API VoteIn

# Résumé

Cette API permet d’enregistrer des scrutins et des votes. Elle peut retourner les résultats d’un scrutin.

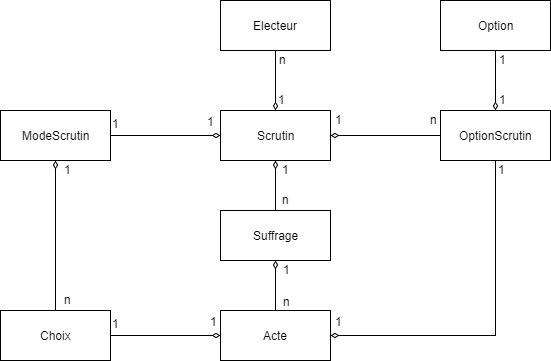
# Projet

La solution a été créée avec Visual Studio 2017. Elle utilise le framework .NET Core et est scindée en plusieurs projets :

* **VoteIn**: Projet principal. Contient les contrôleurs et les réglages de l’API. Les migrations sont aussi localisées dans ce projet.
* **VoteIn.BL**: Repositories
* **VoteIN.DAL**: Définition du contexte EntityFrameworkCore.
* **VoteIn.Model**: Modèles
* **VoteIn.Utils** : Constantes et autres outils

Il serait préférable que les migrations soient situées dans le projet DAL, mais les commandes *dotnet* pour effectuer les migrations doivent être lancées dans le projet qui contient le Startup.cs.

# Structure BDD/Modèle



Pour créer un **mode de scrutin**, on doit définir quels sont les **choix** donnés à la personne qui votera. Lors de la création d’un **scrutin**, on présente les différentes **options** aux votants (les candidats par exemple) ; si c’est un **scrutin** privé, on a aussi besoin de définir une liste d’**électeurs** qui recevront un message pour voter. Lorsqu’un utilisateur donne son suffrage, il **acte** quel est son **choix** pour chaque **option**.

Par exemple pour un scrutin majoritaire classique :

* Les **choix** donnés sont choisi/rejeté
* Les **options** sont les candidats

Chaque individu donne à un candidat le choix « choisi » et aux autres le choix « rejeté ». Le résultat de chaque candidat est son rapport

Autre exemple, pour le jugement majoritaire :

* Les **choix** sont les mentions Très-bien/Bien/Assez-bien/Passable/Insuffisant/À rejeter
* Les **options** sont les candidats.

Pour chaque candidat, son résultat pour chaque mention est le rapport

Cas particuliers : le vote alternatif, Borda, Condorcet et bien d’autres.

Ces **modes de scrutin** sont particuliers, on demande à chaque votant de trier les candidats par ordre de préférence. Pour voter, on doit donc affecter à chaque candidat sa position. Pour pouvoir gérer ces scrutins dont le nombre de **choix** n’est pas déterminé (ici, il dépend du nombre de candidat), on peut définir un **mode de scrutin** numérique (booléen *Numerique* sur *ModeScrutin*). Dans ce cas, il n’y a pas de **choix** créé en base, et on utilise le champ *Valeur*sur *l’****acte*** pour enregistrer la position choisie du candidat.

# Routes

L’API répond aux routes suivantes :

* **GET** /api/{T} : Retourne toutes les objets de type T.
* **GET** /api/{T}/{i} : Retourne l’objet de type T d’id i.
* **POST** /api/{T} : Enregistre un nouvel objet de type T.
* **PUT** /api/{T}/{i} : Modifie l’objet de type T d’id i.
* **DELETE** /api/{T}/{i} : Supprime l’objet de type T d’id i.

On peut définir dans chaque contrôleur les entités filles à inclure lors d’une requête GET via la variable *includes*. Voir *ScrutinController* pour un exemple.

Il existe aussi des routes particulières :

* Pour s’authentifier auprès de l’API :
  + **POST** /api/Auth/register : Accepte un objet de type *RegisterViewModel* et enregistre l’utilisateur dans la table **User**.
  + **POST** /api/Auth/getToken : Accepte un objet de type *AuthViewModel* et retourne un token JWT valide 40 minutes (si les identifiants reçus sont corrects) (durée modifiable dans le fichier *TokenAuthOptions*) Pour protéger une route il faut utiliser l’annotation [Authorize].
* Pour récupérer le résultat d’un **scrutin**:
  + **GET** /api/scrutin/{i}/resultat : Retourne le résultat du scrutin d’id i de manière classique, c’est-à-dire pour chaque **option** le pourcentage et le nombre de suffrages pour chaque **choix**.
  + **GET** /api/scrutin/{i}/resultatVoteAlternatif : Retourne le résultat du scrutin d’id i avec la méthode du vote alternatif. Ce résultat se compose de chaque étape d’élimination avec les scores de chaque candidat à chaque étape.
  + **GET** /api/scrutin/{i}/resultatBorda : Retourne le résultat du scrutin d’id i avec la méthode de Borda. Ce résultat se compose des scores de chaque candidat.
  + **GET** /api/scrutin/{i}/resultatCondorcet : Retourne le résultat du scrutin d’id i avec la méthode de Condorcet. Ce résultat se compose des scores de chaque candidat.
* Pour envoyer un vote formaté en utilisant le modèle du FrontEnd et l’enregistrer dans le modèle explicité ci-dessus.
  + **POST** /api/suffrage/sendVote : Accepte un objet de type *VoteScrutinMajoritaire, VoteAlternatif, VoteJugementMajoritaire* (c.f. *VoteViewModels*) et l’enregistre sous le format **Suffrage**/**Acte**

# Migrations

Pour ajouter et appliquer des migrations, il faut se rendre dans le dossier du projet **VoteIn** et ouvrir une invite de commande. Il est également possible de passer par la console du gestionnaire de packages NuGet, mais les commandes qui demandent une confirmation à l’utilisateur ne fonctionnent pas, comme le drop de la base par exemple.

Les commandes utiles sont :

* dotnet ef migrations add "Nom de la migration"
* dotnet ef database update : Application des migrations.
* dotnet ef migrations remove : Annulation de la dernière migration.
* dotnet ef database drop

# Liens

<https://docs.microsoft.com/en-us/aspnet/core/security/enforcing-ssl>

<https://code.msdn.microsoft.com/How-to-authorization-914d126b>

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity>