| **Projet 12**  Bilan du projet  Version 1.0 |
| --- |
| **Auteur**  Charlotte Vanhuyse |

**Table des matières**

[**Versions**](#_heading=h.p66re9gr1ndj) **4**

[**Introduction**](#_heading=h.5ipo7veyv8b7) **5**

[Objet du document](#_heading=h.1sjalts83qyz) 5

[Références](#_heading=h.9xy84xmakbz) 5

**Choix d’architecture 6**

Problématique 6

[S](#_heading=h.fiq7b7pbdn7v)olutions 6

Solutions envisagées 6

Monolitiques 6

Microservices 6

Choix de l’architecture 6

Complications rencontrées 7

Base de données 7

Tables d’association 7

**Choix des outils de développement 8**

Frameworks déjà utilisés 8

Outil déprécié 8

Problématique 8

API Gateway 8

Nouvelles technologies 9

Problématique 9

Angula[r](#_heading=h.pm1whizddoga) 9

[Auth](#_heading=h.mi3t1xb635d5)0 9

Mapbox 9

Solution 9

[Auth](#_heading=h.mi3t1xb635d5)0 10

Angula[r](#_heading=h.pm1whizddoga) 10

**Communiquer avec une API externe 11**

Problématique 11

Solution 11

Backend 11

requête venant du frontend 11

requête partant du backend 11

Frontend 12

Général 12

Mapbox 12

Implémentation 12

Intégration 12

**Sécuriser mon application 13**

Problématique 13

Solutions 13

Implémentation 13

Unicité 13

Exposition dans les URL 13

**Tester l’application 14**

Tests unitaires 14

Tests d’intégration 14

**Conclusion 15**

**rsions**

| **Auteur** | **Date** | **Description** | **Version** |
| --- | --- | --- | --- |
| Charlotte Vanhuyse | 14/03/2022 | Création du document | 1.0 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. **Introduction**
   1. **Objet du document**

Ce document a pour but de revenir sur mon expérience durant la réalisation du projet 12 de la formation. Je vais revenir sur les problématiques rencontrées tout au long de l’implémentation de l’application, en précisant les obstacles et les solutions mises en place pour les résoudre.

* 1. **Références**

Pour de plus amples informations, se référer :

1. **Projet 12 - Note d’intention** : Note d’intention expliquant les objectifs du projet.
2. **Projet 12 - Dossier de conception fonctionnelle** : Dossier de conception fonctionnelle de l'application.
3. **Projet 12 - Dossier de conception technique** : Dossier de conception technique de l'application.
4. **Choix d’architecture**
   1. **Problématique**

La première problématique rencontrée lors de ce projet était de décider quel type d’architecture serait la plus adaptée à mon projet.

* 1. **Solutions**
     1. ***Solutions envisagées***
        1. Monolithique

L’architecture monolithique consiste en une seule unité composée d’interface utilisateur, de la logique métier et d’une interface de données qui communique avec une base de données.

Ce choix a été envisagé car cette architecture est plus facile à implémenter et à déployer et permet d’avoir une base de données commune pour tout le projet.

D’un autre côté, l’application sera plus compliquée à maintenir dû à l'interdépendance des différents composants et je ne pourrai pas faire le choix d”une techno différente pour la partie “interface utilisateur”.

* + - 1. Microservices

Les microservices consistent en des services indépendants les uns des autres. Chaque composant est un service autonome pouvant être déployé séparément.

Les microservices peuvent évoluer ou changer de technologies indépendamment. Les micro-services facilitent l’adaptation du système à la montée en charge.

* + 1. ***Choix de l’architecture***

J’ai choisi une architecture en microservice. Plus complexe à mettre en place, mais plus facile à maintenir sur le long terme, plus adaptée aussi si je souhaite faire évoluer mon projet dans le futur. Elle me permet aussi de pouvoir envisager d’autres technologies que “Java” pour mon front.

Pour avoir plus de précision vous pouvez vous référer au dossier de conception fonctionnelle et au dossier de conception technique.

* 1. **Complications rencontrées**
     1. ***La base de données***

Suite à ce choix, J’ai souhaité mettre en place une gestion de données indépendante pour chaque microservice mais mes tables sont trop dépendantes les unes des autres ce qui m’obligeait à doubler énormément de code.

J’ai donc fait évoluer mon architecture vers une solution hybride, une architecture basée sur les microservices mais avec une base de données partagée.

* + 1. ***Les tables d’association***

Le fait de travailler avec des tables d’associations a apporté des problèmes dans mon application.

Tout d’abord les requêtes engendraient des boucles infinies dûes aux attributs dû au relation “ManyToMany” et “OneToMany” entre mes tables.

Comme solution, j’ai utilisé des objets DTO (Data Transfert Object) pour mapper les données reçues lors des requêtes.

On trouvera aussi deux types de requêtes dans mon application: des requêtes sur les tables principales ou des requêtes directement sur les tables d’associations selon le besoin.

1. **Choix des outils de développement**

Après avoir déterminé l’architecture de l’application, j’ai choisi les outils qui me paraissaient le plus adaptés au projet.

* 1. **Frameworks déjà utilisés dans le passé**

Au cours de ma formation j’ai déjà pu utiliser plusieurs technologies et j’ai fait le choix d’en utiliser certaines dans ce projet également.

L’application utilise donc :

- Spring Boot pour le backend codé en langage java,

- Maven pour le packaging

- PostgreSQL pour la base de données

- Hibernate/JPA pour l’ORM

- Junit/Mockito pour les tests unitaires et Postman pour les test d’intégration

- Spring Cloud pour m’aider à configurer l’architecture en microservice et faire communiquer les différents service entre eux ( Eureka, Ribbon, Config, Gateway)

- Spring security

* 1. **Outil déprécié**
     1. ***problématique***

Lors de la mise en place du projet, il s’est avéré que certains outils de Spring Cloud étaient dépréciés, notamment celui choisi pour l’api gateway de mon application.

* + 1. ***api gateway***

J’ai fait le choix de changer la technologie qui gère mon microservice “api gateway” par rapport à celle prévue au départ pour être à jour avec l’évolution de Spring Cloud.

Sur mes précédents projets, j’utilisais Zuul mais celui-ci est déprécié et Spring Cloud recommande d’utiliser Gateway, j’ai pu faire évoluer ce service sans avoir à modifier le reste de l’application.

* 1. **Nouvelles technologies**
     1. ***problématique***

Pour d’autres aspects de l’application, j’ai souhaité utiliser des outils que je n’avais jamais utilisés auparavant mais qui correspondaient aux attentes pour mon projet.

La problématique qui s’est imposée à moi fut de déterminer comment appréhender ces différents outils en même temps au sein de ce projet.

* + 1. ***Services choisis***
       1. Angular

Pour mon front, je voulais utiliser “Angular” :

Utilisé Typescript pour le front est plus pertinent que Java, le code s'exécute côté client. Maîtriser “Angular” est aussi un plus pour mon futur si je veux devenir un développeur fullstack.

* + - 1. Auth0

Pour ce projet, j’ai souhaité externaliser le service d’authentification et utiliser Auth0 qui implémente le protocole Oauth2.

Auth0 va gérer l’inscription et la gestion des permissions des utilisateurs. Auth0 possède sa propre base de données “utilisateurs”.

Un utilisateur aura donc un identifiant Auth0 et un identifiant pour mon application.

* + - 1. Mapbox

Pour afficher une carte interactive j’ai choisi Mapbox.

* + 1. ***solutions***

Pour pouvoir être efficace dans la découverte de cet outil, il m’a paru plus logique d'en implémenter un à la fois.

* + - 1. Auth0

Sur un projet précédent, j’ai implémenté moi même un service d’authentification qui suivait le protocole Oauth2. J’ai donc décidé de commencer par Auth0.

* + - * 1. backend

Le problème principal rencontré sur le back de l’application avec Auth0 était de savoir s' il était préférable d’implémenter le flow d'autorisation et d’authentification sur l’api gateway ou directement dans chaque microservice qui expose une api rest.

J’ai fait le choix de le faire dans les microservice directement pour éviter des possibles failles de sécurité et aussi pour régler des problèmes de conflits de cross-origin implémenter dès deux côtés.

* + - * 1. frontend

La problématique avec la partie front de l’application était de réussir à maîtriser deux nouveaux outils en même temps.

Pour faciliter mon travail et la résolution de différents bugs qui pourraient apparaître.

J’ai fait le choix de commencer par implémenter un front avec spring boot que je maîtrise et ainsi pouvoir me concentrer sur le fonctionnement d’Auth0 et son interaction avec mon front. Une fois réussi je suis passé à l’implémentation de mon front à l’aide d’Angular.

* + - 1. Angular

Pour apprendre à utiliser Angular et TypeScript, j’ai suivi les cours proposés par Openclassrooms. Ces cours m’ont permis d’acquérir de bonnes bases pour implémenter le front que je souhaitais pour mon application.

J’ai ensuite pu reprendre l’implémentation de mon premier front en Java et m’appuyer dessus pour faire fonctionner ce nouveau front avec Auth0.

1. **Communique avec une API externe**
   1. **Problématique**

Un autre challenge que je me suis donné pour ce projet était d’utiliser des API externes et donc de les faire communiquer avec mon application.

Il y en a deux nécessaires à mon application :

* Auth0 qui gère l’authentification
* Mapbox qui s’occupe de la mise en place de la carte interactive
  1. **Solution**
     1. ***backend***

Le backend interagit avec Auth0 de deux manières . Elle reçoit des requêtes qui sont accompagnées d’un token communiqué par Auth0 et elle envoie des requêtes pour pouvoir accéder ou modifier la base de données “utilisateur” de Auth0.

* + - 1. Requête venant du frontend

Lorsqu’une requête arrive sur la backend, le “bearer token” présent (le cas échéant) dans le header est analysé. Cette partie à été implémentée dans le package “config” en suivant la doc proposée par Auth0.

* + - 1. requête partant de backend

Pour pouvoir envoyer des requêtes aux endpoints de Auth0, j’ai utilisé RestTemplate.

A chaque requête le bearer token du service, préalablement obtenu, est ajouté au header. Si nécessaire, un objet json est ajouté au body de la requête.

Le token obtenu en amont suit les règles du flow ‘’Client credential’’ qui est configuré dans Auth0. Dans Auth0, les permissions correspondant à l’api management ont été ajoutées à l’api “microservice profil”.

* + 1. ***frontend***

La partie frontend communique avec Auth0, Mapbox et le backend de l’application.

* + - 1. Général

A l’aide de “@angular/common/http”, l’utilisateur peut envoyer des requêtes aux endpoints de ces API Rest. “@auth0/auth0-angular” permet de filtrer les url des requêtes et d’ajouter le token de l’utilisateur à chacune d’elles.

* + - 1. Mapbox
         1. Implémentation

En ce qui concerne Mapbox, j’ai rencontré des difficultés à mettre en place la carte interactive de mon projet car la bibliothèque de mapbox n’est pas adaptée à Angular.

J’ai donc commencé par essayer d’implémenter mon code à l’aide de la bibliothèque “ngx-mapbox-gl” présentée comme un wrapper Angular pour Mapbox.

Malheureusement je n’avais pas anticipé que cette bibliothèque n’était pas compatible avec ma version d’Angular (13) et j’ai rencontré des erreurs que je n’ai pas su résoudre.

Pour pallier à ce problème, j’ai finalement utilisé la bibliothèque de départ “mapbox-gl” et j’ai pu configurer ma carte ainsi que les pop-up et marqueurs nécessaires à mon projet.

* + - * 1. intégration

Lors de l’intégration avec le reste de l’application, j’ai eu besoin de retravailler le code backend des méthodes liées aux données sur les lieux représentés sur ma carte. Les données récupérées sont transformées en suivant le format GeoJson pour pouvoir être utilisées à l’initialisation de la carte.

1. **Sécuriser mon application**
   1. **problématique**

Le dernier point que j'aimerai aborder est la problématique de la sécurité.

Lors de la réalisation du projet, mon mentor a soulevé des failles de sécurité possibles au sein de mon application. Cela m’a donc obligé à réfléchir sur ce sujet.

* 1. **solutions**
     1. ***implémentation***

La première mesure que j’ai prise a été de retravailler le code de mon application pour que les méthodes qui manipulent des données soient majoritairement présentes dans le backend de l’application.

* + 1. ***unicité***

Pour ce qui concerne la base de données, j’ai choisi d’utiliser des Id au format UUID pour l’unicité de mes objets. Ce format est plus sécurisé que de simple id auto-incrémenté plus facilement prévisible.

* + 1. ***exposition dans les url***

Un autre point soulevé a été la présence d’id dans mes url ou body de mes requêtes.

Pour parer à cela, J’ai donc réécrit mes méthodes pour que, lorsque c’était possible, l’id de l’utilisateur soit récupéré directement dans le token présent dans le contexte de l’application plutôt que dans le body de mes requêtes.

Si l’id est toujours présent dans ma requête, il est comparé à l’id présent dans les claims du token.

1. **Tester l’application**

Lors de la formation un des points à travailler était la partie tests d’une application.

J’ai donc appliqué ce que j’ai appris dans mes projet précédents et j’ai utilisé la méthode de TDD pour tester le backend de mon projet.

* 1. **Tests Unitaires**

Les tests unitaires permettent de vérifier le bon fonctionnement de mon code à l’intérieur de mon microservice.

Pour implémenter les tests unitaires de mes microservices j’ai utilisé Junit et Mockito.

* 1. **Tests d’intégration**

Les tests d’intégration permettent d’analyser le fonctionnement global de l’application.

On vérifie que les microservices communiquent correctement entre eux mais aussi avec les composants externes.

J’ai choisi d’utiliser Postman pour implémenter mes test d’intégration.

J’ai rencontré des difficultés au moment de tester les requêtes nécessitant un token venant de Auth0. Lors de la demande de token à Auth0, la réponse comportait bien un token mais le payload était vide.

Il s’est avéré que la requête envoyée à Auth0 était incomplète car il manquait le paramètre “audience” pourtant configuré dans Postman. Il m’a suffit de l’ajouter directement à l’url qui appelle Auth0 pour résoudre ce problème.

1. **Conclusion**

Dans ce projet, j’ai pu utiliser les compétences acquises lors de la formation. Mais j’ai pu aussi apprendre à utiliser de nouveaux outils qui me seront utiles dans ma future carrière professionnelle. Cela m’a rappelé que le développement est un domaine en évolution permanente et qu’il est important de se tenir informé, de continuer à en apprendre sur les outils que j’utilise et de ne pas hésiter à essayer des technologies que je ne connais pas pour m’améliorer.