UFR Sciences et Techniques

UNIVERSITÉ DE ROUEN NORMANDIE

Document d'Architecture Logiciel

Version : 1.0

Date : 12 Janvier 2023

Rédigé par : Anass El Gharbaoui

Relu par : Anass El Gharbaoui, Corentin Pille, Robin Sapin, Sid Ahmed Brahimi, Yacine Ben Ahmed, Michel Nassalang, Mohamed Cherfi

Approuvé par :

UFR Sciences et Techniques

UNIVERSITÉ DE ROUEN NORMANDIE

MISES À JOUR

Version	Date	Modifications réalisés
1.0	25/12/2023	Création
1.1	02/01/2024	Modification d'architecture
1.2	24/01/2024	Architecture statique et liste des composants
1.3	02/02/2024	Reprise du document
1.4	15/05/2024	Mise à jour du document pour la conclusion du projet

1. Objet du document

Le Document Architecture Logicielle (DAL) décrit l'organisation et la structure de notre plateforme Agora du point de vue architectural . Il vise à fournir une vision globale de la manière dont les composants sont organisés et interagissent entre eux.

L'objectif principal de notre groupe est de fournir une plateforme permettant d'héberger l'ensemble des jeux qui seront développés par les membres du deuxième groupe travaillant sur ce projet, afin de pouvoir être joués en ligne avec plusieurs joueurs.

Les trois contraintes majeures qui nous sont imposés sont les suivantes :

- Utiliser la framework PHP Symfony, dans sa version stable la plus récente (STB-REA-01) pour développer le site
- Implémenter l'ensemble des fonctionnalités demandés par le client, et décrit dans le document de Spécification technique des Besoins
- Établir une documentation explicative et précise du code de notre plateforme, afin de permettre aux futurs groupes qui travaillent sur le projet de comprendre le code, ainsi que d'implémenter de nouvelle fonctionnalités sans compromettre les anciennes.

2. Documents applicables et références

Pour la rédaction de ce document on va se baser sur le modèle canva de DAL fourni par l'enseignant responsable, les DAL des versions dernières du projet agora, ainsi que le document de Spécification Techniques des Besoins.

3. Terminologie et sigles utilisés

MVC:	Model vue controller
Agora:	Le nom du projet
Symfony:	Le framework php utilisé pour le développement
PHP:	Langage de programmation libre source.
Twig:	Moteur de templates CSS
Composer:	Le gestionnaire des dépendances de Symfony
MariaBD:	Le système de gestion de base de données
STB:	Spécification technique des besoins.

Table (ou Board)	C'est le composant représentant le salon de jeu d'une partie. Un joueur crée une table, associé à une partie d'un jeu, et attend que la table soit complète avant de lancer la partie. C'est ce composant qui va communiquer avec les services de jeu développés par l'autre équipe
Forum	Espace de discussion ouvert du site. Il existe un forum centré sur chaque jeu
Topic	Sujet de discussion d'un forum. C'est à l'intérieur d'un topic que les joueurs peuvent discuter, autour du sujet du topic.

4. Configurations requise

4.1 Configuration requise pour un utilisateur

Afin de permettre une bonne utilisation du site, la seule configuration requise pour un utilisateur du site est d'y naviguer avec l'un navigateur pris en charge par la plateforme (Safari, Mozilla, Chrome, Edge) (STB-OPE-03), avec un ordinateur, une tablette ou un smartphone (STB-INT-01)

4.2 Configuration de l'environnement de développement

Voici les technologies utilisés pour le développement de la plateforme :

- Ubuntu en version 22.04 (Long Time Support)
- PHP en version 8.3.0 (STB-REA-01)
- Symfony en version 6.4.1 (STB-REA-01)
- Composer en version 2.6.6 pour la gestion des dépendances
- Tailwind en version 3.0 (STB-REA-02)
- MariaDB en version 10.11.2 (STB-REA-04)
- Apache en version 2.4.58
- PHPStan en version 1.10 pour la vérification du code
- phpUnit en version 9.6.13 pour les tests unitaires (STB-REA-04)

4.3 Configuration de l'environnement de déploiement

Voici les technologies qu'on va utiliser en plus des précédentes, pour le déploiement de la plateforme :

- Docker en version 24.0.7 (STB-REA-06)
- GitLab en version 16.5.1 (STB-REA-05)

5. Présentation de la structure général

5.1 Architecture Statique

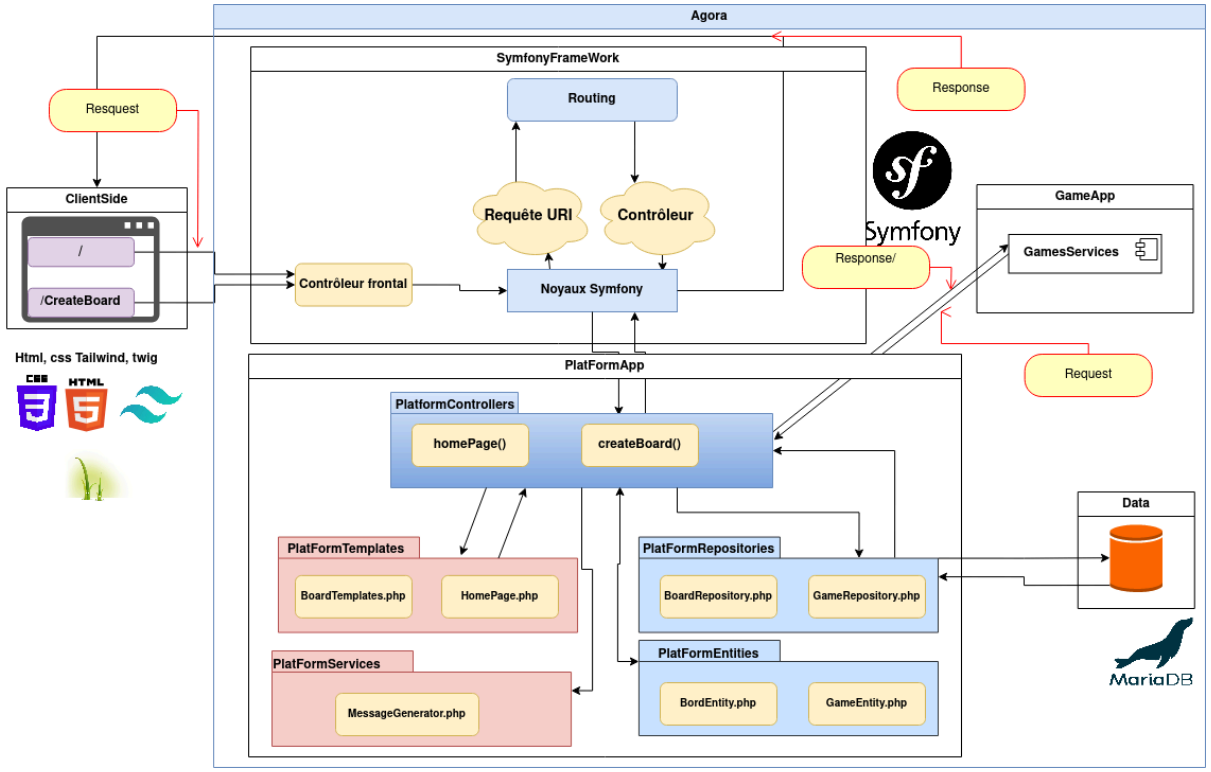

Schéma de l'architecture statique de AGORA côté site, dans le cas d'une requête createBoard()

Concernant l'architecture de notre application, on va se baser sur le modèle MVC (model-view-controller), qui divise l'application en 3 parties :

- Modèle (Model) : Représente les données et la logique métier.

Implémentation dans Symfony : Les entités et les classes Repository sont utilisées pour modéliser les données et interagir avec la base de données. Les entités définissent la structure des données, tandis que les repositories fournissent des méthodes pour accéder aux données.

- Vue (View) : Gère l'affichage des données et interagit avec l'utilisateur.

Implémentation dans Symfony : Les vues sont généralement créées à l'aide de fichiers de modèle (Twig). Les vues sont responsables de la présentation des données et peuvent être rendues avec les données provenant du contrôleur.

- Contrôleur (Controller) : Gère les requêtes de l'utilisateur, interagit avec le modèle pour récupérer les données nécessaires, et passe ces données à la vue appropriée.

Implémentation dans Symfony : Les contrôleurs dans Symfony sont des classes qui héritent de la classe de base AbstractController. Ils contiennent des actions qui répondent aux

requêtes HTTP et coordonnent l'interaction entre le modèle et la vue. Les contrôleurs retournent généralement des réponses HTTP qui peuvent être des vues, du JSON, etc.

5.2 Structure de projet

Selon les bons pratiques de symfony, le répertoire principal de l'application sera découpée de la manière suivante :

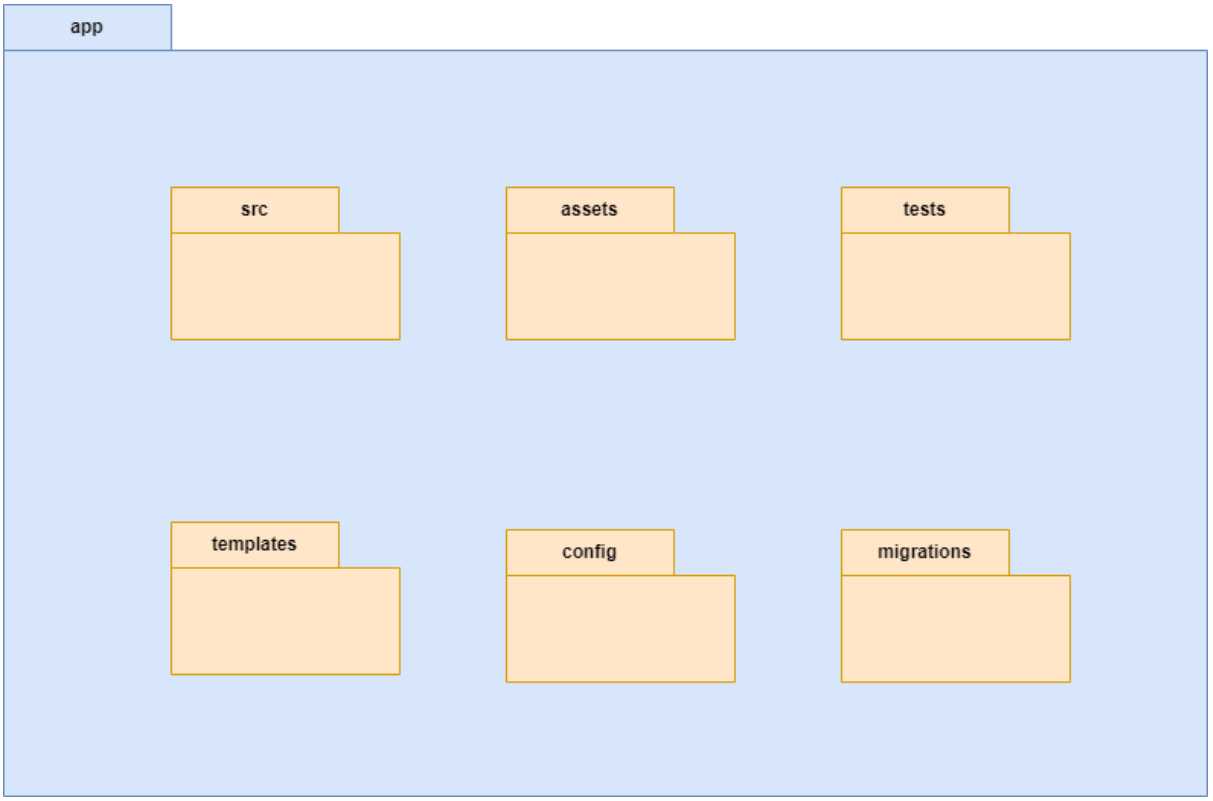

Schéma de l'architecture du répertoire principal du projet AGORA

On retrouve donc 6 sous-répertoires principaux qui sont les suivants :

- **src** : Ce dossier contient le code source de votre application. Il sera divisé en plusieurs sous-dossiers comme Controller, Entity, Repository, etc, qui auront chacun une responsabilité particulière dans l'architecture de votre application. Ce sous-dossier sera détaillé plus loin dans cette partie
- **assets :** Ce dossier est utilisé pour stocker les fichiers statiques de l'application tels que les images, les feuilles de style CSS, les scripts JavaScript, etc. Dans Symfony, les assets peuvent être gérés à l'aide de Webpack Encore ou d'autres outils d'empaquetage.
- **tests :** Ce dossier contient les tests unitaires et fonctionnels de l'application. Les tests permettent de vérifier que l'application fonctionne correctement et de détecter les éventuels bugs.

- **templates :** Ce dossier contient les templates Twig de l'application. Ils permettent de séparer le code PHP de la présentation HTML de l'application. Un détail de ce répertoire sera fait plus loin dans cette partie
- **config :** Ce dossier contient les fichiers de configuration de l'application. Ils permettent de configurer les différents composants de Symfony et les bundles utilisés dans l'application.
- **migrations :** Ce dossier est utilisé pour gérer les migrations de la base de données. Les migrations sont des scripts SQL qui permettent de modifier la structure de la base de données de manière incrémentale et réversible. Ils sont créés dans Symfony à l'aide du composant Doctrine.

Le code source de notre application est donc contenu dans le sous-dossier src, dont le détail est le suivant :

Schéma de l'architecture du répertoire "src" du projet AGORA

On retrouve donc 9 sous-répertoires qui sont les suivants :

- **Controller :** Ce dossier contient les contrôleurs de l'application. Les contrôleurs sont des classes PHP qui gèrent les requêtes HTTP entrantes et génèrent les réponses HTTP à envoyer aux clients. Un contrôleur est composé de fonctions, associées chacunes à une route (un url spécifique de l'application) qui vont exécuter la fonction lorsqu'un utilisateur envoie une requête à cette route. Les contrôleurs vont interagir avec les autres composants de l'application (Service, Form, Entity…) pour effectuer les tâches nécessaires.
- **Data :** Ce dossier va contenir des fichiers détaillant les différents paramètres que pourront avoir des formulaires de recherches (comme la recherche de tables, d'utilisateurs , etc…).
- **DataFixtures :** Ce dossier contient les fixtures de données de l'application. Les fixtures sont des données d'exemple qui sont chargée dans la base de données de l'application pour les besoins de test ou de démonstration.
- **Entity :** Ce dossier contient les entités de l'application. Les entités sont des classes PHP qui représentent les objets métier de l'application et sont mappées sur les tables de la base de données à l'aide de Doctrine ORM. Ce sont les données de ces entités qui forment principalement l'ensemble des données contenues dans la base de données.
- **Form :** Ce dossier contient les formulaires de l'application. Chaque fichier décrit un type différent de formulaire qui pourra être réutiliser à certains endroits de l'application
- **Repository :** Ce dossier contient les repositories de l'application. Les repositories sont des classes PHP qui permettent d'effectuer des opérations de base de données personnalisées sur les entités de l'application. Les repositories peuvent être utilisés pour effectuer des requêtes complexes ou pour implémenter des règle métier spécifiques.
- **Security :** Ce dossier contient les composants de sécurité de l'application. Les composants de sécurité sont des classes PHP qui permettent de gérer l'authentification et l'autorisation des utilisateurs de l'application. Les composants de sécurité peuvent interagir avec les autres composants de l'application, tels que les contrôleurs et les entités, pour effectuer les tâches nécessaires.
- **Service :** Ce dossier contient les services de l'application. Les services sont des classes PHP qui fournissent des fonctionnalités réutilisables pour l'application. Ce sont notamment ces classes qui contiendront la majorité du code métier qui sera invoqué par les contrôleurs de l'application

- **TwigExtension :** Ce dossier contient les extensions Twig de l'application. Les extensions Twig sont des classes PHP qui permettent d'ajouter des fonctionnalités personnalisées au moteur de template Twig. Les extensions Twig peuvent être utilisées pour créer des filtres, des fonctions ou des tags Twig personnalisés.

Chaque sous dossier sera séparé en deux sous-dossier, un sous-dossier “Game” et “Platform” afin de bien faire la distinction entre les fichiers des deux groupes d’études travaillant sur le projet. Un détail de chaque fichier de ce répertoire est fourni dans le document technique “Partie Plateforme : Description de la bibliothèque de code”

Le dossier src va donc contenir l’essentiel des éléments composant la partie “Back” de notre application. Pour la partie visuelle (“front”), c’est le répertoire templates qui va contenir l’ensemble des fichiers twig qui composeront la vue de notre plateforme web. Le détail de ce répertoire est le suivant :

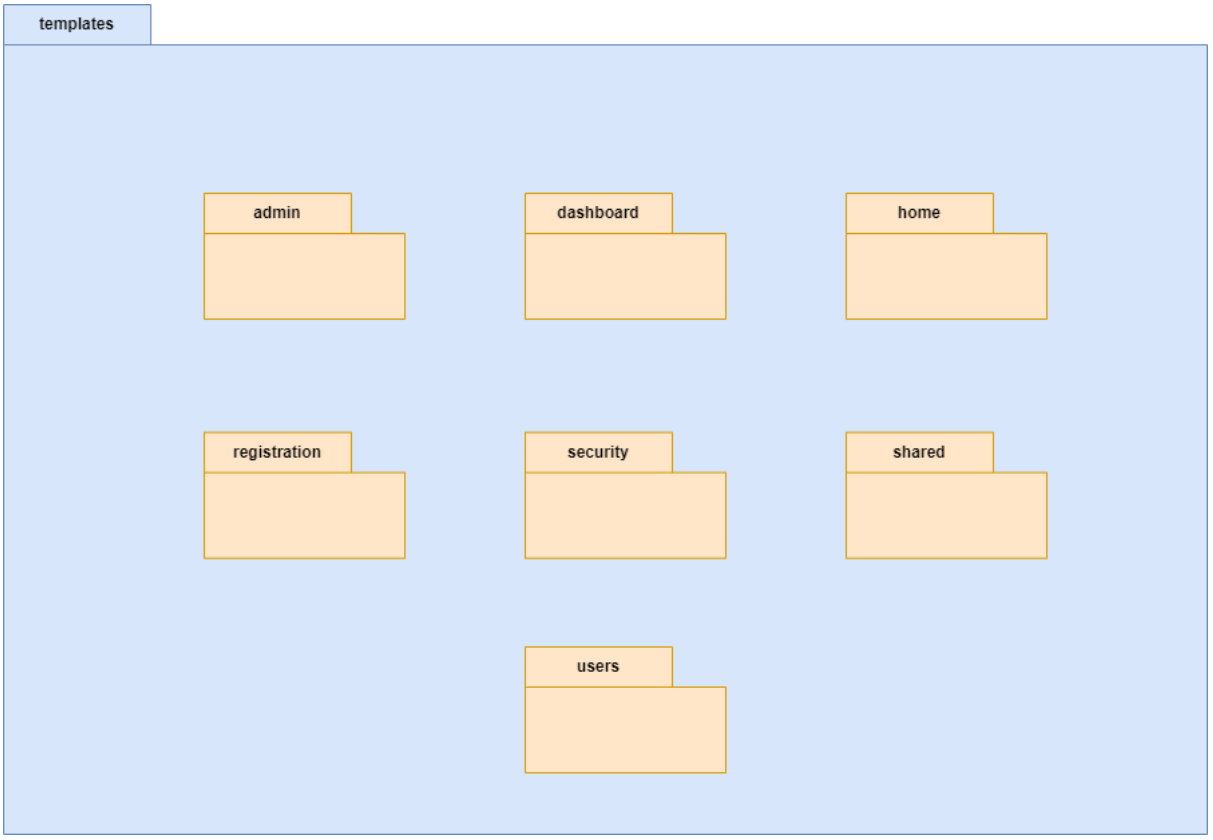

Schéma de l’architecture du répertoire “templates” du projet AGORA

On retrouve donc 7 sous-répertoires qui sont les suivants :

- **admin :** Ce répertoire contient les templates qui forment l’interface d’administration du site
- **dashboard :** Ce répertoire contient les templates qui forment l’interface d’un utilisateur connecté (un joueur donc) sur le site.

- **home :** Ce répertoire contient les templates qui forment la vue d'un utilisateur non-connecté sur le site (notamment les pages d'accueil)
- **registration :** Ce répertoire contient les templates décrivant les pages d'inscription d'un utilisateur à la plateforme
- **security :** Ce répertoire contient les templates décrivant les pages de connexion d'un utilisateur à la plateforme, ou en lien avec les données de connexion de l'utilisateur (comme la page de récupération de mot de passe)
- **shared :** Ce répertoire contient les templates réutilisables à de nombreux endroits de la plateforme (comme la navbar) afin de factoriser au mieux le code
- **users :** Ce répertoire contient les templates représentants les pages de paramétrage des différents types de compte utilisateur de la plateforme

5.3 Déroulement d'une requête symfony

- Client demande URL
  - Le contrôleur frontal de Symfony traite la demande en premier.
- Noyau reçoit la demande d'URL
  - Noyau appelle le service de routing.
  - Le service de routing indique le contrôleur à appeler pour l'URL demandée par le client.
- Noyau appelle le contrôleur
  - Contrôleur, si nécessaire, peut appeler le modèle et générer la vue.
- Une fois le contrôleur a terminé de générer la vue
  - Il renvoie le résultat au noyau.
- Noyau transmet la réponse au client.

6. Définitions des entités

Cette partie fournit une description de chaque entité utilisée par la partie plateforme du projet AGORA. Le détail des méthodes de chaque entité est fourni dans le document technique "Partie Plateforme - Description de la bibliothèque de code"

1. Board

Cette classe entité représente une table de jeu. C'est donc sur elle que doivent se retrouver les différents joueurs de la partie. Elle doit aussi contenir les différents paramètres de la partie, à savoir le nombre de joueurs, le temps d'inactivité avant exclusion d'un joueur, et la liste des joueurs invités.

Attributs :

- id (Int) : L'id de la table dans la base de donnés "Boards"
- nbUserMax (Int): Le nombre de joueurs pouvant rejoindre la table
- status (String) : Le statut actuel de la table, pouvant varier entre WAITING (elle n'est pas complète et la partie n'est pas lancée), IN_GAME (la table est complète et la

partie est lancée), PAUSED (la partie a été lancé mais un joueur a été retiré de la table) et FINISHED (la partie s'est terminée)
- creationDate (DateTime) : La date de création de la table
- invitationTimer (DateTime) : La date de péremption d'une invitation de la table
- inactivityHours (Int) : Le nombre d'heures d'inactivité d'un joueur avant son exclusion
- inactivityTimer (DateTime) : La date et l'heure de l'exclusion du joueur actuel en cas d'inactivité. Elle se calcule avec hoursBeforeExclusion et la date et l'heure du dernier coup joué par le précédent joueur
- game (Game) : Le jeu dont la table va en lancer une partie. Cette relation sert notamment pour filtrer les tables selon le jeu
- nbInvitations (int) : Le nombres d'invitations actives liées à la table
- listUsers (Collection<User>) : La liste des utilisateurs qui sont actuellement inscrits à la table
- partyId (int) : L'id de la partie lancé par la table (généré depuis les contrôleurs de la partie Jeux de l'application)
- invitedContacts (Collection<User>) : La liste des joueurs qui ont été invités à la table.

Un board possède donc une relation ManyToOne avec l'entité Game (plusieurs tables peuvent lancer une seule partie d'un seul jeu), ainsi que deux relations ManyToMany avec User (la première indique qu'un joueur peut rejoindre plusieurs tables, et que plusieurs joueurs peuvent rejoindre la même table ; la deuxième indique qu'un joueur peut être inviter à plusieurs tables, et que plusieurs joueurs peuvent être invités à la même table)

2. Game

Cette entité représente le concept de jeu sur la plateforme. Elle va contenir les informations descriptives concernant l'un des jeux disponibles sur la plateforme, ainsi que certaines variables utilisées par les différents composants de l'application.

Attributs:
- id(Int) : L'identifiant du jeu (clé principal)
- name (String) : Le nom du jeu
- descrRule (String) : Une chaîne servant de phrase descriptive courte du jeu
- imgURL (String) : L'URL de l'image miniature du jeu
- label (String) : Un identifiant utilisé pour retrouver les contrôleurs gérant la vue du jeu associé au label. C'est un attribut important puisqu'il assure la communication entre les contrôleurs de la partie Jeux et de la partie Plateforme
- isActive (bool) : Indique si le jeu est disponible à jouer ou pas
- minPlayers (int) : Le nombre de joueurs minimum requis pour lancer une partie de ce jeu (utilisé dans le formulaire de création de table)
- maxPlayers (int) : Le nombre maximum de joueurs pouvant rejoindre une partie de ce jeu (utilisé dans le formulaire de création de table)
- boards (Collection<Board>) : La liste des tables de jeu qui ont ou vont lancer une partie de ce jeu

Un Game possède donc une relation OneToMany avec l'entité Game (Une partie d'un jeu peut être lancée par plusieurs tables).

3. User

Cette entité représente un joueur de la plateforme. Un utilisateur du site voulant se créer une table va passer par un formulaire, qui va à son terme créer une instance d'entité User représentant le nouveau compte joueur de l'utilisateur

Attributs:

- id (int) : L'identifiant du joueur (clé principal)
- email (string) : L'email renseigné par le joueur durant la création de son compte
- roles (Array) : Ce tableau va contenir l'ensemble des rôles que peut avoir un User sur la plateforme. Un user peut être ROLE_USER (joueur), ROLE_MODERATOR (modérateur) ou bien ROLE_ADMIN (administrateur)
- password (string) : Le mot de passe renseigné par l'user lors de la création de son compte, hashé afin de garantir la protection de son mot de passe en cas de fuite
- isVerified (bool) : Indique si le joueur a vérifié son compte via l'email de validation envoyé depuis la plateforme à son adresse email
- favoriteGames (Collection<Game>) : La liste des jeux que le joueur a désigné comme favoris
- boards (Collection<Board>) : La liste des tables sur lesquelles le joueur s'est inscrit
- notifications (Collection<Notification>) : La liste des notifications qui ont été envoyées à ce joueur
- contacts (Collection<User>) : La liste des autres joueurs que le joueur a ajouté dans sa liste de contacts
- isBanned (bool) : Indique si le joueur est actuellement banni de la plateforme
- createdAt (DateTime) : La date de création du compte

Un User possède donc une relation OneToMany avec l'entité Notification (un joueur peut recevoir plusieurs notifications), et des relations ManyToMany avec les entités Board (plusieurs joueurs peuvent rejoindre les mêmes tables), Game (plusieurs joueurs peuvent ajouter en favoris les mêmes jeux) et User (plusieurs joueurs peuvent ajouter en contact les mêmes autres joueurs)

4. Notification

Cette entité représente une notification. Une notification est une alerte envoyée à un utilisateur pour l'informer d'une activité ou d'un événement spécifique qui s'est produit sur le site et qui est susceptible de l'intéresser (par exemple le démarrage d'une partie, une invitation à une table, etc…)

Attributs :

- id (int) : L'identifiant de la notification (clé principal)
- receiver (User) : Le joueur qui est récepteur de cette notification
- content (String) : Le contenu textuelle de la notification
- createdAt (DateTime) : La date de création de la notification
- isread (bool) : Indique si le joueur a cliqué sur le lien "Marqué comme lû" associé à la notification
- type (String) : Le type de la notification (exemple : "Message", "Invitation", "Info")
- title (String) : Le titre de la notification

Une notification possède une relation ManyToOne avec l'entité User (plusieurs notifications peuvent être envoyées au même joueur).