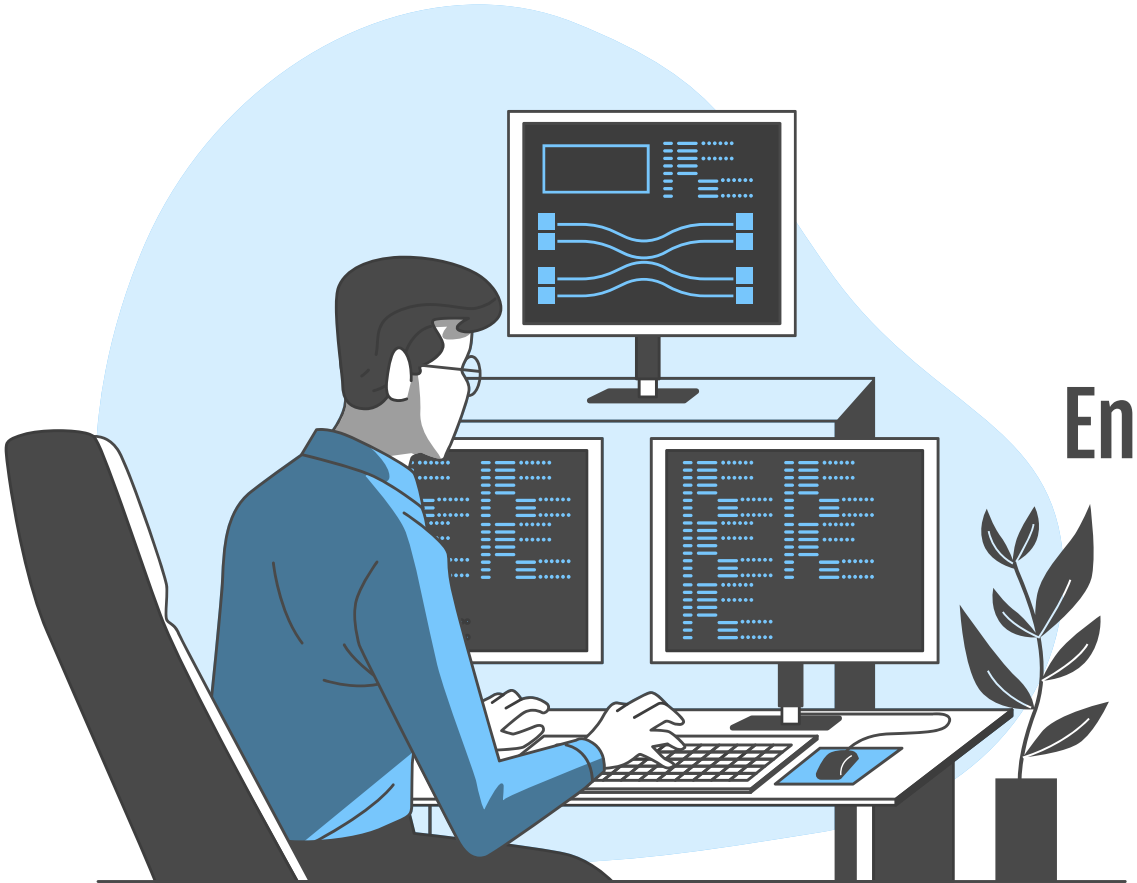

Environnement DOTNET et C# (ALTN71)

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**.*
- ❑ *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 Gitlab ou 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*** » de ***Master*** 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** 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**).
- ❑ **Acceptée après délai** mais **perte d'1 démi point sur la note** en retard.

Complétude

- ❑ **10 points** par projet.
- ❑ **% de tests** automatisées et de couverture de code. Doit inclure les tests des versions.
- ❑ Peut couvrir plus de cas à la discrétion des étudiants. (Pris en compte dans cette note).
- ❑ Accordée (**totalement** si **réalisée** à **la date du rendu prévue et fixée par l'enseignant**).
- ❑ **Acceptée après délai** mais **perte d'1 à 5 point si** retard excessif (à la discrétion de l'enseignant).

Qualité

- ❑ **Evaluation humaine**
- ✓ Nommage limpide et parlant
- ✓ Fonctions courtes
- ✓ Algorithmes simples
- ✓ Tests unitaires (couverture de code incluse)
- ✓ Qualité du design (pas discriminatoire)

- ❑ **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.

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*** »

- ❑ 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.

Les rôles

Joueur

- ❑ Possède un compte dans **Keycloak.**
- ❑ Lance des parties et fait des choix dans les donjons.
- ❑ 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, les scores, les parties.
- ❑ Crée/modifie des événements spéciaux.
- ❑ Peut désactiver ou réinitialiser un joueur.

Fonctionnalités attendues => Front-End

- ❑ Page d'accueil, règles du jeu, connexion.
- ❑ Interface de jeu interactive (choix, résultats, progression).
- ❑ Tableau de scores et historique personnel.
- ❑ Tableau de bord admin (rôle administrateur) et historique général

Fonctionnalités attendues => Services

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

Fonctionnalités attendues => Back-end(Traitement brute)

- ❑ Logique de génération aléatoire de donjons.
- ❑ 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)*
- ❑ *Une API Gateway pour centraliser les appels.*
- ❑ *Utilisation de Docker pour le déploiement.*
- ❑ *La gestion des rôles.*
- ❑ *Un fichier **Readme.md** pour expliquer son projet.*

Lancement de l'application

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

Version 1

La structure du projet et les tests

Contenue de la version

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.*

Ajout d'un Readme.md

Quelques commandes

 # 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 >>*

C’est quoi la suite ?

- ❑ *Finalisation de la définitions des modèles.*
- ❑ *Ajout d’Entity Framework Core*
- ❑ *Utilisation de EFCoreInMemory*
- ❑ *Mise à jour des tests unitaires*
- ❑ *Mise à jour du readme global*