



Projet Génie Logiciel

Création d'une application web intégrant une API

Grenet Rebecca

Roche Eléa

08.01.2023

Table des matières :

I. Contexte	2
II. Procédure d'installation et de test de l'application.	2
III. Thématique de l'application.	3
A. Les cinémas	3
B. Les films	4
C. Les salles de cinéma	5
D. Les séances	5
IV. Modélisation des données	6
V. Documentation de l'API.	7
A. Liste des points d'entrée et description des données renvoyées	7
B. Commandes CURL	9
VI. Organisation de l'équipe, répartition des tâches dans le groupe et planning.	14
VII. Bilan et perspectives sur le projet.	15

I. Contexte

Dans le cadre du projet de Génie Logiciel au sein de l'ENSC, nous avons réalisé une application web intégrant une API, qui sera utilisée pour une application mobile que nous réaliserons au semestre suivant. Ce projet a été réalisé à l'aide de la technologie ASP.NET Core MVC, et l'outil de versioning de code GitHub.

Le contexte métier choisi est celui de la gestion de cinémas d'une même filiale, comme Pathé Cinémas. En effet, à travers notre application web, nous souhaitons pouvoir accéder mais aussi gérer des informations clés pour un cinéma, comme l'ajout de film ou de séances. Ces informations sont stockées et actualisées dans une base de données relationnelle SQLite.

II. Procédure d'installation et de test de l'application.

Concernant la procédure d'installation de notre application web, nous avons rédigé le fichier ReadMe.md qui se trouve à la racine du dépôt GitHub, afin que cela soit plus accessible pour toute personne disposant de notre dépôt.

Dans un premier temps, il est nécessaire de cloner le dépôt GitHub sur son ordinateur. Pour ce faire, il faut se placer à l'endroit souhaité dans ses documents, et ouvrir un terminal, puis coller la ligne suivante :

```
git clone https://github.com/ensc-glog/projet-elea_rebecca_gr3.git
```

Cette commande vous permet de récupérer le projet. Par la suite, il se peut que l'ordinateur n'est pas l'outil Entity Framework Core CLI, dans ce cas, vous devez coller cette ligne de commande dans le terminal :

```
dotnet tool install --global dotnet-ef
```

Vous pouvez vérifier que l'installation s'est bien effectuée à l'aide de la commande suivante `dotnet ef`.

Après cette installation, il faut ouvrir le projet sur un éditeur de code (tel que Visual Studio Code, ou Sublime Text). Une fois cela effectué, vous devez ouvrir un terminal dans votre logiciel, et lancer la commande suivante :

```
dotnet watch run
```

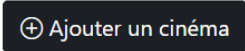
Cela va lancer le projet, et ouvrir l'application web sur le navigateur web de l'ordinateur, en vous dirigeant automatiquement sur la page d'accueil.

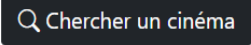
III. Thématique de l'application.

Notre application web concerne la gestion de plusieurs cinémas appartenant à une même filiale. Celle-ci permet ainsi d'accéder et de modifier les données des différents cinémas, avec respectivement leurs différentes salles et séances. De même, il sera possible d'ajouter les informations d'un film, de le consulter et de le modifier. Ainsi, nous avons quatre classes métiers, à savoir : cinéma, film, salle de cinéma et séance.

Par ailleurs, nous avons fait attention lors de l'implémentation à respecter les principes de l'acronyme SOLID, avec notamment le principe de responsabilité unique (S). En effet, dans chaque controller des modèles, nous avons implémenté les opérations de persistance de données (CRUD), ce qui respecte également le concept énoncé précédemment.

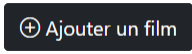
A. Les cinémas

Lorsque nous arrivons sur la partie Cinéma de notre application web, nous avons directement accès à tous les cinémas disponibles, en y indiquant leur nom ainsi que leur adresse. Sur cette page, trois actions sont alors possibles pour l'utilisateur. La première concerne l'ajout d'un cinéma ; l'utilisateur accèdera, à l'aide du bouton , à un formulaire dans lequel il devra indiquer les informations nécessaires comme l'adresse, le responsable du cinéma, ou bien le prix d'une place. La deuxième action est la recherche

de cinéma  qui s'effectue selon la ville. L'utilisateur devra indiquer la ville souhaitée parmi celles disponibles, et il sera alors affiché les villes disponibles. Enfin, il est possible de consulter un cinéma, pour ce faire, il suffit de cliquer sur le bouton "Plus d'info". A partir de cette page nous avons quatre actions qui sont possibles, à savoir :

- Modifier les informations du cinéma à travers un formulaire, à partir du bouton "Modifier".
- Voir tous les films disponibles pour ce cinéma, à partir du bouton "Voir les films disponibles".
- Voir les différentes salles du cinéma, à partir du bouton "Voir les salles du cinéma".
- Retourner à la liste des différents cinémas grâce au bouton "Retour à la liste".


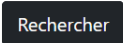
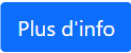
B. Les films

En ce qui concerne les films, nous avons les mêmes possibilités que celles du cinéma, à savoir l'action , et l'action de voir toutes les informations pour un film. Lorsque l'on arrive sur le formulaire de création, il est nécessaire de remplir les informations suivantes : le nom du film, le nom du réalisateur, un résumé, le genre du film, ainsi que la date de sortie et la durée du film.

Lorsque nous cliquons sur le bouton "Plus d'info", nous obtenons les mêmes informations que celles énoncées précédemment pour le formulaire de création. Sur la page correspondant à ce dernier bouton, nous avons de nouveau plusieurs actions disponibles, à savoir :

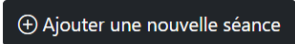
- Voir les différentes séances pour ce film, à partir du bouton "Voir les séances".
- Modifier les informations de ce film à travers un formulaire, à partir du bouton "Modifier".
- De supprimer un film, et ainsi toutes les salles et séances associés à ce même film, à partir du bouton "Supprimer".
- De retourner à la liste des films grâce au bouton "Retour à la liste".

C. Les salles de cinéma

Concernant les salles de cinéma, la page principale affiche toutes les salles disponibles pour tous les cinémas. Pour autant, il est possible de les trier grâce à la recherche suivante :   . Comme pour le Film, il est possible de créer une salle, en y indiquant le cinéma associé, le nombre de places, ainsi que le numéro de la salle. Enfin, en cliquant sur  d'une salle, nous obtenons les mêmes informations que celles énoncées précédemment, et nous avons également la possibilité de :

- Modifier les informations de cette salle à partir du bouton “Modifier”.
- Supprimer la salle, et donc toutes les séances associées à cette salle, à partir du bouton “Supprimer”.
- Retourner à la liste des salles à partir du bouton “Retour à la liste”.

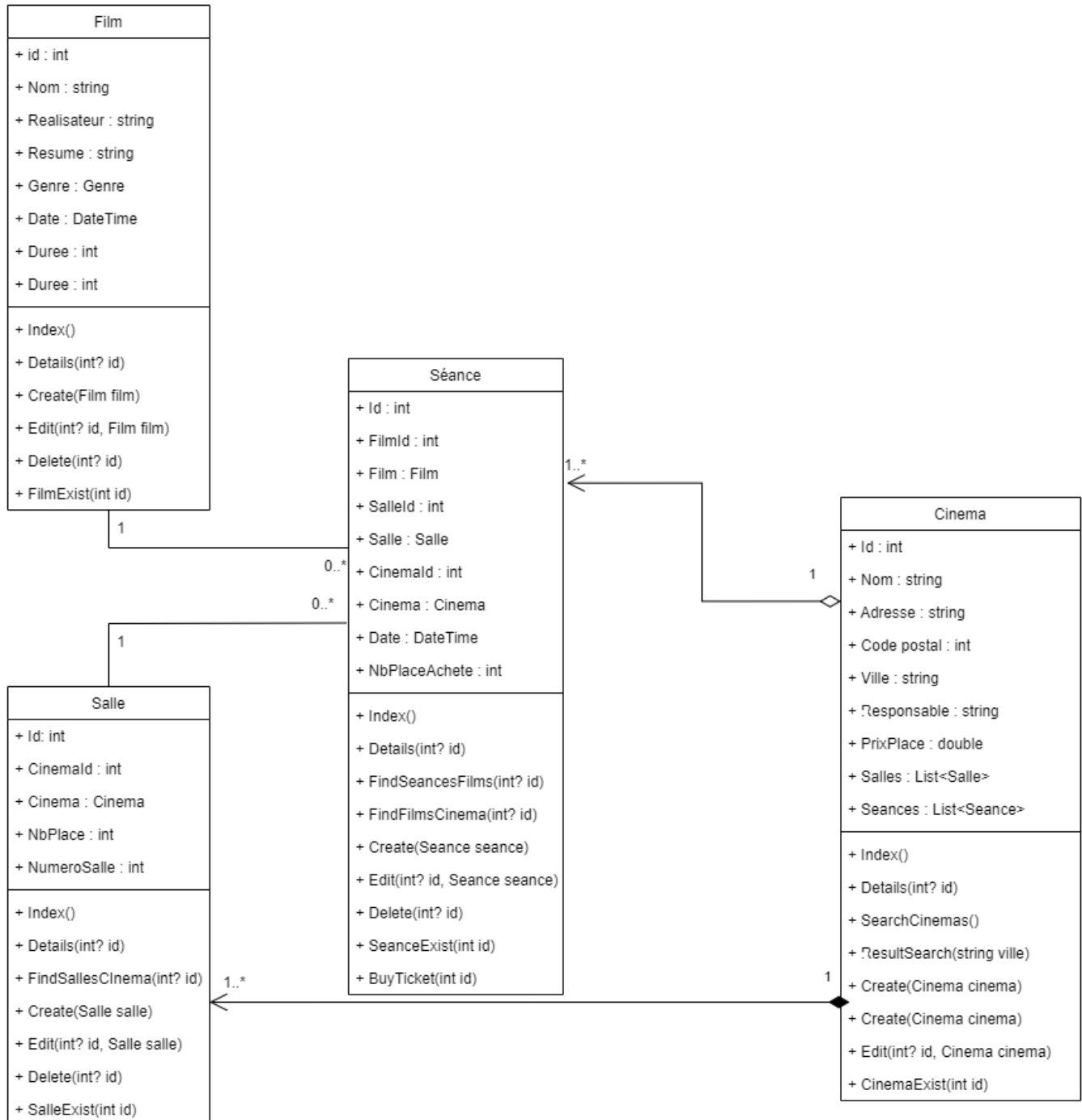
D. Les séances

Pour finir, notre dernière classe métier est celle des séances de cinéma. Nous pouvons ajouter de nouvelles séances en cliquant sur  . Pour chacune de ces séances, il est indiqué le film qui sera diffusé, la date de sortie du film, le cinéma dans lequel a lieu cette séance ainsi qu'un extrait du résumé. En cliquant sur “Plus d'info”, nous sommes dirigés sur une page qui affiche les mêmes informations que sur la page précédente pour le film considéré, avec en plus le résumé complet, la date de la séance, le numéro de la salle, la durée du film ainsi que le nombre de places achetées pour cette séance. Nous avons également les actions suivantes :

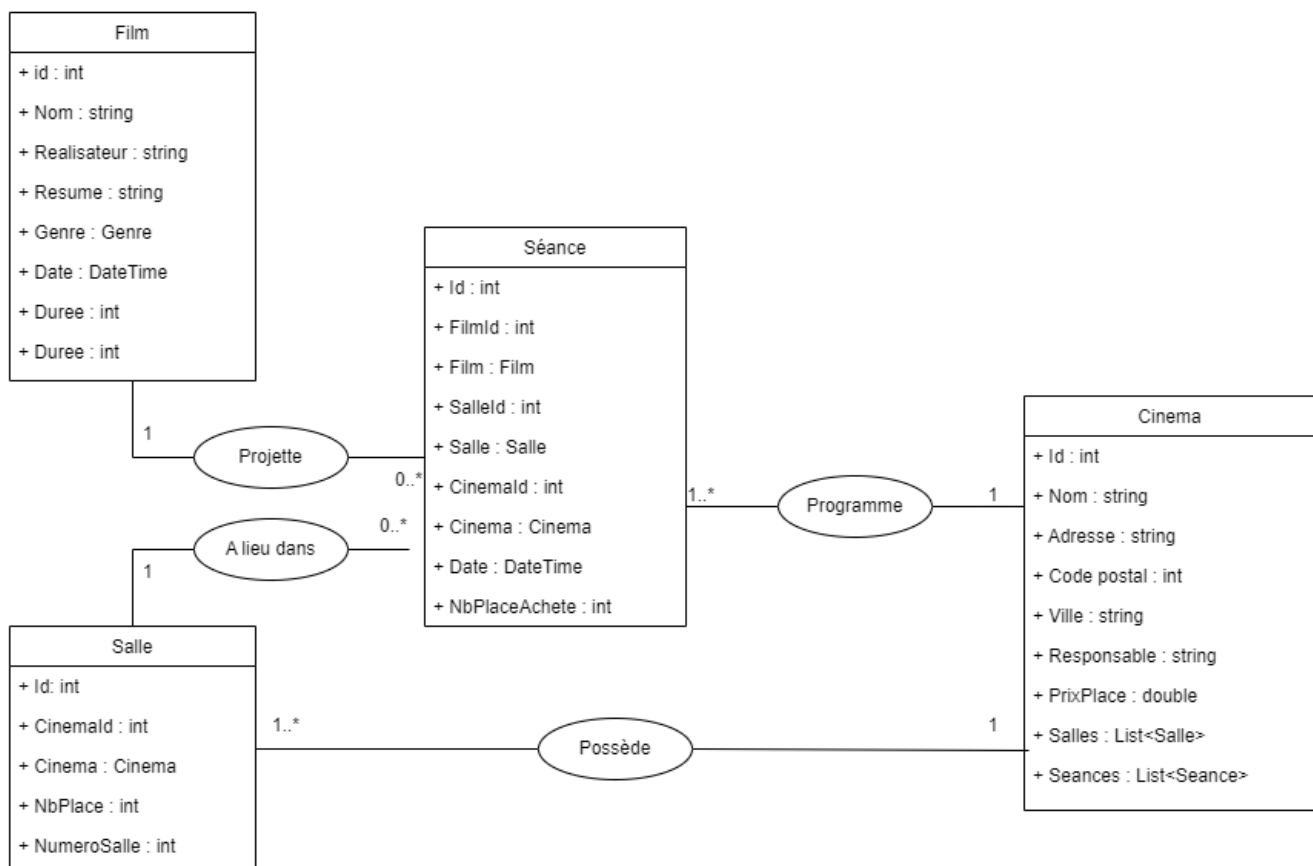
- Acheter un ticket pour cette séance, grâce au bouton “Acheter un ticket”. L'utilisateur sera alors redirigé vers la page d'accueil, avec un message l'alertant si le ticket a bien été réservé ou s'il n'y avait plus de places disponibles pour cette séance.
- Modifier les informations de cette séance à partir du bouton “Modifier”.
- Supprimer la séance, à partir du bouton “Supprimer”.
- Retourner à la liste des séances.

IV. Modélisation des données

A. Diagramme de classe



B. Modèle relationnel



V. Documentation de l'API.

A. Liste des points d'entrée et description des données renvoyées

Pour chacune des méthodes implémentées, les données sont renvoyées au format JSON.

Point d'entrée	Description des données renvoyées
CinemaApiController	
GetCinemas()	Renvoie la liste de tous les cinémas.
GetCinema(int id)	Renvoie le cinéma associé à l'identifiant id donné en paramètre.

GetCinemas(string ville)	Renvoie la liste des cinémas associés à la ville donnée en paramètre.
PostCinema(Cinema cinema)	Permet d'ajouter le cinéma entré en paramètre à la liste des cinémas.
PutCinema(int id, Cinema cinema)	Permet de modifier le cinéma associé à l'identifiant id donné en paramètre en le remplaçant par le cinéma donné en paramètre.
DeleteCinema(int id)	Permet de supprimer le cinéma associé à l'identifiant donné en paramètre.
FilmApiController	
GetFilms()	Renvoie la liste de tous les films.
GetFilm(int id)	Renvoie le film associé à l'identifiant id donné en paramètre.
PostFilm(Film film)	Permet d'ajouter le film entré en paramètre à la liste des films.
PutFilm(int id, Film film)	Permet de modifier le film associé à l'identifiant id donné en paramètre en le remplaçant par le film donné en paramètre.
DeleteFilm(int id)	Permet de supprimer le film associé à l'identifiant donné en paramètre.
FilmExist(int id)	Permet de vérifier si un film existe ou non. Retourne un booléen.
SeanceApiController	
GetSeances()	Renvoie la liste de toutes les séances.
GetSeance(int id)	Renvoie la séance associée à l'identifiant id donné en paramètre.
GetFilms(int id)	Renvoie la liste des films associée à l'identifiant id donné en paramètre. Cela est dans l'API de séance car ça permet d'afficher seulement les films qui ont des séances et donc qui sont à l'affiche.
GetSeanceFilms(int id)	Renvoie la liste de toutes les séances d'un film, l'identifiant id donné en paramètre est donc l'identifiant d'un film.

PostSeance(Seance seance)	Permet d'ajouter une séance entrée en paramètre à la liste des séances.
PutSeance(int id, Seance seance)	Permet de modifier la séance associée à l'identifiant id donné en paramètre en la remplaçant par la séance donnée en paramètre.
DeleteSeance(int id)	Permet de supprimer la séance associée à l'identifiant donné en paramètre.
SeanceExist(int id)	Permet de vérifier si une séance existe ou non. Retourne un booléen.
SalleApiController	
GetSalles()	Renvoie la liste de toutes les salles.
GetSalle(int id)	Renvoie la salle associée à l'identifiant id donné en paramètre.
GetSallesCinema(int id)	Renvoie la liste de toutes les salles du cinéma associé à l'identifiant id
PostSalle(Salle salle)	Permet d'ajouter une salle entrée en paramètre à la liste des salles.
PutSalle(int id, Salle salle)	Permet de modifier la salle associée à l'identifiant id donné en paramètre en la remplaçant par la salle donnée en paramètre.
DeleteSalle(int id)	Permet de supprimer la salle associée à l'identifiant donné en paramètre.
SalleExist(int id)	Permet de vérifier si une salle existe ou non. Retourne un booléen.

B. Commandes CURL

Afin de vérifier que les méthodes CRUD (Create, Read, Update, Delete) de notre API renvoyaient bien le résultat souhaité, nous avons utilisé Postman, mais aussi les commandes CURL.

Par ailleurs, pour chacune de ces commandes, il vous faudra modifier le port utilisé pour lancer l'application web (égale à 7216 dans nos requêtes). Celui-ci est indiqué juste

après avoir lancé la commande `dotnet watch run` dans un terminal. Voici quelques exemples ci-dessous :

- **Cinéma :**

Commande CURL permettant de voir tous les cinémas :

```
curl -X GET -k -H 'Content-Type: application/json' -i 'https://localhost:7216/api/CinemaApi'
```

Commande CURL permettant de voir un cinéma suivant son identifiant unique, ici 9 :

```
curl -X GET -k -H 'Content-Type: application/json' -i  
'https://localhost:7216/api/CinemaApi/9/GetCinema'
```

Commande CURL permettant de voir tous les cinémas suivant la ville, ici celle de Paris :

```
curl -X GET -k -H 'Content-Type: application/json' -i  
'https://localhost:7216/api/CinemaApi/Paris/GetCinemas'
```

Commande CURL permettant d'ajouter un cinéma :

```
curl -X POST -k -H 'Content-Type: application/json' -i https://localhost:7216/api/CinemaApi/ --data '{  
  "Nom" : "Cinema So Ouest",  
  "Adresse": "108 rue Rivay",  
  "CodePostal" : "92300",  
  "Ville": "Levallois-Perret",  
  "Responsable": "Rebecca Grenet",  
  "PrixPlace": "7"  
}'
```

Commande CURL permettant de modifier un cinéma, ici celui étant associé à l'identifiant 14

:

```
curl -X PUT -k -H 'Content-Type: application/json' -i https://localhost:7216/api/CinemaApi/14 --data '{  
  "Id": "4",  
  "Nom" : "Madeleine",  
  "Adresse": "36 avenue du Maréchal Foch",  
  "CodePostal" : "13000",  
  "Ville": "Marseille",  
  "Responsable": "Rebecca Grenet",  
}
```

```
"PrixPlace": "7"  
}'
```

Commande CURL permettant de supprimer un cinéma, ici celui étant associé à l'identifiant unique 10 :

```
curl -X DELETE -k -H 'Content-Type: application/json' -i 'https://localhost:7216/api/CinemaApi/10'
```

- Film :

Commande CURL permettant de voir tous les films :

```
curl -X GET -k -H 'Content-Type: application/json' -i 'https://localhost:7216/api/FilmApi'
```

Commande CURL permettant de voir un film suivant son identifiant unique, ici 2 :

```
curl -X GET -k -H 'Content-Type: application/json' -i 'https://localhost:7216/api/FilmApi/2'
```

Commande CURL permettant d'ajouter un film :

```
curl -X POST -k -H 'Content-Type: application/json' -i https://localhost:7216/api/FilmApi/ --data '{  
  "Nom" : "Le château ambulant",  
  "Realisateur": "Hayao Miyazaki",  
  "Resume": "Jadis protégée par des animaux géants, la forêt se dépeuple à cause de l'homme.  
Blessé par un sanglier rendu fou par les démons, le jeune guerrier Ashitaka quitte les siens et part à  
la recherche du dieu-cerf qui seul pourra défaire le sortilège qui lui gangrène le bras.",  
  "Genre": "Drame",  
  "Date": "2000-01-12",  
  "Duree": "133"  
}'
```

Commande CURL permettant de modifier un film, ici possédant l'identifiant 7 :

```
curl -X PUT -k -H 'Content-Type: application/json' -i https://localhost:7216/api/FilmApi/7 --data '{  
  "Id" : "7",  
  "Nom": "Princesse Mononoké",  
  "Realisateur": "Hayao Miyazaki",
```

```
"Resume": "Jadis protégée par des animaux géants, la forêt se dépeuple à cause de l'homme. Blessé par un sanglier rendu fou par les démons, le jeune guerrier Ashitaka quitte les siens et part à la recherche du dieu-cerf qui seul pourra défaire le sortilège qui lui gangrène le bras.",  
"Genre": "Drame",  
"Date": "2000-01-12",  
"Duree": "133"  
'
```

Commande CURL permettant de supprimer un film, ici celui étant associé à l'identifiant 7 :

```
curl -X DELETE -k -H 'Content-Type: application/json' -i 'https://localhost:7216/api/FilmApi/7'
```

- Séance :

Commande CURL permettant de voir toutes les séances :

```
curl -X GET -k -H 'Content-Type: application/json' -i 'https://localhost:7216/api/SeanceApi'
```

Commande CURL permettant de voir une salle suivant son identifiant unique, ici 2 :

```
curl -X GET -k -H 'Content-Type: application/json' -i  
'https://localhost:7216/api/SeanceApi/GetSeance/2 '
```

Commande CURL permettant de voir tous les films que proposent un cinéma suivant l'identifiant unique du cinéma , ici 9 :

```
curl -X GET -k -H 'Content-Type: application/json' -i  
'https://localhost:7216/api/SeanceApi/GetFilms/9 '
```

Commande CURL permettant de voir toutes les séances pour un film suivant l'identifiant unique du film, ici 6 :

```
curl -X GET -k -H 'Content-Type: application/json' -i  
'https://localhost:7216/api/SeanceApi/GetSeancesFilm/6'
```

Commande CURL permettant d'ajouter une séance:

```
curl -X POST -k -H 'Content-Type: application/json' -i https://localhost:7216/api/SeanceApi/ --data '{  
  "Cinemald": "1",  
  "Salleld": "1",  
  "Duree": "133",  
  "Date": "2000-01-12",  
  "Genre": "Drame",  
  "Resume": "Jadis protégée par des animaux géants, la forêt se dépeuple à cause de l'homme. Blessé par un sanglier rendu fou par les démons, le jeune guerrier Ashitaka quitte les siens et part à la recherche du dieu-cerf qui seul pourra défaire le sortilège qui lui gangrène le bras." }'
```

```
"FilmId": "5"  
'
```

Commande CURL permettant de modifier la séance suivant son identifiant unique, ici 2 :

```
curl -X PUT -k -H 'Content-Type: application/json' -i https://localhost:7216/api/SeanceApi/2 --data '{  
  "Id" : "2",  
  "CinemaId": "1",  
  "SalleId": "1",  
  "FilmId": "4"  
'
```

Commande CURL permettant de supprimer une séance suivant son identifiant unique, ici 8 :

```
curl -X DELETE -k -H 'Content-Type: application/json' -i https://localhost:7216/api/SeanceApi/8
```

- Salle :

Commande CURL permettant de voir toutes les salles :

```
curl -X GET -k -H 'Content-Type: application/json' -i 'https://localhost:7216/api/SalleApi'
```

Commande CURL permettant de voir une salle suivant son identifiant unique, ici 3 :

```
curl -X GET -k -H 'Content-Type: application/json' -i 'https://localhost:7216/api/SalleApi/3 '
```

Commande CURL permettant d'ajouter une salle :

```
curl -X POST -k -H 'Content-Type: application/json' -i https://localhost:7216/api/SalleApi/ --data '{  
  "CinemaId": "1",  
  "NbPlace": "7",  
  "NumeroSalle": "1"  
'
```

Commande CURL permettant de modifier une salle suivant son identifiant unique, ici 7:

```
curl -X PUT -k -H 'Content-Type: application/json' -i https://localhost:7216/api/SalleApi/7 --data '{  
  "Id" : "7",
```

```
"CinemaId": "1",  
"NbPlace": "7",  
"NumeroSalle": "1"  
}'
```

Commande CURL permettant de supprimer une salle suivant son identifiant unique, ici 5 :

```
curl -X DELETE -k -H 'Content-Type: application/json' -i https://localhost:7216/api/SalleApi/5
```

VI. Organisation de l'équipe, répartition des tâches dans le groupe et planning.

Dans un premier temps, nous avons commencé par choisir le contexte de notre application web, puis s'est suivi la réalisation d'un diagramme de classe afin de se représenter notre base de données, et par la suite les différentes requêtes que nous souhaitons faire.

La réalisation de l'application web s'est effectuée à l'aide de l'outil de gestion de versions Git et son code a été partagé sur la plateforme GitHub. Cela nous a permis de travailler en parallèle sur des fonctionnalités différentes, à l'aide de la création de branche. Nous avons souhaité respecter les conventions de nommage des branches, avec la branche de développement develop, et les branches feature pour les nouvelles fonctionnalités, qui dérivent de develop. Ainsi, une fois que notre implémentation était fonctionnelle, nous fusionnons nos branches sur la branche develop, qui représente la branche principale avec les derniers changements fonctionnels. Une fois que le projet était terminé, et donc prêt à être livré, nous avons alors fusionné la branche develop et master.

En parallèle de GitHub, nous avons un document Word qui reprenait les fonctionnalités à implémenter avec pour chacune d'entre elles, un ordre de priorité : élevé, moyenne, faible. Nous avons ainsi commencé par réaliser les fonctionnalités les plus

importantes, tout en faisant attention à répartir les tâches au préalable pour ne pas effectuer le travail en double.

VII. Bilan et perspectives sur le projet.

Le projet de génie logiciel nous a permis de comprendre les avantages de l'architecture MVC, mais aussi de mettre en pratique la gestion de projet avec notamment la répartition des tâches et le respect du délai demandé. De plus, cela nous a permis de découvrir le framework Dotnet MVC pour la réalisation d'applications web. Nous avons appris à rechercher activement sur la documentation mise à disposition, notamment pour le routage de nos méthodes pour les API. Enfin, la plateforme GitHub nous a permis de comprendre l'importance de créer des branches individuelles afin de pouvoir travailler sans avoir de conflit.

Par ailleurs, nous pouvons remarquer qu'il n'existe pas de distinction entre les visiteurs de l'application web et l'administrateur. En effet, toute personne a la possibilité de créer un film, comme une salle de cinéma par exemple. Cela amène donc à des problèmes de gestion des cinémas. Une piste d'amélioration serait donc de créer une interface utilisateur différente de l'interface administrateur, ce qui nous permettrait de bien séparer les informations auxquelles auraient accès le public, et les informations et modifications dont aurait accès l'administrateur.

Concernant les perspectives de ce projet, la réalisation de notre API au cours de ce semestre sera utilisée dans le cadre de la réalisation d'une application mobile. C'est pourquoi, nous avons pensé à cela pour chacune de nos requêtes afin d'avoir des informations exploitables lors du futur développement de l'application.