P_DEV

Concevoir une médiathèque audio/vidéo partagée avec un protocole P2P

Chef de projet : CARREL Xavier
VELICKOVIC Mateja
ÉCOLE DES MÉTIERS TECHNIQUES LAUSANNE

Table des matières

1 Analyse préliminaire

1.1 Introduction

Le projet BitRuisseau a pour but de consulter et de partager ses fichiers audio et vidéo sur une médiathèque (locale ou non) en utilisant un protocole défini avant le démarrage du projet, chaque utilisateur pourra se connecter à un topic (sujet) et pourra consulter les différentes médiathèques présentes dans la communauté.

1.2 Objectifs

Chaque utilisateur pourra :

- Demander sur le réseau quels sont les médiathèques opérationnelles.
- S'annoncer opérationnelle.
- Demander le catalogue à une médiathèque.
- Publier son catalogue.

Fonctionnalités avancées :

- Demander un fragment de média à une médiathèque.
- Fournir un fragment de média à une médiathèque.

1.3 Gestion de projet

Pour la gestion de ce projet, j'ai principalement utilisé l'outil IceScrum pour la création de tâches et l'écriture des tests les concernant, GitHub m'a permis de versionner de mon projet afin de garder un suivi de ce dernier et de noter le temps passé sur une tâche en la référençant dans le titre de chaque commit.

2 Analyse / Conception

2.1 Domaine

Les données manipulées par le programme concernent la plupart du temps des métadonnées de contenu multimédia ainsi que des informations sur les médiathèques qui sont partagées dans le programme.

Ces dernières représentent la structure et le contenu que les utilisateurs finaux s'échangent et consultent via un broker.

En ce qui concerne les échelles de temps, l'échange des médiathèques et des médias sont fait en temps réel.

Ces données sont destinées à des utilisateurs qui souhaitent partager ou consulter du contenu multimédia au sein d'une communauté ou même bien peut des passionnés qui souhaitent simplement partager différents médias.

2.2 Concepts

Protocole MSP 1.0

MEDIA **S**HARING **P**ROTOCOL

Version du protocole : 1.0

Protocole : MQTT

Topic : mediaplayer

Message type :

```
MEDIA_STATUS_REQUEST    // Transporte rien du tout

MEDIA_STATUS            // Transporte un objet de type Mediatheque
```

Class media et mediatheque :

```
class Media
{
    string Filname
    int Size
    enum Mediatype {
        MP3,
        MP4,
        MOV,
        GIF,
        PNG,
        JPEG,
        JPG,
        WAV,
    }
}

class Mediatheque
{
    bool IsAvailable          // Je suis disponible et non connecté.
    string DisplayName
    List<Media> Medias

}
```

La définition du protocole 1.0 a été faite en date du 13 novembre 2024 en début d'après-midi, cette dernière se basait principalement sur quel protocole nous allions travailler ainsi que les messages qui vont être envoyés sur le sujet (topic) prédéfini, qui dans notre cas est : mediaplayer.

Nous avons aussi défini différentes classes pour les médias et les médiathèques afin de tous avoir la même classe et de ne pas se retrouver avec des informations différentes.

En ce qui concerne le type de message nous en avons défini deux nouveau à ajouter, un premier pour demander le statut d'une médiathèque et le second qui va transporter un objet de type médiathèque qui va donc contenir toutes les informations de cette dernière.

Figure 1 : Définition d'un protocole

2.3 Analyse fonctionnelle

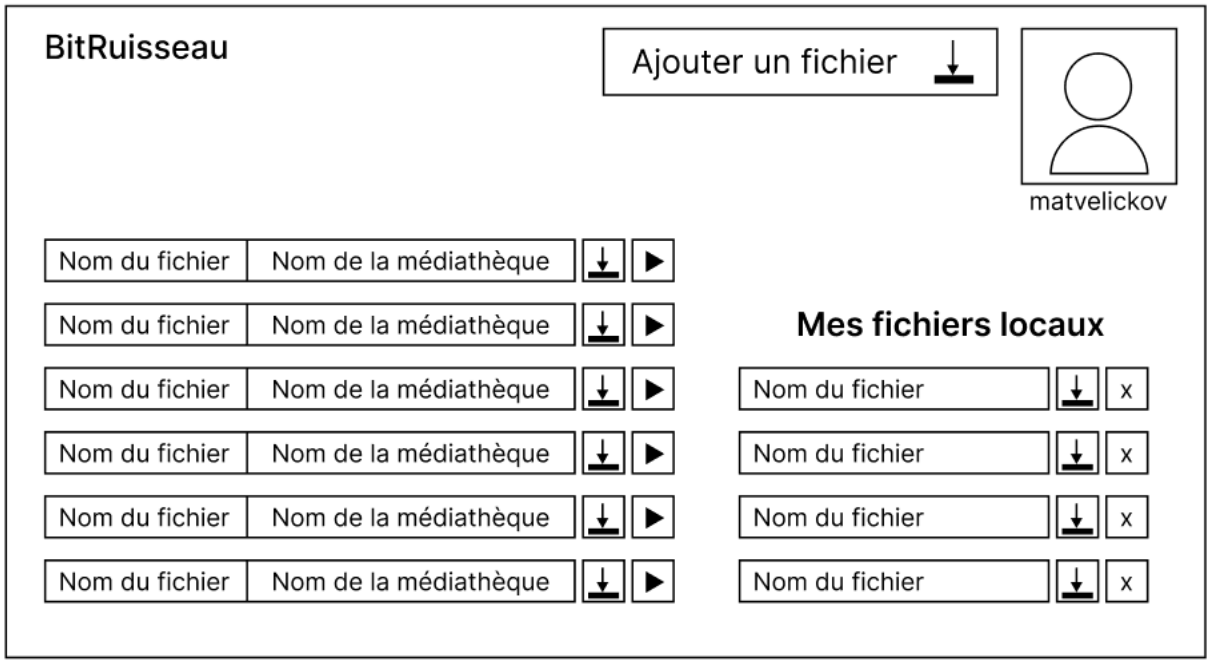

Figure 2 : Maquette Low Fidelity de l'application BitRuisseau

Voici la maquette Low Fidelity qui a été réalisée sur Figma en une dizaine de minutes, l'utilisateur peut voir les différents fichiers publiés sur la communauté ainsi que ses fichiers locaux que lui-même a ajouter via un bouton situé en haut à droit de son interface.

L'utilisateur peut aussi voir son profil avec son nom d'utilisateur en haut à droite de l'interface aussi.
(voir figure 2)

---

1. Pouvoir lister le contenu de la médiathèque localement

En tant qu'utilisateur, je veux pouvoir visionner les médias qui ont été chargés localement afin de les consulter.

Tests d'acceptance :

- **Étant donné** que je veuille lister les médias chargés
- **Lorsque** je suis sur la page d'accueil
- **Alors** je peux voir tous les médias qui ont été chargés.

---

2. Pouvoir visualiser la liste des contenus disponibles dans la communauté

En tant qu'utilisateur, je veux pouvoir visionner les médias disponibles dans la communauté afin de consulter les médias de différents utilisateurs.

Tests d'acceptance :

- **En tant qu'utilisateur** je veux lister les médias de la communauté
- **Lorsque** je suis sur la page de la communauté
- **Alors** les médias publiés sont affichés.

---

3. Pouvoir visualiser les médiathèques opérationnelles

En tant qu'utilisateur, je veux pouvoir visualiser les médiathèques opérationnelles afin de les consulter.

Tests d'acceptance :

- **En tant qu'utilisateur** que je désire consulter les médiathèques opérationnelles
- **Lorsque** j'arrive sur l'interface des médiathèques
- **Alors** les médiathèques opérationnelles s'affichent.

---

4. Pouvoir s'annoncer opérationnelle

En tant qu'utilisateur, je veux pouvoir m'annoncer opérationnelle auprès du broker afin de partager mon statut.

Tests d'acceptance :

- **En tant qu'utilisateur** que je désire m'annoncer opérationnelle
- **Lorsque** je définis mon statut comme *opérationnelle*
- **Alors** je transmets mon statut au broker.

---

5. Demander le catalogue à une médiathèque

En tant qu'utilisateur, je veux demander le catalogue d'une médiathèque afin de consulter son contenu.

Tests d'acceptance :

- **En tant qu'utilisateur** que je désire consulter le catalogue d'une médiathèque
- **Lorsque** je clique sur une médiathèque
- **Alors** son catalogue s'affiche.

6. Publier son catalogue à la vue des autres médiathèques

En tant qu'utilisateur, je veux pouvoir publier mon catalogue afin que les autres médiathèques puissent le consulter.

Tests d'acceptance :

- **En tant qu'utilisateur** que je désire partager mon catalogue
- **Lorsque** je clique sur **Partager**
- **Alors** mon catalogue est publié.

---

7. Pouvoir demander un fragment de média à une médiathèque

Tests d'acceptance :

- **En tant qu'utilisateur** que je désire demander un média
- **Lorsque** je demande ce média à une médiathèque
- **Alors** le média m'est envoyé sous forme de fragment.

---

8. Pouvoir transmettre un fragment de média à une médiathèque

Tests d'acceptance :

- **En tant qu'utilisateur** que je désire transmettre un fragment de mon média
- **Lorsque** je le transmets à une médiathèque
- **Alors** le fragment est envoyé.

---

9. Pouvoir ajouter/charger un média depuis un dossier local

En tant qu'utilisateur, je veux pouvoir charger un média localement afin de le partager.

Tests d'acceptance :

- **Sélectionner un média :**
    - **En tant qu'utilisateur** que je veuille sélectionner un média
    - **Lorsque** j'appuie sur *Charger un média*
    - **Alors** une interface s'ouvre pour la sélection.

- **Charger un média :**
  - **En tant qu'utilisateur** que je sois sur l'interface de sélection
  - **Lorsque** je sélectionne et confirme un média
  - **Alors** il est chargé dans l'application.

2.4 Stratégie de test

- Aucun test effectué

3 Réalisation

3.1 Points de design spécifiques

3.1.1 Conversion des images en tableaux de bytes

Pour chaque image qui a dû être envoyée au broker, j'ai créé une méthode pour que l'image soit convertie en tableau de bytes qui lui par la suite sera envoyé sur le broker et récupéré par les différents utilisateurs (voir figure 3).

```
/// <summary> Convert an image to a byte's array
2 références
public static byte[] ImageToByteArray(Image imageToByte)
{
    ImageConverter _imageConverter = new ImageConverter();
    byte[] bytes = (byte[])_imageConverter.ConvertTo(imageToByte, typeof(byte[]))!;
    return bytes;
}
```

Figure 3 : Méthode pour la conversion des images en tableau de bytes

3.1.2 Conversion des tableaux de bytes en images

Et en ce qui concerne la situation inverse, dans le cas ou c'est mon broker qui demande à recevoir des images, nous faisons simplement l'inverse en convertissant le tableau de bytes en Image (voir figure 4).

```
/// <summary> Convert a byte's array to image
1 référence
public Image ByteArrayToImage(byte[] bytes)
{
    MemoryStream ms = new MemoryStream(bytes, 0, bytes.Length);
    ms.Write(bytes, 0, bytes.Length);
    Image imageBack = Image.FromStream(ms, true);
    return imageBack;
}
```

Figure 4 : Méthode pour la conversion des tableaux de bytes en images

3.2 Déroulement

Le projet s'est bien déroulé, on s'est rendu compte avec mes camarades que si une classe était juste à peine différente que celle d'un autre camarade, tout plantait et le transfert de fichier ne fonctionnait pas, il fallait donc uniformiser tout cela et établir un protocole commun.

3.3 Mise en place de l'environnement de travail

Le projet est accessible sur GitHub dans un repository qui est privé, ce dernier contient la documentation.

- /doc : Contient la documentation du projet (rapport), les notes prises en cours ainsi que l'auto-évaluation de ce projet.

- /matvelickov-bitRuisseau : contient le code source du projet.

- /resources : contient les différentes images utilisées dans le README.md du projet

- .gitignore : exclu certains fichiers qui ne doivent pas être suivis par Git

- README.md : documentation en markdown

Description du matériel :

- Ordinateur de l'ETML avec Windows 10 installé
- Visual Studio 2022
- IceScrum
- Suite office pour la documentation et communication du projet

3.4 Description des tests effectués

- Retrouver dans le point 2.3

3.5 Bilan

3.5.1 Erreurs restantes

Il reste pour l'instant qu'une seule erreur sur le programme qui est assez dérangeante, cette dernière se produit lors de la construction du projet, des attributs dans l'AssemblyInfo du bitRuisseau ainsi que du Backend du projet rentrent en collisions les uns avec les autres car la plupart sont identiques, c'est donc un problème que je n'ai pas encore réussi à régler car ces fichiers sont regénérés automatiquement par la classe MSBuild.

3.5.2 Stories

Pour le travail que j'ai effectué et que le travail que j'aurais pensé effectuer, je trouve que la différence n'est pas énorme et je suis très satisfait du travail que j'ai pu produire ces 8 semaines passées, la transmission de vidéos ainsi que de fragments d'images n'est pas encore implémentée dans le projet mais cela ne change en rien le bon fonctionnement du programme avec un code propre, respectant les normes de l'ETML et avec différents commentaires qui pourraient aider dans le cas où le projet serait repris par une personne tierce.

3.5.3 Dette technique

En ce qui concerne la dette technique de ce projet, elle peut être retrouvée dans le code sous forme de commentaires ainsi que dans la liste des tâches dans le menu déroulant d'affichage (voir figure 5).

Description	Projet	Fichier	Ligne
TODO vérifier	Backend, bit-ruis...	Program.cs	15
TODO LWT	bit-ruisseau	Form1.cs	70
TODO Select the uploaded file on upload	bit-ruisseau	Form1.cs	149
TODO Display horizontaly the items	bit-ruisseau	Form1.cs	150
TODO Check if the file's sizes are identical	bit-ruisseau	Form1.cs	227
TODO Move the original file in the project folder bin/Debug/	bit-ruisseau	Form1.cs	228
TODO Check the id of the uploaded the media (duplicates files)	bit-ruisseau	Form1.cs	247

Figure 5 : Liste des tâches à fixer ou à réaliser

Elle a été établie au fur et à mesure de la réalisation du projet pour ne pas oublier certaines choses à réaliser ou à fixer, comme indique sur la figure ci-dessus, certaines tâches n'ont pas encore été corrigées mais elles le seront au plus vite afin de garantir un meilleur fonctionnement du programme.

3.6 Recours à l'intelligence artificielle

J'ai en effet utilisé l'IA dans ce projet lorsque les outils que j'avais à disposition pour m'aider ne me suffisaient pas ou ne m'aidaient pas assez, la plupart de l'utilisation de cette dernière s'est faite via l'extension de GitHub Copilot sur Visual Studio Code afin d'essayer de résoudre les problèmes que j'ai rencontrés ces trois dernières semaines comme je l'ai cité dans ma conclusion, ce qui finalement ne m'aura même pas aidé car j'ai tout de même dû recréer le projet et repartir à blanc.

Pour le reste du projet mon aide principale a été mes camardes ainsi que les recherches rares que j'effectuait sur internet pour des petits doutes ou de légers soucis rencontrés.

4 Conclusions

Pour conclure ce projet, je trouve le sujet vraiment très intéressant, au début de celui-ci je me suis surtout et malheureusement trop concentré sur l'aspect visuel ainsi que sur les fonctionnalités secondaires de l'application au lieu de me concentrer directement sur le sujet principal qui a été notre protocole et la communication entre les différents agents.
Ce qui m'a causé beaucoup de soucis que j'aurais pu corriger bien plus rapidement si j'avais commencé par la communication avec le broker, comme par exemple le fait que j'ai dû reconstruire tout le projet de A à Z et de tout recommencer à cause d'un problème de compatibilité avec le Frontend et le Backend, j'ai donc perdu 1.5 après-midi à régler ce problème alors que j'aurais pu continuer à travailler sur le projet, c'est donc le point sur lequel je m'améliorerais si ce projet serait à refaire.

Actuellement le projet est dans un bon état, il peut recevoir ainsi que transmettre des messages sur le topic mediaplayer, il peut échanger des médiathèques ainsi que des images en différents formats (pas encore testé mais bien avancé), les fonctionnalités telles que le chargement de fichiers localement ou bien le visionnage de ces derniers est terminé mais pourra évidemment être amélioré.

5 Annexes

5.1 Journal de travail

Voir documents dans la release.