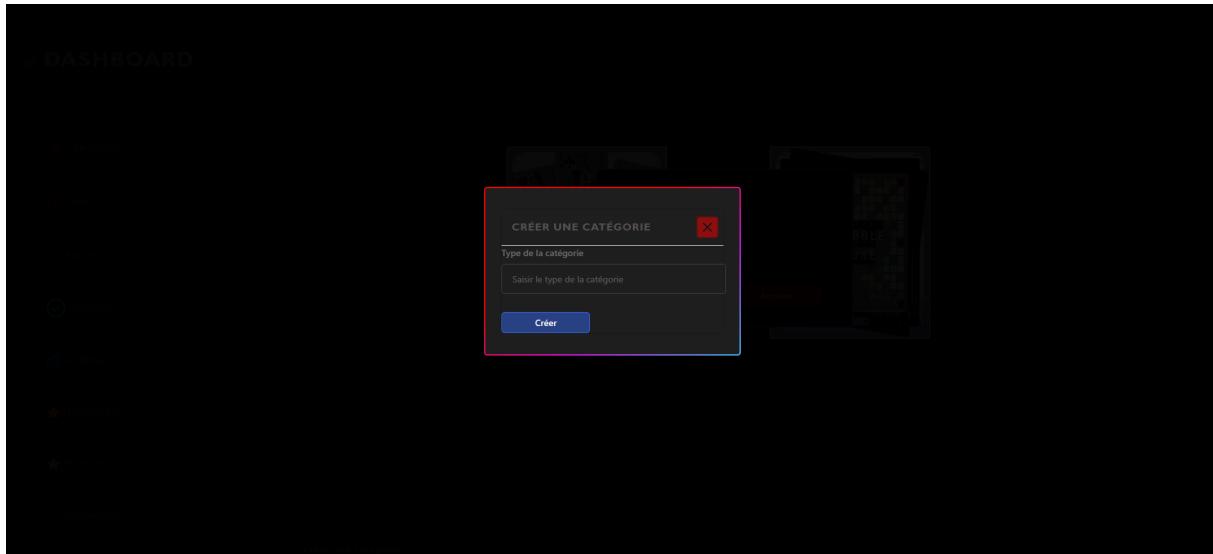
The screenshot shows the VISIPLUS dashboard with a sidebar on the left containing various navigation options like Utilisateurs, Jeux, Authors, and Catégories. The 'Catégories' option is highlighted with a yellow box. On the right, there are two categories displayed: 'JEU DE CARTE' and 'JEUX DE RÉFLEXION'. Each category has a 'MODIFIER' button and a 'SUPPRIMER' button. A yellow arrow points from the 'Catégories' button in the sidebar to the 'SUPPRIMER' button of the 'JEU DE CARTE' category.

Warning de confirmation de suppression d'une catégorie

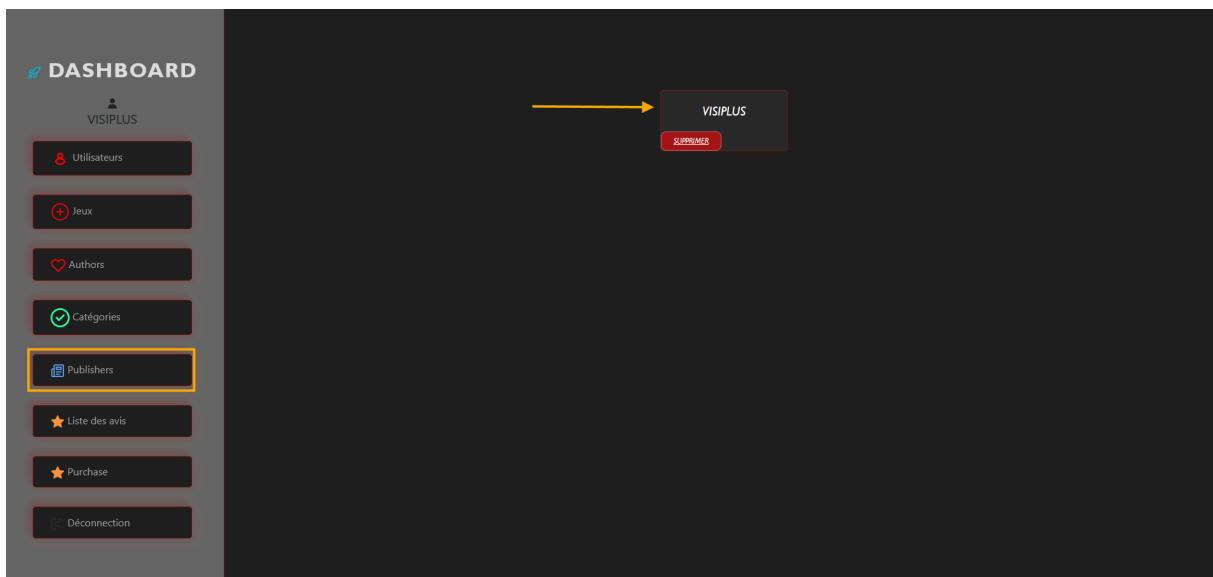
The screenshot shows the VISIPLUS dashboard with the 'Catégories' section selected. A confirmation dialog box is overlaid on the screen. The dialog has a yellow background and contains the following text:
ATTENTION !
Vous êtes sur le point de **supprimer** la catégorie **Jeu de carte**.
Cette action est irréversible et supprimera tous les jeux de cette catégorie.
At the bottom are two buttons: 'Confirmer' (Confirm) and 'Annuler' (Cancel).



Formulaire de création de catégorie



Page de gestions des publishers





Affichage de la liste des jeux lié à un publisher

The screenshot shows the Visiplus dashboard with a sidebar on the left containing navigation links: Utilisateurs, Jeux, Authors, Catégories, Publishers (highlighted with a yellow border), Liste des avis, Purchase, and Déconnexion. The main area displays three game cards: 'LE BRIDGEUR CARTES À JOUER DE BRIDGE' (Author: Le Bridgeur, Genre: Jeu de cartes, Edition: 2016, Category: Jeu de carte, Description: Le Bridgeur Cartes à Jouer de Bridge (vendues par 10)), 'CODE CONNAISSANCES' (Author: Le Bridgeur, Genre: Jeu de cartes, Edition: 2016, Category: Jeu de carte, Description: Le Bridgeur Cartes à Jouer de Bridge (vendues par 10)), and 'SCRABBLE DELUXE' (Author: Le Bridgeur, Genre: Jeu de cartes, Edition: 2016, Category: Jeu de carte, Description: Le Bridgeur Cartes à Jouer de Bridge (vendues par 10)).

Affichage de la description d'un jeu lié à ce publisher

The screenshot shows the Visiplus dashboard with the same sidebar as the previous screenshot. The main area is a modal window displaying the details for 'LE BRIDGEUR CARTES À JOUER DE BRIDGE'. It includes a large image of the cards, the title, identifier, author, genre, edition, category, description, and publication information.

Warning de confirmation de suppression d'un publisher



DASHBOARD

VISIPLUS

- 8 Utilisateurs
- + Jeux
- Authors
- Catégories
- Publishers**
- ★ Liste des avis
- ★ Purchase
- Déconnexion

ATTENTION !

Vous êtes sur le point de **supprimer** le publieur VISIPLUS! Cette action est irréversible et supprimera tous les jeux liés à ce publieur

Confirmer **Annuler**

Onglet de gestion des avis

DASHBOARD

VISIPLUS

- 8 Utilisateurs
- + Jeux
- Authors
- Catégories
- Publishers**
- ★ Liste des avis**
- ★ Purchase
- Déconnexion

ISCOD Note : 5

Publié le :05-07-25 à 00:30

Une expérience exceptionnelle

Paul DOUMER Note : 8

Publié le :05-04-25 à 14:30

Une expérience satisfaisante

Paul DOUMER Note : 8

Publié le :05-04-25 à 14:30

Une expérience satisfaisante

ISCOD Note : 10

Publié le :06-07-25 à 10:34

Commentaire de test



Toastr

The screenshot shows a dark-themed application interface. A central modal window titled "CRÉE UN JEU" is open, prompting the user to enter various details about a game. The fields include "Lien de l'image" (with a placeholder "Insérez un lien vers l'image du jeu"), "Nom" (placeholder "Renseignez le nom du jeu"), "Auteur" (placeholder "Renseignez le nom de l'auteur"), "Genre" (placeholder "Renseignez le genre"), "Edition" (placeholder "Renseignez le numéro d'édition"), "Catégorie" (placeholder "Renseignez la catégorie"), and "Ajoutez une description". At the bottom of the modal is a blue "Créer" button. In the top right corner of the main window, there is an orange notification bar with a warning icon and the text "Veuillez remplir tous les champs correctement."

The screenshot shows a dashboard with a sidebar containing links like "Utilisateurs", "Jeux", "Authors", "Catégories", "Publishers", "Liste des avis", "Purchase", and "Déconnexion". A "Créer un jeu" button is located at the bottom of the sidebar. The main area features a modal window titled "MODIFIER UN JEU" for editing a game. The fields are identical to the creation modal: "Lien de l'image" (with the URL "https://m.media-amazon.com/images/I/81YDYNtCI"), "Nom" (set to "CODENAMES"), "Auteur" (placeholder), "Genre" (placeholder), "Edition" (set to "2000"), and "Catégorie" (placeholder). Below these fields is a large text area containing the description: "Le jeu culte de déduction plébiscité par des milliers de joueurs ! Trouvez les bons mots et évitez l'assassin – Jeu de société addictif en équipe pour des soirées inoubliables". At the bottom of the modal are "Valider" and "Annuler" buttons. In the top right corner of the main window, there is a green notification bar with a checkmark icon and the text "CODENAMES Mise à jour réussie".



The screenshot shows a modal window titled "CRÉE UN JEU" (Create a Game) over a dark dashboard background. The form fields include:

- Lien de l'image: image/ascension-tactics-p-image-101194-grande.jpg
- Nom: EVERSPACE
- Auteur: Le Bridgeur
- Genre: Action
- Édition: 3333
- Catégorie: Jeu de carte
- Description: Ajoutez une description
- Description: Description

A yellow arrow points from the bottom right of the modal to the "Créer" (Create) button at the bottom.

At the top right of the dashboard, a green notification bar says "✓ EVERSPACE Jeu créé avec succès".

The dashboard shows a sidebar with navigation links: Utilisateurs, Jeux, Authors, Catégories, Publishers, Liste des avis, Purchase, and Déconnexion. A "Créer un jeu" button is at the bottom of the sidebar.

The main area displays six game cards:

- BRIDGEUR CARTES À JOUER DE BRIDGE (Auteur: Le Bridgeur, Genre: Jeu de carte, Édition: 2017)
- CODE! CODEGAMES! (Auteur: Le Bridgeur, Genre: jeu d'agilité, Édition: 2020)
- ZOO DARTS SCRABBLE DELUXE (Auteur: Le Bridgeur, Genre: jeu de plateau, Édition: 2000)
- MAFIA (Auteur: Le Bridgeur, Genre: Action, Édition: 2023)
- EVERSPACE (Auteur: Le Bridgeur, Genre: Action, Édition: 2025)
- EVERSPACE (Auteur: Le Bridgeur, Genre: Action, Édition: 1333)

A search bar at the top right says "Saisissez un mot". A green notification bar at the top right says "✓ Jeux chargés avec succès". Below it, another bar says "Vous êtes connecté en tant que VISIPLUS".

Conclusion

Cette nouvelle version du game_up dispose de toutes les méthodes CRUD et fonctionnalités demandées tout en respectant les bonnes pratiques de la méthode SOLID. Ces méthodes peuvent être vérifiées en faisant des tests avec les différents endpoints répertoriés dans le tableau un peu plus haut avec postman ou insomnia.



J'ai réalisé le frontend pour permettre de rendre le site plus facile à testé et plus agréable. Toutes les méthodes CRUD y sont possibles, cependant, par manque de temps, j'ai uniquement rendu possible les affichages de la liste de jeux, utilisateurs, liste des catégories, liste des auteurs, la liste des publications, une barre de recherche des jeux ainsi que la création, modification et suppression des jeux, utilisateur, catégories et auteurs, sur le compte client, il est possible donner un avis et inversement sur compte administrateur, il est possible d'accéder aux avis .

L'ajout des jeux dans la wishlist ou la liste des achats ne sont pas actuellement disponibles. Cependant avec assez de temps, je pourrais ajouter ces fonctionnalités permettant d'améliorer le projet.

J'ai réalisé les tests avec junit atteignant un coverage de 65%. Cette couverture du test réduit les risques d'avoir des bugs ou le crash du backend.

En conclusion toutes les fonctionnalités demandées sont définies dans le projet et peuvent être testées avec postman. La couverture des tests unitaire atteint les 65%, ce qui est un indicateur de performance, l'API python est également intégré ainsi que le côté frontend. Cependant il est à noter que toutes les fonctionnalités ne sont pas implémentées dans le frontend. Toutefois je pourrais améliorer le projet en général frontend en particulier si je dispose de plus de temps.



REFERENCES

- [PRIMENG : <https://primeng.org/carousel>, *Bibliothèque de composants UI*];
- [BOOTSTRAP : <https://getbootstrap.com>, *Framework CSS et JS*];
- [CANVAS : <https://www.canva.com>, *création de bannière de portfolio et des design des pages connexion et inscription*];
- [POSTIMAGE : <https://postimages.org>, *convertir votre image en lien*];
- [SHIELDS : <https://shields.io/badges>, *personnaliser les icônes des réseaux sociaux connus*];
- [UIVERSE : <https://uiverse.io/forms>, *librairie UI pour les composant HTML et CSS dynamiques*];
- [FREEPIK : <https://fr.freepik.com>, *télécharger des icônes gratuitement (limité à partir d'un nombre donné)*];



☞ Prenez connaissance de ce tableau qui vous indique les critères sur lesquels vous serez évalué pour chacune des compétences. Les critères vous indiquent également les points d'attention que vous devez avoir dans le rendu de votre copie.

Compétences évaluées	Critères d'évaluation
C.14. Concevoir une architecture adéquate, à partir des exigences et attributs de qualité en réalisant des diagrammes d'architecture et en les formalisant dans un support technique à destination de l'équipe de développement afin de faciliter son usage, son adoption, sa robustesse et son évolutivité.	<ul style="list-style-type: none">- Les diagrammes UML proposés sont cohérents et décrivent parfaitement l'application proposée.- Le document est clair dans sa présentation et son contenu.- La description du travail effectué met en avant les qualités et les défauts de la démarche de travail.
C.16. Implémenter un logiciel de qualité, en choisissant des structures de données adaptées et des algorithmes pertinents afin d'assurer la robustesse du logiciel.	<ul style="list-style-type: none">- L'API Spring est fonctionnelle.- Les bonnes pratiques de Java et Spring sont respectées.- Les classes Models sont cohérentes.- Des DTOs sont utilisés par l'API.- Hibernate est utilisé pour gérer la couche data.



C.17. Tester le logiciel et l'application à plusieurs niveaux en utilisant les méthodologies de test éprouvées afin de garantir la conformité du logiciel au regard des spécifications et la non-régression des fonctionnalités déjà développées.	<ul style="list-style-type: none">- Les tests d'intégrations et les tests unitaires sont fournis.- Le taux de couverture est au moins de 70%.- Spring Security est en place.- La gestion des rôles est en place.
C.18. Concevoir une application d'analyse de données massives en intégrant un programme d'apprentissage automatique (machine learning) au développement du logiciel et en utilisant des réseaux de neurones, des algorithmes d'optimisation et de recommandation afin de faire ressortir les tendances utilisateurs.	<ul style="list-style-type: none">- Le code d l'API Python est complété par l'algorithme ML (non entraîné).- L'API Spring interroge l'API Python qui lui renvoie des données de tests.



Instructions

Question 1. Fournir l'API Spring complétée.

Question 2. Fournir l'API Python complétée.

Question 3. Fournir la documentation complète.