École Nationale Supérieure de Cognitique

Projet de Génie Logiciel

Création de la partie back-end de l'application Baroudeur

Élèves :

LARAN Baptiste

LEGER Corentin

Professeur :

Sébastien Bertrand

Année universitaire 2021 - 2022

Sommaire

Introduction

1. Contexte

Malgré une crise pandémique globale, chaque année des millions de touristes arpentent les rues des villes de toute la planète. Le tourisme est en effet une activité clé pour une grande partie de la population et permet à petits et grands de découvrir d'innombrables lieux tout autour du globe. Cependant, il devient parfois compliqué de profiter pleinement de cette expérience de découverte pour certaines personnes, notamment dans des villes, où on est souvent presque uniquement guidé vers les points d'intérêt les plus importants et où on peut déjà savoir ce qu'on va y trouver à l'avance !

Un groupe Transpromotion d'étudiants de l'ENSC s'est donc porté sur le design et la création d'une application d'une application de tourisme permettant de ramener une part de mystère dans l'exploration de villes tout en promouvant la culture et l'histoire : *Baroudeur*. Le but de cette application est de redéfinir la découverte de villes grâce à des concepts innovants.

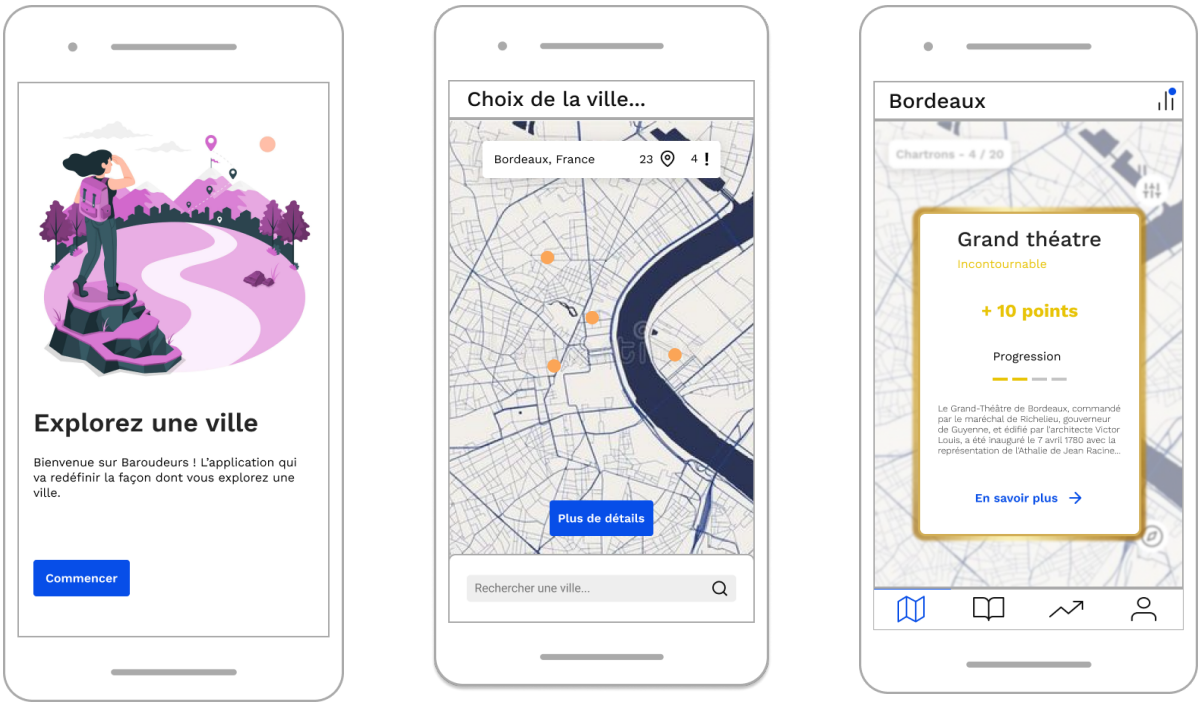


Figure 1 : Maquettes Figma de l'application *Baroudeur*

Dans le cadre du module de Génie Logiciel, l'objectif de ce projet est donc de créer la partie back-end de l'application mobile *Baroudeur* par la mise en place de différentes méthodes et pratiques.

Baroudeur étant une application de tourisme, il a donc été nécessaire d'intégrer dans la base de données des informations sur l'utilisateur lui-même, mais aussi d'autres sur les aspects géo-touristiques de villes tels que des points d'intérêt, des quêtes ... et les informations qui y sont associées.

Le but du projet n'est cependant pas d'implémenter une base de données exhaustive et des fonctionnalités complexes directement utilisables. Celles-ci sont assez basiques et le cœur du travail se trouve dans la méthodologie de conception. Il y a donc de nombreuses informations que nous avons volontairement laissées de côté par rapport à celles dont nous aurions besoin afin de respecter totalement la démarche UX réalisée en amont.

2. Exigences métier

La liste d'exigences métier de l'application fournie en début de projet est exposée ci-dessous.

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

3. Exigences techniques

La liste d'exigences techniques de l'application fournie en début de projet est exposée ci-dessous.

Code	Description	Validation
ET_01	L'application est réalisée à l'aide de la technologie ASP.NET Core MVC.	Oui
ET_02	Les données persistantes sont stockées dans une base de données relationnelle SQLite.	Oui
ET_03	L'interface entre les classes métier et la SGBDR exploite l'outil Entity Framework Core.	Oui
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.	Oui
ET_05	L'ensemble du code source respecte la convention camelCase.	Oui
ET_06	Les noms des classes, propriétés, méthodes, paramètres et variables sont choisis avec soin pour refléter leur rôle.	Oui
ET_07	L'application dispose de tests automatisés de type "smoke	Oui

	tests” pour vérifier le comportement des contrôleurs	

Organisation

1. Gestion du git et répartitions des tâches

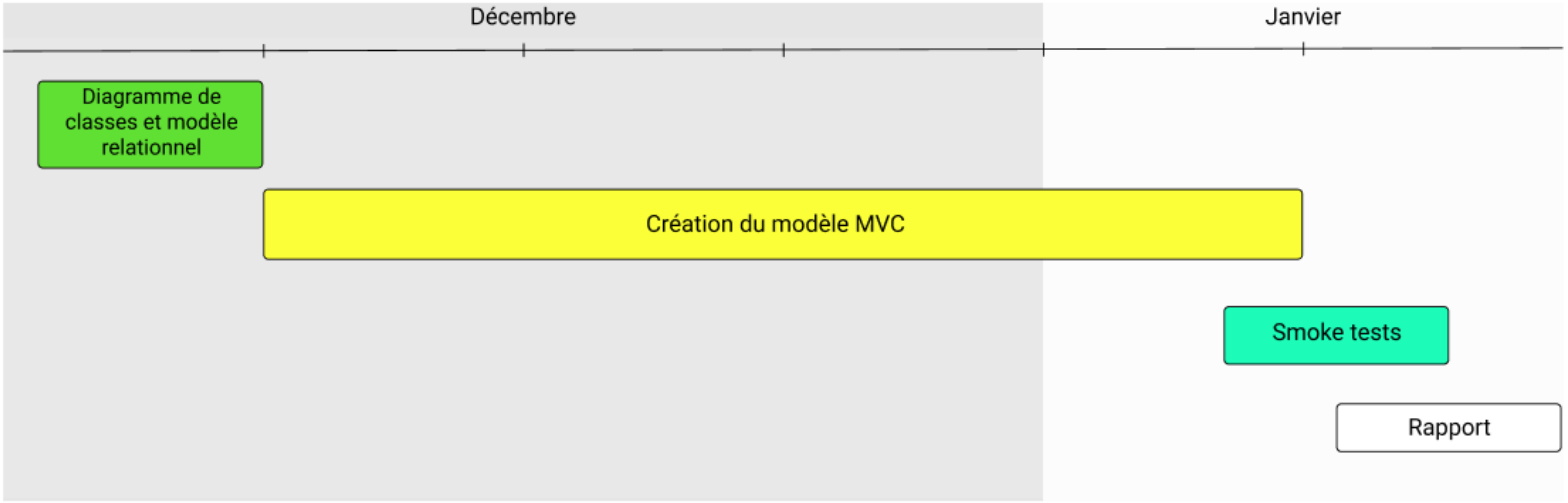
Que ce soit en cours ou en confinés à l’étranger, nous avons eu la chance de réaliser la quasi-intégralité du projet à deux en présentiel, nous permettant d’échanger nos idées et de répartir le travail très facilement. Nous avons donc dans une écrasante majorité du temps codé sur un seul ordinateur pour éviter les conflits via Github que nous n’aurions pas eu le temps de régler.

Répartition des tâches		
Tâche	**Baptiste**	**Corentin**
Diagramme des classes & modèle relationnel	**x**	**x**
Création de l’application		**x**
Création de Models		**x**
Initialisation de la base de données	**x**	
Affichage des Points Of Interest	**x**	
Mise à jour de la base de données		**x**
Affichage des Discoveries	**x**	
Affichage des Users		**x**
Ajustements divers	**x**	**x**
Smoke tests	**x**	**x**

API	x	
Rédaction du rapport	x	x

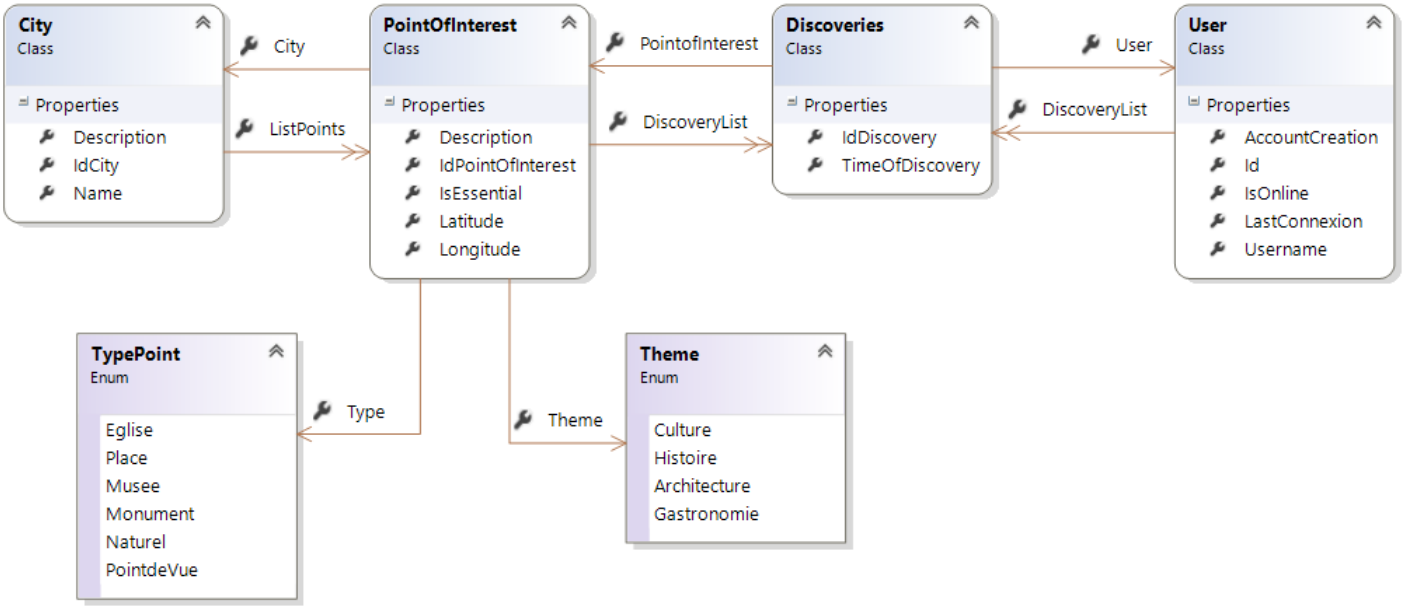
2. Planning

Voici ci-dessous le planning que nous avons suivi afin de réaliser le projet. Les étapes étant regroupées dans Création du modèle MVC sont celles marquées en jaune dans la la répartition des tâches.



Modélisation des données

Diagramme des classes métier et modèle relationnel



Documentation de l’API

L’API implémente l'accès aux modèles PointsOfInterest, User et Discoveries.

Exemple de modèle de PointOfInterest - api/APIPoint

```
{
    "id": 1,
    "name": "Grand Theatre",
    "description": "Théâtre majestueux dont la façade date de 1780, accueillant des opéras, concerts et spectacles de danse.",
    "isEssential": true,
    "theme": 1,
    "pointType": 4,
    "latitude": "44.84313851912286",
    "longitude": "-0.5711948257750703",
    "city": null,
    "cityId": 1,
    "discoveries": null
},
```

API	**Description**	**Corps de la demande**	**Réponse** (format JSON)
GET api/APIPoint/	Obtenir tous les points	None	Tableau des points
GET api/APIPoint/{id}	Obtenir un point par ID	None	Point
POST api/APIPoint/{id}	Ajouter un nouveau point	Point	Point
PUT api/APIPoint/{id}	Mettre un jour un élément existant	Point	None
DELETE api/APIPoint/{id}	Supprimer un point	None	None

Nous pouvons retrouver les mêmes actions avec les adresses des Users et Discoveries :

api/APIUser/

```
{
    "id": 1,
    "username": "blaran",
    "accountCreation": "2021-01-01T00:00:00",
    "lastConnection": "2022-01-12T00:00:00",
    "isOnline": true,
    "discoveries": null
  },
```


---

api/APIDiscovery/

```
{
    "id": 1,
    "userId": 1,
    "user": null,
    "pointId": 1,
    "point": null,
    "timeOfDiscovery": "2021-01-01T07:22:16"
  },
```

Smoke Tests

Des smokes tests ont été réalisés sur l'ensemble des contrôleurs de vue, ainsi que les contrôleurs d'API de l'application.

Smoke tests des controllers

Nous avons premièrement testé les controllers, pour cela nous avons réalisé un test sur les pages html City, Discovery, Home, Point et User, et sur certaines des pages qui y étaient associées.

```
// Tests pour les Controllers
        [Theory]

        [InlineData("/City")]
        [InlineData("/City/Create")]
        [InlineData("/Discovery/Index")]
        [InlineData("/Home/Index")]
        [InlineData("/Point")]
        [InlineData("/Point/Details/1")]
        [InlineData("/User")]
        [InlineData("/User/Delete/2")]
```

Smoke tests des API

On teste ici les API pour User, Point et Discovery.

```
// Tests pour les API
         [Theory]
         [InlineData("/api/APIUser")]
         [InlineData("/api/APIPoint")]
         [InlineData("/api/APIDiscovery")]
```

Dans la partie Assert du test, on modifie juste le “textl/html” par “application/json” car le format des fichiers que l’on teste est ici du json.

```
            // Assert
              response.EnsureSuccessStatusCode(); // Status Code 200-299
              Assert.Equal("application/json; charset=utf-8",
                  response.Content.Headers.ContentType.ToString());
```

Partie Assert pour le test des API

```
              // Assert
              response.EnsureSuccessStatusCode(); // Status Code 200-299
              Assert.Equal("text/html; charset=utf-8",
                  response.Content.Headers.ContentType.ToString());
```

Partie Assert pour le test des Controllers

On a au final les 11 tests qui sont des réussites :

```
Réussi!  - échec :     0, réussite :    11, ignorée(s) :     0, total :    11, durée : 875 ms -
n/Debug/net6.0/baroudeursTests.dll (net6.0)
baptistelaran@MacBook-Air-de-Baptiste baroudeursTests %
```

Bilan et perspectives sur le projet

Pour conclure, ce projet a permis d'améliorer nos compétences sur le langage C# et sur ASP.NET Core. Des notions abordées en cours et en travaux dirigés ont pu être utilisées à nouveau afin de réaliser un ce travail, qui s'approche de ce que nous pourrions avoir à faire dans nos futurs métiers.

Pour améliorer le projet, on aurait pu modifier les classes PointType et Theme, en effet on a ici seulement des énumérable, qui nous permettent d'associer une valeur et un type à chaque point d'intérêt, mais pas d'avoir accès à tous les points d'un même type ou d'un même thème. Pour cela on aurait dû créer une ICollection de User dans les deux classes, ce qui nous aurait par exemple permis de faire des recherches filtrées, ou bien de faire des statistiques.

Une partie de l'application n'a pas fonctionné comme prévue. En effet nous souhaitions pour chaque User et PointOfInterest afficher les Discoveries qui leur étaient associés. Cela n'a pas marché pour une raison que nous ignorons à ce jour.

Enfin, une base de données avait déjà été commencée par le pôle technique du projet transpromo Baroudeur, dont nous ne faisions pas partie, codée en React Native afin de pouvoir développer l'application sur iOS et Android. Nous n'avons pas repris exactement le même modèle car le but n'était pas d'obtenir une base de données fonctionnelle (on a par exemple ici un nombre de points d'intérêt très limité, et la plupart sont uniquement des points célèbres par soucis de simplicité pour trouver des informations à leur sujet) mais surtout de pouvoir aborder des aspects d'UX et de développement d'un même projet, afin d'avoir une meilleure vision du déroulement d'un projet à partir d'une simple idée.

Nous avons par ailleurs comme potentiel projet de continuer le développement de Baroudeur en parallèle avec notre formation. Dans ce sens, ce projet a donc été une vraie opportunité pour apprendre et mieux nous préparer aux difficultés que nous pouvons rencontrer sur notre chemin !