Projet de Génie Logiciel INFO5

Chevrier-Pivot Yohan
Letourneur Yann

Analyse du projet

Acteurs :

Joueur

Le joueur est un utilisateur de l'application qui participe aux parties de jeu de rôle en incarnant un ou plusieurs personnages. Il peut créer des personnages puis les proposer à un maître de jeu pour validation. Le joueur peut consulter les informations publiques des personnages, ainsi que les informations privées de ses propres personnages. Il est responsable de la rédaction de la biographie de ses personnages, peut créer des épisodes, demander leur validation et en révéler de manière irréversible certains paragraphes secrets. Le joueur peut également consulter les parties et aventures passées, et transférer la propriété d'un personnage ou demander un changement de maître de jeu sous réserve des conditions définies.

Maître de jeu

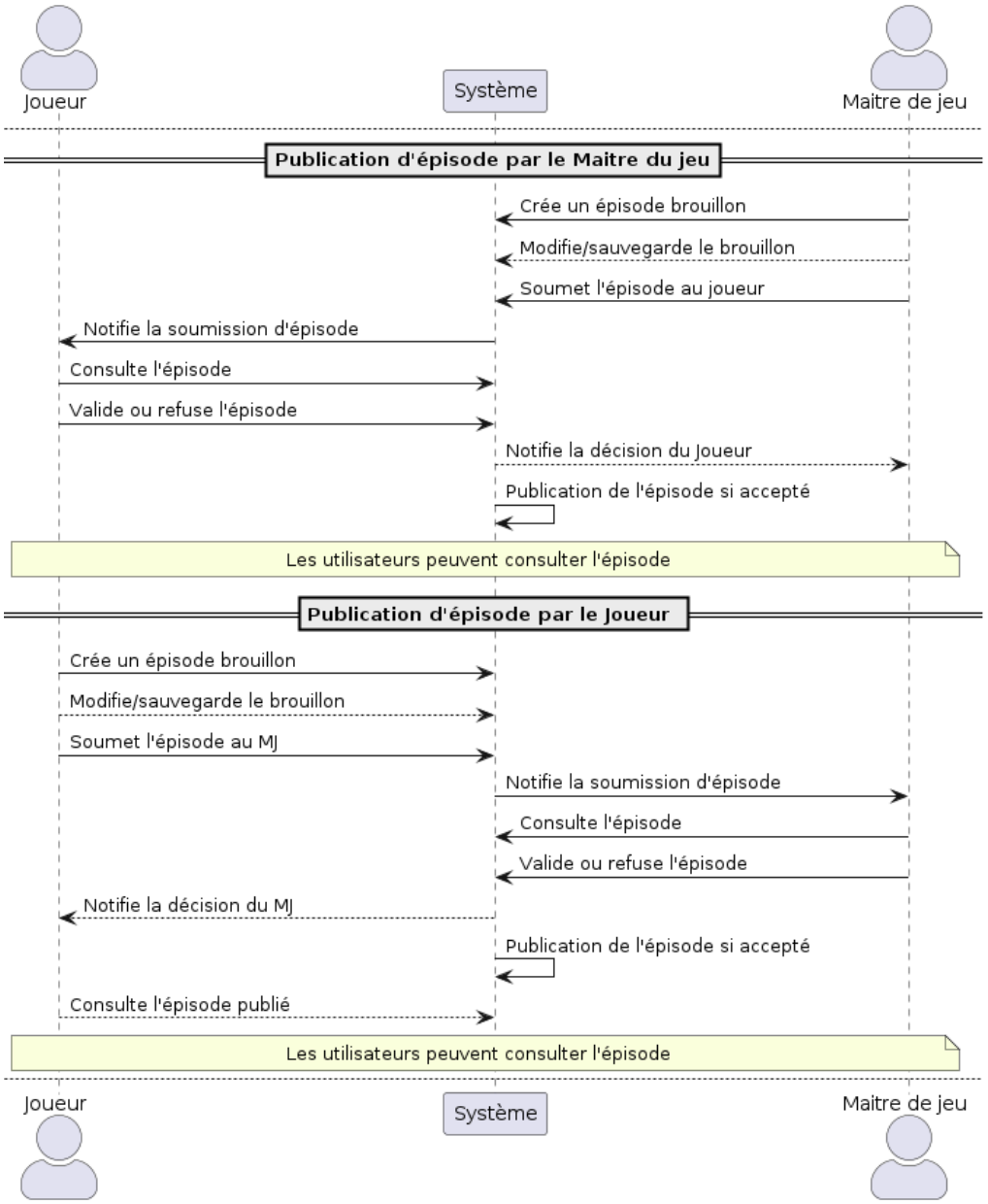
Le maître de jeu est un joueur disposant de responsabilités supplémentaires lors de l'organisation et de l'animation de parties. Il accepte ou rejette les personnages qui lui sont proposés et peut consulter leur biographie privée avant acceptation. Le maître de jeu participe à la rédaction et à la validation des épisodes de biographie, en collaboration avec les joueurs, et peut créer ou modifier des épisodes avant leur validation définitive. Il propose et organise des parties, puis gère les personnages participants. À l'issue d'une partie, le maître de jeu saisit le résumé des événements et clôture la partie, la transformant en aventure définitive. Il peut également accepter ou refuser les demandes de changement de maître de jeu pour un personnage.

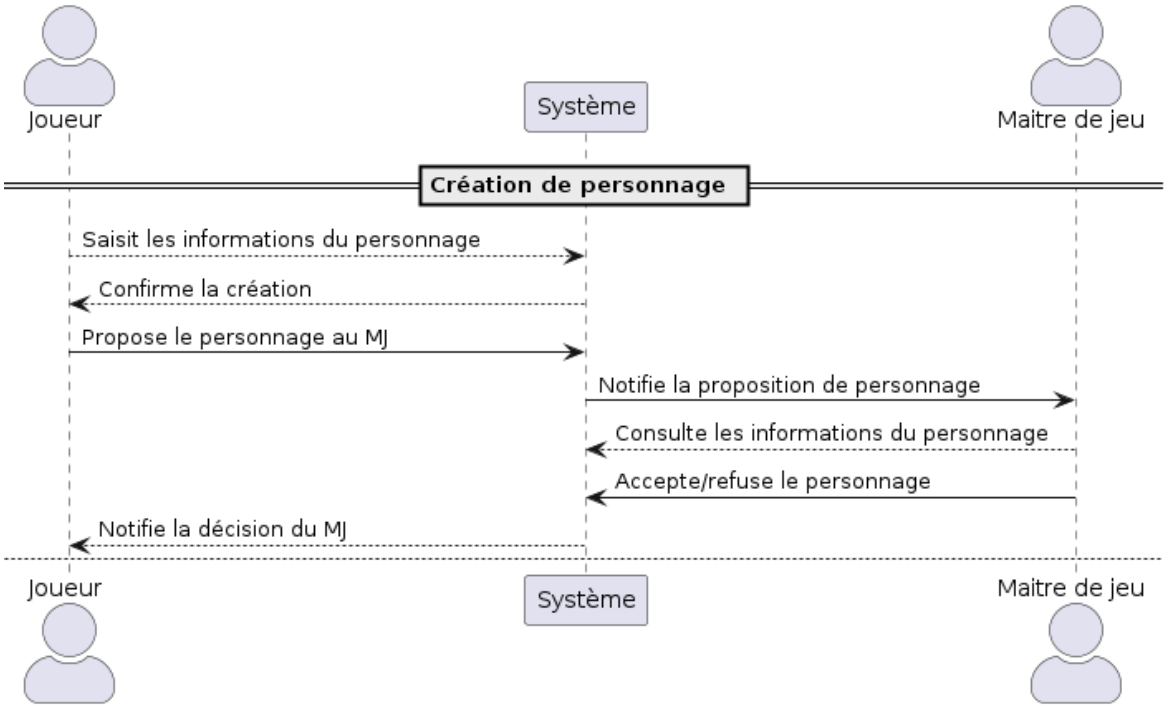
Système

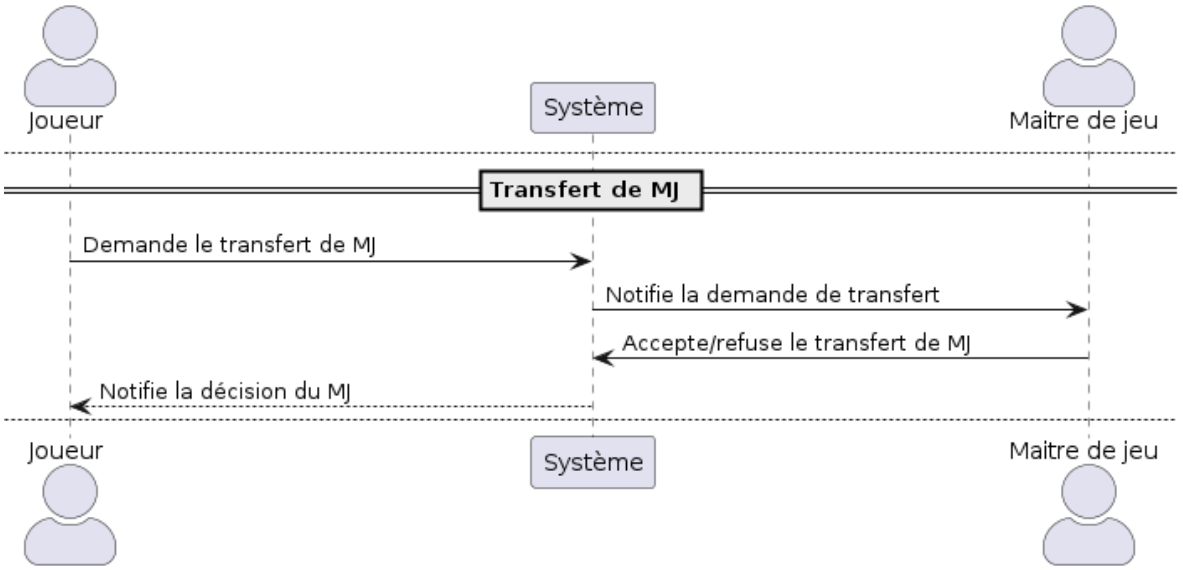
Le système représente l'application elle-même et assure la gestion des données et des règles. Il permet l'authentification des utilisateurs, la création et la gestion des personnages, des biographies, des épisodes et des parties. Le système vérifie les conditions de validité des actions, notamment lors des validations d'épisodes, des transferts de personnages ou des changements de maître de jeu, et garantit le respect des règles d'accès aux informations publiques et privées. Il assure également la visibilité des parties proposées, la transformation des parties clôturées en aventures consultables, ainsi que la cohérence globale des informations affichées aux utilisateurs.

Diagramme de cas d'utilisation :

Diagramme de séquence système:




Joueur
Système
Maitre de jeu
Création de personnage
Saisit les informations du personnage
Confirme la création
Propose le personnage au MJ
Notifie la proposition de personnage
Consulte les informations du personnage
Accepte/refuse le personnage
Notifie la décision du MJ
Joueur
Système
Maitre de jeu


Joueur
Système
Maitre de jeu
Transfert de MJ
Demande le transfert de MJ
Notifie la demande de transfert
Accepte/refuse le transfert de MJ
Notifie la décision du MJ
Joueur
Système
Maitre de jeu

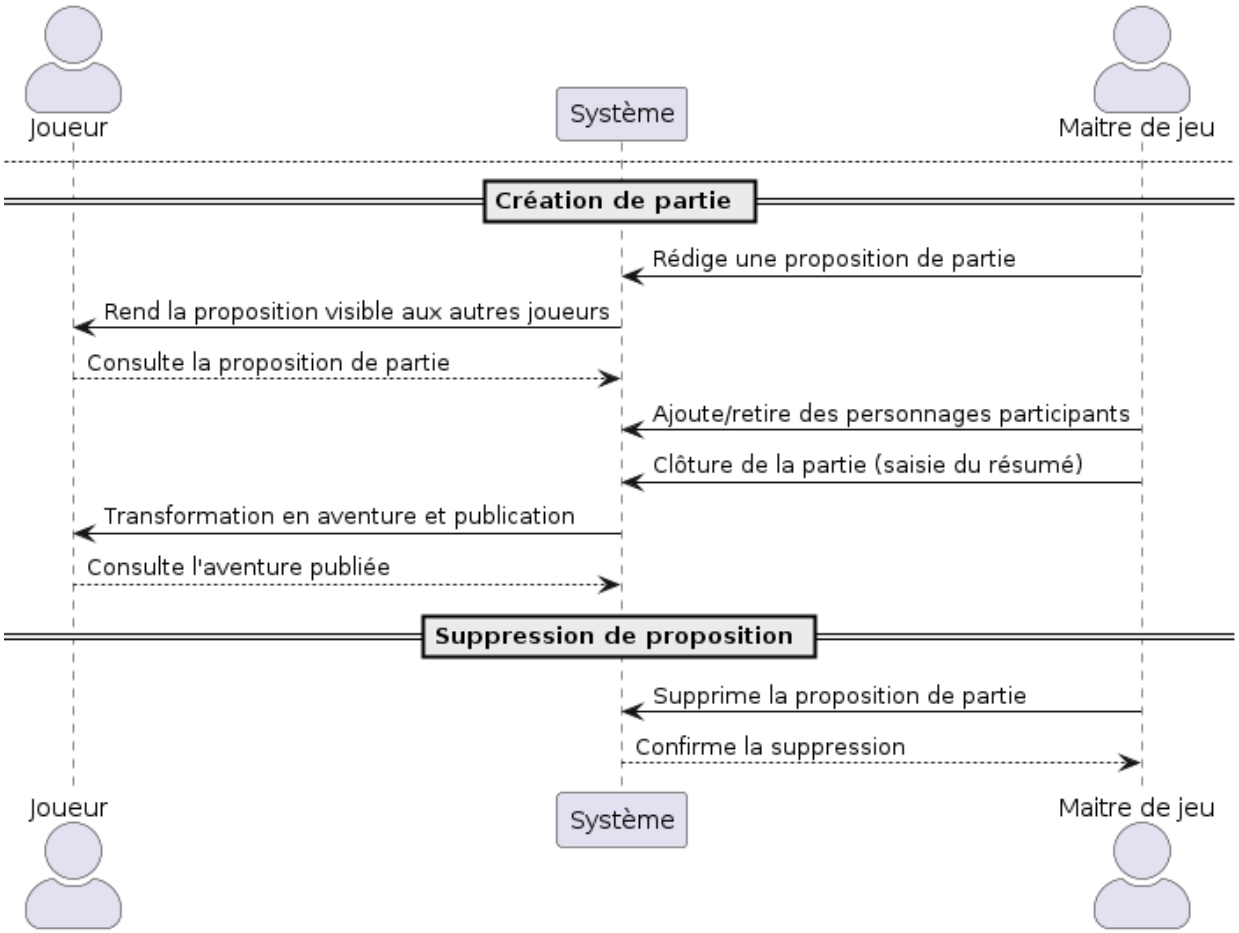
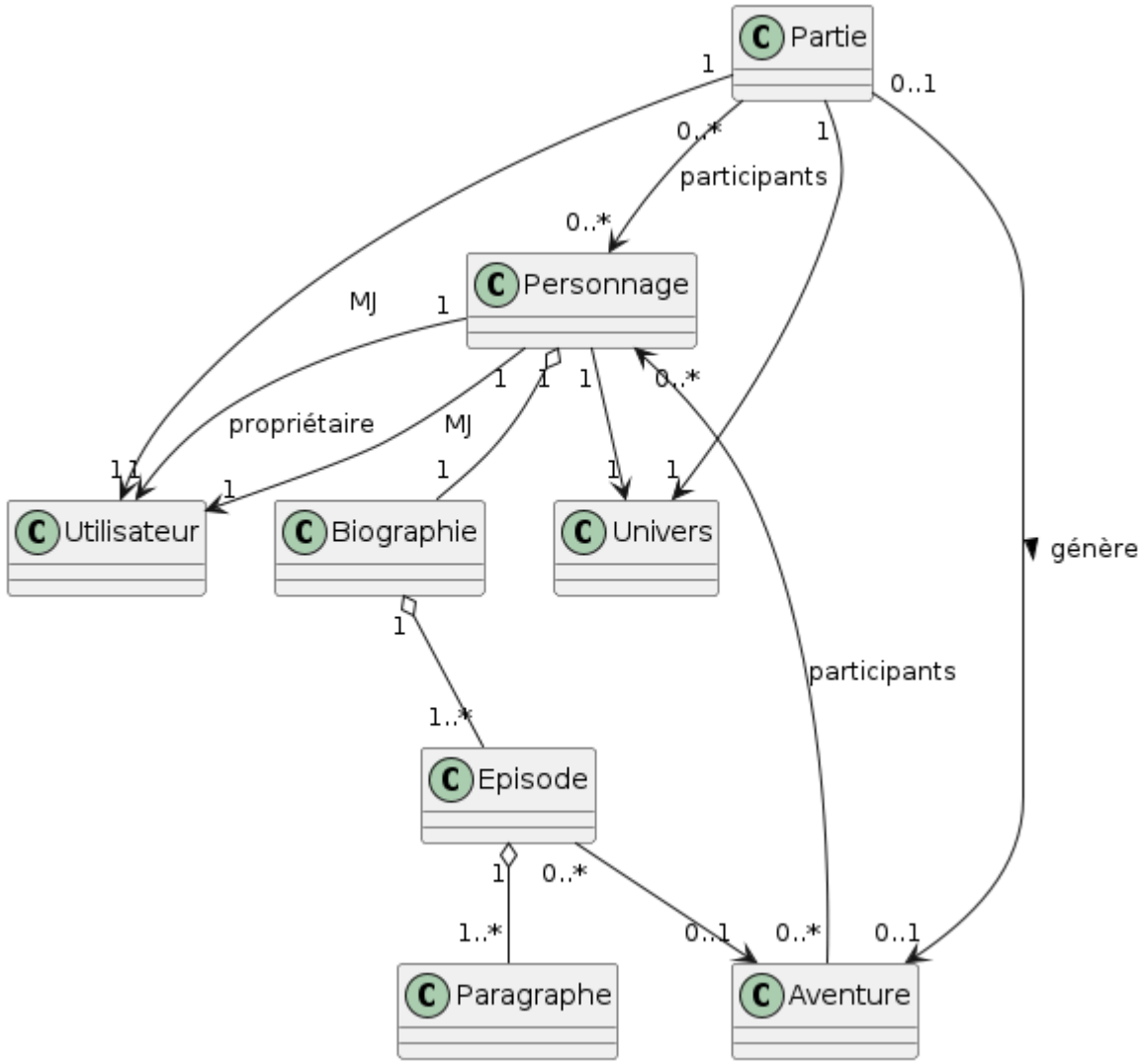

Joueur
Système
Maitre de jeu
Création de partie
Rédige une proposition de partie
Rend la proposition visible aux autres joueurs
Consulte la proposition de partie
Ajoute/retire des personnages participants
Clôture de la partie (saisie du résumé)
Transformation en aventure et publication
Consulte l'aventure publiée
Suppression de proposition
Supprime la proposition de partie
Confirme la suppression
Joueur
Système
Maitre de jeu

Diagramme de classe d'analyse



Conception

Diagramme d'architecture (MVC)

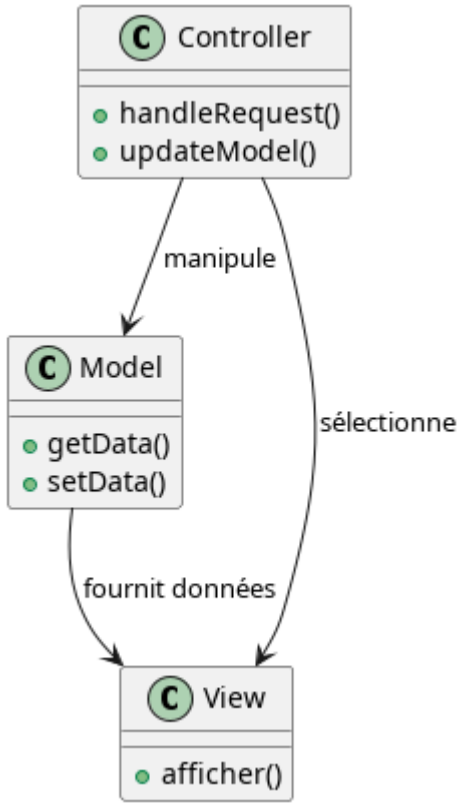
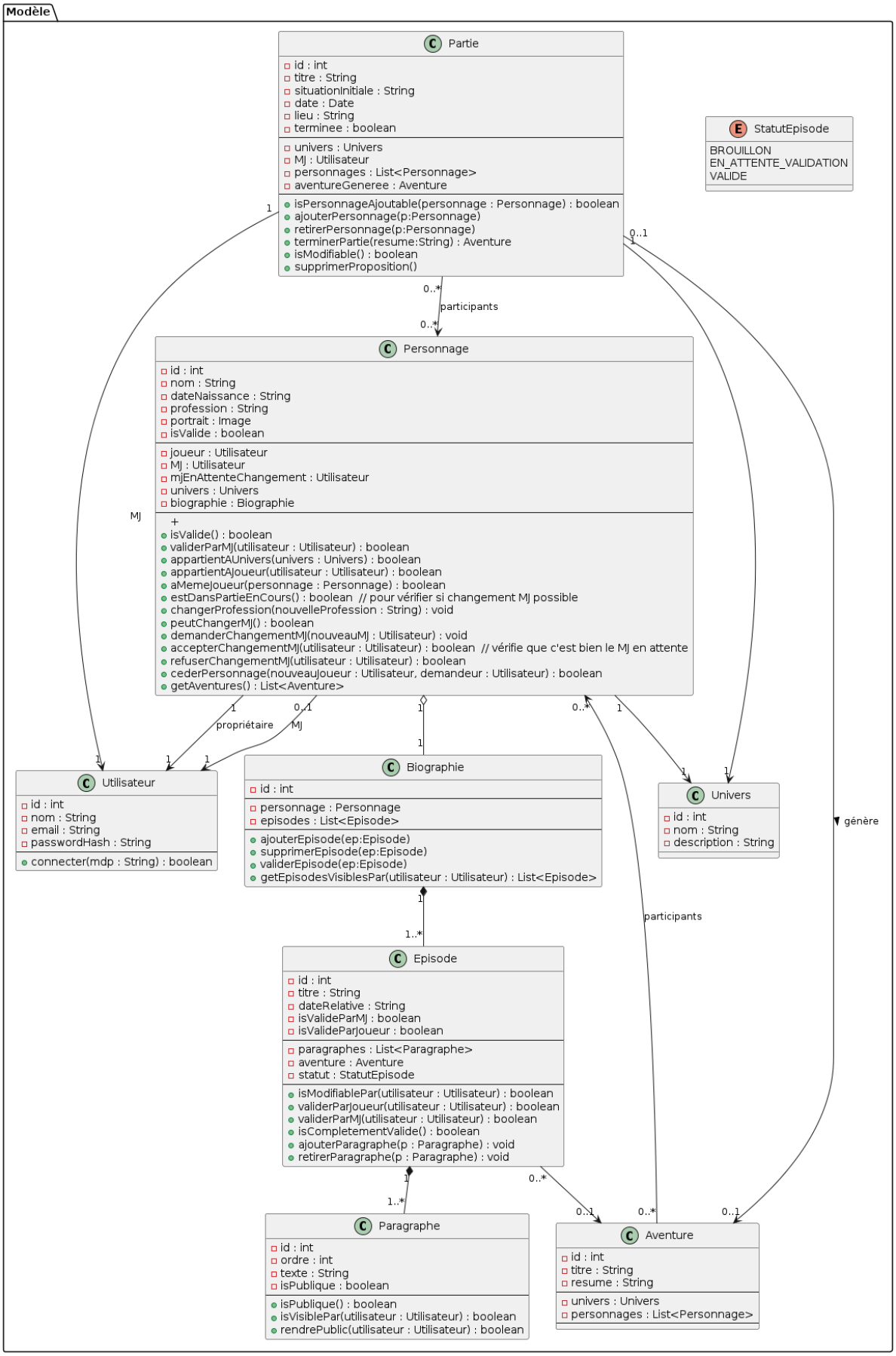
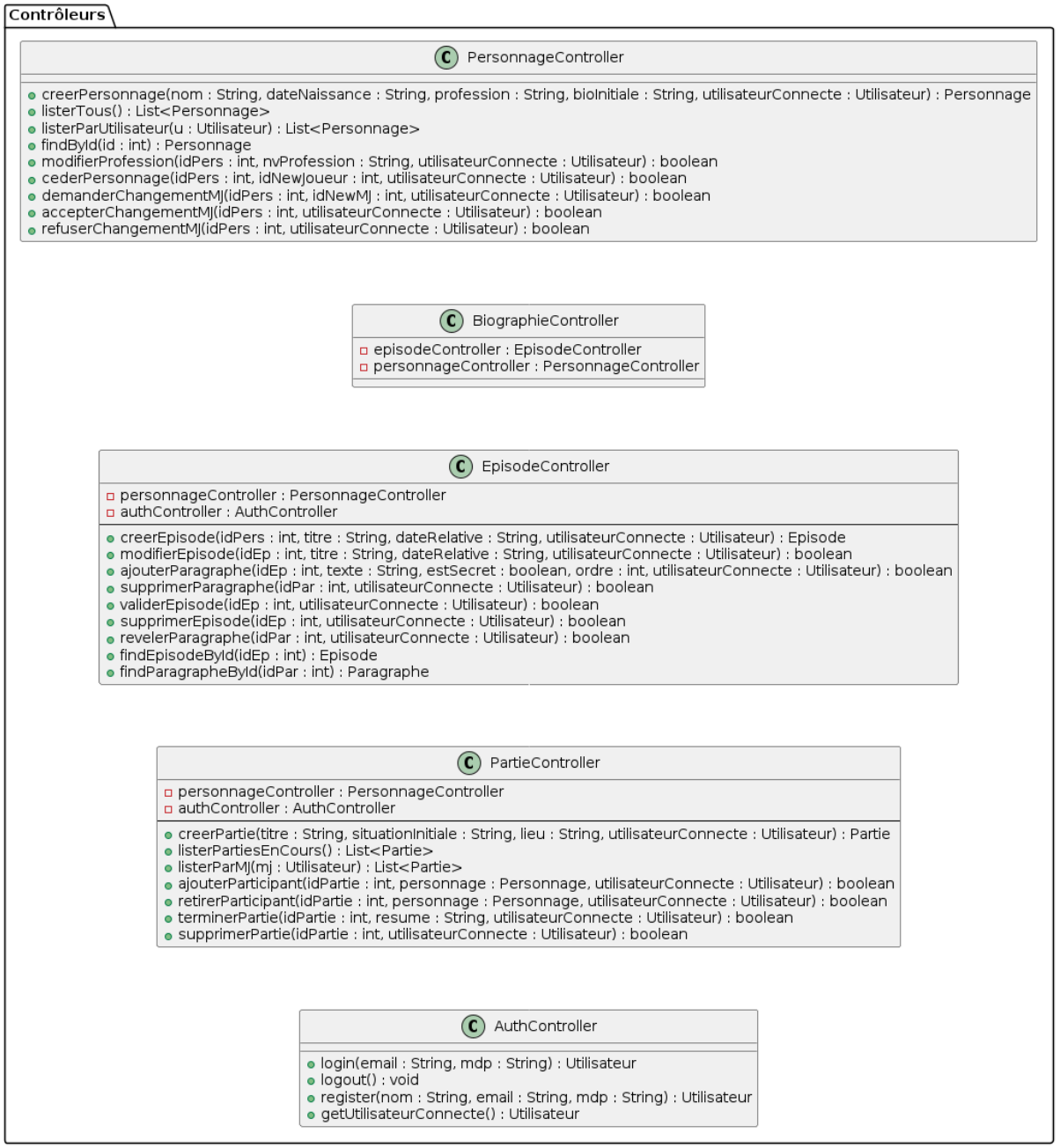


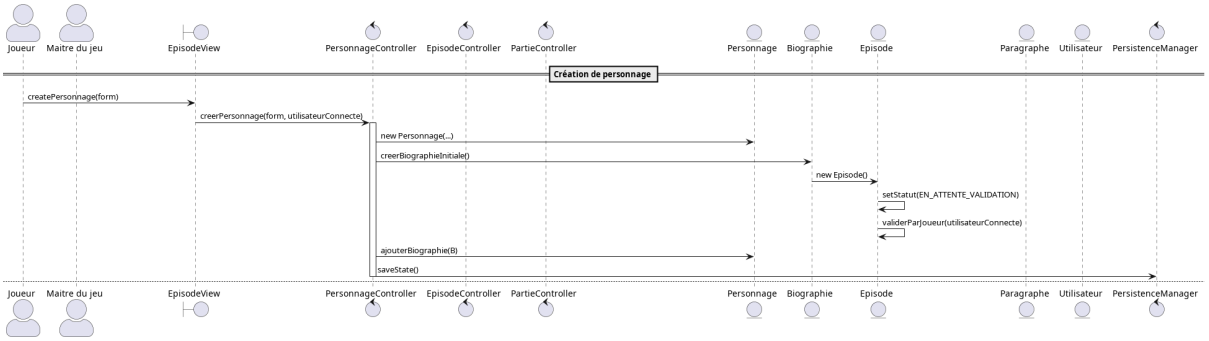
Diagramme de classes logiciel

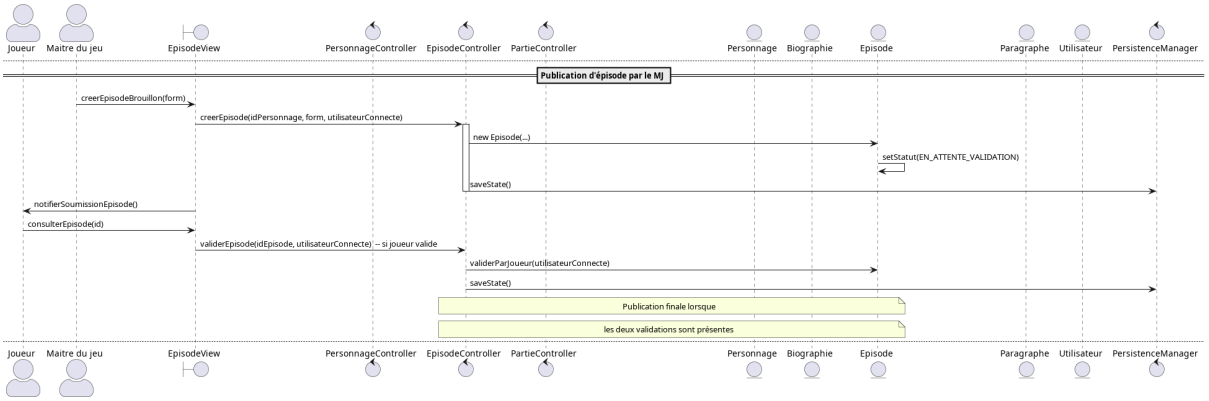


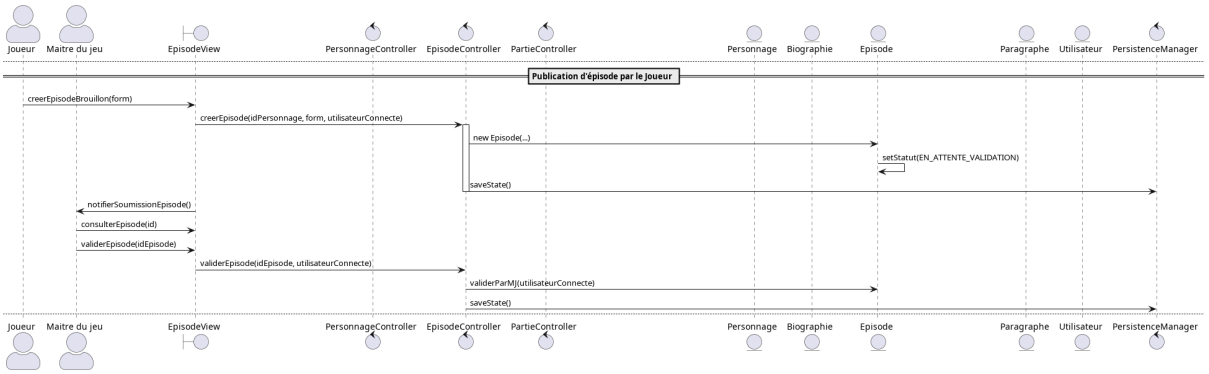

Contrôleurs
PersonnageController
creerPersonnage(nom : String, dateNaissance : String, profession : String, bioInitiale : String, utilisateurConnecte : Utilisateur) : Personnage
listerTous() : List<Personnage>
listerParUtilisateur(u : Utilisateur) : List<Personnage>
findById(id : int) : Personnage
modifierProfession(idPers : int, nvProfession : String, utilisateurConnecte : Utilisateur) : boolean
cederPersonnage(idPers : int, idNewJoueur : int, utilisateurConnecte : Utilisateur) : boolean
demanderChangementMJ(idPers : int, idNewMJ : int, utilisateurConnecte : Utilisateur) : boolean
accepterChangementMJ(idPers : int, utilisateurConnecte : Utilisateur) : boolean
refuserChangementMJ(idPers : int, utilisateurConnecte : Utilisateur) : boolean
BiographieController
episodeController : EpisodeController
personnageController : PersonnageController
EpisodeController
personnageController : PersonnageController
authController : AuthController
creerEpisode(idPers : int, titre : String, dateRelative : String, utilisateurConnecte : Utilisateur) : Episode
modifierEpisode(idEp : int, titre : String, dateRelative : String, utilisateurConnecte : Utilisateur) : boolean
ajouterParagraphe(idEp : int, texte : String, estSecret : boolean, ordre : int, utilisateurConnecte : Utilisateur) : boolean
supprimerParagraphe(idPar : int, utilisateurConnecte : Utilisateur) : boolean
validerEpisode(idEp : int, utilisateurConnecte : Utilisateur) : boolean
supprimerEpisode(idEp : int, utilisateurConnecte : Utilisateur) : boolean
revelerParagraphe(idPar : int, utilisateurConnecte : Utilisateur) : boolean
findEpisodeById(idEp : int) : Episode
findParagrapheById(idPar : int) : Paragraphe
PartieController
personnageController : PersonnageController
authController : AuthController
creerPartie(titre : String, situationInitiale : String, lieu : String, utilisateurConnecte : Utilisateur) : Partie
listerPartiesEnCours() : List<Partie>
listerParMJ(mj : Utilisateur) : List<Partie>
ajouterParticipant(idPartie : int, personnage : Personnage, utilisateurConnecte : Utilisateur) : boolean
retirerParticipant(idPartie : int, personnage : Personnage, utilisateurConnecte : Utilisateur) : boolean
terminerPartie(idPartie : int, resume : String, utilisateurConnecte : Utilisateur) : boolean
supprimerPartie(idPartie : int, utilisateurConnecte : Utilisateur) : boolean
AuthController
login(email : String, mdp : String) : Utilisateur
logout() : void
register(nom : String, email : String, mdp : String) : Utilisateur
getUtilisateurConnecte() : Utilisateur

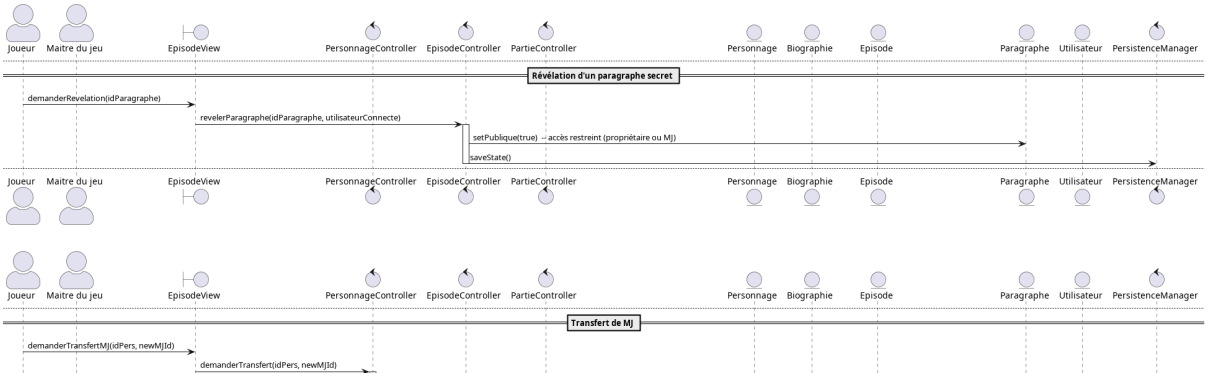

Vues
VuePersonnage
-
afficherPersonnage(idPers, utilisateurConnecte : Utilisateur) : void
listerPersonnages(utilisateurConnecte : Utilisateur) : void
VueBiographie
afficherBiographie(idPers, utilisateurConnecte : Utilisateur) : void
VuePartie
afficherPartie(idPartie, utilisateurConnecte : Utilisateur) : void
listerParties(utilisateurConnecte : Utilisateur) : void
VueEpisode
afficherEpisode(idEp, utilisateurConnecte : Utilisateur) : void
VueConnexion

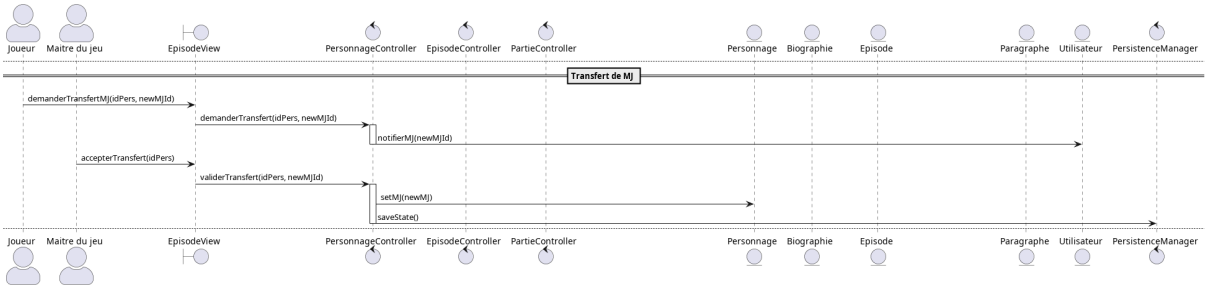
Diagrammes de séquence

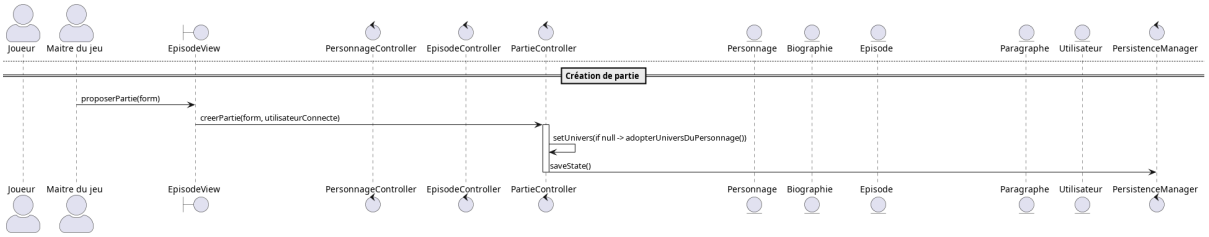












Manuel Utilisateur

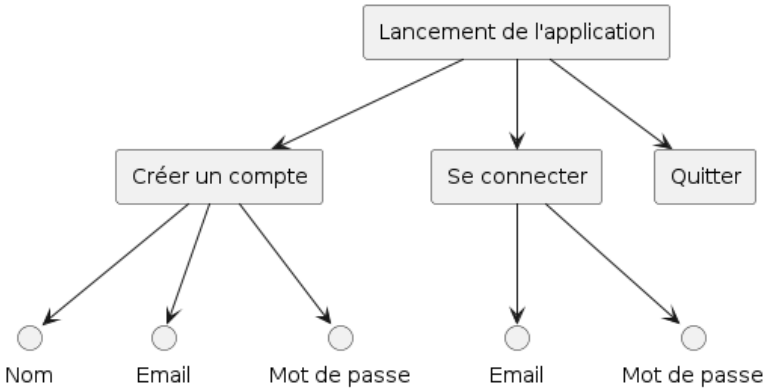
Pour lancer l'application, se placer dans le dossier *projet* puis exécuter la commande *mvn -f pom.xml exec:java -Dexec.mainClass="polytech.info5.gl.projet.App"*

Les données relatives à l'application sont sauvegardées dans le fichier *state.json* du dossier *data*, données automatiquement chargées et sauvegardées à chaque utilisation, permettant ainsi la persistance des informations.

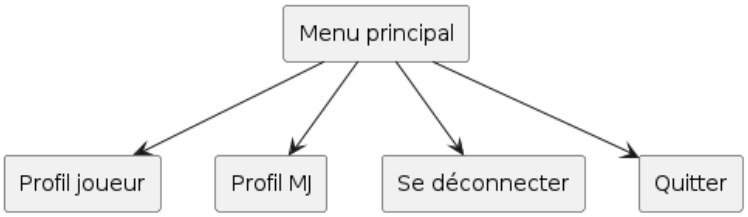
Ensuite, notre application est entièrement guidée et navigable depuis la console à l'aide du numéro associé à chaque commande (affiché à côté de l'action associée).

Organisation des actions dans la console

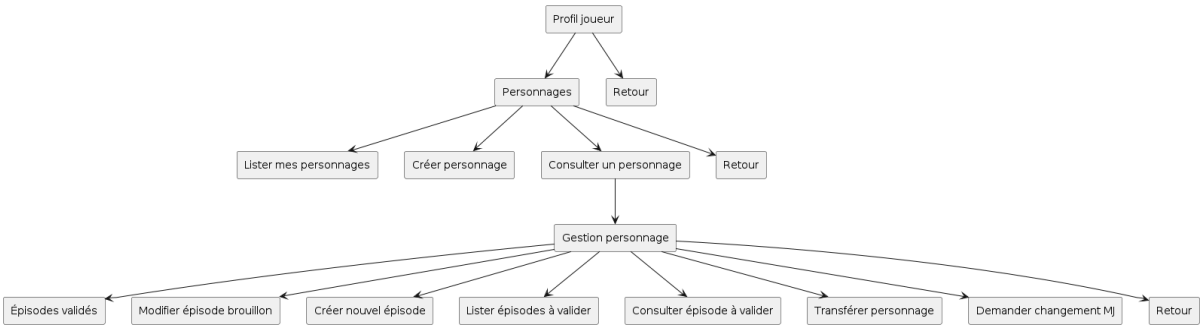
Au lancement de l'application (si l'on est pas déjà connecté à un compte) la console nous donne ces possibilités d'actions :



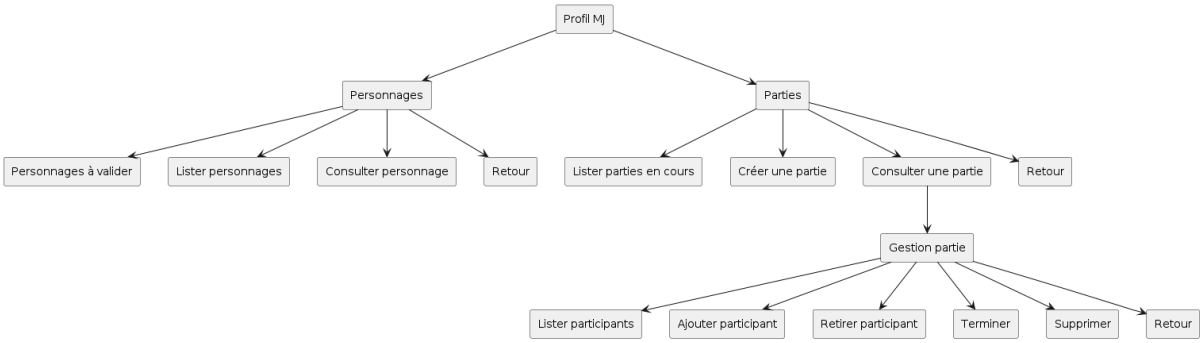
Ensuite, une fois identifié (via création de compte ou connexion), nous nous retrouvons dans le menu principal :



Voici les options disponibles lorsque l'utilisateur souhaite accéder à ses fonctionnalités en tant que joueur :



Enfin, les fonctionnalités MJ :



Bilan sur les outils de modélisation utilisés (PlantUML)

Afin de modéliser nos diagrammes UML, nous avons choisi d'utiliser l'outil PlantUML.

Pourquoi PlantUML ?

Familiarité : PlantUML nous avait déjà été recommandé lors des cours de bases de données de l'année dernière, donc nous disposions déjà d'une certaine expérience avec cet outil. De plus, Yann a également utilisé PlantUML durant son stage.
Documentation : PlantUML propose une documentation complète, claire et facile à prendre en main, ce qui facilite l'apprentissage et l'utilisation de ses fonctionnalités.
Popularité : PlantUML est largement utilisé, ce qui permet de trouver facilement des exemples en ligne ainsi que des réponses à des problématiques similaires rencontrées par la communauté.
Intégration : VS Code, notre éditeur de code principal, dispose d'une extension PlantUML permettant de créer et de visualiser les diagrammes UML directement dans l'éditeur, améliorant ainsi le confort et la rapidité de travail.
Format textuel : PlantUML utilise un format textuel pour décrire les diagrammes, ce qui facilite la gestion des versions avec des systèmes de contrôle de version comme Git. Cela permet également de modifier rapidement les diagrammes sans avoir à manipuler une interface graphique.

Inconvénients de PlantUML

Disposition des éléments : La disposition automatique des éléments générée par PlantUML n'est pas toujours optimale (par exemple : flèches qui se croisent alors qu'il existe une disposition des éléments pour laquelle il n'y a aucun croisement). Les possibilités de personnalisation sont limitées et nécessitent parfois l'utilisation de nombreux ajustements pour obtenir le rendu souhaité. Il n'existe pas de moyen simple permettant de positionner manuellement ou précisément les éléments du diagramme.
Dimensions des diagrammes complexes : Pour les diagrammes comportant un grand nombre d'éléments, PlantUML tend à produire des diagrammes très larges ou très longs, ce qui complique leur lisibilité ainsi que leur mise en forme dans le rapport en A4.
Performance : La génération de diagrammes très complexes peut être relativement lente, surtout couplée à l'utilisation de Live Share (qui nous permettait de collaborer en temps réel sur les diagrammes).
Tolérance aux erreurs : PlantUML est strict concernant sa syntaxe, la moindre erreur empêche la génération complète du diagramme. Cela rend difficile la visualisation progressive lors de l'écriture de diagrammes complexes.
Stylisation limitée : Bien que PlantUML offre des options de personnalisation, elles restent limitées et très fonctionnelles.
Description textuelle : La nature textuelle de PlantUML oblige à avoir un visualisateur / générateur de rendu pour pouvoir avoir une idée du rendu final et rend difficile la bonne compréhension du diagramme avec seulement le code source.

Adaptations et solutions

Modularisation des diagrammes : Diviser les diagrammes complexes en plusieurs sous-diagrammes (un par page grâce à au mot-clé newpage) permet d'améliorer la lisibilité et de réduire les problèmes de mise en page.
Validation incrémentale : le "live preview" de l'extension VS Code permet de visualiser les modifications en temps réel, facilitant la détection des erreurs de syntaxe immédiatement.

Pour conclure, PlantUML s'est révélé être un outil efficace et adapté à nos besoins de modélisation UML. Grâce à sa simplicité d'utilisation et à son intégration avec notre environnement de développement, nous n'avons pas rencontré de problèmes majeurs ni de limitations insurmontables lors de son utilisation, et nous avons pu mener le projet à bien sans avoir besoin d'utiliser d'autres outils de modélisation en complément.