

**Rapport**  
T.E.R Master Informatique

# L'ARBRE DU VIVANT



Réalisé par

---

Stefan STEFANOVSKI  
Basil DALIE  
Karil SQUALLI

---

Sous l'encadrement de M.  
**William PUECH**



Département Informatique de la Faculté des Sciences  
UNIVERSITÉ MONTPELLIER  
Bt. 16 - CC 12 - Place Eugène Bataillon  
34095 Montpellier cedex 05

*Nous tenons à remercier vivement notre encadrant de projet M. William Puech pour le temps passé ensemble et le partage de son expertise et l'aide précieuse qu'il nous a apporté nous ont permis de mener à bien notre projet.*

*Nous souhaitons également remercier Jacques Exertier pour ses brillantes introductions à l'application **L'arbre du vivant** et son aide tout au long du projet, ainsi que Vincent Lhuillier pour ses retours sur le travail réalisé et son soutien sur les aspects techniques du projets.*

*Nous remercions enfin Melanie Debiais-Thibaud pour avoir joué le rôle de l'utilisateur final.*

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Présentation de l'application</b>	<b>6</b>
2.1	Lancement . . . . .	6
2.2	Interface principale . . . . .	7
2.3	Organisation des images et des commentaires . . . . .	8
<b>3</b>	<b>Objectifs du projet</b>	<b>10</b>
3.1	Récupération des données à partir d'un serveur . . . . .	10
3.2	Interface web pour l'application . . . . .	10
<b>4</b>	<b>Travail réalisé</b>	<b>11</b>
4.1	Partie Serveur . . . . .	11
4.1.1	Base de donnée . . . . .	11
4.1.2	API web . . . . .	12
4.2	Partie Application C# . . . . .	14
4.2.1	Analyse du code . . . . .	14
4.2.2	Implémentation . . . . .	18
4.3	Partie Application Web . . . . .	20
4.3.1	Introduction . . . . .	20
4.3.2	Les différentes pages de l'application . . . . .	22
4.3.3	Compréhension de la structure du code . . . . .	25
<b>5</b>	<b>Bilan du projet</b>	<b>28</b>
5.1	Autocritique . . . . .	28
5.2	Difficulté . . . . .	28
5.3	Perspectives . . . . .	29
<b>6</b>	<b>Conclusion</b>	<b>30</b>
<b>A</b>	<b>Code source du projet</b>	<b>31</b>

<b>B</b>	<b>API Web</b>	<b>32</b>
B.1	Diagramme de flow . . . . .	33
B.2	Page Inscription . . . . .	34
B.3	Page Connexion . . . . .	34
B.4	Page Telechargement . . . . .	35
B.5	Page Mes Arbres . . . . .	35
*		

# Chapter 1

## Introduction

Dans le cadre du module TER inscrit dans le programme pédagogique du M1 AIGLE à la faculté des sciences de Montpellier. Il nous a été demandé de réaliser un projet en équipe sur une durée de 4 mois. Le sujet qui nous a été attribué a consisté à intégrer de nouvelles fonctionnalités à un logiciel existant nommé "L'arbre du vivant" développé par deux employés d'Ubisoft, et qui permet de visualiser, de concevoir et d'éditer un arbre représentant les différents taxons existant ou ayant existé.

Le but de notre projet a été d'adapter ce logiciel afin de le rendre utilisable par exemple pour une classe d'étudiants en biologie.

Nous avons ainsi réalisé une interface web permettant à chaque utilisateur d'obtenir l'exécutable, et soumettre ou récupérer des arbres. Nous avons également modifié le fonctionnement de l'application de manière à ce qu'il télécharge les différentes données (images, sons, commentaires, etc.) associées aux taxons depuis un serveur.

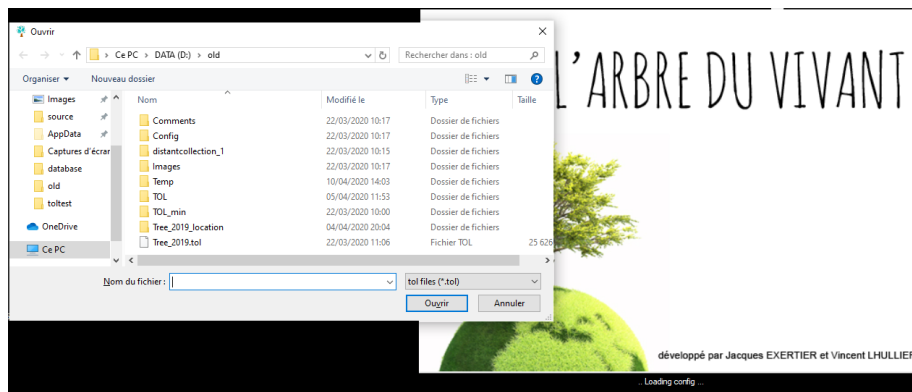
## Chapter 2

# Présentation de l'application

Dans les pages qui vont suivre, nous vous présenterons l'application "L'arbre du vivant" tel qu'elle était quand elle nous a été livrée.

### 2.1 Lancement

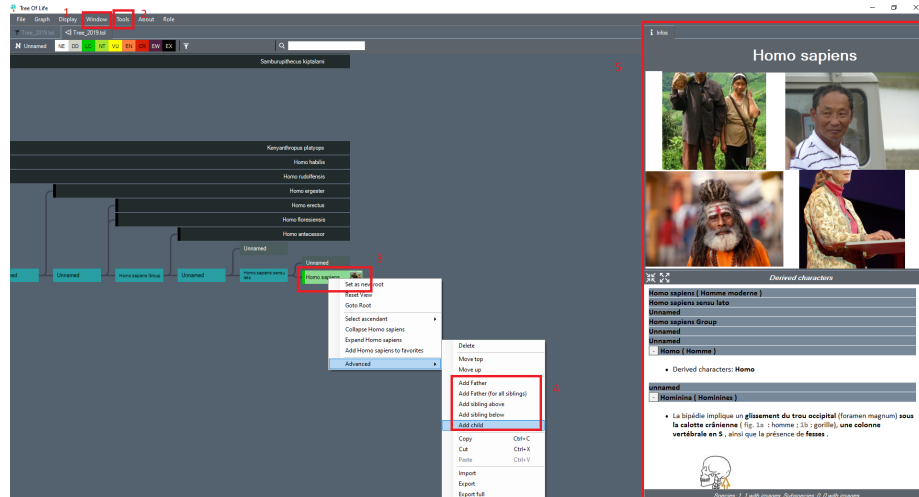
Au démarrage de l'application, une fenêtre d'attente est affichée, et une boîte de dialogue s'ouvre pour demander à l'utilisateur de charger un arbre au format .TOL (format binaire de représentation des arbres), ou alors au format XML. Une fois cette étape réalisée, l'application parcourt l'arbre chargé et initialise la structure de donnée permettant de stocker les taxons.



Ecran d'attente et chargement du fichier

## 2.2 Interface principale

Une fois le chargement terminé, on obtient une fenêtre composée d'un menu et du graphe principal de l'application. Sur ce graphe, on peut dérouler un noeud de l'arbre et le réduire à l'aide de la molette de la souris, on peut aussi sélectionner un taxon en cliquant sur le noeud correspondant. En effectuant un clic-droit sur un taxon, on peut appliquer un ensemble d'opérations sur le noeud correspondant de l'arbre(3), on peut notamment en faire la nouvelle racine de l'arbre ou encore lui ajouter un enfant, un parent intermédiaire ou un frère (4). Sur le menu, on a un onglet 'Window' (1) qui permet d'afficher et de cacher un ensemble de contrôles utilisateurs. Ces contrôles peuvent être placés sur l'écran dans 5 zones redimensionnables (haut, bas, droite, gauche, centre) afin de composer l'interface à partir des outils dont l'utilisateur a besoin pour sa tâche. ils comprennent notamment la vue d'information (5), un graphe additionnel, la fenêtre d'édition d'un taxon, un formulaire de recherche etc..



Interface principale de l'arbre du vivant

Sur la capture d'écran précédente, vous pouvez voir que l'on a fixé à droite la vue d'information de l'application, qui permet d'afficher les informations associées à un taxon. Celle-ci se divise en 2 parties : la première permet de naviguer à travers les images disponible pour le taxon sélectionné, et la seconde permet d'afficher la commentaire du taxon (la description du taxon) sélectionné ainsi que celles de ses noeuds ancêtres. Pour les animaux, on trouve également un bouton permettant de lancer la lecture de son cri dans la zone au dessus de la liste des images.

Sur le menu du haut, on a également un onglet 'Tools' qui permet d'accéder à un ensemble d'outils, comme la fenêtre d'affichage des collections d'images, ou encore, l'outil de mise à jour de l'arbre avec les images et les sons stockés sur le disque dur.

## 2.3 Organisation des images et des commentaires

Dans l'application "L'arbre du vivant", les images et les commentaires associés aux taxons doivent faire partie d'une "collection". La collection peut par exemple regrouper toutes les images issues d'un site web en particulier. Il est possible d'activer et désactiver certaines collections afin de n'afficher que les images ou commentaires d'une source en particulier.

Name	Number	Use it	Default
00_EE_Valfla...	3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
01_Arkive	85700	<input checked="" type="checkbox"/>	<input type="checkbox"/>
02_INPN	35350	<input checked="" type="checkbox"/>	<input type="checkbox"/>
03_HBW ois...	86627	<input checked="" type="checkbox"/>	<input type="checkbox"/>
04_Unofficial	3	<input checked="" type="checkbox"/>	<input type="checkbox"/>
05_Wikipedia	22484	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Ammonites	903	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ASSER_EXE...	1574	<input checked="" type="checkbox"/>	<input type="checkbox"/>
EOL_Animalia	499779	<input checked="" type="checkbox"/>	<input type="checkbox"/>
EOL_Plantas	516904	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Les collections d'images peuvent être de deux types différents : il y'a d'abord les collections locales dont les images sont présentes sur le disque dur, puis il y'a les collections de liens dont les images sont situées sur un serveur distant et accédées par une URL.



Dans le répertoire des données de l'application (qui correspond à la variable `TaxonPath`), les fichiers de descriptions des images et des commentaires sont classés dans les sous-répertoires 'Images/' et 'Comments/'. Chacun de ces répertoires contient un sous-répertoire par collection. Chaque répertoire associé à une collection d'image contient :

- Un fichier *infos.xml* qui contient les métadonnées de la collection. La liste des images qu'il compose s'ils'agit d'une image unique.
- Un fichier `Link{collection}.csv` qui contient la liste des URLs s'ils'agit d'une collection de lien.

## Chapter 3

# Objectifs du projet

### 3.1 Récupération des données à partir d'un serveur

En l'état originel du projet, l'utilisation complète des fonctionnalités de l'application requiert le stockage préalable de l'ensemble des images, commentaires et sons associés aux taxons. Ces données ont un poids d'environ 500 Go, la majorité de cet espace correspondant aux images. De plus, au premier démarrage, il est nécessaire de placer le fichier de description de l'arbre dans le même répertoire que les répertoires de stockage des images, des sons et des commentaires, l'emplacement du fichier descriptif de l'arbre étant utilisé comme valeur pour la variable `TaxonPath`, qui correspond au répertoire parent des répertoires 'Images', 'Sounds' et 'Comments'.

Ces limitations sont incompatibles avec l'utilisation de l'application pour une classe d'étudiants. En effet, il n'est pas vraisemblable de fournir aux étudiants l'ensemble des données des taxons sur chaque poste de travail. L'objectif pour notre équipe était donc d'implémenter une récupération automatique des images, sons et commentaires depuis un serveur de manière à ce que l'utilisateur puisse utiliser l'application en disposant du seul fichier exécutable.

### 3.2 Interface web pour l'application

La seconde exigence formulée par nos encadrants était de fournir aux utilisateurs une interface web dotée d'un système d'authentification, et qui permet de télécharger l'exécutable de l'application ainsi que les arbres créés avec le logiciel, ainsi que de soumettre de nouveaux arbres. On nous a également demandé de réaliser un système de permission de manière à ce que chaque étudiant puisse consulter uniquement ses propres arbres, alors que les enseignants devaient pouvoir consulter les arbres de tous les étudiants.

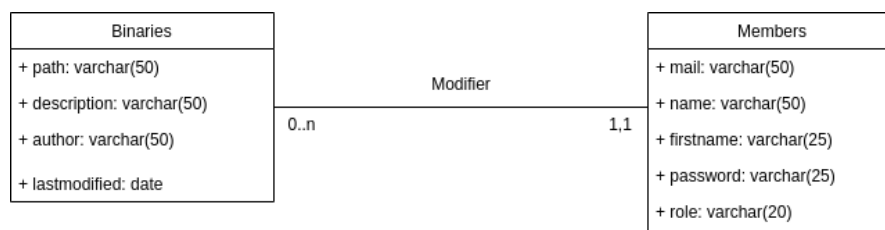
## Chapter 4

# Travail réalisé

### 4.1 Partie Serveur

#### 4.1.1 Base de donnée

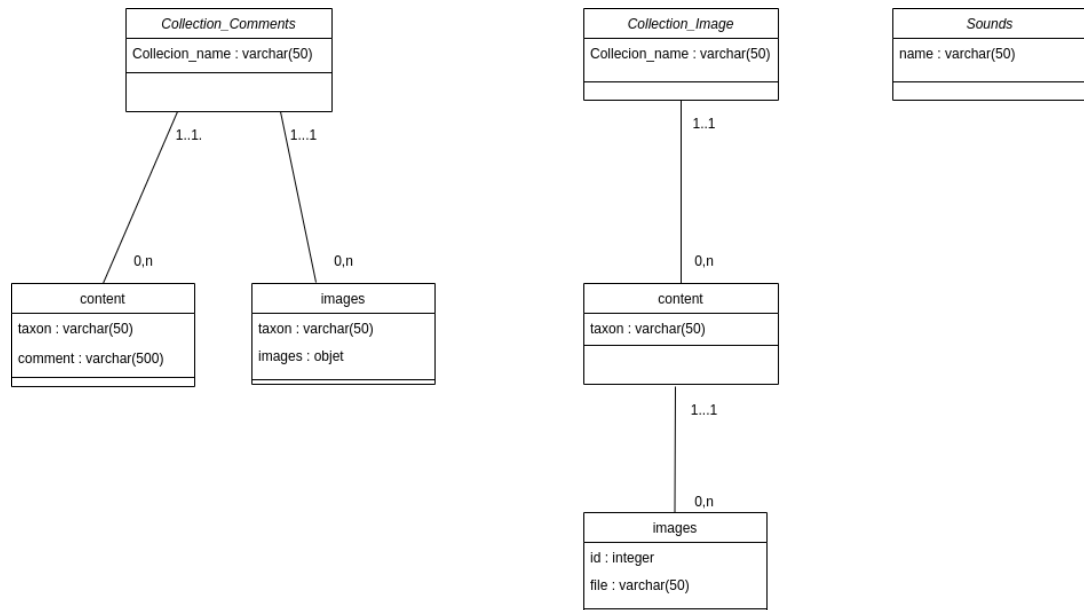
Pour commencer, nous avons crée une base de données Mongodb qui fait la liason entre les clients et les arbres qu'ils soumettent, voir diagramme ci dessous (chaque type d'objet JSON est transformé en une table).



La table *Membre* contient tous les informations sur l'utilisateur (mail, nom, prenom, etc.). La table *Binaires* contient les informations sur les arbres soumis par un membre (description, author, etc.).

Un membre peut être un étudiant ou un enseignant. L'enseignant a la possibilité de consulter tous les fichiers soumis par les étudiants, et un étudiant ne peut consulter que les fichiers soumis par lui-même.

L'autre partie des données de la base contient les informations associées aux taxons



La première table représente une collection de commentaires. Elle est composée de deux listes d'information, `content` et `images`. Dans la liste `content`, chaque objet contenu est un couple (taxon; fichier de description). Dans la liste `images`, chaque objet est constitué d'un nom de taxon et de la liste des fichiers images qu'on trouve dans son commentaire.

Dans `Collection_images` on stocke pour chaque taxon, la liste de ses images. Pour chaque image est indiqué le nom du fichier et son index (utilisé afin de différencier plusieurs images d'un même taxon).

Pour les sons, on stocke seulement le nom du taxon, l'URL vers le son pouvant être trivialement déduit à partir de celui-ci, les fichiers de sons étant au format `nomtaxon.wma`.

#### 4.1.2 API web

La seconde partie de notre travail a consisté à mettre au point une interface permettant à l'application "L'arbre du vivant", ainsi qu'à notre application web, de communiquer avec la base de donnée. Nous avons ainsi décidé de mettre au point une API REST avec la bibliothèque `Express.js`. Celle-ci est divisée en 5 fichiers :

- `users-controller.js` qui contient les routes associées à la gestion des membres de l'application web.
  - `GET /user?accessToken={token}` : sert à identifier un utilisateur de la base de donnée à partir d'un JSON Web token. <sup>1</sup>

<sup>1</sup>Pour implémenter le mécanisme d'authentification sur le site web, nous avons eu recours à l'utilisation des JSON web token, qui permet au client de s'authentifier de façon sécurisée

- POST /auth/login : sert à authentifier un utilisateur à partir de son adresse e-mail et de son mot de passe.
- POST /auth/register : sert à enregistrer un utilisateur sur la base de donnée.
- `binaries-controller.js` qui contient les routes associées aux arbres :
  - GET /binaries : Permet de récupérer la liste des arbres.
  - GET /user-binaries : Permet de récupérer les arbres soumis par un utilisateur actuellement connecté sur le site.
  - GET /binaries/{file} : Permet de télécharger un arbre en particulier.
  - POST /binaries : Permet de télécharger un arbre vers le serveur.
  - DELETE /binaries?path={path} : Permet de supprimer un arbre du serveur.
- `comments-controller.js` qui contient les routes associées aux commentaires des taxons :
  - GET /comments/{collection} : Permet de récupérer l'index d'une collection de commentaires (Liste des paires (taxon; fichier de commentaire)).
  - GET /comments/{collection}/{taxon} : Permet de récupérer le contenu du fichier de commentaire associé à un taxon.
  - GET /comments/{collection}/{taxon}/{file} : Permet de récupérer une image inscrite dans un commentaire
- `images-controller.js` qui contient les routes associées aux images des espèces et des sous-espèces.
  - GET /images/{collection} : Permet de récupérer l'index d'une collection d'images (Pour chaque espèce, la liste de ses images)
  - GET /images/{collection}/{species}/{index} : Permet de récupérer l'image (indexée par un entier) d'une espèce en particulier.
- GET /sounds/{taxon} qui contient les routes associées aux cris des espèces et sous-espèces animales.
  - GET /sounds/{taxon} : Permet de récupérer le cri d'un specimen d'une espèce animale en particulier.
  - GET /sounds : Permet de récupérer l'index des sons.

---

auprès du serveur grace à un token signé et crypté qu'il envoie dans l'en-tête de chaque requete HTTP

- `setup-controller.js` qui contient deux routes nécessaire à l'initialisation de l'application "L'arbre du vivant".
  - `GET /appdata/{file}` : sert à récupérer un fichier nécessaire à l'initialisation du programme.
  - `GET /state` : indique à l'application que le serveur est disponible.

En annexe A.1, vous pouvez consulter un diagramme de flow illustrant l'activité d'un utilisateur qui :

- Crée un compte sur l'application
- Se connecte
- Télécharge l'exécutable et un arbre
- Lance l'application et consulte les images et commentaires d'un taxon
- Fait des modifications et upload le nouvel arbre sur le serveur.

## 4.2 Partie Application C#

### 4.2.1 Analyse du code

Ensuite, nous avons dû étudier le code de l'arbre du vivant de manière à savoir quels éléments du code (plus de 35.000 lignes de code écrit en C#) devait être modifiées.

#### Initialisation des collections d'images

La première étape dans la récupération et l'affichage des images sur le serveur concerne l'instanciation des objets associés aux collections d'images (classe `ImageCollection`). Celle-ci est initiée au moyen de la méthode `BuildDictionary` de la classe `TaxonImages` qui parcourt l'ensemble des répertoires associés aux collections d'images et instancie un objet de classe `ImageCollection` dont les champs sont affectés selon les valeurs définies dans le fichier `infos.xml` de la collection.

#### Affichage d'une image

L'affichage d'une image dans la vue d'information est géré par le contrôle utilisateur `TaxonImageControlStatic`. Celui-ci expose une méthode `StaticPaint` qui sert à dessiner une image dans l'espace occupé par le contrôle.

`StaticPaint` appelle la méthode `GetCurrentImageToDraw` de `VignetteData` qui renvoie l'objet correspondant à l'image. Dans la méthode `GetCurrentImageToDraw` Si l'image n'a jamais été chargée, une image alternative est renvoyée et la méthode `RegisterDelayedGetFullImage` de la classe `TaxonImages` est appelée, si l'image a déjà été chargée la dernière image chargée est renvoyée.

RegisterDelayedGetFullImage ajoute une requête d'obtention de l'image à un dictionnaire des requêtes d'obtention d'images en attente.

Pendant ce temps, la méthode OnTimerRequestFullEvent est appelée à l'intervalle régulier et pour chaque requête en attente, déclenche l'appel de BWFullImage\_doWork en tâche de fond.

BWFullImage\_doWork effectue la récupération du fichier image par récupération soit sur le système de fichier, soit par le lien associé.

Quand la tâche de récupération est finie, la méthode BWFullImage\_completed est appelée et enchaîne sur la fonction OnImageLoaded de la classe VignetteData qui met à jour l'image courante. Ainsi au prochain appel de la méthode StaticPaint, c'est l'image réelle du taxon qui sera affichée.

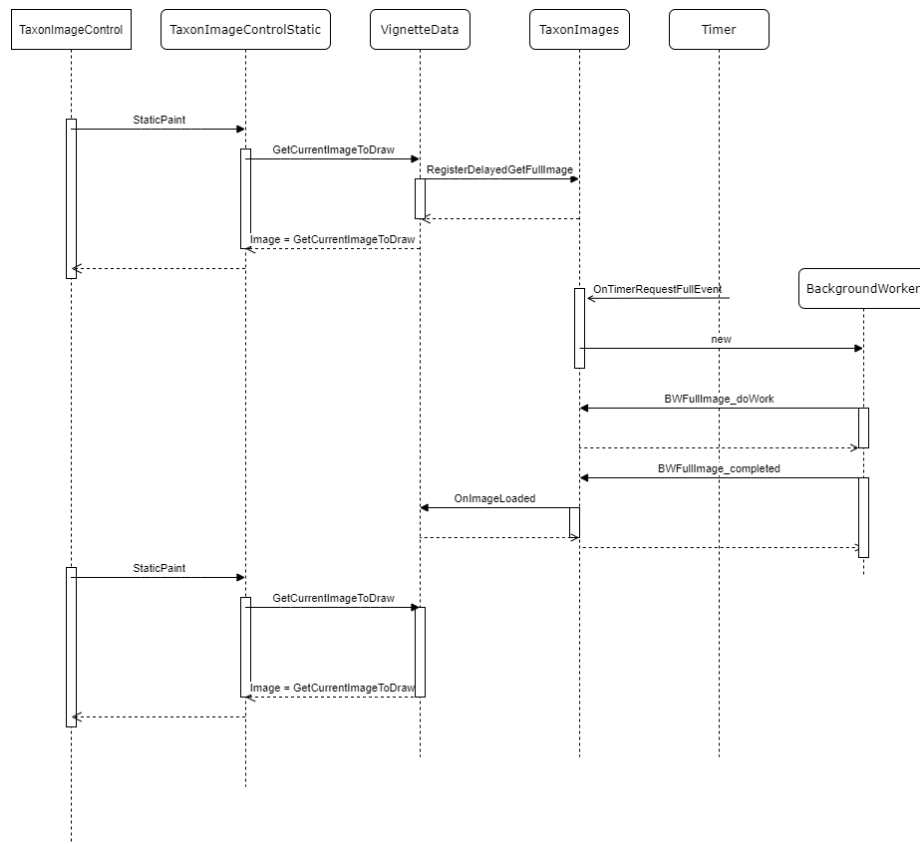


Diagramme de séquence illustrant la récupération et l'affichage d'une image

### Initialisation des collections de commentaires

Les collections de commentaires sont initialisées dans la méthode BuildList de la classe TaxonComments de la même manière que les collections d'images : les répertoires de chaque collection sont parcourus et un objet CommentsCollection est créé, alimenté à partir des informations du fichiers \_infos.xml.

## Récupération et affichage des commentaires

Quand un taxon est sélectionné, cela déclenche l'appel de la méthode `RefreshContent` de la classe `TaxonMultiCommentControl`. Celle-ci appelle la méthode `GetComments` de la classe `TaxonComments`, qui à son tour ajoute une requête d'obtention de commentaire à la liste des requêtes d'obtention de commentaire en attente.

Pendant ce temps, la fonction `TaxonCommentsLoop` (qui est exécutée en boucle dans un thread séparé) :

1. Récupère une requête d'obtention de commentaire
2. Lit le fichier de commentaire depuis le disque dur
3. Modifie les balise d'insertions d'image de manière à ajouter le chemin vers l'image (fonction `TransformHTMLComment`).
4. Remplit la propriété `Result` de la requête avec le contenu du fichier qui vient d'être lu.
5. S'il s'agit du noeud racine, appelle la requête de complétion, et sinon appelle empile une requête pour obtenir les commentaires du noeud parent

La requête de complétion associée à la requête correspond à la méthode `OnCommentLoaded` de la classe `TaxonMultiCommentControl`, qui à son tour appelle `SetComments` de la même classe.

C'est la méthode `SetComments` qui construit le document `HTML` et donne son contenu au formulaire `Windows Forms WebBrowser`.



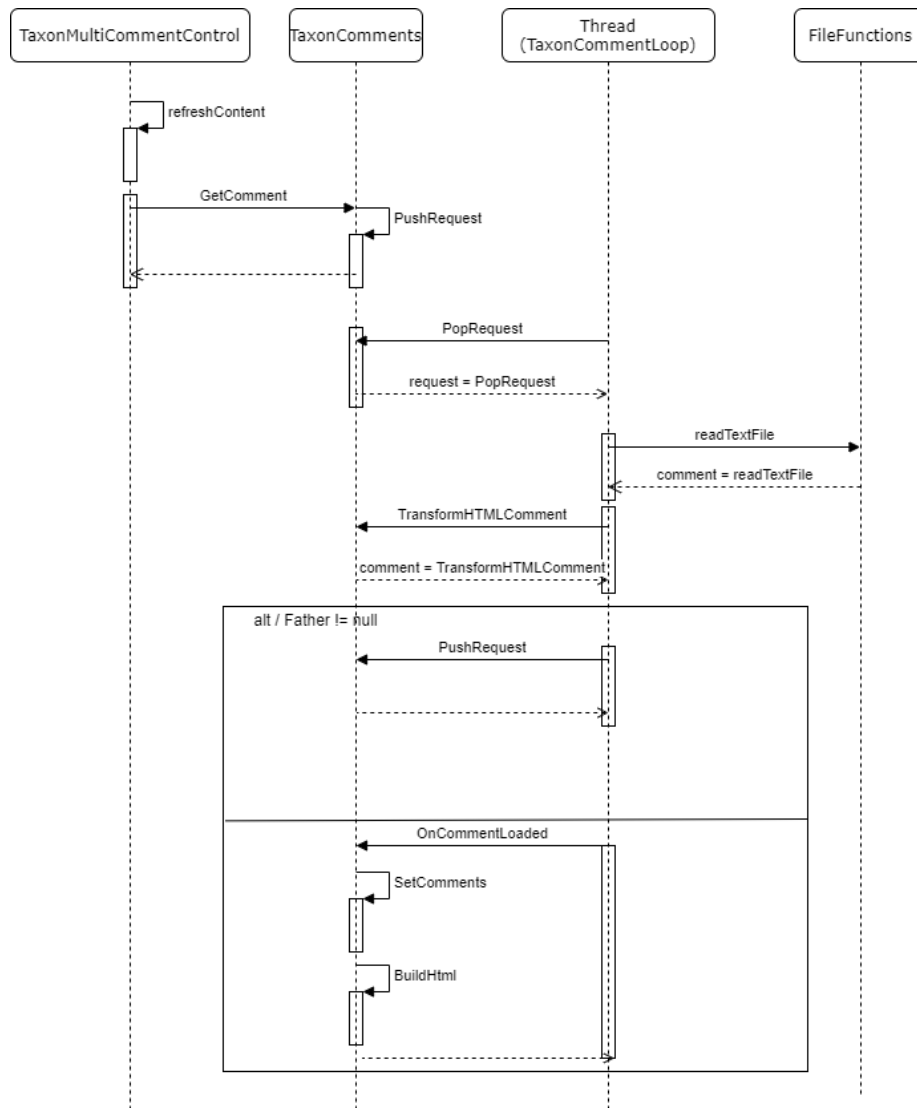


Diagramme de séquence illustrant la récupération et l’affichage des commentaires

### Récupération des sons

Vincent Lhuillier, développeur de l’*“arbre du vivant”* a conçu un lecteur de fichiers audio qui tourne en arrière-plan et peut s’insérer dans un contrôle Windows Forms. Celui-ci possède une propriété `File` qui représente le chemin vers le fichier audio à lire.

Quand un nouveau taxon est sélectionné, la méthode `UpdateSoundControl` est appelée, celle-ci récupère le taxon actuellement sélectionné et déduit de son

nom le chemin vers le fichier audio.

### **Sauvegarde de la configuration**

Lorsque l'utilisateur quitte proprement l'application, l'état de l'application (sous-fenêtres activées, arbre chargé, etc.) est sauvegardé dans un fichier de configuration `TreeOfLifeConfig_auto.xml`.

## **4.2.2 Implémentation**

### **Récupération des images depuis le serveur**

En plus des collections locales et des collections de liens, nous avons décidé d'ajouter un troisième type de collection d'images : les collections distantes. Celle-ci dispose d'une propriété supplémentaire spécifiée dans le fichier `_infos.xml` : `Location`. Elle indique l'adresse à laquelle on peut récupérer l'index d'une collection d'image (comme défini dans la section sur l'API web). Lorsqu'on initialise un `TaxonImageCollection`, si la propriété `Location` est présente, une requête HTTP est envoyée pour récupérer l'index de la collection d'images. Ce résultat est exploité pour alimenter une liste nommée `DistantReferences`. Ce qui permet notamment d'afficher sur l'outil de gestion des collections le nombre d'image présent dans la collection.

Dans la classe `TaxonImageCollection`, une méthode `GetDistantImageLink` est ajoutée afin de retourner le lien vers une image sur le serveur étant donné le nom du taxon et son index.

Dans la méthode `BWFullImage_DoWork`, après avoir testé si la collection est une collection de lien, on teste également s'il s'agit d'une collection distante, et dans le cas échéant, on télécharge l'image à l'adresse retournée par `GetDistantImageLink`.

### **Récupération des commentaires depuis le serveur**

Comme pour les images, un type correspondant aux collections de commentaire distantes est ajouté avec un attribut `Location` pointant vers l'index de la collection distante.

Dans la méthode `TaxonCommentLoop`, on vérifie si la collection indiquée dans la requête d'obtention de commentaire est une collection distante, et dans le cas échéant on télécharge le fichier de commentaire.

Dans la fonction `TransformHTMLComment`, dans le cas où la collection est distante, on ajoute aux balises `<img>` la route de la collection afin que `WebBrowser` aille télécharger les images depuis le serveur.

### **Récupération des sons depuis le serveur**

Pour récupérer les sons correspondant à un taxon, on a ajouté une méthode `FindSound` qui vérifie si un fichier audio nommé `{taxon}.wma` existe dans le

répertoire des sons. Si celui-ci n'existe pas, on récupère le fichier depuis le serveur.

Dans la méthode `UpdateSoundControl`, la propriété `File` est affectée avec le résultat de `FindSound`.

On a également ajouté une liste statique `availableSounds` accessible depuis toutes les classes de l'application, qui contient le nom des taxons ayant un cri associé. Celle-ci est remplie à partir de la route de l'API web permettant de récupérer l'index des sons. Cette étape est nécessaire notamment pour faire fonctionner l'outil de mise à jour des sons de l'arbre.

### **Premier démarrage de l'application**

Malgré le déplacement des images, des commentaires et des sons sur le serveur. L'application nécessite que soient présents les répertoires 'Images', 'Comments', 'Sounds', les répertoires correspondant à chaque collection (d'images ou de commentaires) ainsi que le fichier `_infos.xml` correspondant à chacune d'entre elles. Ainsi, nous avons décidé d'ajouter une propriété booléenne `dataInitialized` au fichier de configuration. Si celle-ci est fausse au démarrage, une requête HTTP est envoyée au serveur afin de récupérer le fichier `init.zip` contenant tous les fichiers précédemment cités. Le contenu de l'archive est placé dans un répertoire TOL, à l'intérieur du répertoire AppData de l'utilisateur Windows courant.

L'utilisateur a également la possibilité de choisir de travailler avec des données situées sur son ordinateur. Nos encadrants ayant insisté sur le fait que le travail rendu soit rétrocompatible avec le code qu'il nous ont livré.

Il a également la possibilité de choisir de travailler avec un arbre vide, s'il souhaite en créer un nouveau.

The screenshot shows a window titled "TreeOfLife". Inside, there are two main sections. The first section, "Selection of data initialization mode", contains a radio button for "Online mode" which is selected, and a text input field containing "http://localhost:8888/". Below this is a radio button for "Offline mode" which is unselected, followed by an empty text input field and a button labeled "Select data directory ...". The second section, "Tree", contains a radio button for "Load a tree" which is selected, followed by an empty text input field and a button labeled "Select ...". Below this is a radio button for "Empty tree" which is unselected. At the bottom of the window are two buttons: "Validate" and "Quit", with the "Quit" button highlighted by a blue rectangle.

Formulaire affiché lors du premier démarrage de l'application

## 4.3 Partie Application Web

### 4.3.1 Introduction

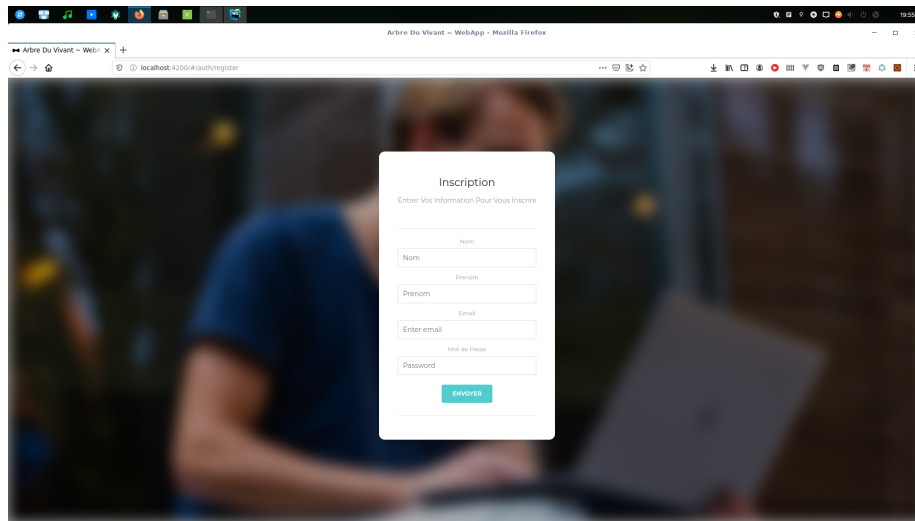
En ce qui concerne l'interface web, celle-ci a été développée en utilisant les dernières technologies dans le standard web, parmi les plus importantes :

- Angular 9 : le framework Single-Page-Application développé par Google, est un framework front-end parmi les plus utilisés par la communauté des développeurs. Basé sur le langage TypeScript, il permet d'écrire des applications extensibles et maintenables avec facilité.
- TypeScript : TypeScript est un langage de programmation libre et open source développé par Microsoft qui a pour but d'améliorer et de sécuriser la production de code JavaScript. C'est un sur-ensemble de JavaScript (c'est-à-dire que tout code JavaScript correct peut être utilisé avec TypeScript).

- API Rest : est un ensemble de standard qui specifient comment échanger les données entre un client et server a travers le protocole HTTP.
- Jest : Est un framework de test unitaire développé par Facebook, qui facilite la création de stub, mock et l'écriture de test unitaire en général.

### 4.3.2 Les différentes pages de l'application

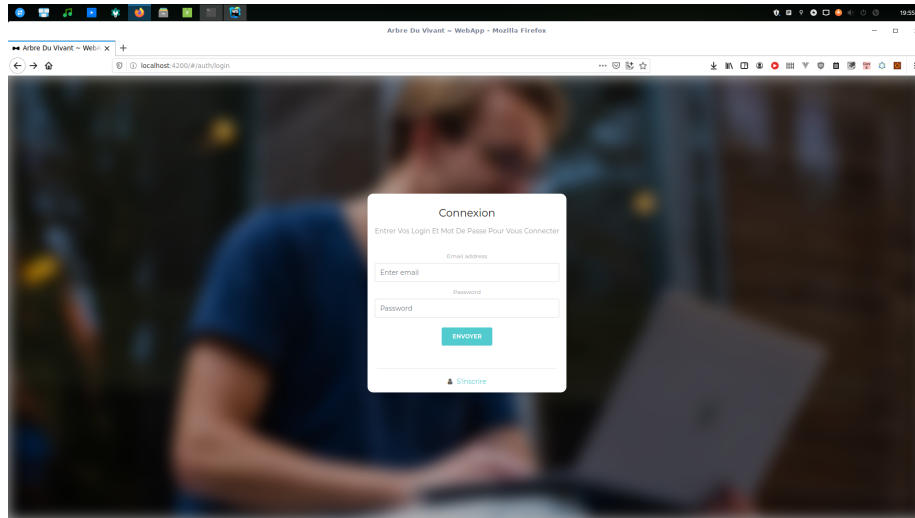
#### Page d'inscription

The image shows a web browser window displaying a registration page titled "Inscription". The page has a blurred background of a person working on a laptop. In the center, there is a white registration form with the following fields: "Nom", "Prénom", "Email", and "Mot de Passe" (Password). Each field has a corresponding input box. Below the "Mot de Passe" field is a green button labeled "Envoyer". The browser's address bar shows the URL "localhost:4200/auth/register".

La page d'inscription permet à l'utilisateur de s'inscrire grâce à un formulaire. Après avoir rempli ce dernier, quand l'utilisateur clique sur le bouton 'Envoyer', une vérification des données est effectuée pour s'assurer que tous les champs sont bien remplis avec les valeurs adéquates (entier, chaîne, date, etc.). Puis une requête d'inscription est envoyée à l'API web via la route `POST /auth/register`

Pour effectuer ces traitements, Angular offre plusieurs outils et modules pour gérer la récupération et la validation d'un formulaire, notamment grâce au module `ReactiveFormsModule` et `FormsModule`. Pour ne citer que celle-ci, la directive `[ngModel]` permet de relier la valeur d'un formulaire directement à une variable de notre code, ce qui facilite grandement la récupération des données à partir d'un formulaire.

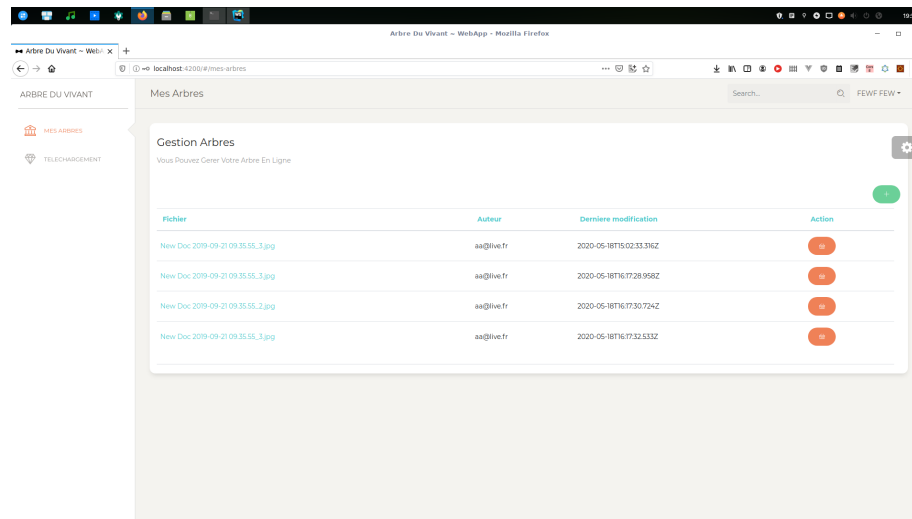
## Page de connexion



La page de connexion permet à l'utilisateur de se connecter pour pouvoir accéder au contenu de l'application web, un utilisateur non connecté sera automatiquement redirigé vers cette page. En ce qui concerne le formulaire, on gère la validation des données avant de les envoyer au serveur ainsi que l'affichage des erreurs (par exemple, dans le cas où l'adresse e-mail est déjà utilisée). En cas de succès l'utilisateur sera redirigé vers sa page principale de gestion des arbres.

Pour effectuer cette tâche, nous avons eu recours à l'authentification grâce aux tokens JSON : une technique qui consiste à assigner un jeton à l'utilisateur quand les informations de connexion sont correctes. Grâce à ce jeton, que l'utilisateur enverra dans toutes les prochaines requêtes dans un champ header de la requête HTTP, le serveur pourra facilement l'authentifier en se basant sur les différents champs qui se trouvent dans le jeton JSON.

## Page de gestion des arbres

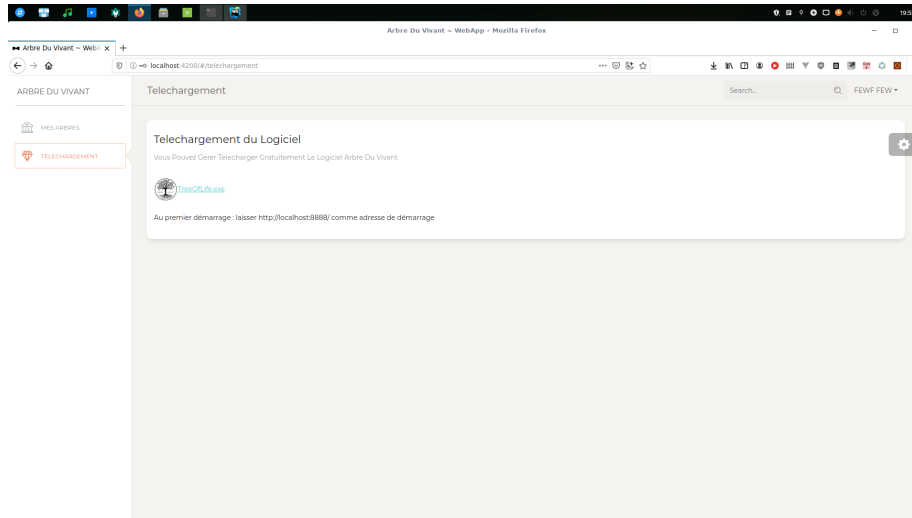


La page de gestion des arbres permet à l'utilisateur de gérer les arbres (fichiers .TOL) qu'il soumet à son enseignant pour obtenir une éventuelle correction ou un retour. L'utilisateur peut notamment ajouter et supprimer les arbres soumis. Grâce à la technologie Single-Page-Application qu'offre le framework Angular : l'ajout et la suppression des arbres pour l'utilisateur est pleinement facilitée, puisqu'il n'y a aucun rafraîchissement de la page lors de l'envoi et de la suppression du fichier, ce qui permet d'avoir une navigation et une manipulation fluide sans interruption.

Grâce aux routes exposées par le serveur, le navigateur envoie les informations des fichiers à ajouter et à supprimer dans des requêtes AJAX.



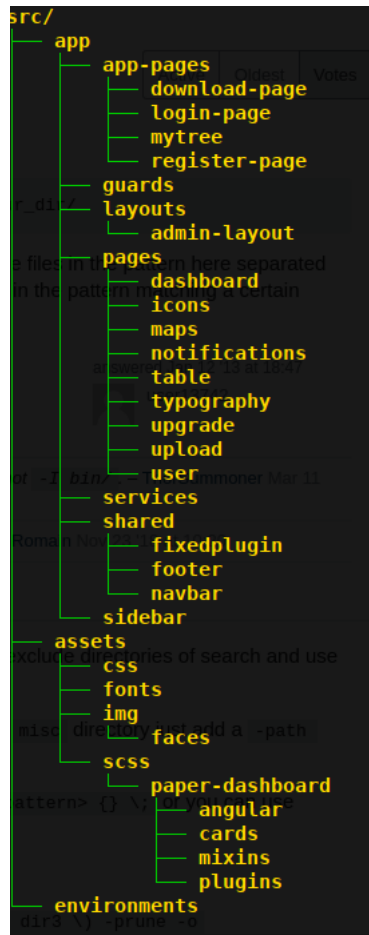
## Page de téléchargement



La page de téléchargement permet à l'utilisateur de télécharger l'application de bureau pour manipuler les arbres.

### 4.3.3 Compréhension de la structure du code

Pour bien éclaircir le fonctionnement de l'application web ainsi que les différentes technologies utilisées, on va décrire le contenu des différents dossiers de notre projet.



Organisation du code de l'application web

- **App-Pages** : Ce dossier contient toutes les pages navigables de notre application web. Chaque sous-dossier contient un composant Angular, qui est composé de trois fichiers :
  - Un fichier HTML qui contient la structure de la page.
  - Un fichier de style qui contient les différents styles appliqués à la page (ça peut être du code CSS ou SCSS si on choisit d'utiliser un préprocesseur CSS)
  - Un fichier en TypeScript qui contient toute la logique et la gestion d'événements du composant.
- **Guards** : Les "Guards" permettent de contrôler l'accès à une "route" (e.g. autorisation) ou alors le départ depuis une "route" (e.g. enregistrement ou publication obligatoire avant le départ). On y trouve généralement des fichiers TypeScript composés d'une classe qui implémente l'interface

`CanActivate`, ce qui force la classe a avoir une methode `canActivate` qui doit retourner `True` si l'utilisateur peut continuer à consulter une page par exemple, ou `false` dans le cas contraire.

- **Layout et Pages et Shared** : ensemble de fichiers et de modules qui contiennent plusieurs composants graphiques réutilisables dans différentes situations.
- **Services** : Ce dossier contient un ensemble de fichiers TypeScript. Chacun de ces fichiers contient une classe qui permet d'effectuer des requete AJAX vers les différents services et routes proposées par le serveur. On utilise notamment la classe `HttpClient` fournie par Angular, cette dernière contient plusieurs méthodes (`get`, `post`, `delete`, etc.) qui lancent chacune une requete HTTP vers le serveur.
- **Assets**: contient les différentes ressources (images, icônes, sons, polices de caractères) requises pour le fonctionnement de l'applkication web.

## Chapter 5

# Bilan du projet

### 5.1 Autocritique

Le principal objectif que nous nous étions fixé au sein du groupe était d’avoir une application fonctionnelle, et un site web clair pour guider les utilisateurs.

Nous avons pu finir tous les tâches prévu au départ du projet, l’application *L’arbre du vivant* est utilisable pour tout le monde sans avoir télécharger les données avant le démarrage. De plus, le site web que nous avons créé permet a un utilisateur de s’enregistrer, de télécharger l’application et enfin de soumettre un travail.

Nous sommes globalement satisfaits de notre gestion du temps, et nous pensons que notre organisation y a beaucoup contribué. L’expérience de travailler avec des industriels nous a appris à mieux comprendre les besoins clients et à savoir poser des questions afin de lever les zones d’ombres face à une exigence client. De plus, le fait de savoir que le fruit de notre travail allait être utilisé à la fin du projet nous a poussé à nous investir d’avantage dans le projet et nous a permis de nous initier aux spécificité du développement logiciel dans l’industrie.

### 5.2 Difficulté

Le code de l’application *L’arbre du vivant* étant plus complexe que celui des projets sur lesquelles nous avons l’habitude de travaillé. La phase d’analyse du code nous a pris beaucoup de temps et a donc retardé dans l’implémentation des fonctionnalités demandées.

Enfin, la communication entre les membres de l’équipe et les encadrants était perturbé à cause de la pandémie du COVID-19 qui a causé beaucoup de restriction de mouvements.

## 5.3 Perspectives

Ce projet peut-être enrichi de multiples manières, pour l’instant, on peut par exemple imaginer l’implémentation d’un éditeur *Markdown* pour éditer les commentaires des taxons. La traduction dans d’autres langues serait aussi intéressante. Enfin, ce projet ne fonctionne aujourd’hui que sous la plateforme windows, il serait intéressant d’en faire une version multi-plateforme.

Les perspectives de développement de cette application sont donc multiples, et l’expérience acquise durant ce projet nous permet d’affirmer qu’elles sont réalisables en des temps décents.

## Chapter 6

# Conclusion

Le développement de ce projet nous a permis d'enrichir notre connaissances dans des langages que nous connaissions mal (C#, Angular, MongoDB...), ainsi que d'améliorer notre capacité de travail en équipe dans des situations nouvelles.

Le travail de groupe a été une réussite grâce à une bonne organisation et une bonne méthode de travail accompagnés d'outils performants. Les capacités de chacun ont été pleinement exploitées.

Après avoir obtenu la confirmation d'Ubisoft pour un hébergement du serveur nous ne serons capables de le lancer qu'après avoir fini l'année universitaire. Nous sommes cependant heureux d'être les premières à avoir travaillé sur ce projet initié par les employés d'Ubisoft, car nous penserons qu'il a un grand potentiel au sein de la faculté et ce dernier facteur nous a servi de source de motivation tout au long du projet.

## Appendix A

# Code source du projet

Les sources du projets sont disponibles sur les deux dépôts git suivants:

- <https://github.com/karimsqualli/treeoflife/commits/master> qui contient le code modifié de l'application "l'arbre du vivant".
- <https://github.com/Dvassily/TreeOfLife-web> qui contient les sources de l'API web et de l'application web.

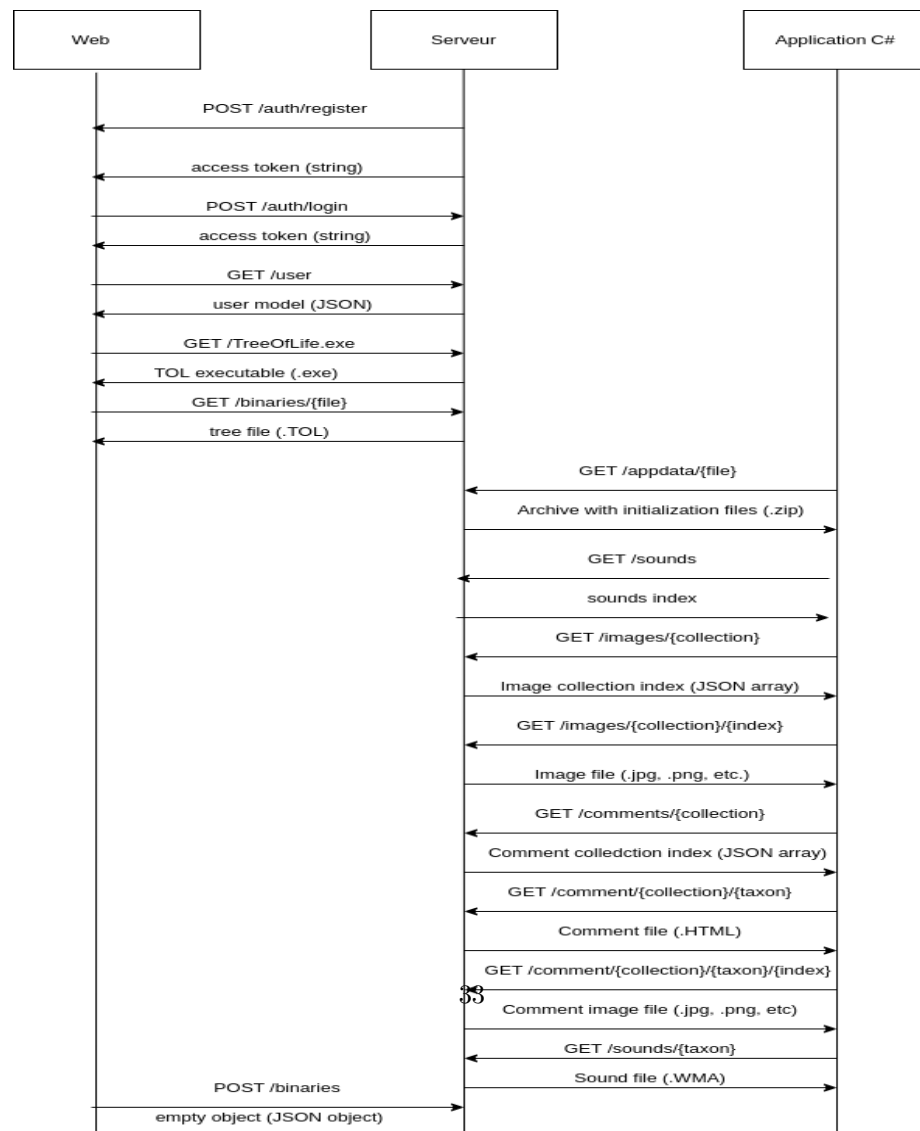




# Appendix B

## API Web

### B.1 Diagramme de flow



## B.2 Page Inscription

Arbre Du Vivant - WebApp - Mozilla Firefox

localhost:4200/#auth/register

**Inscription**  
Entrez Vos Informations Pour Vous Inscrire

Nom

Prénom

Email

Mot de Passe

Password

Envoyer

## B.3 Page Connexion

Arbre Du Vivant - WebApp - Mozilla Firefox

localhost:4200/#auth/login

**Connexion**  
Entrez Vos Login Et Mot De Passe Pour Vous Connecter

Email address

Enter email

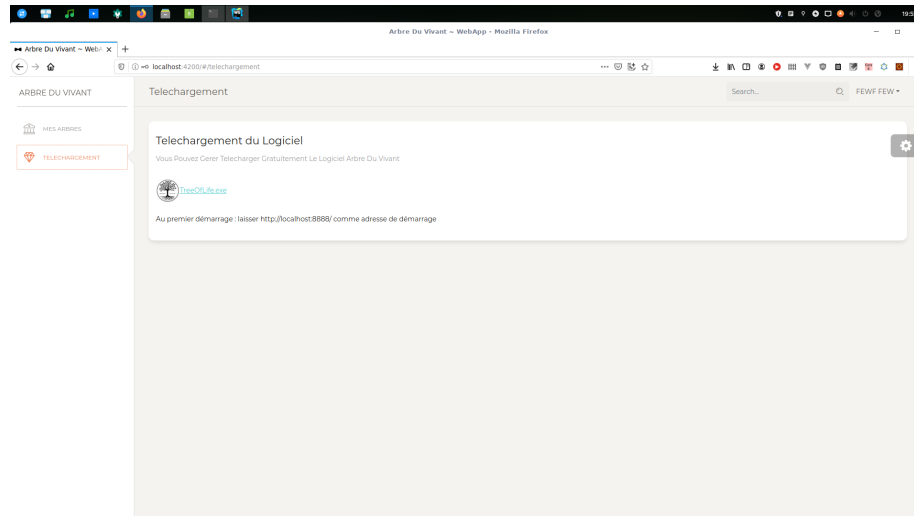
Password

Password

Envoyer

S'inscrire

## B.4 Page Telechargement



## B.5 Page Mes Arbres

