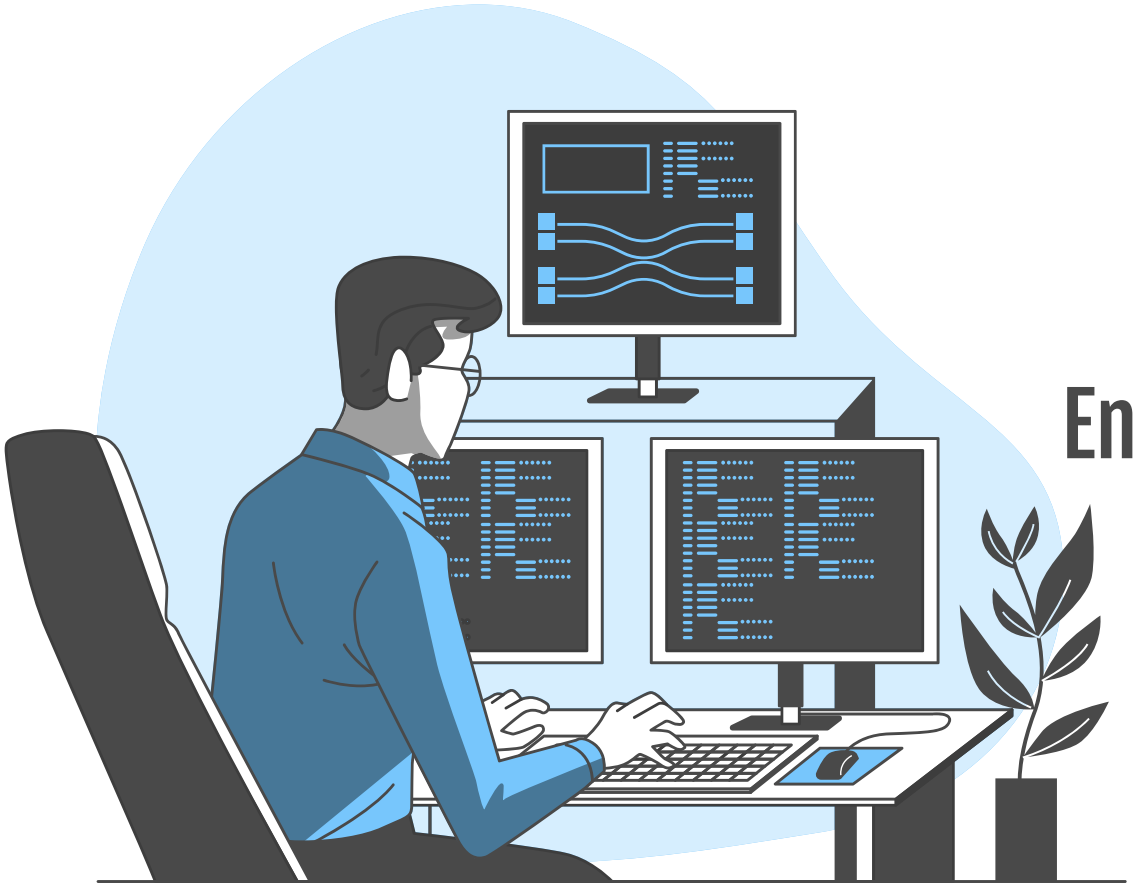

Environnement DOTNET et C# (ALTN71)

Thierry TAGNE
EFFREI – (2025-2026)

BlazorGameQuest

Jeu d’aventure

Projet en continu – Développement Agile

Objectifs pédagogiques

- *Comprendre et utiliser le framework .NET avec C#.*
- *Appliquer les principes du développement piloté par les tests (TDD).*
- *Manipuler les événements, LINQ, tâches asynchrones et WebAPI.*
- *Implémenter une authentification OAuth avec Keycloak.*
- *Utiliser Entity Framework Core avec **EFCoreInMemory** ou alors une base de données de son choix mais qui devra tourner obligatoirement sous Docker.*
- *Créer une interface utilisateur moderne avec **Blazor** et **JavaScript**.*

Principe

Product Owner : M. TAGNE

Equipes de développement

- ❑ Binômes (à former et à envoyer au plus tard le 26 Septembre 2025).
- ❑ Même Projet pour tout les binômes.

Unique Projet

- ❑ Etapes à rendre régulièrement (après certains cours).
- ❑ Nouvelles fonctionnalités à chaque cours (ou tous les deux cours).
- ❑ Utilisation de Github.

Notation

Notation => Versions

Existence d'une branche de « *Prod* »

- ❑ Correspond à la dernière version corrigée

Rendu d'une version

- ❑ Dès que possible : « ***Merge Request*** » vers ***Prod***
- ❑ Deadline : ***Validée par l'enseignant***
- ❑ Centralise les remarques et les discussions
- ❑ Contiendra les notes (temporaires et finales)

Notation => Evaluation

Note intermédiaire divisée en 2 moitiées

- ❑ **Livraison** – par rendu (notes fixes)
- ❑ **Complétude** – final (notes préliminaires)

Note finale = Note Intermédiaire x %Qualité

- ❑ Va de 50% à 100%
- ❑ Lisibilité, Maintenabilité, etc.
- ❑ Estimée en cours de projet, figée en fin

Livraison

- **2 points max** par version.
- **% de tests** automatisées et de couverture de code.
- Accordée (**totalement** si **réalisée** à **la date du rendu prévue et fixée par l'enseignant et respect des attendus de la version**).
- **Acceptée après délai** mais **perte d'1 démi point sur la note** en retard. Et Idem si non respect du processus et consignes de livraison.

Complétude

- ❑ **10 points** par projet.
- ❑ **% de tests** automatisées et de couverture de code. Doit inclure les tests des versions.
- ❑ Accordée (**totalement** si **réalisée** à **la date du rendu prévue et fixée par l'enseignant et respect des attendus de la version**).
- ❑ **Acceptée après délai** mais **perte d'1 à 5 points sur la note** si retard excessif. Et Idem si non respect du processus et des consignes de livraison.).

Qualité

- ❑ **Evaluation humaine**
- ✓ Nommage limpide et parlant
- ✓ Fonctions courtes
- ✓ Algorithmes simples
- ✓ Tests unitaires (couverture de code incluse)
- ✓ Qualité du design (pas discriminatoire, Mais doit être facile d'utilisation)

- ❑ **Evaluation en continue** (par version mais sans garantie).

- ❑ Accordée à la date du rendu final

Scénarii et Fonctionalités attendues

Scénario

- Les ***joueurs*** incarnent des aventuriers explorant des donjons générés ***aléatoirement***.
- Ils doivent faire des choix dans chaque salle (combattre, fouiller, fuir...) pour gagner des points.
- Les ***administrateurs*** peuvent gérer les joueurs (Création, Désactivation, Export de la liste des joueurs, etc.).

Scénario => déroulement d'une partie

- ❑ Le joueur clique sur « ***Nouvelle aventure*** »
- ❑ Un donjon est généré (suite de salles aléatoires)
  - Chaque salle a une difficulté particulière. On doit donc générer un nombre de salles aléatoires (maximum 5).
  - A chaque salle, le joueur fait un choix (combattre, fouiller, fuir…)
  - Selon le choix, il gagne ou perd des points, peut trouver des objets ou subir des pièges
  - La partie s'arrête après ***X*** salles ou si le joueur « ***meurt*** » (lorsqu'il a Zéro point ou alors lorsqu'il a un total de points négatif, ou alors si son choix plus haut l'entraine à la mort)

- ❑ Le score final est enregistré et visible dans le classement.

Scénario => Exemple de cycle du jeu

- ***Accueil*** → « ***Nouvelle aventure*** »
- ***Salle 1***: « *Un gobelin apparaît. Que faites-vous ?* »
  - [Combattre] → Test de réussite, gain/perte de points
  - [Fuir] → Moins de points gagné, mais sécurité
  - [Fouiller] → Chance de trouver un objet précieux ou de tomber dans un piège
- ***Salle 2*** : « *Un coffre mystérieux* »
  - [Ouvrir] → Trésor ou piège
  - [Ignorer] → Passe à la salle suivante
- ...
- **Fin de partie** : *Le **joueur parcours toute les salles*** ou ***meurs*** ou ***arrête le jeu***. Score affiché, sauvegardé, classement des joueurs mis à jour. Historique à préserver.

Scénario => Les salles

- ***Les salles*** *doivent être toutes différentes.*

- *Donc si nous avons* ***5 salles*** *sur lesquelles nous feront un choix aléatoire, nous devons avoir des salles différentes avec des enjeux différents pour rendre le jeu plus agréable.*

Les rôles

Joueur

- Possède un compte dans **Keycloak.**
- Lance des parties et fait des choix dans les salles.
- Gagne des points selon ses actions.
- Consulte son historique et le classement général.
- Peut enregistrer un ***état de jeu*** et reprendre plus tard.

Administrateur

- Possède un compte dans **Keycloak.**
- Accède à un tableau de bord.
- Gère les joueurs.(Création, export)
- Peut désactiver ou réinitialiser un joueur (réinitialiser son statut de joueur et effacer toute son historique).

Fonctionnalités attendues => Front-End

- ❑ Page d'accueil, règles du jeu, connexion. (Pas de page d'inscription, ll y aura un lien entre le joueur créé et son identifiant ***Keycloak*** à prévoir dans votre base de données).
- ❑ Interface de jeu interactive (choix, résultats, progression).
- ❑ Tableau de scores et historique personnel.
- ❑ Tableau de bord admin (rôle administrateur), Gestion des joueurs et historique général.

Fonctionnalités attendues => Services

- ***Endpoints*** sécurisés selon le rôle (joueur/administrateur)
- Gestion des parties, scores, joueurs, ...
- Authentification et Authorisation via **Keycloak** (OpenID Connect)
- Documentation via **Swagger**

Fonctionnalités attendues => Back-end(Traitement brute)

- ❑ Logique de génération aléatoire de donjons (suite de salles).
- ❑ Calcul des scores.
- ❑ Gestion des rôles et droits

Attendu de la version finale

- ❑ *Un frontend Blazor (même basique mais « user friendly »).*
- ❑ *Choix de **l'architecture** à utiliser et explication.*
- ❑ *Création et présentation de la structure globale de son projet.*
- ❑ *Plusieurs microservices : Authentification, ...*
- ❑ *Utilisation de **EFCore (avec EFCoreInMemory)** ou d'une **base de données sous Docker**.*
- ❑ *Une API Gateway pour centraliser les appels.*
- ❑ *Utilisation de Docker pour le déploiement.*
- ❑ *La gestion des rôles (administrateur et joueur)*
- ❑ *Un fichier **Readme.md** pour expliquer son projet.*

Lancement de l'application

- *Déployer dans docker et démarrer toutes les images.*
- *Configurer Keycloak.*
- *Lancer l'interface web :*
  Joueur : http://localhost:5000
  Admin : http://localhost:5000/admin

Configuration de Keycloack et Initialisation des données #1

- *Les joueurs / utilisateurs :*
  *- Nous aurons **2 utilisateurs** avec le rôle « **joueur** » à créer dans **Keycloak**. Ces utilisateurs dans **Keycloak** devront avoir pour login un id_unique « **user1** » pour le jouer 1 et « **user2** » pour le joueur 2 et pour mot de passe « **1234** ».*
  *- Nous devrons initialiser notre base de données avec **2 joueurs** en faisant le lien avec le compte utilisateur dans **Keycloak**.*

- *Nous aurons un seul administrateur avec pour rôle « **administrateur** » qui lui, sera configuré uniquement au niveau de **Keycloak** et qui aura pour login « **admin** » et mot de passe « **admin** ».*

- *La Gateway devra tourner uniquement sous le **port 5000** et le seul protocole accepté est **http**.*

Plannification des rendus

Action	Date
Présentation V1	24/09/2025
Présentation V2	06/10/2025
Rendu V1	13/10/2025 à 23h59
Présentation V3 et V4	15/10/2025
Rendu V2	27/10/2025 à 23H59
Présentation V5	06/11/2025
Rendu V3	10/11/2025 à 23h59
Rendu V4	24/11/2025 à 23h59
Rendu V5	07/12/2025 à 23h59

Version 1

La structure du projet et la définition des cas de tests

Contenu de la version 1

Initialisation du projet

- ❑ *Création de la structure globale de son projet (Solution, micro-services, etc.).*
- ❑ *Création d'un projet Client Blazor WebAssembly.*
- ❑ *Identifier l'ensemble (du moins l'essentiel) des pages pour son projet.*
- ❑ *Mise en place du routing et des composants de base (css, js, etc.)*
- ❑ *Page d'accueil, menu, navigation, etc.*
- ❑ *Création de l'interface « **Nouvelle aventure** » (sans logique métier).*
- ❑ *Création d'un composant Blazor affichant une salle de jeu statique*

Développement Orienté Tests

- ❑ *Créer un projet Tests*
- ❑ *Utilisation de xUnit ou MSTest.*
- ❑ *Définition des cas de tests pour les fonctionnalités du projet. (Ici il s'agit juste de dire ce que vous ferez, sous forme textuelle mais pas de réaliser forcement des test)*

Ajout d'un Readme.md (Au début du quel nous mettez les noms du binôme, au début et pas à la fin)

Quelques commandes utiles

 # 1. Créer un dossier racine

- ❑ *mkdir* BlazorGameQuest1234
- ❑ *cd* BlazorGameQuest1234

 # 2. Créer la solution

- ❑ *dotnet new sln -n* BlazorGameQuest1234

 # 3. Créer le projet Blazor WebAssembly (client)

- ❑ *dotnet new blazorwasm -n* BlazorGame.Client *--no-https*

 # 4. Créer les projets Web API

- ❑ *dotnet new webapi -n AuthenticationServices --no-https*

 # 5. Créer le projet Shared (modèles partagés)

- ❑ *dotnet new classlib -n SharedModels*

Ajouter un projet dans la solution

- ❑ *dotnet sln add << nom_du_projet >>*

Version 2

Modélisation et base de données

Contenu de la version 2

Modélisation et base de données

- ❑ Définir les principaux modèles et leurs propriétés (Joueurs, Administrateurs, Donjons, Salles, ... (Astuce : mettre ces modèles dans un projet à part partagé par les autres projets. Par exemple SharedModels.csproj)
- ❑ Mise en place de **EF Core** avec PostgreSQL ou Une autre base de données (Possibilité aussi de configurer ***EFCore InMemory***)
- ❑ Migrations initiales et configuration du DbContext (si utilisation de EFCore avec une base de donnée)
- ❑ Création des micro-services et les endpoints pour gérer vos modèles. Ces API sont pour le moment non sécurisés.
- ❑ *Ajout de Swagger dans les API*
- ❑ *Test via Swagger IHM*

Tests

- ❑ *Mettre à jour ses tests unitaires au besoin.*
- ❑ *Vérifier la couverture de code si des tests unitaires ont été écrits..*

Mettre à jour le Readme.md du projet.

Version 3

Déroulé d’une partie et logique métier

Contenu de la version 3

Déroulement d'une partie

- ❑ Génération aléatoire d'un ou plusieurs donjons (suite de salles)
- ❑ Interface de jeu interactive : choix dans chaque salle
- ❑ Calcul du score selon les actions
- ❑ Sauvegarde de la partie et du score

Tests

- ❑ *Enrichir les tests unitaires*
- ❑ *Vérifier la couverture de code (essayer d'atteindre au moins 80%)*

Mettre à jour le Readme.md du projet.

Version 4

Administration et Visualisation

Contenu de la version 4

Génération des interfaces de gestion

- ❑ Historique personnel et classement général
- ❑ Tableau de bord admin : liste des joueurs (avec action désactivation etc.), liste des scores, classement général, liste des parties , export des joueurs.
- ❑ *Visualiser dans Postman et/ou swagger la liste des salles, des évènements, la liste des joueurs etc....*
- ❑ *Concevoir les interfaces **blazor** pour cette partie « admin »*

Tests

- ❑ *Enrichir les tests.*
- ❑ *Vérifier la couverture de code*

Mettre à jour le Readme.md du projet.

Version 5

Sécurité et finalisation

Contenu de la version 5

Sécurisation de l'application et finalisation

- ❑ Intégration de **Keycloak** :
  - Authentification OpenID Connect
  - Attribution des rôles joueur et admin
  - Sécurisation des API
  - Créér une page de connexion dans le projet Blazor et faire que seul les utilisateurs dans **Keycloak** puisse se connecter
  - Faire que toutes les pages du projet web soit accessibles uniquement à un utilisateur authentifié
- ❑ Enrichir la documentation Swagger
- ❑ Déploiement sous Docker
- ❑ Configurer la Gateway comme seul point d'entrée d'appel vers les APIs.
- ❑ Prévoir un test pour vérifier le bon déroulement d'une partie

Tests

- ❑ *Enrichir les tests unitaires.*
- ❑ *Vérifier la couverture de code (Il faut atteindre au minimum 80% de couvertures de code)*

Mettre à jour le Readme.md du projet.

C’est quoi la suite ?

Rendu du projet le ***07 Décembre 2025*** *au plus tard* ***à 23h59***