Paul Bernard

François Domecq

ATTENTION !

Si vous voyez ce message, vous n'avez sûrement pas utilisé la bonne branche.

(ce rapport est inchangé à partir d'ici)

⚠️ Cette branche est encore utilisée activement (après la date de rendu de rapport). Nous y ajoutons des tests et des fonctionnalités supplémentaires en vue du projet React Native du second semestre.

La branche qui correspond au rendu du 14/01, soumissible à l'évaluation est disponible à l'adresse :

https://github.com/ensc-glog/projet-2021-bernard-domecq-gr1/tree/rendu

Projet de Génie logiciel

Création de la partie Back-end d'une application mobile

ENSC 2A - Année 2020-2021

Table des matières

Introduction

Sujet

L'objectif de ce projet était de réaliser la partie Back-end d'une application mobile que nous utiliserons dans le cadre du projet de développement mobile au semestre 8. Pour réaliser ce projet, nous devions utiliser la technologie ASP.NET Core MVC. Ce framework permet de générer des applications web et des API à l'aide du modèle de conception Model-View-Controller auquel nous avons été introduit au cours de ce semestre. Cette réalisation d'API sera utilisée au second semestre pour la création d'une application mobile. Pour réaliser ce projet, nous devions également respecter certaines exigences métier et techniques listées ci-dessous :

Exigences métier

Code	Description
EF_01	L'application offre accès aux données métier via des vues web (HTML).
EF_02	L'application expose une API anonyme donnant accès aux données métier. Cette API utilise le format JSON.

Exigences techniques

Code	Description
ET_01	L'application est réalisée à l'aide de la technologie ASP.NET Core MVC.
ET_02	Les données persistantes sont stockées dans une base de données relationnelle SQLite.
ET_03	L'interface entre les classes métier et la SGBDR exploite l'outil Entity Framework Core.
ET_04	L'application respecte autant que possible les grands principes de conception étudiés en cours : séparation des responsabilités, limitation de la duplication de code, KISS, YAGNI, etc.
ET_05	L'ensemble du code source respecte la convention camelCase.
ET_06	Les noms des classes, propriétés, méthodes, paramètres et variables sont choisis avec soin pour refléter leur rôle.
ET_07	L'application dispose de tests automatisés de type "smoke tests" pour vérifier le comportement des contrôleurs

Thématique de l’application

Le choix du thème de l’application étant libre, nous avons fait le choix de concevoir un réseau social au sein de l’école : Ensciens. Notre application serait très similaire à la partie fil d’actualité, visualisation des profils des différents réseaux sociaux comme Facebook, Twitter.. Cependant, nous avons pris la décision de ne pas concevoir de messagerie instantanée ce qui aurait ajouté une grosse charge de travail. Notre réseau social permettrait donc à chaque utilisateur d’avoir son propre profil et de le personnaliser. Il aurait également accès à un fil d’actualité sur lequel sont répertoriés les différentes posts publiés par les autres utilisateurs, mais également les différents bureaux et clubs de l’école. En effet, l’application prendrait en compte la vie associative et permettrait aux différentes associations de créer des publications directement sur le réseau social afin d’annoncer les différents évènements à venir. Enfin, chaque publication dispose d'un fil de commentaire, sur lesquels les élèves, bureaux et clubs peuvent réagir à une publication.

Modélisation des données

Diagramme de classes métier

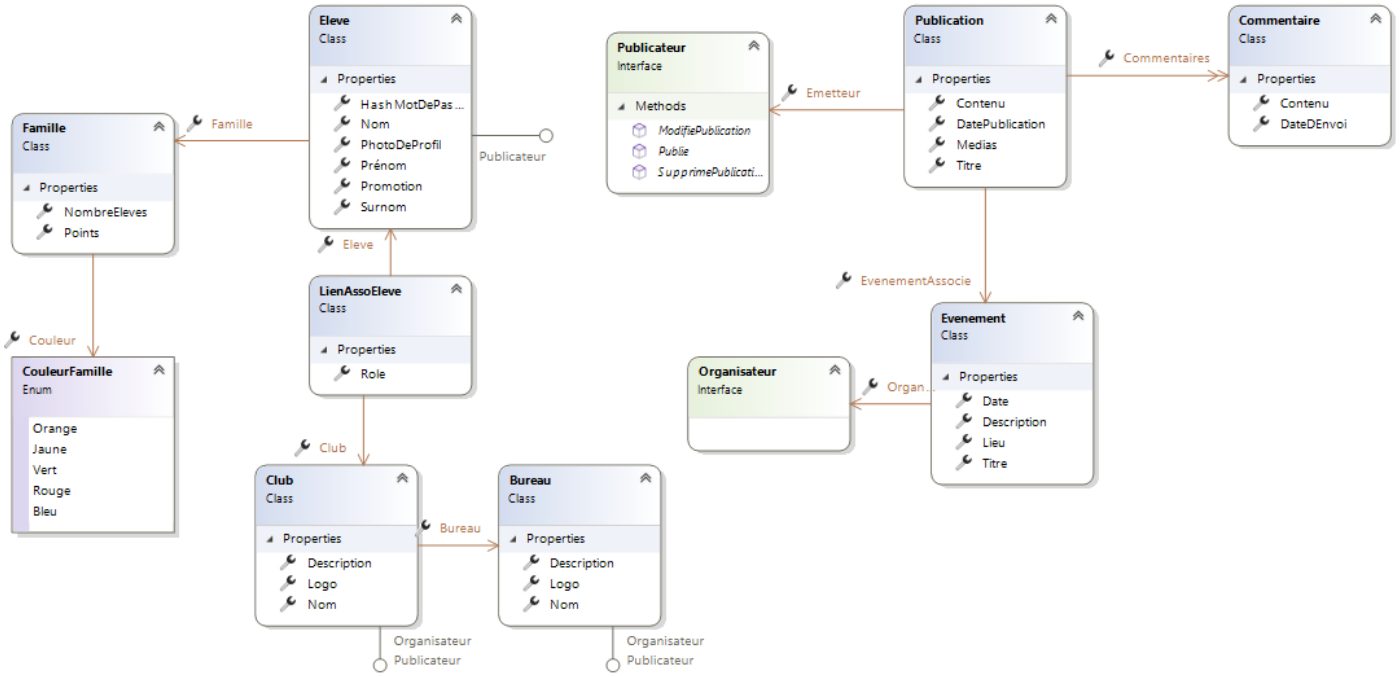

figure 1 : Diagramme de classes métier initial

Sur la figure ci-dessus se trouve le diagramme de classes que nous avions établi initialement.
La partie de droite concerne les parties liées au fil d'actualité et à la vie associative de notre application. En effet, nous avons fait le choix de créer une interface Publicateur implémentée par les classes Eleve, Club et les Bureau. Cela permet à ces trois classes de poster des publications qui peuvent optionnellement annoncer un évènement et contenir des commentaires laissés par les utilisateurs.
Les classes Bureau et Club implémentent également l'interface Organisateur. Cela leur donne la possibilité de créer des événements.
Pour ce qui concerne la partie de gauche de ce diagramme, nous pouvons voir que chaque élève a un champ Famille qui caractérise la famille à laquelle il appartient. Nous voyons également que les élèves peuvent appartenir à des clubs qui eux-même appartiennent à un Bureau. Les élèves ayant généralement un rôle au sein des associations, il nous a paru logique de créer une classe associative entre Eleve et Club ainsi qu'entre Eleve et Bureau (pas représenté sur le diagramme ci-dessus).

Nous avons donc commencé à instancier nos modèles en suivant le diagramme ci-dessus. Cependant, nous avions commis certaines erreurs de conception, puis nous avons rencontré des problèmes avec les interfaces. Cela nous a amené à faire des changements dans nos classes métier.

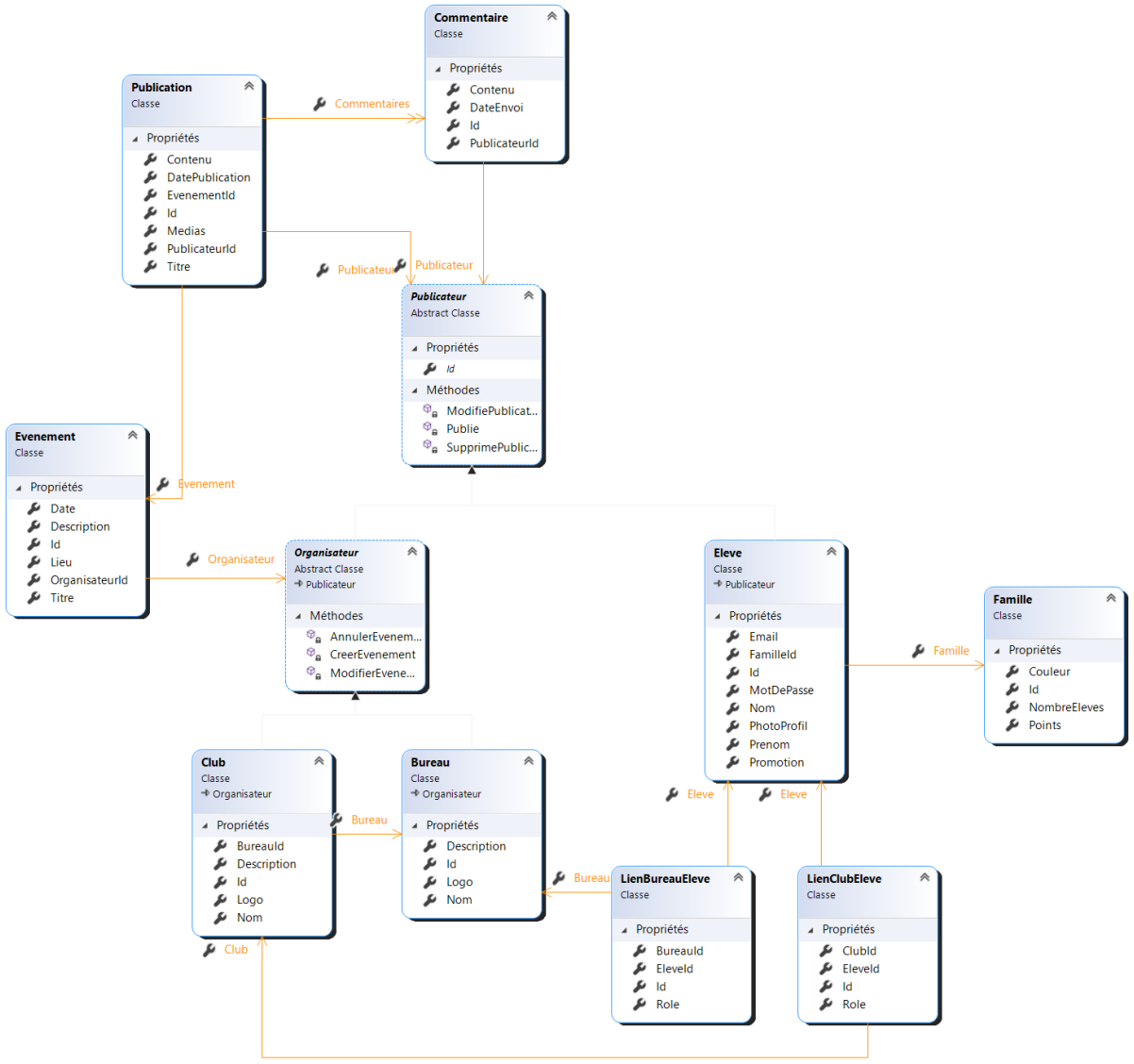

figure 2 : Diagramme de classes métier final

Comme nous pouvons le constater, les interfaces Publicateur et Organisateur ont été remplacées par deux classes abstraites : Entity Framework ne permet pas d'utiliser des interfaces dans ses modèles, donc la migration était impossible. Ainsi, Bureau et Club héritent de la classe abstraite Organisateur (qui peut créer des évènements), et Eleve et Organisateur héritent de la classe abstraite Publicateur, qui peut publier un post ou en commenter un.

Modèle relationnel

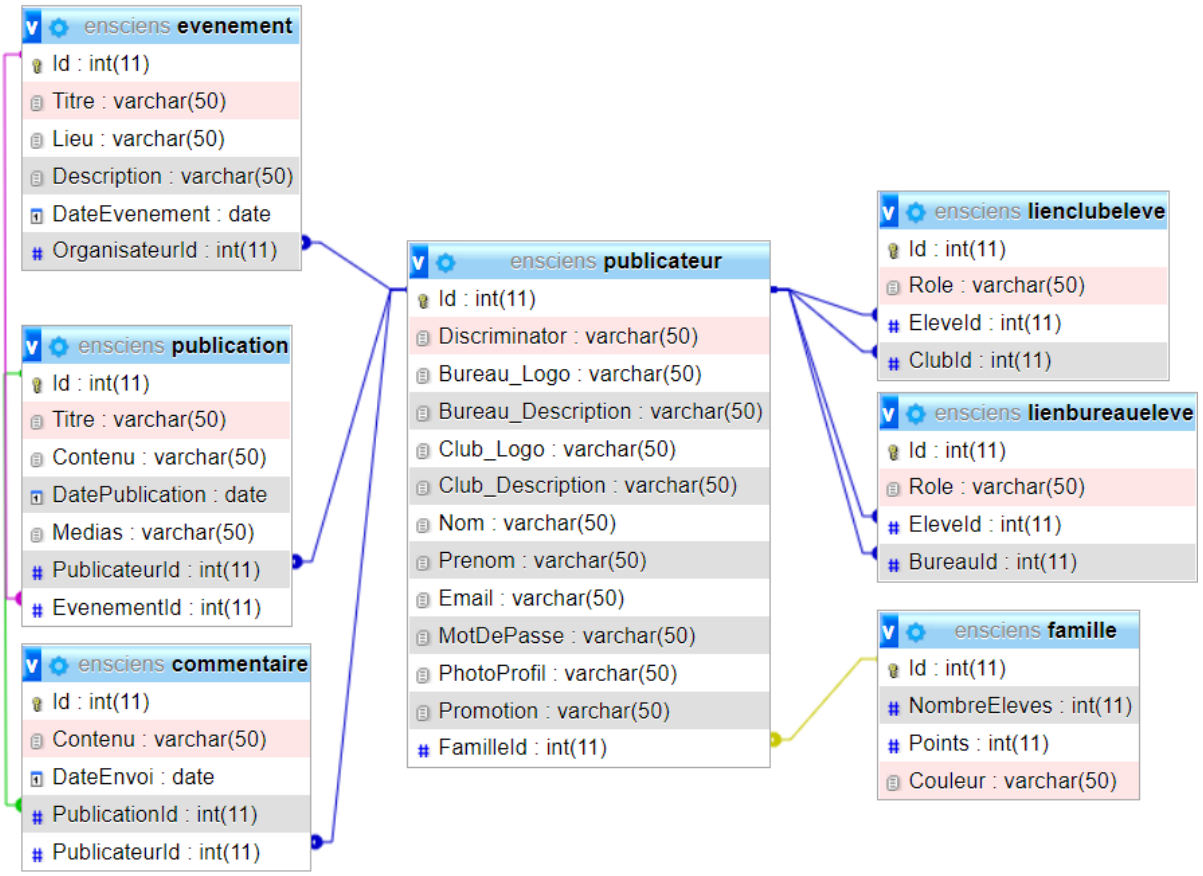

figure 3 : Modèle relationnel

La figure ci-dessus présente le modèle relationnel de notre base de données. Comme nous pouvons le voir, la table publicateur dispose d'un champ Discriminator. Ce champ permet de différencier les différents types de publicateurs, à savoir les élèves, les bureaux et les clubs. Les liens entre bureau/club et élèves sont gérés dans les tables lienbureauxeleve (resp. lienclubeleve) qui ont deux clés étrangères EleveId et BureauId (resp. ClubId) qui correspondent à deux clés primaires de la table publicateur. La table evenement dispose d'une clé étrangère OrganisateurId en provenance de la table publicateur. Seuls les bureaux et clubs peuvent organiser des évènements puisqu'elles héritent de la classe mère Organisateur. La table publication dispose de deux clés étrangères en provenance de la table publication (resp. evenement) qui correspond au publicateur du post (resp. à l'évènement annoncé dans le post, ce champ peut être null). Et enfin, chaque commentaire dispose d'une clé étrangère PublicationId pour savoir à quel post appartient quel commentaire et d'une autre clé étrangère PublicateurId pour savoir qui a publié le commentaire.

Gestion de projet

Répartition des tâches

Ci-dessous se trouve notre tableau de répartition des tâches. Nous avons effectué certaines tâches ensemble comme pour la définition des objectifs ainsi que la modélisation de notre application. Nous nous sommes ensuite répartis les tâches concernant la programmation, à l'exception de l'affichage des vues et des tests puis nous avons finalisé le projet ensemble. Nous avons, commenté le code et rédigé le rapport ensemble.

	Paul	François
I - Lancement du projet		
Définition de l'application		
Modélisation du diagramme de classes métier		
II - Réalisation du projet		
Implémentation des classes métier		
Création des controllers associés		
Création du modèle de données et migration des données		
Création de l'API Web		
Modification des vues		
Tests automatisés		
III - Finalisation du projet		
Relecture du code et commentaires		
Rédaction du rapport		

Planning

Planning initial

figure 4 : Planning initial

La figure ci-dessus représente le planning initial que nous nous étions fixés. Nous avions choisi de garder 1 semaine pour la phase de modélisation afin de commencer le projet sur de bonnes bases. Selon nous, la création du modèle et des controllers associés ne devait pas nous prendre beaucoup de temps car notre modélisation nous semblait cohérente. La première semaine de vacances était dédiée à la création du modèle de données (fichier SeedData), de l'API et de la mise en forme des vues web. Enfin, tout au long du projet, nous souhaitions ajouter des tests automatisés de type smoke tests. La longue phase où nous n'avions "rien prévu" nous permettait d'avoir une marge en cas de problèmes.

Planning final

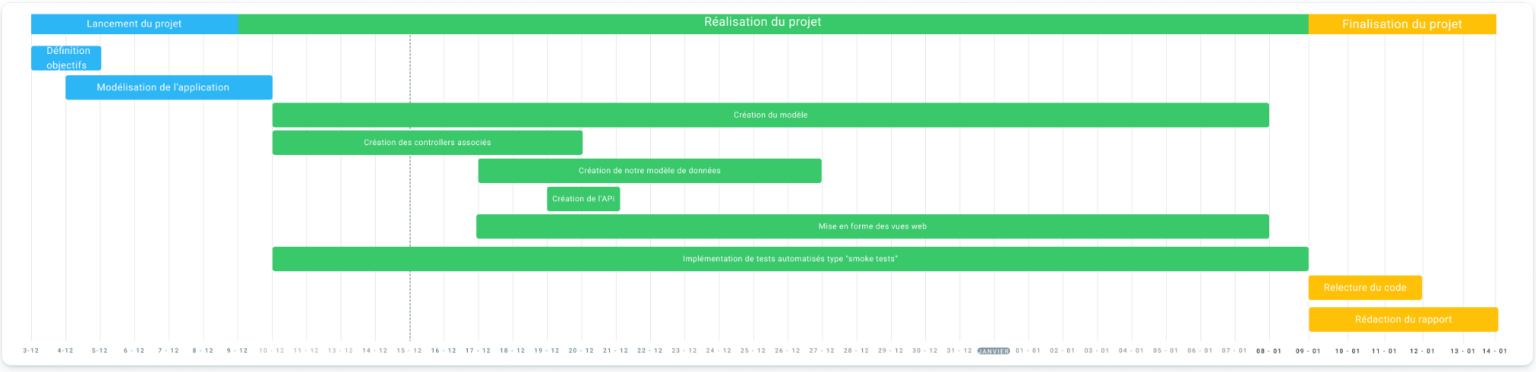

figure 5 : Planning final

En réalité, le planning qui a été suivi est totalement différent de celui prévu initialement. Tout d'abord, nous avons rencontré beaucoup de difficultés à créer nos classses métier, notamment avec les interfaces. C'est pourquoi la création du

modèle nous a pris beaucoup plus de temps que prévu. De même, la création de notre modèle de données (fichier SeedDate) nous a pris plus de temps que prévu car il nous a fallu chercher tous les clubs existants à l'école, le rôle de chaque élève au sein des bureaux. Les adhérents aux différents clubs ont cependant été inventés car chercher quels élèves étaient membres de tels clubs nous aurait pris trop de temps. Nous avons également fait le choix de ne pas ajouter tous les élèves de l'école dans notre modèle de données mais seulement certains élèves, la plupart de deuxième année. Enfin, nos classes métiers ayant évolué régulièrement au cours du projet, il nous a logiquement fallu régulièrement modifier les vues web associées.

Détail d'implémentation

Authentification

Un aspect important que nous souhaitions gérer dans notre projet était la sécurisation des données de l'API. Nous avons donc ajouté un système d'authentification au niveau des requêtes qui visent à modifier, supprimer ou ajouter des informations concernant les élèves, les bureaux, les clubs, les évènements, les publications ou les commentaires. Chaque requête API sensible doit comporter en entête l'adresse mail et le mot de passe d'un élève habilité à effectuer l'action demandée. Pour les familles, aucune modification majeur n'est autorisée, quel que soit le requêteur, mais les responsables de famille du Bureau des Familles peuvent ajouter ou enlever des points à une famille.

Cet aspect est détaillé dans la partie Documentation de l'API.

Il ne nous a pas semblé utile d'ajouter la même sécurité pour les vues HTML, puisque le projet est prévu pour être utilisé via une application, le site web étant essentiellement utile pour évaluer l'état de la base de données et la manipuler aisément. Cependant, il nous aurait suffi de copier-coller les fonctions d'authentifications dans les contrôleurs de vues si besoin.

Les publicateurs

La classe Publicateur étant la classe mère des classes Eleve, Organisateur (et donc Bureau et Club), nous avons rencontré des difficultés au moment de l'affichage des noms des publicateurs de posts ou de commentaires. En effet, lors de la création de classes métiers, nous avions configuré le champ Nom différemment dans la classe Eleve de la classe Bureau et de la classe Club. Ainsi, nous nous sommes retrouvés avec une table Publicateur avec trois champs nom :

- Nom correspondant à NomEleve
- Bureau_Nom
- Club_Nom

Nous avons mis du temps à trouver la source du problème. Une fois que nous l'avons trouvée, nous avons ajouté un attribut Nom abstrait dans la classe mère Publicateur. Puis dans le PublicationController, avec les modifications suivantes :

```
public async Task<IActionResult> Index()
{
    var ensciensContext = _context.Publication.Include(p => p.Evenement).Include(p=>p.Publicateur);
    return View(await ensciensContext.ToListAsync());
}
```

```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var publication = await _context.Publication
        .Include(p => p.Evenement)
        .Include(p=>p.Commentaires)
        .ThenInclude(c=>c.Publicateur)
        .Include(p=>p.Publicateur)
        .FirstOrDefaultAsync(m => m.Id == id);
```

Nous avons ensuite pu récupérer dans les View Publication/Index (resp. Publication/Details) le publicateur du post (resp. les publicateurs des différents commentaires).
Il nous a également été difficile de manipuler le langage CSHTML pour présenter les vues, qui n'est pas naturel à manipuler. Par exemple, dans le code suivant :
@Html.DisplayFor(modelItem => item.Contenu)
La partie droite de la fonction anonyme n'autorise pas à mettre du code C#, et ses valeurs possibles sont limitées.

Documentation de l'API

Bureau

`GET /api/BureauApi/`

Cette commande renvoie un objet JSON contenant la liste de tous les bureaux

`GET /api/BureauApi/{id}`

Cette commande renvoie un objet JSON contenant les informations du bureau ayant pour id **l'id rentré dans l'URL**.

`PUT /api/BureauApi/{id}`

Cette commande modifie les informations du bureau ayant pour id **l'id rentré dans l'URL** avec le nouvel objet bureau rentré en paramètre.

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à modifier le bureau correspondant. Un tel élève est soit un président, soit un vice-président de la structure, soit un des respos famille pour le BDF.

`POST /api/BureauApi/`

Cette commande crée un nouveau bureau avec l'objet bureau rentré en paramètre.

`DELETE /api/BureauApi/{id}`

Cette commande supprime le bureau ayant pour id **l'id rentré dans l'URL.**

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à supprimer le bureau correspondant. Un tel élève est soit un président, soit un vice-président de la structure, soit un des respos famille pour le BDF.

Club

`GET /api/ClubApi/`

Cette commande renvoie un objet JSON contenant la liste de tous les clubs.

`GET /api/ClubApi/{id}`

Cette commande renvoie un objet JSON contenant les informations du club ayant pour id **l'id rentré dans l'URL**.

`PUT /api/ClubApi/{id}`

Cette commande modifie les informations du club ayant pour id **l'id rentré dans l'URL** avec le nouvel objet club rentré en paramètre.

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à modifier le club correspondant. Un tel élève est soit un président, soit un vice-président du club.

`POST /api/ClubApi/`

Cette commande crée un nouveau club avec l'objet club rentré en paramètre.

`DELETE /api/ClubApi/{id}`

Cette commande supprime le club ayant pour id **l'id rentré dans l'URL.**

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à supprimer le club correspondant. Un tel élève est soit un président, soit un vice-président du club

.

Commentaire

`GET /api/CommentaireApi/`

Cette commande renvoie un objet JSON contenant la liste de tous les commentaires.

`GET /api/CommentaireApi/{id}`

Cette commande renvoie un objet JSON contenant les informations du commentaire ayant pour id **l'id rentré dans l'UR**L.

`PUT /api/CommentaireApi/{id}`

Cette commande modifie les informations du commentaire ayant pour id **l'id rentré dans l'URL** avec le nouvel objet commentaire rentré en paramètre.

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à publier au nom du publicateur : - Si le publicateur est un bureau ou un club, les identifiants en entête doivent correspondre à un élève président, vice-président ou respo famille de la structure. - Si le publicateur est un élève, les identifiants doivent correspondre à cet élève.

`POST /api/CommentaireApi/`

Cette commande crée un nouveau commentaire avec l'objet commentaire rentré en paramètre.

`DELETE /api/CommentaireApi/{id}`

Cette commande supprime le commentaire ayant pour id **l'id rentré dans l'URL.**

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à publier au nom du publicateur : - Si le publicateur est un bureau ou un club, les identifiants en entête doivent correspondre à un élève président, vice-président ou respo famille de la structure. - Si le publicateur est un élève, les identifiants doivent correspondre à cet élève.

Eleve

`GET /api/EleveApi/`

Cette commande renvoie un objet JSON contenant la liste de tous les élèves.

Sécurité : Le mot de passe n'est pas affiché

```
private string SecurePassword(string password)
        {
            return new String('*',password.Length);
        }
```

Puis, nous l'appellons dans la commande GET : /api/EleveApi/. Ainsi, lors de l'appel de la commande, le résultat affiché est celui-ci :

GET /api/EleveApi/{**id**}

Cette commande renvoie un objet JSON contenant les informations de l'élève ayant pour id **l'id rentré dans l'URL**.
Comme pour la commande précédente, le mot de passe est crypté à l'aide de la fonction securePassword.

PUT /api/EleveApi/{id}

Cette commande modifie les informations de l'élève ayant pour id **l'id rentré dans l'URL** avec le nouvel objet élève rentré en paramètre.

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe de l'élève à modifier.

`POST /api/EleveApi/`

Cette commande crée un nouvel élève avec l'objet élève rentré en paramètre.

`DELETE /api/EleveApi/{id}`

Cette commande supprime l'élève ayant pour id **l'id rentré dans l'URL.**

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe de l'élève à supprimer.

Evenement

`GET /api/EvenementApi/`

Cette commande renvoie un objet JSON contenant la liste de tous les évènements.

`GET /api/EvenementApi/{id}`

Cette commande renvoie un objet JSON contenant les informations de l'évènement ayant pour id **l'id rentré dans l'URL.**

`PUT /api/EvenementApi/{id}`

Cette commande modifie les informations de l'évènement ayant pour id **l'id rentré dans l'URL** avec le nouvel objet évènement rentré en paramètre.

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à modifier l'évènement. Un tel élève est soit un président, soit un vice-président, soit un respo famille de la structure organisatrice.

`POST /api/EvenementApi/`

Cette commande crée un nouvel évènement avec l'objet évènement rentré en paramètre.

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à publier un nouvel évènement au nom du bureau ou du club organisateur. Un tel élève est soit un président, soit un vice-président, soit un respo famille de la structure organisatrice.

`DELETE /api/EvenementApi/{id}`

Cette commande supprime l'évènement ayant pour id **l'id rentré dans l'URL.**

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à supprimer l'évènement associé au bureau ou au club organisateur. Un tel élève est soit un président, soit un vice-président, soit un respo famille de la structure organisatrice.

Famille

`GET /api/FamilleApi/`

Cette commande renvoie un objet JSON contenant la liste de toutes les familles.

`GET /api/FamilleApi/{id}`

Cette commande renvoie un objet JSON contenant les informations de la famille ayant pour id **l'id rentré dans l'URL**.

`PUT /api/FamilleApi/{id}?nbPoints={nbPoints}`

Ajoute **nbPoints** points à la famille d'identifiant **id**. nbPoints peut être négatif, auquel cas la famille perd |**nbPoints**| points.

Publication

`GET /api/PublicationApi/`

Cette commande renvoie un objt JSON contenant la liste de toutes les publications.

`GET /api/PublicationApi/{id}`

Cette commande renvoie un objet JSON contenant les informations de la publication ayant pour id **l'id rentré dans l'URL**.

`PUT /api/PublicationApi/{id}`

Cette commande modifie les informations de la publication ayant pour id **l'id rentré dans l'URL** avec le nouvel objet publication rentré en paramètre.

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à modifier la publication :
- Si le publicateur est un bureau ou un club, les identifiants en entête doivent correspondre à un élève président, vice-président ou respo famille de la structure.
- Si le publicateur est un élève, les identifiants doivent correspondre à cet élève.

`POST /api/PublicationApi/`

Cette commande crée une nouvelle publication avec l'objet publication rentré en paramètre.

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à publier au nom du publicateur :
- Si le publicateur est un bureau ou un club, les identifiants en entête doivent correspondre à un élève président, vice-président ou respo famille de la structure.
- Si le publicateur est un élève, les identifiants doivent correspondre à cet élève.

`DELETE /api/PublicationApi/{id}`

Cette commande supprime la publication ayant pour id **l'id rentré dans l'URL.**

Sécurité : La requête à l'API doit avoir en entête les champs Mail et Password contenant respectivement l'adresse mail et le mot de passe d'un élève autorisé à supprimer la publication :
- Si le publicateur est un bureau ou un club, les identifiants en entête doivent correspondre à un élève président, vice-président ou respo famille de la structure.
- Si le publicateur est un élève, les identifiants doivent correspondre à cet élève.

Liste et description des procédures de test automatisé

Nous avons également réalisé quelques tests automatisés de type “smoke tests” au cours de ce projet pour vérifier le comportement de nos controllers et de nos API. Pour réaliser ces tests, nous avons donc suivi le patron AAA :

- Arrange
- Act
- Assert

Tests sur les View

`Get_EndpointsReturnSuccessAndCorrectContentType`

Ce test prend en paramètre un URL et vérifie que l’URL redirige vers une page qui existe et est au format adapté (HTML).

Nous avons réalisé huit tests avec les urls suivants :

```
[InlineData("/")]
[InlineData("/Home/Index")]
[InlineData("/Eleve/")]
[InlineData("/Eleve/Create")]
[InlineData("/Bureau/")]
[InlineData("/Bureau/Edit/61")]
[InlineData("/Famille/")]
[InlineData("/Publication/")]
```

`Get_EndpointsReturnError`

Ce test prend en paramètre un URL, si jamais l’URL n’existe pas dans notre application, alors le test renvoie une erreur 404 à l’utilisateur.
Les URL que nous avons testé avec ce test sont les suivants :

/Famille/Edit/2	Nous ne souhaitons pas qu’un utilisateur puisse modifier les informations d’une famille. Nous nous assurons donc que l’URL Famille/Edit/2 renvoie vers page 404.
/Famille/Delete	De même, nous ne souhaitons pas qu’un utilisateur puisse supprimer une des familles de l’école. Nous nous assurons donc que l’URL Famille/Delete renvoie une page 404.

/ControleurInexistant	Pour s'assurer que le serveur ne nous renvoie une "véritable" page qu'à condition que l'on tente d'atteindre un contrôleur existant, nous vérifions que /ControleurInexistant renvoie bien une page 404.

Tests sur les Api

`Get_EndpointsReturnSuccessAndCorrectContentType`

Ce test prend en paramètre un URL et vérifie que l'endpoint de l'API avec la méthode GET renvoie un résultat, et que ce résultat est au format JSON.
Nous avons testé les URL suivantes avec ce test :

```
[InlineData("/api/EleveApi")]
[InlineData("/api/BureauApi")]
[InlineData("/api/ClubApi")]
[InlineData("/api/CommentaireApi")]
[InlineData("/api/EvenementApi")]
[InlineData("/api/PublicationApi")]
[InlineData("/api/EleveApi/1")]
[InlineData("/api/EvenementApi/1")]
```

`Post_EndpointsReturnError`

Ce test vise à s'assurer qu'il est impossible de créer une nouvelle famille au sein de notre application par le biais d'une requête API. Les différentes étapes de ce test sont les suivantes :

- Assert : Création d'un URL et du contenu de requête : les informations d'une famille à créer.
- Act : On teste la création d'une nouvelle famille
- Assert : Le test renvoie un code d'erreur 405

Bilan et perspectives sur le projet

Ce projet nous a permis de nous familiariser avec la technologie ASP.NET CORE qui était nouvelle pour nous. C'est également le premier projet où nous utilisons l'architecture logicielle MVC. Lors de ce projet, nous avons également été initié à la création d'API, API qui sera réutilisée pour le projet de développement mobile au 2ème semestre. De plus, nous avons également eu l'occasion de réaliser des tests automatisés à l'aide du framework xUnit, les tests étant un aspect très important du génie logiciel. Enfin, nous avons pu améliorer nos compétences dans le versionnage de code et le travail collaboratif avec une utilisation de Git et Github plus poussée que lors de nos projets précédents. Ce projet nous a donc permis d'approfondir tous les concepts abordés dans le cours de Génie Logiciel et de les appliquer à la réalisation d'un projet.

Pour la continuité de ce projet, il nous paraîtrait important de peaufiner notre système d'authentification : par exemple, actuellement, seuls les présidents de bureau peuvent modifier ou supprimer un bureau, mais ce sont aussi les seuls qui peuvent créer un post. Les responsables communication des associations ne le peuvent pas et c'est dommage. Nous aimerions raffiner notre système d'authentification pour permettre de mieux filtrer les élèves.
Quelques détails de sécurité nous semblent important à mettre en place : par exemple, un club ne doit pouvoir être créé que par le responsable club du bureau auquel le club est rattaché.
Nous devrions aussi ajouter tous les élèves de l'école dans notre base de données et définir les rôles de chaque élève au sein des clubs. Nous aimerions aussi améliorer le système de gestion et de stockage des images dynamiques : actuellement, il n'est pas possible de créer une nouvelle image, il est seulement possible de pointer vers une des images préexistantes sur le serveur.
Enfin, nous comptons utiliser les API que nous avons créée lors du projet de développement mobile. Nous souhaitons ainsi créer une application mobile faisant office de réseau social au sein de l'école. Ainsi, les élèves pourront utiliser une application propre à l'ENSC pour suivre l'évolution de la vie étudiante et associative de l'école. Cela représenterait une alternative pour les élèves qui ne souhaitent pas installer Facebook mais qui veulent être tenu au courant de la vie associative.