

Présentation de mon projet

Chapitre 1 - Gestion des salles de réunion 1.1 Description métier de l'application Un client de Aurera passe un contrat pour gérer des salles de formation sur la plateforme eClassRoom. Cette plateforme est hébergée sur Azure. Le client va utiliser un site Web pour renseigner ses administrateurs, ses formateurs et ses stagiaires et créer des salles de formation. Pour créer une salle, le client et son représentant, identifié par un email, renseigne le formateur et les stagiaires, choisi son type de machine virtuelle (VM) Linux ou Windows, modèle de VM, OS avec détails (sku, offer, version) et le modèle de disk ISO préconfiguré. L'appuie sur le bouton Provisionner permet la création des machines virtuelles dans Azure et on met à disposition dans un portail l'accès aux fichiers unitaires RDP pour que chaque stagiaire puisse se connecter à sa VM.

1.2 Structure de projet initiale Dans le répertoire SRC, je veux construire une arborescence pour un projet Visual Studio 2019 de type WebAssembly Blazor avec modèle Entity Framework pour PostgreSQL et une couche de services et des End Points REST en Web API .NET C#. Tu peux m'aider ? [Create Workspace...]

1.3 Le modèle Entity Framework (EF) Voici une description simple du modèle de données. La société Aurera possède des clients pour la solution eClassRoom. Un client possède les attributs suivants : nom de société, adresse, code postal, ville, pays, email de l'administrateur, mobile optionnel. On fournit un mot de passe à l'administrateur d'un client. cela implique qu'il existe une table des utilisateurs. Chaque utilisateur possède un email, nom, prénom, mot de passe et un role (admin, formateur, stagiaire). Un client peut créer des salles de formations. Une salle de formation a les attributs suivants: un nom de formation, un formateur, des stagiaires, une date de début, une date de fin, les caractéristiques des machines virtuelles (type Windows ou Linux, type de VM Azure, sku, offer, version, disk ISO spécifique, nom marketing de la VM). Chaque stagiaire possède une machine virtuelle et via le site web, il peut accéder au fichier RDP de connexion. Les VM sont supervisées et leurs utilisations est tracées pour permettre de créer des factures mensuelles pour les clients. Tu peux me générer le modèle dans une synthèse en français puis le code entity framework en C# dans un projet VS EFModel ? Tu peux créer le DbContext ?

Arborescence à créer :

```
SRC └── EFModel └── EFModel.csproj └── EClassRoomDbContext.cs └── Models └── Client.cs └──
Utilisateur.cs └── SalleDeFormation.cs └── MachineVirtuelle.cs └── Facture.cs
```

OK pour lier les stagiaires à une salle

Tu peux me créer les fichiers src\Client\Client.csproj et src\Server\Server.csproj avec .NET 8

1.4 Page de Login Tu pourrais me coder la page Login.razor et son code Login.razor.cs qui permet de saisir email et mot de passe. Sur l'appuie du bouton "Connecter", la page fait un appel httpClient vers le end point Web API /login. Le controller IIS va utiliser une couche de Service (que tu vas créer) qui utilise la couche Entity Framework et qui va exécuter la requête de vérification qu'un utilisateur existe et que le mot de passe est correct.

Pour installer le package NuGet **Npgsql.EntityFrameworkCore.PostgreSQL**, exécute cette commande dans le terminal à la racine de ton projet Server:

```
dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL
```

Ou via Visual Studio :

1. Clique droit sur le projet **Server** > **Gérer les packages NuGet**.
2. Recherche `Npgsql.EntityFrameworkCore.PostgreSQL`.
3. Clique sur **Installer**.

Après installation, tu pourras utiliser `options.UseNpgsql(...)` dans ta configuration du DbContext.

Tu peux me créer le fichier appsettings.json avec des valeurs par défaut ?

On continue avec le projet Client. Je veux que tu me code la page de gestion des clients avec les fichiers razor et razor.cs . Je veux pouvoir créer, modifier ou supprimer des clients. La page razor fait des appels httpclient vers un controller IIS /users qui fait ensuite appel à une couche de services qui fait les accès à EFModel. Tu mets les DTO dans un projet séparé nommé Shared.

Voici la structure recommandée :

Les DTO sont placés dans le projet Shared. La page Blazor (Clients.razor et Clients.razor.cs) utilise ces DTO et fait des appels HTTP vers users. Le code ci-dessous est prêt à l'emploi.

Je voudrais que tu me documentes le code généré sur répertoire SRC. Que peux tu écrire en français dans le texte pour faire la documentation d'architecture technique ?

Voici un exemple de texte pour documenter l'architecture technique du code généré dans le répertoire **SRC** :

2. Architecture technique de la solution

La solution eClassRoom est structurée en plusieurs projets distincts dans le dossier **SRC**, chacun ayant un rôle précis dans l'architecture en couches :

2.1. Présentation générale

L'application repose sur une architecture **multi-couches** :

- **Client** : Application Blazor WebAssembly pour l'interface utilisateur.
- **Server** : API REST .NET (Web API) pour l'accès aux données et la logique métier.
- **EFModel** : Modèles de données Entity Framework Core et accès à la base PostgreSQL.
- **Shared** : Objets de transfert de données (DTO) partagés entre le client et le serveur.

2.2. Détail des projets

2.2.1. Client

Le projet **Client** contient l'application Blazor WebAssembly :

- Affichage des pages (login, gestion des clients, etc.).
- Appels HTTP vers l'API REST du serveur via `HttpClient`.
- Utilisation de composants BootstrapBlazor pour l'UI.
- Les pages Razor sont organisées dans le dossier **Pages**.

2.2.2. Server

Le projet **Server** expose des endpoints REST (contrôleurs Web API):

- Chaque contrôleur reçoit les requêtes du client, les valide et appelle la couche de services.
- La couche de services (dossier **Services**) centralise la logique métier et les accès aux données via Entity Framework.
- La configuration du serveur (connexion PostgreSQL, CORS, Swagger, etc.) est réalisée dans **Program.cs** et **appsettings.json**.

2.2.3. EFModel

Le projet **EFModel** regroupe:

- Les entités du modèle de données (dossier **Models**): **Client**, **Utilisateur**, **SalleDeFormation**, **MachineVirtuelle**, **Facture**.
- Le contexte de base de données **EClassRoomDbContext** configuré pour PostgreSQL.
- Ce projet est référencé par le serveur pour l'accès aux données.

2.2.4. Shared

Le projet **Shared** contient les **DTO** (Data Transfer Objects):

- Les DTO servent à échanger des données entre le client et le serveur de façon sécurisée et adaptée.
- Ils évitent d'exposer directement les entités EF côté client.

2.3. Communication et flux de données

- Le **client** envoie des requêtes HTTP (GET, POST, PUT, DELETE) vers les endpoints du **serveur**.
- Le **serveur** traite les requêtes via ses contrôleurs, qui délèguent la logique à la couche de services.
- La couche de services utilise **Entity Framework** pour interagir avec la base PostgreSQL.
- Les données sont échangées sous forme de **DTO** définis dans le projet **Shared**.

2.4. Sécurité et bonnes pratiques

- Les mots de passe doivent être stockés de façon sécurisée (hashage côté serveur).
- Les accès à l'API sont à sécuriser (authentification, autorisation).
- L'utilisation de DTO permet de maîtriser les données exposées au client.

2.5. Extensibilité

- L'architecture permet d'ajouter facilement de nouvelles entités, pages ou services.
- Les projets sont découplés pour faciliter la maintenance et les évolutions.

Ce texte peut être adapté et enrichi selon les besoins spécifiques de ton projet et les choix techniques retenus.

Je voudrais que tu reprennes le code et me génère la documentation dans le folder Docs tel qu'indiquer dans mes instructions.

Documente moi la page Login.razor et son code-behind. Explique moi les dépendances et les interactions Services et EF. Fais le dans le folder de haut niveau Docs.

Documente moi la page Clients.razor et son code-behind. Explique moi les dépendances et les interactions Services et EF. Fais le dans le folder de haut niveau Docs. Merci mon ami !

On continue avec le projet Client. Je veux que tu me code la page de gestion des machines virtuelles avec les fichiers razor et razor.cs . Je veux pouvoir créer, modifier ou supprimer des VMs. Utilise le modèle EF pour connaître les attributs. La page razor fait des appels httpclient vers un controller IIS /users qui fait ensuite appel à une couche de services qui fait les accès à EFModel. Tu mets les DTO dans un projet séparé nommé Shared. N'oublie pas de créer la documentation de page et les dépendances et les interactions Services et EF. Fais le dans le folder de haut niveau Docs.

On continue avec le projet Client. Je veux que tu me code la page de gestion des salles de formation avec les fichiers razor et razor.cs . Je veux pouvoir créer, modifier ou supprimer des salles. Utilise le modèle EF pour connaître les attributs. La page razor fait des appels httpclient vers un controller IIS /users qui fait ensuite appel à une couche de services qui fait les accès à EFModel. Tu mets les DTO dans un projet séparé nommé Shared. TU créé aussi la couche services. N'oublie pas de créer la documentation de page et les dépendances et les interactions Services et EF. Fais aussi la documentation métier. Faisle dans le folder de haut niveau Docs.

On continue avec le projet Client. Je veux que tu me code la page de gestion des utilisateurs avec les fichiers razor et razor.cs . Je veux pouvoir créer, modifier ou supprimer des utilisateurs. Utilise le modèle EF pour connaître les attributs. La page razor fait des appels httpclient vers un controller IIS /users qui fait ensuite appel à une couche de services qui fait les accès à EFModel. Tu mets les DTO dans un projet séparé nommé Shared. Tu créé aussi la couche services. N'oublie pas de créer la documentation de page et les dépendances et les interactions Services et EF. Fais aussi la documentation métier. Faisle dans le folder de haut niveau Docs.

Infrastructure Azure

Chapitre 2 - Azure On passe au service technique Azure. Je veux créer des objets dans Azure : des resource groups, des VMs. Tu saurais faire cela de manière modulaire pour chaque étape et le tout avec le SDK Azure SDK version 2.0. Je ne veut pas utiliser les API Fluent mais la nouvelle version. Tu peux me coder cela ? Crée la documentation Azure dans le Folders 'Docs'.

Tu peux me coder cette partie Azure Infrastructure ?

Ajoute ici d'autres méthodes pour créer VNet, Subnet, IP publique, NIC, etc.

Ja'i besoin de créer un Service Azure Métier de plus haut-niveau. Je veux le code pour pouvoir créer une salle de formation: il faut donc provisionner autant de machines virtuelles qu'il y a de stagiaires. Il faut donc faire les appels aux bonnes fonctions du service AzureInfrastructure et passer les paramètres user name password etc. Tu peux me coder ce service métier ?

La méthode ProvisionnerSalleAsync doit retourner un objet complexe qui contient la SalleDeFormationsDto et pour chaque stagiaire, la liste des adresses ip public ; ainsi on pourra donner aux stagiaires cette information pour qu'ils se connectent. Je veux stocker ces informations via le modèle Entity Framework dans une Table ProvisionningVM. Il faut donc que tu améliores le modèle EF et ajoute dans la couche service ces ajouts en bases de données. Tu saurais me coder cela et me le documenter en technique et en métier dans le folder 'Docs' ?

Tu peux me recréer un diagramme de séquence au format PlantUML à partir de la doc et du code des deux services ?