INSTITUT NATIONAL DE L'INFORMATION GÉOGRAPHIQUE ET FORESTIÈRE

ENSG Géomatique

Université Gustave Eiffel

Ecole Nationale des Sciences Geographiques
Projet Informatique

Responsable de la mention : Victor Coindet

Rapport de projet de fin d'année :

Amaryllis Vignaud, Axelle Gaigé, Mélodia Mohad, Thomas de Beaumont, Valentin Mathiot

Titre du stage :

Plateforme pour la visualisation d'images orientées

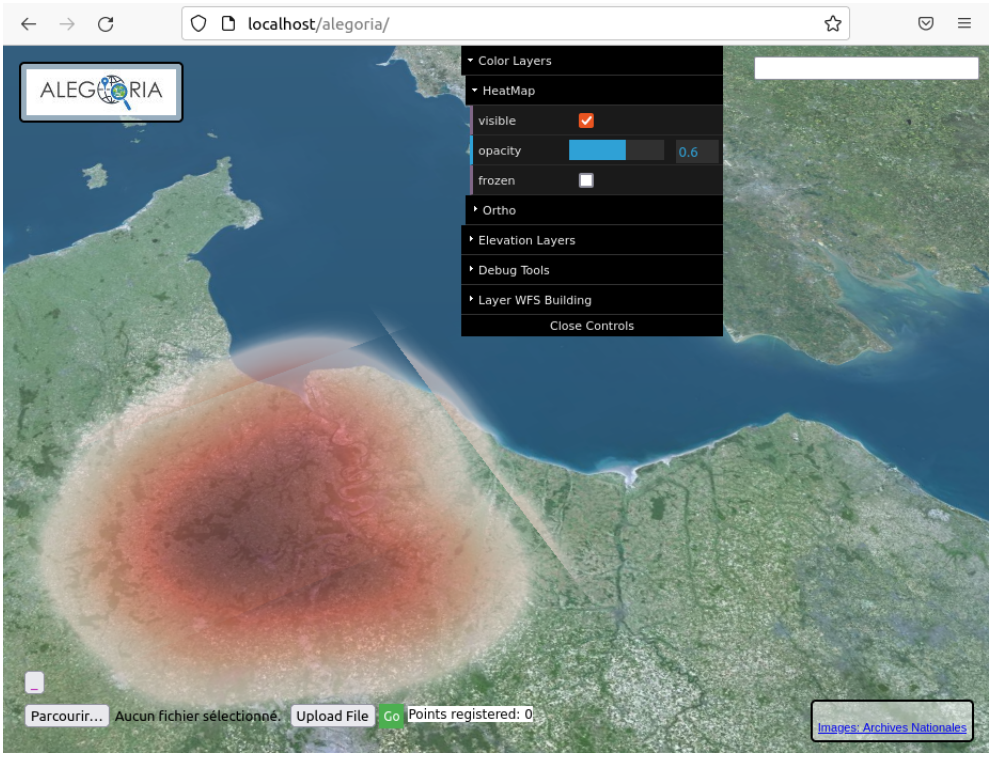


Table des matières

Table des figures

1 Glossaire

API Application Programming Interface

BDD Base de données

IGN Institut national de l'information géographique et forestière

MVC Model-View-Controller ou Modèle-Vue-Contrôleur

REST REpresentational State Transfer

SPARQL Protocol and RDF Query Language

SQL Structured Query Language

VM Virtual Machine

WebGL Web Graphics Library

1.1 Définitions

API REST

Une API est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications. Lorsqu'un utilisateur utilise un ordinateur, par exemple sur une application web, il peut souhaiter récupérer des informations. C'est alors que l'API entre en jeu : l'utilisateur va envoyer ses paramètres, l'API va les recevoir et faire les traitements nécessaires en fonction de la demande. L'avantage de l'API est que l'utilisateur n'a pas besoin de connaître le code derrière, seulement d'envoyer les paramètres nécessaires.

Une API dite REST (ou RESTful), est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web, notamment en utilisant des protocles HTTP. Les protocoles HTTP "permettent aux logiciels d'un appareil de communiquer avec les logiciels d'un autre appareil (ou du même appareil) même s'ils utilisent des systèmes d'exploitation et des architectures différents. Le client peut demander des ressources avec un langage que le serveur comprend, et le serveur renvoie la ressource avec un langage que le client accepte. Le serveur renvoie la ressource au format JSON (JavaScript Object Notation), XML (Extensible Markup Language) ou texte, mais de nombreuses API prennent en charge d'autres langages" (ref https://www.uptrends.fr/qu-est-ce-que/rest-api).

iTowns

iTowns est une plateforme de visualisation et d'exploitation de données géographiques 3D web.

MicMac

MicMac est un logiciel de photogrammetrie open source.

HeatMap

Une HeatMap est une carte représentant la densité d'un objet dans l'espace.

Orientation

L'orientation correspond à la façon dont se place l'image dans l'espace.

2 Introduction

2.1 Contexte

2.1.1 Contexte du projet

Notre projet s'inscrit dans le cadre du projet Alegoria.

"Le projet ALEGORIA, fondé par l'ANR (Agence Nationale de la Recherche), a pour objectif de faciliter la valorisation des collections de fonds institutionnels iconographiques décrivant le territoire français à différentes périodes allant de l'entre-deux-guerres à nos jours.

Les collections en vue sont de tailles variables, entre des milliers et des centaines de milliers d'éléments, et sont constituées de représentations iconographiques, qui sont la plupart du temps des photographies aériennes ou terrestres obliques et verticales de l'environnement. Contrairement à l'exploitation bien ancrée qui est faite de l'imagerie satellitaire où les pratiques professionnelles sont nombreuses (scientifiques, civils et militaires) et les données bien identifiées et indexées, la valorisation de ces collections reste confidentielle et dispersée. Elles sont réparties dans diverses institutions, partiellement numérisées, généralement pas ou peu documentées et faiblement géoréférencées. Elles représentent pourtant un riche patrimoine, peu connu du grand public et exploité de manière contrainte par leurs principaux utilisateurs (chercheurs, institutions et collectivités locales), en consultation directe à la bibliothèque ou par le biais de bibliothèques numériques classiques en ligne.

La mise en valeur d'un tel patrimoine bénéficierait d'outils permettant d'automatiser leur collecte, leur traitement et leur indexation, au sein même de la collection mais aussi entre les collections, pour être capable de les croiser puis de mieux les étudier. Leur mise en œuvre au sein d'applications implique de faciliter leur appréhension par des utilisateurs non spécialistes, elle suppose de fournir des outils conviviaux pour l'exploration visuelle de ces collections." (ref http://alegoria.ign.fr/)

2.1.2 Outils du projet

Le projet a pour but, comme évoqué précédemment, de développer des outils pour mettre en valeur le patrimoine existant sur les différentes collections de vue :

- Un moteur d'indexation et de recherche multimodal et à grande échelle, couplant la recherche par le contenu et par les métadonnées dans les collections intra-domaines ainsi que dans les collections inter-domaines.
- Un moteur web pour l'affichage immersif de ces contenus, permettant la navigation spatio-temporelle et l'interaction dans l'environnement 3D enrichi de collections photographiques anciennes. Notre projet s'inscrit dans le cadre de l'amélioration de cet outil.

Ainsi, notre projet doit permettre de visualiser des images provenant d'une base de données, dans une interface 3D web. Ces images doivent être orientées dans l'espace grâce à l'utilisation de MicMac et à l'ajout de points d'appuis pour effectuer cette orientation.

2.2 Besoin du client

Les besoins du client dans le cadre de notre projet sont les suivants :

— Notre commanditaire souhaite visualiser des images provenant d'une base de données dédiées. Le premier besoin du commanditaire est donc de créer une base avec pour fonction le stockage des données des images.
— Le commanditaire souhaite également mettre en place une API pour faciliter l'utilisation de MicMac.
— Le commanditaire souhaite également afficher des Heatmaps (nous définiront le terme plus bas) sur son interface.

Tous ces composants doivent pouvoir interagir avec l'application cliente.

3 Analyse fonctionnelle

Nous allons maintenant réaliser l'analyse fonctionnelle de notre application. L'analyse a porté à la fois sur la partie cliente de l'application et sur la partie serveur de celle-ci. Nous nous intéresserons également à l'architecture de l'API qui sera mise en place.

3.1 Partie cliente

Nous allons d'abord nous intéresser à l'analyse fonctionnelle de la partie cliente. Pour cela nous avons réalisé un diagramme de classe de l'application déjà existante (voir figure 1). Nous avons également réaliser un diagramme de déploiement de l'ensemble de l'architecture côté cliente (voir figure 2).

Diagramme de classe de l'application cliente

Pour commencer, nous allons présenter la structure de l'interface web existante à notre arrivée. Nous avons choisi de présenter ce diagramme car il nous semblait important de montrer le contenu de l'application de départ, celle-ci n'étant pas documentée.

La partie cliente est découpée en trois grandes briques, correspondant chacune à un dépôt GitHub du projet nécessaire pour la création de l'application cliente :

- alegoria : cette brique contient le fichier index.html permettant d'instancier l'interface de notre application web. Elle contient également des fichiers PHP et JavaScript. Ce sont les fichiers JavaScript, appelés par le fichier html, qui font appel aux fichiers PHP pour lancer MicMac et différentes opérations de ce logiciel. Nous pouvons également souligner que certains fichiers sont inutilisés.
- iTowns : cette brique contient quant à elle les fichiers nécessaires pour l'affichage du fond de carte 3D, et les outils nécessaires à la navigation dans celle-ci. Celle-ci est basée sur Three.js et est codé en JavaScript/WebGL.
- photogrammetric-camera contient l'ensemble des paramétrages nécessaires à la création de la caméra utilisée dans l'application cliente. Par exemple, comment la caméra va se placer dans l'espace 3D (orientation, centre de la caméra...).

Nous pouvons remarquer sur ce diagramme que c'est le fichier index.html qui relie l'ensemble des dépôts et les différents fichiers de l'application. Il y a donc un couplage très fort entre celui-ci et l'ensemble des fichiers/dossiers de l'application.

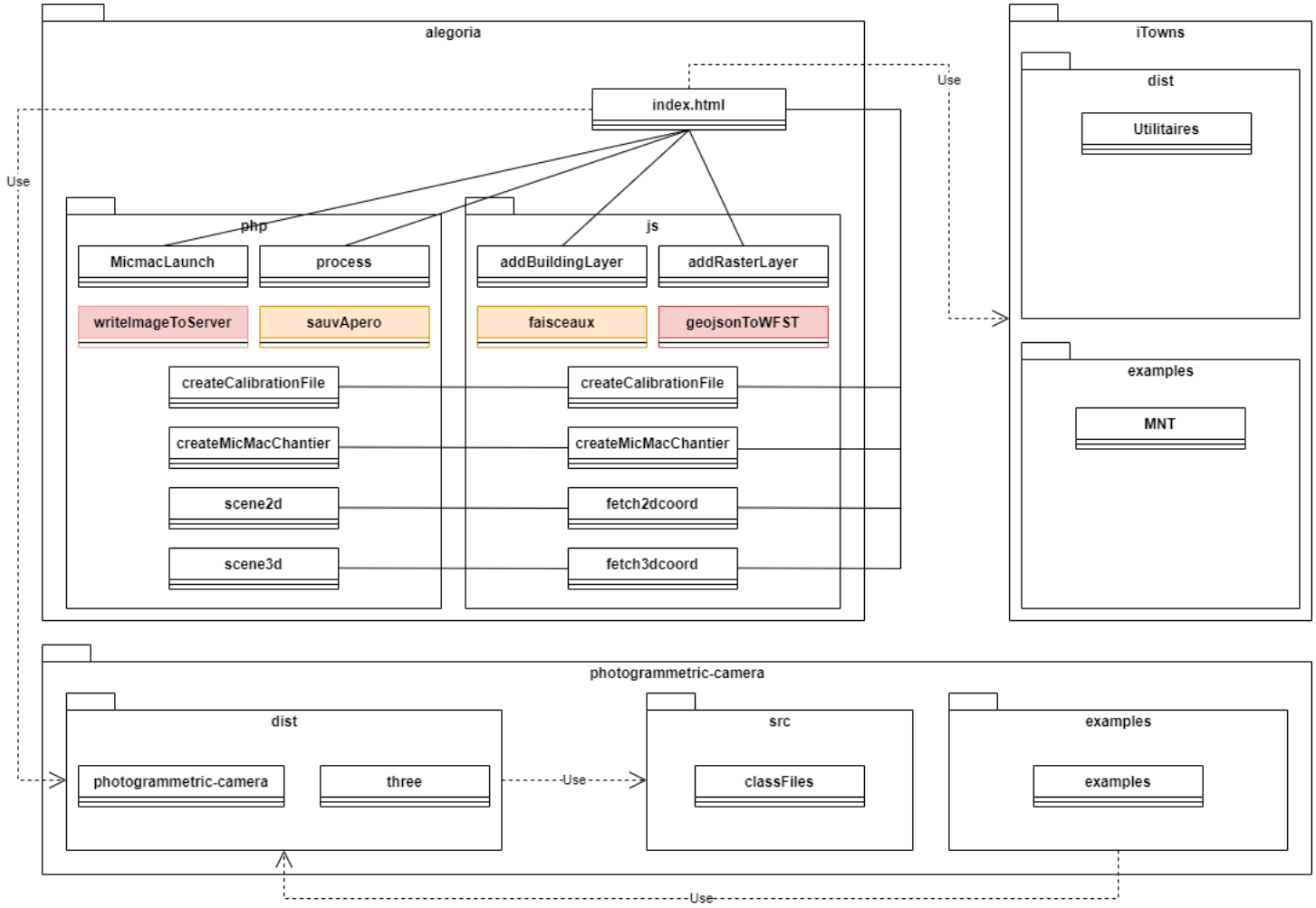


FIGURE 1 – Diagramme de classe de l'application cliente

Architecture globale

Pour finir de présenter la partie cliente, allons analyser comment les composants de la partie cliente s'imbriquent entre eux à l'aide du diagramme de déploiement sur la page ci-dessous. Ce diagramme présente l'ajout de nouvelles fonctionnalités comme l'intégration d'une API ou le requêtage sur Search Engine, qui n'existaient pas auparavant.

Le diagramme de classe présenté précédemment correspond à l'interface de notre application. Ce diagramme de déploiement montre que l'interface va évoluer avec l'ajout de nouvelles interactions entre l'interface et divers composants.

En effet, à partir de l'interface, l'utilisateur va sélectionner une image. Une fois cette image sélectionnée, l'interface va chercher à récupérer les paramètres d'orientation de l'image. L'interface interroge donc la base de données, et en parallèle l'outil Search Engine d'Alegoria. Si l'image ne possède pas de données d'orientation, celle-ci ne renverra rien à l'interface. Cependant, si les paramètres d'orientation existe, l'image et ses métadonnées seront retournées à l'interface.

Une fois l'image téléchargée dans l'interface, l'utilisateur peut souhaiter la placer dans l'espace en 3D. Pour cela, l'utilisateur va demander de calculer l'orientation de l'image depuis celle-ci. L'interface va alors envoyer une requête à l'API avec les paramètres d'orientation et différents points qu'il aura créé lui-même, appelés points d'appuis (voir après la figure **??**). Ces points permettent d'aider MicMac à placer correctement l'image dans l'espace.

Une fois le message reçu par l'API, celle-ci va mettre en forme les paramètres pour demander le calcul d'orientation de l'image à MicMac. Pour orienter convenablement l'image, MicMac va calculer auparavant trois fichiers : le fichier gcp, le fichier appui et le fichier calib. Ces fichiers contiennent respectivement :

- Les points d'appuis 3D (dans l'espace, en orange sur l'image) de l'image à orienter pour le fichier gcp ;
- Les points d'appuis 2D (sur l'image, en bordeaux sur l'image) de l'image à orienter dans l'espace pour le fichier appui ;
- Les paramètres de calibration du capteur pour le fichier calib

Une fois ces trois fichiers calculés, l'API va lancer la commande nécessaire au calcul de l'orientation de l'image. Une fois le résultat obtenu, MicMac envoie le résultat à l'API. En parallèle, les trois fichiers nécessaires au calcul de l'orientation sont supprimés.

Une fois le résultat reçu par l'API, celle-ci va renvoyer les résultats en plaçant correctement l'image dans l'espace. L'interface envoie ensuite les paramètres d'orientation nouvellement calculé à la base, afin de les enregistrer pour une utilisation future.

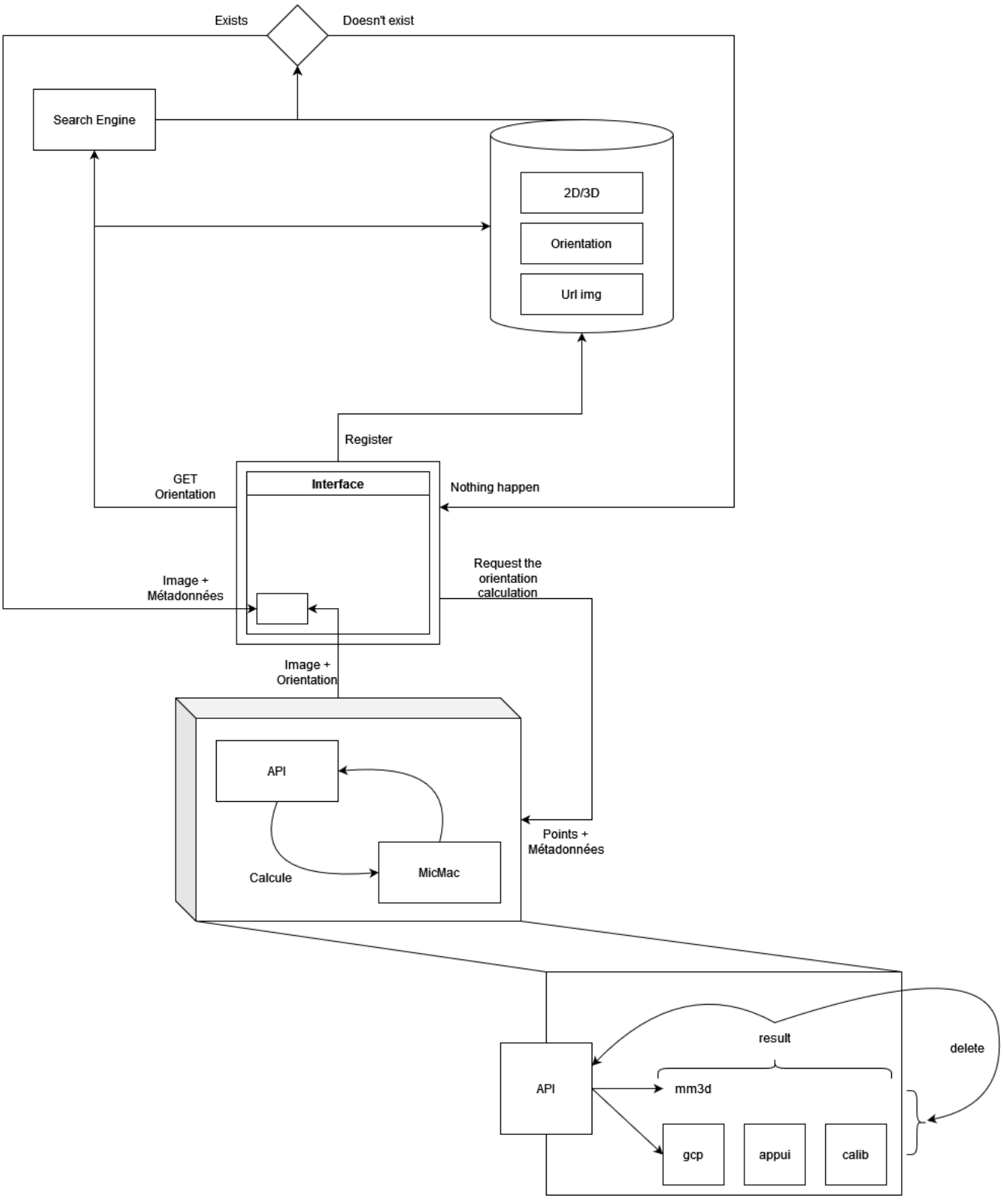
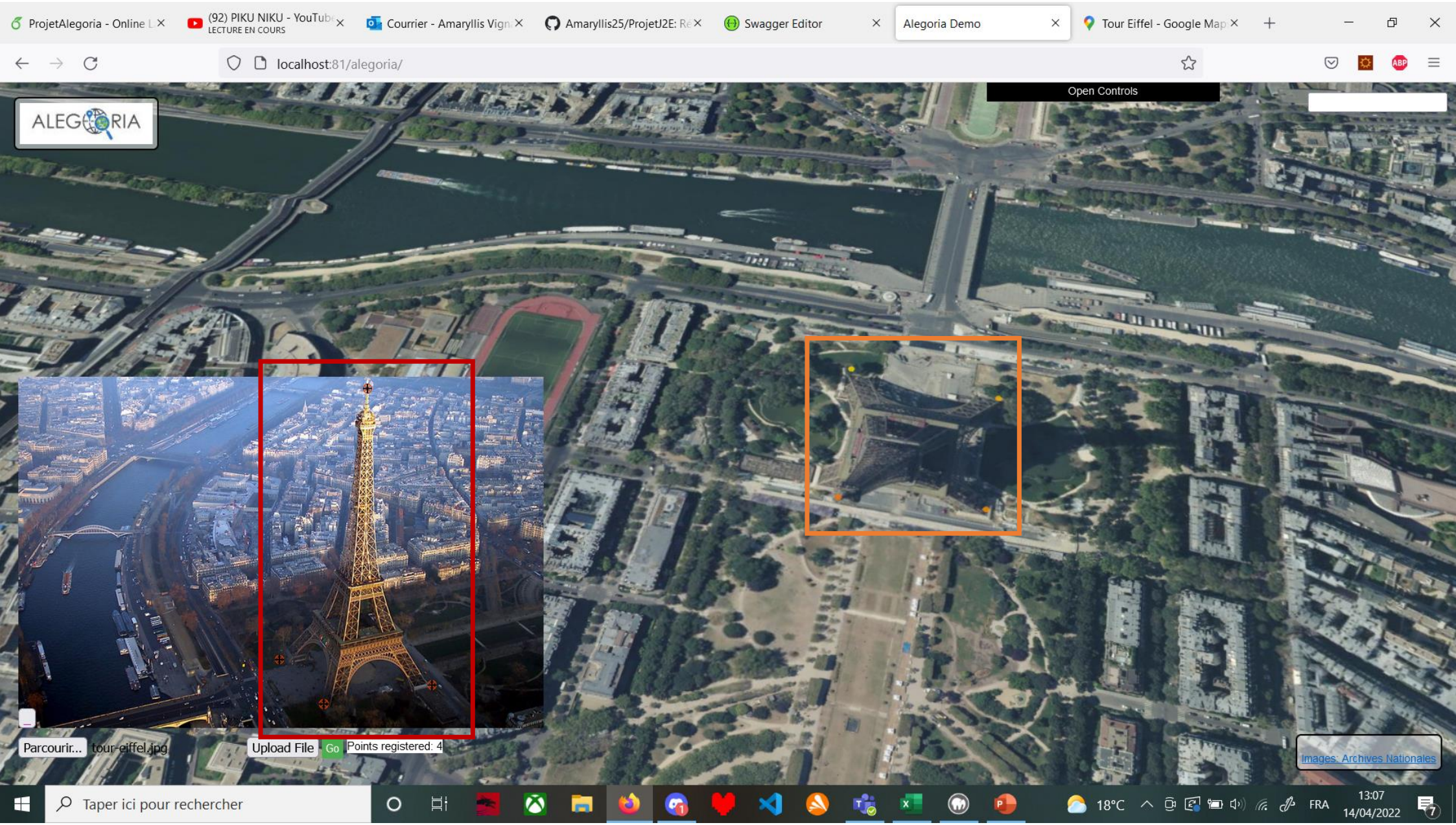


FIGURE 2 – Diagramme de déploiement de la partie cliente


ProjetAlegoria - Online L
(92) PIKU NIKU - YouTube
LECTURE EN COURS
Courrier - Amaryllis Vign
Amaryllis25/ProjetJ2E: Re
Swagger Editor
Alegoria Demo
Tour Eiffel - Google Map
localhost:81/alegoria/
ALEGORIA
Open Controls
Parcourir...
tour-eiffel.jpg
Upload File
Go
Points registered: 4
Images: Archives Nationales
Taper ici pour rechercher
18°C
FRA
13:07
14/04/2022

3.2 Partie serveur

Après nous être intéressé à la partie cliente, nous allons nous intéresser à l'analyse fonctionnelle de la partie serveur. Pour cela nous avons réalisé un modèle conceptuel pour définir les relations entre les différents concepts et entités (voir figure 4). Nous avons également réaliser un schéma relationnel pour modéliser les relations existantes entre plusieurs informations, et de les ordonner entre elles (voir figure 5). Nous avons également réalisé un diagramme de déploiement pour visualiser la relation entre chaque brique de la partie serveur (voir figure 3).

Déploiement de la partie serveur

Dans un premier temps, nous allons commencer par analyser comment les composants de la partie serveur s'imbriquent entre eux :

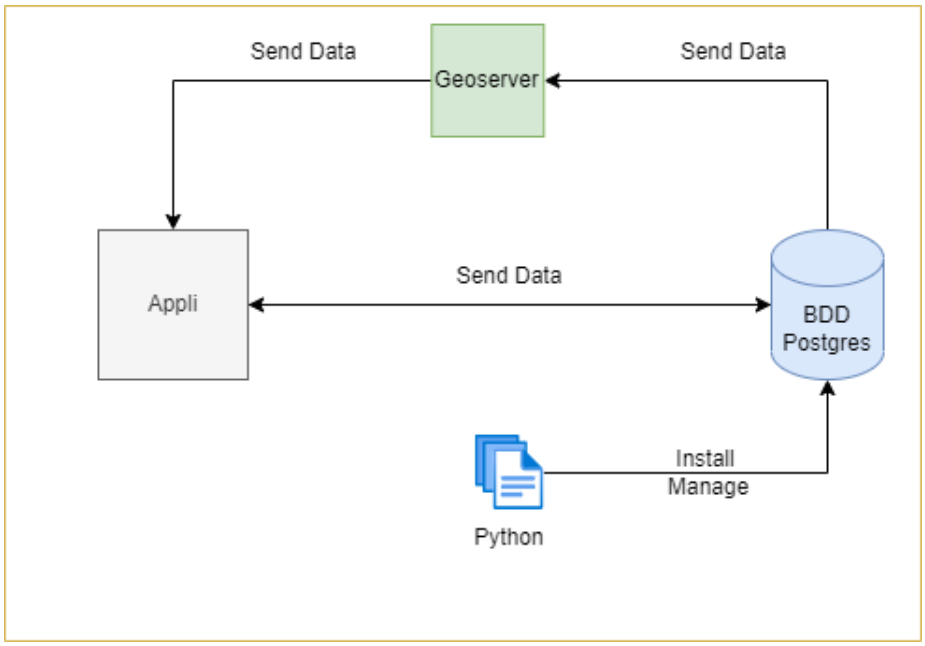


FIGURE 3 – Diagramme de déploiement de la partie serveur

Il faut souligner qu'avant notre arrivée, seule l'application cliente existait ainsi que des petits scripts pour créer une base de données. Cependant, la plupart des données étaient stockées dans des dossiers en local dans l'application client afin de pouvoir les utiliser. Dans notre diagramme, nous avons donc décidé de rajouter trois briques dans la partie serveur, chacun communiquant avec l'application :

- Une base de données : celle-ci va permettre de stocker les données de l'ensemble des images du commanditaire ;
- Des fichiers écrits en python : ceux-ci vont permettre d'effectuer l'entretien de la base de données. Nous en parlerons plus loin ;
- Un GeoServer : celui-ci va interagir avec la base de données. En récupérant les données de celles-ci, il va permettre de créer des "HeatMap" (ou cartes de chaleur). Une fois ces cartes créées, elles seront envoyées sous forme de flux à l'application cliente.

La base de données communique également avec l'application cliente et inversement. En

effet, un utilisateur doit pouvoir retrouver et utiliser des données depuis la base de données, mais l'utilisateur doit également pouvoir ajouter des données à la base, notamment lorsqu'il souhaite créer un géoréférencement ou des points d'appuis pour l'orientation de l'image.

Architecture de la base de données

La base de données, devant être entièrement créée, nous allons regarder plus précisement l'architecture qui a été réfléchit à cet effet (voir figure 5). La base de données a pour but de référencer l'ensemble des images qui pourront être utilisées dans la partie cliente. Elle doit donc référencer chaque image avec les métadonnées associées. Une image va donc posséder un id unique, un temps de pose, un identifiant unique et une taille. Cependant il faut noter que ce ne sont pas les seules informations associées à une image.

En effet, chaque image provient d'une source. Et donc une source peut fournir plusieurs images. Par exemple plusieurs lots de données d'images proviennent de l'IGN. Une image possède également un masque, cependant un masque peut être utilisé pour plusieurs images. De plus chaque image peut posséder un ou plusieurs points d'appuis, en fonction des interactions avec l'utilisateur de l'application. Ainsi, une image va référencer une source et un masque, tandis que la table des points d'appuis va référencer l'image associée.

Chaque image peut posséder un ou plusieurs géoréférencements. De plus chacun de ces géo-référencements est relié à plusieurs paramètres : interne (focal, point principal, déviation...), externe (quaternion..) et de transformation (matrice de transformation...). Ainsi, chaque géo-référencement va référencer une image en particulier et l'ensemble des tables des paramètres associés (pour le géoréférencement et la caméra).

D'autres contraintes s'appliquent à notre architecture :
— Elle doit être évolutive. En effet, la base de données qui existaient auparavant contenait une multitude de tables qui manquait de cohérence. Par exemple l'utilisateur pourrait souhaiter rajouter de nouvelles tables (par exemple la possibilité de référencer les données par lot grâce à une nouvelle table).
— Elle doit être facilement manipulable, et donc cohérente, pour y ajouter facilement des données, en supprimer ou simplement les modifier.

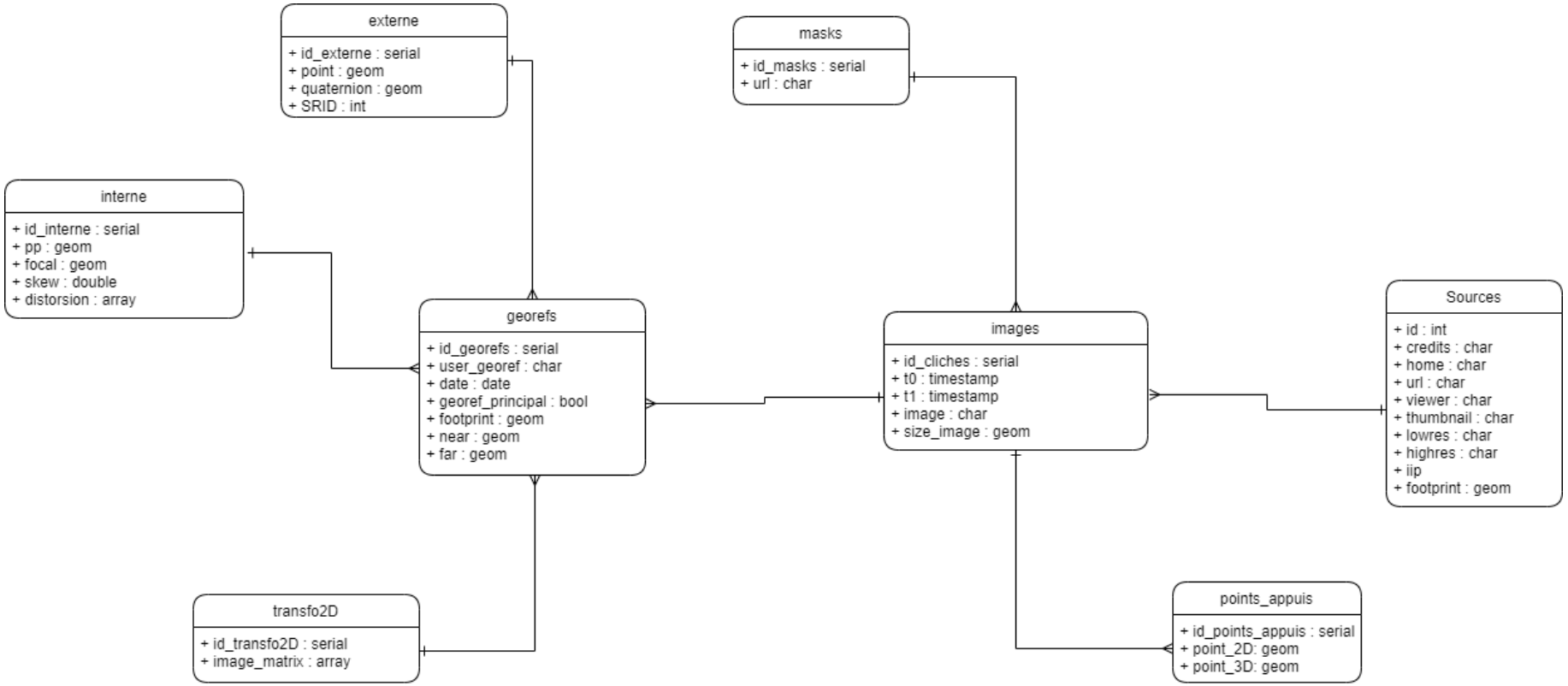


FIGURE 4 – Modèle conceptuel de la base de données

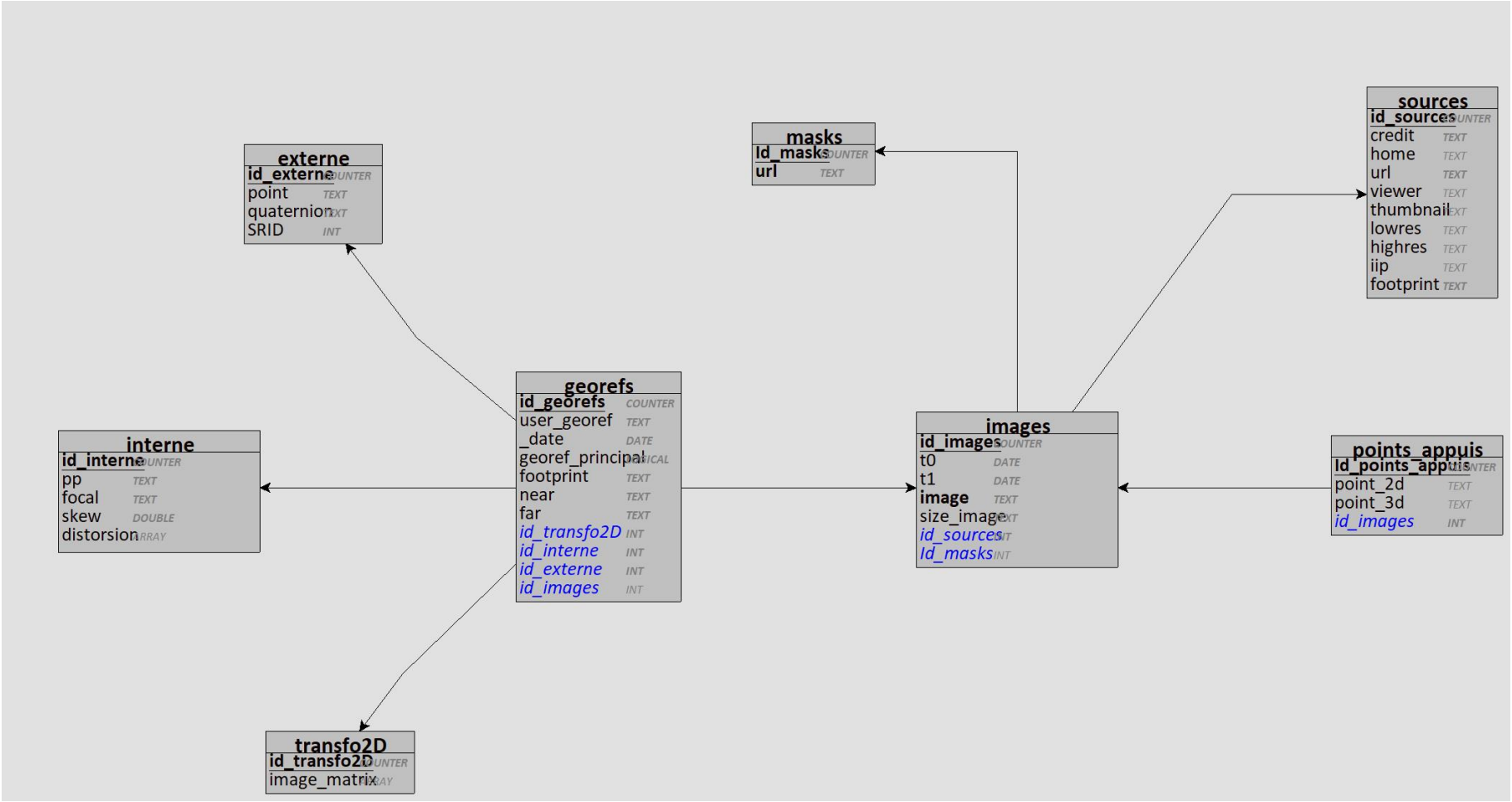


FIGURE 5 – Schéma relationnel de la base de données

Entretien de la base de données

Auparavant, nous avons évoqué l'utilisation de scripts python pour l'entretien de la base de données. Des fichiers SQL intégrant du langage python ont également été créés à cet effet. Nous avons créé un diagramme de cas d'utilisation à cet effet :

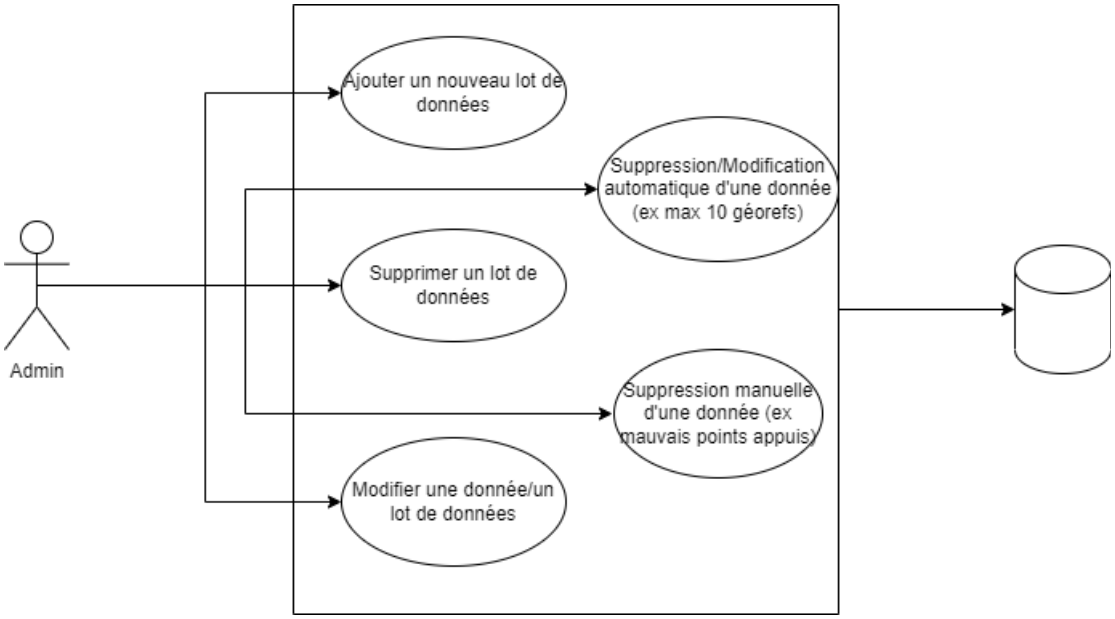


FIGURE 6 – Diagramme de cas d'utilisation

Nous avons détaillé cinq cas d'utilisation de ces scripts avec le diagramme ci-dessus :
- Le cas où l'administrateur souhaite ajouter un nouveau lot de données à la base, c'est-à-dire des données provenant d'une source spécifique ;
- Le cas où l'administrateur souhaite supprimer un lot de données ;
- Le cas où l'administrateur souhaite supprimer une donnée en particulier. Par exemple s'il voit qu'un utilisateur a enregistré des données erronées ;
- Le cas où l'administrateur souhaite modifier une donnée (ou un lot, selon la donnée modifiée) ;
- Le cas où une donnée doit être supprimée/modifiée automatiquement. Par exemple une donnée ne possède qu'un géoréférencement principal. Un autre exemple peut être d'éviter d'avoir un nombre infini de géoréférencement pour une image, ce qui alourdirait beaucoup la base de données.

Un diagramme d'activité (voir figure 8) et un diagramme de séquence (voir figure 7) ont été établit pour expliquer le fonctionnement du processus d'entretien de la base de données avec l'un des cinq cas ci-dessus. Nous avons pris le cas où l'utilisateur souhaite retirer une donnée (les diagrammes des autres cas sont semblables).

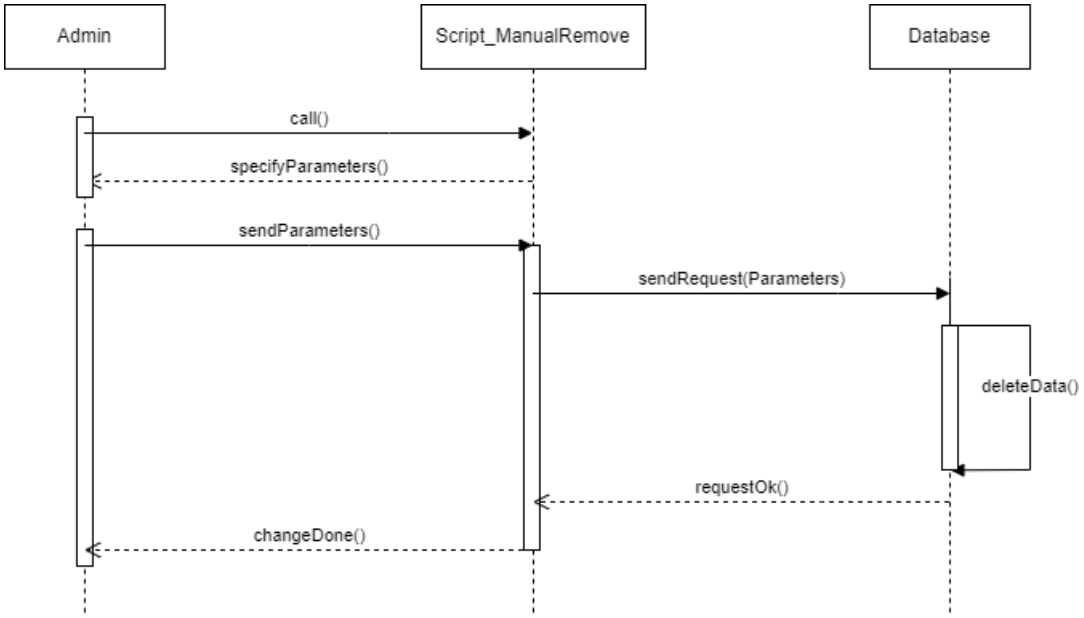


FIGURE 7 – Diagramme de séquence

L'administrateur va utiliser un des scripts d'entretien de la base de données. Pour cela il va appeler le fichier correspondant. Lorsque ce fichier est appelé, le script va lui envoyer un message lui indiquant de spécifier les différents paramètres nécessaires pour pouvoir interagir avec la base de données.

Une fois que l'administrateur a spécifié les paramètres nécessaires à la suppression d'une donnée, le script va envoyer une requête à la base de données avec ceux-ci. La base de donnée va ensuite effectuer la requête pour supprimer une donnée. Une fois la requête effectuée, la base de données retourne un message signifiant que les changements ont bien été effectué. Ainsi, le script pourra dire à l'administrateur que la suppression de la donnée s'est bien déroulée.

Après avoir eu un aperçu des différents messages envoyé entre les différents composants du système, nous allons nous intéresser à la modélisation fonctionnelle de notre système. Les deux diagrammes sont sensiblement similaires dans ce qu'ils représentent, bien que le premier s'intéresse aux flux de messages et le second aux flux entre les activités.

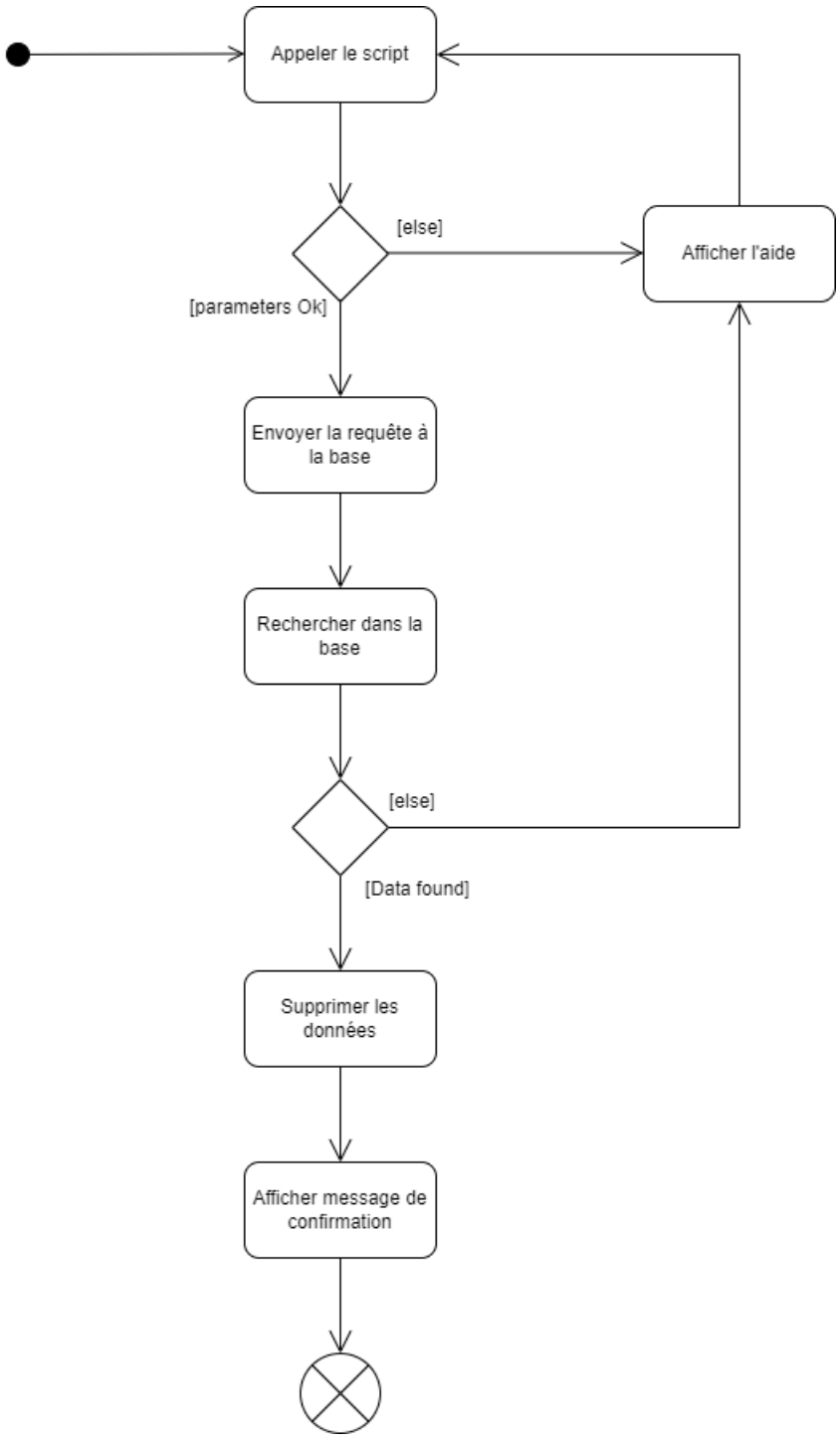


FIGURE 8 – Diagramme d'activité

3.3 Étude technique

Nous allons maintenant évoquer les solutions techniques que nous avons choisi d'utiliser lors de notre projet.

Il faut préciser que lorsque nous avons démarré celui-ci, beaucoup de choses étaient déjà

construites, notamment sur la partie cliente. Nous avons choisi de garder les technologies utilisées dans le projet :

— Pour la partie serveur, la base de données sous PostgreSQL, notamment parce que le commanditaire sait utiliser cette technologie. Cependant, nous avons dû créer l'ensemble des scripts pour la construire et l'administrer ;
— Pour la partie client, nous avons choisi de garder ce qui existait déjà pour les technologies utilisées. Nous avons donc l'utilisation d'iTowns pour l'implémentation de la 3D dans l'application. Il y a également l'utilisation de npm et NodeJS pour la construction d'un partie de l'application. Enfin sous Ubuntu, on utilise le client Apache pour lancer l'application.

Par ailleurs de nouvelles technologies ont été implémentées comme la présence d'une API sous Swagger. L'API fonctionne sous NodeJS. Ainsi le client va interagir avec l'API pour créer les différents fichiers d'orientation MicMac d'une image. L'API a également été créé pour promouvoir l'utilisation de MicMac. En effet, l'utilisateur doit juste rentrer les paramètres nécessaires pour effectuer une commande MicMac, sans pour autant connaître le code qu'il y a derrière. Cela permettra également de promouvoir ce logiciel dédié à la photogrammétrie à tout type de public. De plus, n'importe qui pourrait réutiliser notre API pour sa propre application.

De nouvelles librairies ont également été implémentées comme plpython3 pour utiliser python directement dans PostgreSQL et pouvoir gérer la base de données directement depuis le client pgAdmin.

Au niveau des langages utilisés, le PHP qui était auparavant utilisé pour les fichiers de calibration a été remplacé par du JavaScript par soucis d'uniformisation. Celui-ci a donc disparut du projet.

4 Gestion de projet

Dans cette partie, nous allons parler de la gestion de projet, c'est-à-dire comment nous nous sommes organisés durant celui-ci pour travailler. Nous allons d'abord évoqué la planification du projet avant d'évoquer le rôle de chacun dans celui-ci.

4.1 Méthode de gestion de projet

Lors de notre projet, nous avons choisi d'appliquer les méthodes agiles vues en cours. C'est-à-dire que le projet est découpé sous forme de sprints, qui correspondent chacun à un nombre de jours prédéfinis. De plus, chaque personne dans l'équipe a un rôle prédéfini. Les différents rôles présents dans notre projet sont :

— le Product Owner qui va faire le lien avec le client ;
— le Scrum Master qui va gérer l'équipe de développeur ;
— les développeurs.

A chaque sprint, différentes tâches sont prédéfinies, qui doivent être réalisées dans le temps imparti. Ces tâches sont appelées User Story. Chacune d'entre elle possède un nombre de points correspondant à son niveau de difficulté (appelé User Point). A chaque fin de Sprint, la somme totale des User Point est calculée : la somme de ces points est appelée vélocité et permet de définir la capacité de travail de l'équipe. Nous y reviendrons un peu plus loin.

En appliquant les méthodes agiles, nous avons également différentes réunions à effectuer. Une fois par jour, nous avons fait des Daily Meeting, qui correspondent à un point rapide sur les avancées ou difficultés de chacun. A la fin de chaque sprint, nous faisions :

— Une review pour déterminer si les objectifs du sprint ont bien été rempli ;
— Une retrospective pour déterminer si des actions doivent être prise pour le futur sprint ;
— Un planning, pour définir les tâches a effectué dans le sprint suivant

Ci-dessous, nous présentons le planning type d'un sprint au cours de notre projet :

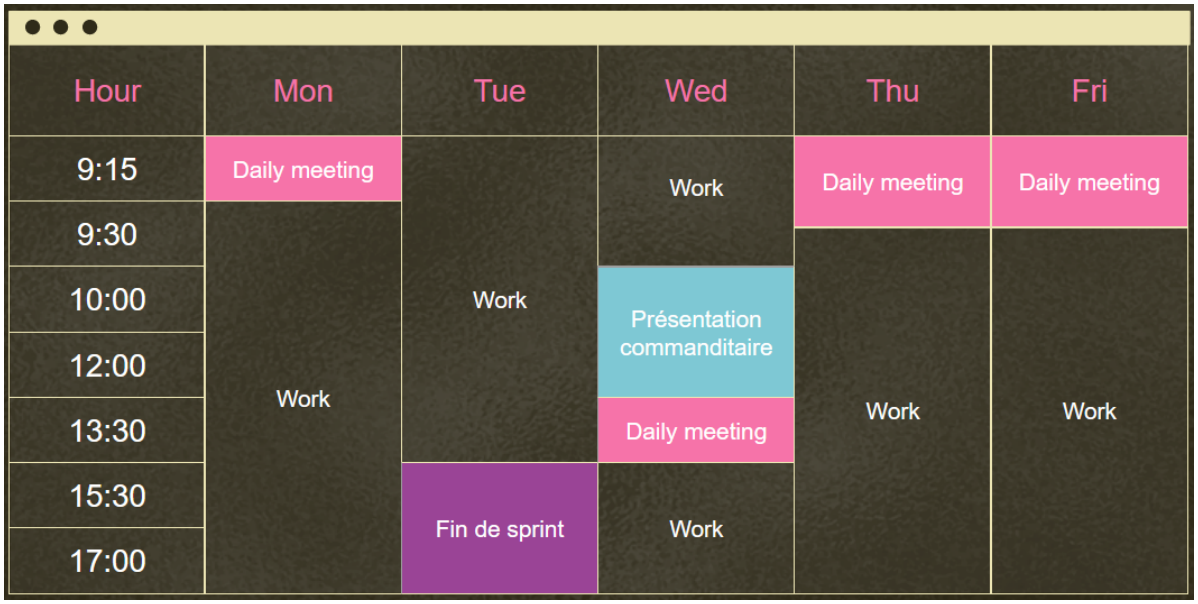


FIGURE 9 – Planning d'un sprint

4.2 Planification

Pour résumé l'ensemble des tâches effectuées lors de notre projet, nous avons réalisé un diagramme de Gantt présent sur la page suivante.

Le projet se déroulant sur six semaines, celui-ci a été découpé en six sprints, chacun avec un objectif différent.

Le Sprint 1 a pour objectif la **prise en main du projet**. Ce sprint avait pour but de prendre contact avec le commanditaire, avec le démarrage de l'analyse de l'existent. Un travail conséquent d'architecture a été démarré, avec l'installation des différents composants du logiciel.

Le Sprint 2 a pour objectif l'**Architecture et documentation**. Le travail d'architecture a été découpé en trois composantes : un pour la base de données, un pour l'API et un pour la partie cliente. Une documentation sur les métadonnées de la base de données a été rédigée en parallèle. Un des développeurs de l'équipe s'est également intéressé à GeoServer pour la création de cartes de chaleur.

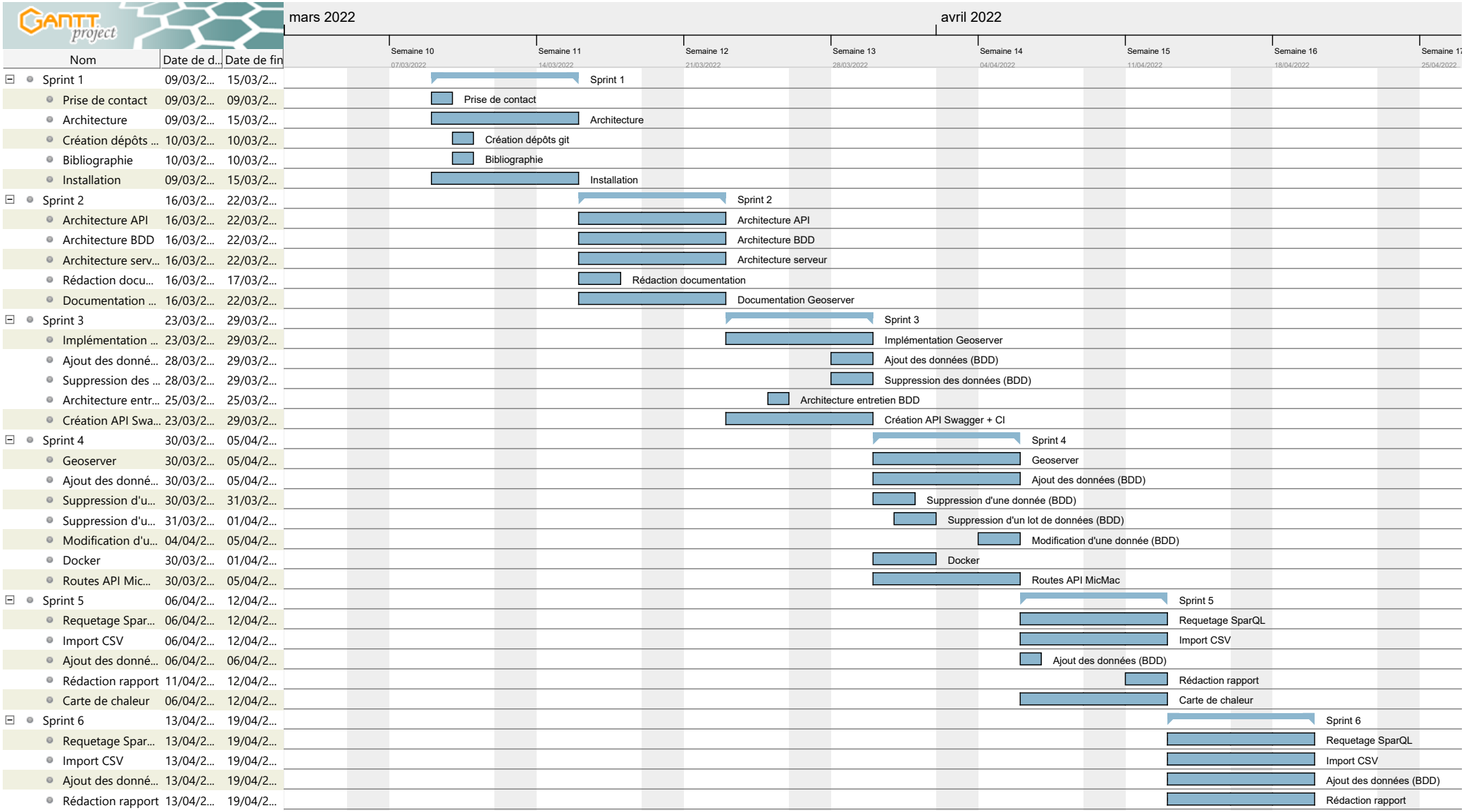
Le Sprint 3 a pour objectif l'**Architecture et implémentation**. Dans ce sprint, un travail approfondi sur GeoServer a été réalisé avec la recherche de requêtes pour la création de cartes de chaleur. L'architecture des fichiers permettant l'entretien de la base de données a également été réalisé. En parallèle, l'API pour MicMac a été implémentée ainsi que différents tests d'intégration continus. Les fonctionnalités d'ajout et de suppression des données ont également commencé à être développé.

Le Sprint 4 a pour objectif l'**Implémentation et dockerisation**. Elle reprend ainsi la suite du sprint précédent avec l'implémentation des fonctionnalités en cours et de nouvelles. Ainsi l'implémentation de l'ajout de données continu, ainsi que la suppression d'une donnée. Les fonctionnalités de suppression d'un lot de données et de modification d'une donnée. Un Docker a également été créé pour MicMac et toutes les routes de l'API ont été implémentées pour lancer les commandes souhaitées par le commanditaire. Enfin, un travail de finalisation pour GeoServer a été effectué : finalisation des requêtes SQL sous GeoServer et création de flux sont nécessaires pour afficher les cartes de chaleur.

Le Sprint 5 a pour objectif l'implémentation de **Nouvelles fonctionnalités**. Le commanditaire souhaite pouvoir récupérer des données depuis l'outil Search Engine d'Alegoria pour pouvoir les mettre dans la base de données. Pour cela, il faut créer un script permettant leur import. Le rapport doit également commencé à être rédigé. Le GeoServer étant implémenté et les cartes disponibles sous forme de flux, il faut également pouvoir les importées dans le client. L'ajout des données étant complexe, la User Story est toujours présente lors de ce sprint.

Le Sprint 6 a pour objectif la **finalisation du projet**. Les tâches précédentes qui sont toujours en cours doivent être finalisées. Un septième sprint sur trois jours a permis de finaliser l'ensemble des tâches en cours, de rédiger l'ensemble de la documentation nécessaires et de réaliser nos présentations.

Diagramme de Gantt



Nom	Date de d...	Date de fin
Sprint 1	09/03/2...	15/03/2...
Prise de contact	09/03/2...	09/03/2...
Architecture	09/03/2...	15/03/2...
Création dépôts ...	10/03/2...	10/03/2...
Bibliographie	10/03/2...	10/03/2...
Installation	09/03/2...	15/03/2...
Sprint 2	16/03/2...	22/03/2...
Architecture API	16/03/2...	22/03/2...
Architecture BDD	16/03/2...	22/03/2...
Architecture serv...	16/03/2...	22/03/2...
Rédaction docu...	16/03/2...	17/03/2...
Documentation ...	16/03/2...	22/03/2...
Sprint 3	23/03/2...	29/03/2...
Implémentation ...	23/03/2...	29/03/2...
Ajout des donné...	28/03/2...	29/03/2...
Suppression des ...	28/03/2...	29/03/2...
Architecture entr...	25/03/2...	25/03/2...
Création API Swa...	23/03/2...	29/03/2...
Sprint 4	30/03/2...	05/04/2...
Geoserver	30/03/2...	05/04/2...
Ajout des donné...	30/03/2...	05/04/2...
Suppression d'u...	30/03/2...	31/03/2...
Suppression d'u...	31/03/2...	01/04/2...
Modification d'u...	04/04/2...	05/04/2...
Docker	30/03/2...	01/04/2...
Routes API Mic...	30/03/2...	05/04/2...
Sprint 5	06/04/2...	12/04/2...
Requetage Spar...	06/04/2...	12/04/2...
Import CSV	06/04/2...	12/04/2...
Ajout des donné...	06/04/2...	06/04/2...
Rédaction rapport	11/04/2...	12/04/2...
Carte de chaleur	06/04/2...	12/04/2...
Sprint 6	13/04/2...	19/04/2...
Requetage Spar...	13/04/2...	19/04/2...
Import CSV	13/04/2...	19/04/2...
Ajout des donné...	13/04/2...	19/04/2...
Rédaction rapport	13/04/2...	19/04/2...

4.3 Vélocité

Précédemment, nous avons défini la notion de vélocité, définissant la quantité de travail moyen qui peut être effectué lors d'un sprint :

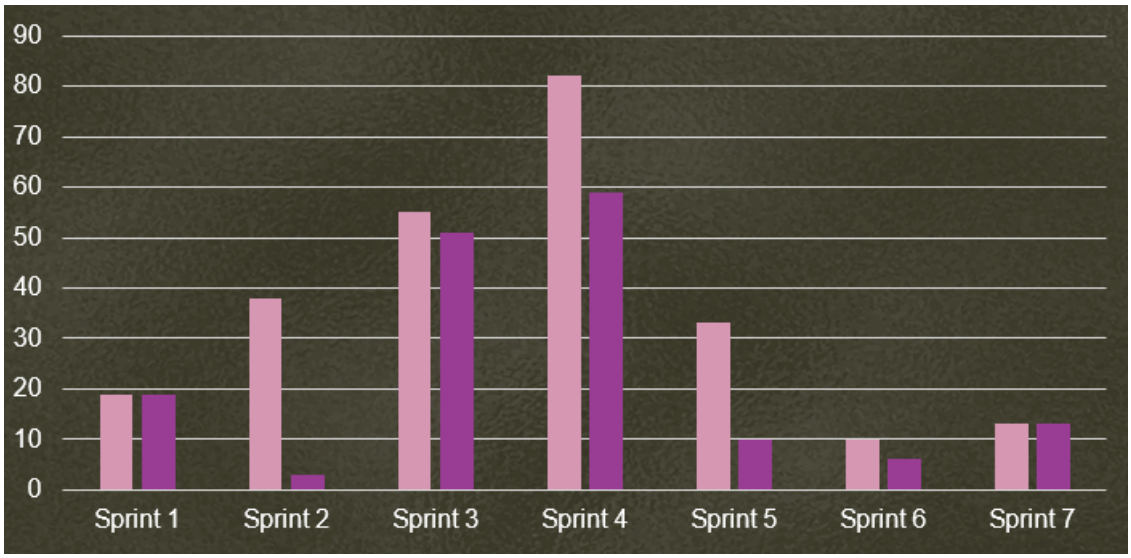


FIGURE 10 – Diagramme de vélocité au cours du projet

Lors du premier sprint, nous avons réussi à faire les User Storys désirées, sans pour autant réussir à en faire plus. Cela s'explique notamment par le fait que lors de cette semaine, nous avons pris en main le projet. Lors du second sprint, nous pouvons constater que nous avons très peu de User Point comparé au prévision de début de sprint. Cette semaine a été un moment de flou sur ce que souhaitait réellement faire le commanditaire, et c'est ce qui nous a fait perdre du temps.

A partir du troisième sprint, l'équipe a fait un bon impressionnant de productivité, ayant totalisé plus de 50 points en fin de sprint. Ce résultat a pu être produit après que nous ayons compris ce que souhaitait réellement le commanditaire. Cela s'explique aussi par le fait que nous avons pu commencer à coder. Nous avons presque atteint l'objectif que nous nous étions fixé. Au quatrième sprint, nous avons peut-être été un peu trop ambitieux : nous nous sommes fixé pour objectif d'atteindre plus de 75 points, ce qui après coup nous semble beaucoup. Nous avons réussi à effectuer au total près de 60 points.

Au cinquième sprint, le nombre de user points est tombé relativement bas. Une des fonctionnalités majeures a pu être réalisées pendant ce sprint, cependant d'autres User Storys assez conséquentes ont été établit, qui nous ont demandé plus de travail que prévu. Au sixième sprint, nous avons choisi de continuer les User Story en cours. De plus, nous avons dû en parallèle rédiger le rapport à rendre et sur le début de la préparation de l'ensemble des rendus du rapport. Il ne s'agit de taches difficiles en soit mais elles peuvent se révéler longues. Le dernier sprint correspondant à celui de la fin du projet, l'ensemble des fonctionnalités en cours ont pu être finies, et l'ensemble des rendus également.

4.4 Rôles

L'application des méthodes agiles nécessite la répartition des rôles au sein de notre équipe. Les rôles ont été répartis entre les cinq membres de l'équipe, et ceux-ci sont restés fixent tout au long du projet :

- Le rôle de Product Owner a été attribué à Valentin ;
- Le rôle de Scrum Master a été attribué à Mélodia ;
- Les autres membres de l'équipe, Amaryllis, Axelle et Thomas ont tenu le rôle de développeur.

4.4.1 Apport personnel

Amaryllis Vignaud :

Le projet m'a beaucoup apporté, aussi bien sur le plan professionnel que relationnel. Il m'a permis de créer du lien avec des camarades, mais également de mettre en oeuvre tout ce que nous avons pu voir depuis le début de l'année. J'ai pour ma part beaucoup participé à la mise en place de la base de données avec Mélodia, et ai par ailleurs réalisé un bon travail d'architecture sur celle-ci, comme sur la partie cliente. J'ai beaucoup apprécié le travail en méthodes agiles, avec les points réguliers et les User Storys, qui permettaient de se fixer des objectifs réalisables. C'est aussi une première expérience avec un commanditaire et de se rendre compte la communication est importante. Le sujet me semblait bien complexe au début, mais au fur et à mesure des discussions, tout s'éclaircissait. J'ai grandement apprécie travailler avec mon équipe, il y a toujours eu une bonne ambiance dans celle-ci tout au long du projet, ce qui s'avère motivant pour travailler.

Axelle Gaigé :

De manière générale, ce projet m'a permis de développer mes compétences professionnelles. De plus, il m'a appris à prendre de la distance, en effet il y avait beaucoup de travail à réaliser et nous ne pouvions donc pas tout faire parfaitement. Ce projet m'a également obliger à approfondir des domaines vers lesquels je ne me serai pas forcément dirigés : j'ai notamment pu faire du SPARQL ou bien utiliser le framework iTowns.

Mélodia Mohad :

Ce dernier projet d'école m'a aidé à enrichir mes compétences professionnelles et relationnel. J'ai pu redécouvrir la méthode agile sur un projet plus long et en apprécié ses bien fait. En effet grâce au point régulier avec le commanditaire nous avons pu nous approprier au mieux le projet et essayer de lui proposer une solution qui colle à ces attentes. Les daily meeting ont qu'en a eux participer à maintenir une cohésion de groupe et permis je pense à chacun de garder une vue global du projet tout en travaillant sur des points spécifique. J'ai pour ma part travailler sur l'architecture de la BDD avec Amaryllis. Ce fut ma partie préféré de ce projet malgré les quelques soucis de compréhension nous avons su je trouve travailler à deux de manière optimal avec un résultat dont nous sommes contente. La deuxième partie du travaille sur la BDD fut d'implémenter les fonctionnalités permettant sa gestion. J'ai eu plus de mal pour cette partie car j'ai du faire face à plusieurs petit problème qui ont ralentit l'implémentassions d'une fonctionnalité qui me semblait simple.

Ce projet conclut mes 3 ans d'étude à l'ENSG et notamment mon années de spécialisation. Je n'imagine pas meilleur équipe pour avoir mener à bien ce projet. Je suis fière de nous et de notre travaille.

Thomas de Beaumont :

Ce projet fut très enrichissant pour moi autant dans le domaine professionnel que personnel. C'était la première fois que nous étions confronté a la mise en place et au déploiement complet d'une application web et cela m'a appris a essayer d'avoir un point de vue global sur les différentes briques de l'application ainsi que sur les interactions entre elles. J'ai aussi pris en compétence de multiples domaines : NodeJS, PostgreSQL, SPARQL, Docker... Du coté personnel j'ai beaucoup appris de notre équipe, toujours de bonne humeur et travaillant dans la bonne ambiance quels que soient les obstacles auxquels nous avons put être confrontés.

Valentin Mathiot :

Ce projet de groupe m'a beaucoup appris tant sur le plan social que professionnel. Nous avons déjà réalisé au cours de cette année un projet de groupe mais celui actuel, s'étalant sur une plus longue durée, m'a permis d'avoir une première approche sur un projet au sein d'une entreprise. Au sein de cette équipe dynamique et agréable, j'ai pu monter en compétence et approfondir mes connaissances. Nous avions eu tout de suite une bonne organisation de travail ce qui a largement aidé sur toute la suite du projet. De mon coté, j'étais davantage à travailler sur la partie client et notamment sur la partie GeoServer. L'ensemble des compétences que j'ai pu acquérir que ca soit en programmation ou sur l'organisation de travail, seront des outils précieux pour mon parcours professionnel.

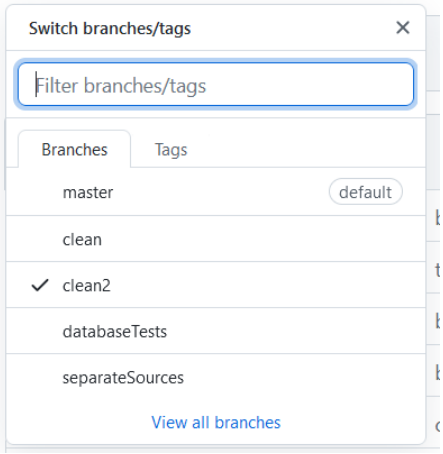
5 Difficultés rencontrées

Lors de notre projet, nous avons rencontré un certain nombre de difficultés. Ces difficultés ce sont aussi bien présentées au début du projet, que lors de sa réalisation après quelques semaines.

5.1 Début de projet

Dépôts GitHub

Au début de notre projet, après la rencontre avec le commanditaire, celui-ci nous a donné accès à quatre dépôts sur GitHub. Cela représente déjà un nombre assez conséquent de dépôts.



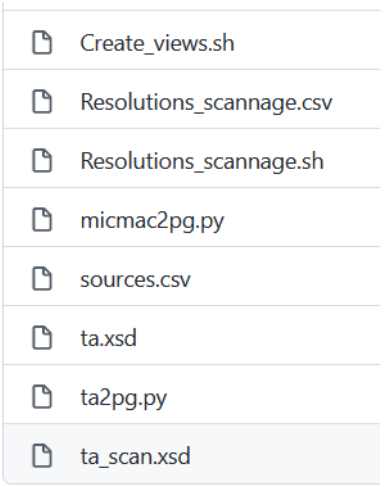
Nous avons rencontré deux problèmes majeurs sur l'ensemble de ces dépôts :

- Aucune documentation claire n'a été rédigée, et ce sur aucun des dépôts GitHub. Nous avons donc dû trouver nous mêmes comment effectuer les diverses installations ;
- Ces dépôts ne sont pas nettoyés. Il y a sur certains un nombre conséquent de branches et nous avons dû trouver lesquelles utiliser.

Documentation

Pour la documentation, il n'y avait aucune précision de comment les différentes dépôts s'imbriquaient les uns dans les autres. Il y a trois dépôts à récupérer pour construire l'application partie cliente, avec plusieurs installations à effectuer en parallèle. Par exemple pour l'un des dépôts, il fallait reconstruire le projet avec npm mais cela n'était pas précisé. Nous avons donc perdu du temps sur les diverses installations à effectuer.

Côté serveur, il n'y avait qu'un seul et unique dépôt. Cependant celui-ci ne disposait d'aucune documentation claire sur comment construire la base de données ou encore à quoi correspondent les différents attributs de celles-ci. Il n'y a également eu aucun travaille de réflexion sur l'architecture de celles-ci, et nous n'avions donc pas accès à des diagrammes pour mieux comprendre sa structure. Il faut également noté que pour créer la base de données, il y avait un grand nombre de fichiers différents, avec des noms parfois peu évocateur comme le montre l'image ci à droite. De plus ceux-ci n'étaient pas tous utiles à la construction de la base de données.



Nous pouvons donc dire qu'il y avait un vrai manque d'organisation et de documentation dans les dépôts la première fois que nous les avons utilisé.

Au cours du projet, d'autres soucis se sont posés à cause du manque de documentation.

Par exemple, le virtuoso pour requêter les données du projet Alegoria n'était pas documenté clairement.

5.2 Temps de chargement

Un autre soucis rencontré au cours du projet est lié à l'optimisation de l'existant. En effet le temps de chargement des tuiles du fond 3D d'iTowns est relativement long.

5.3 Communication

Au cours du projet, nous avons rencontré quelques difficultés de communication avec notre commanditaire. En effet, notre commanditaire étant parfois occupé, nous étions bloqué sur l'avancement de notre projet, notamment lorsque nous avons voulut effectuer les premières installations de l'application.

D'autres problèmes ont été rencontrés avec le commanditaire, notamment le fait que nous n'avons pas compris exactement ce que souhaitait faire le commanditaire avec son application. Ainsi, nous avons dû revenir au moins trois fois sur l'architecture de la base de données avant que celle-ci soit bonne pour notre commanditaire.

5.4 Données

Un des soucis majeurs de ce projet est les données fournies par le commanditaire. Le problème est que les données proviennent de sources différentes, et donc que les métadonnées ne sont pas écrites de la même manière entre les fichiers. Cela complique les imports pour la base de données : il faudrait un script par sources différentes, et par type de fichiers (xml, csv...). Au départ, le choix a été fait de ne se concentrer que sur les données de l'IGN pour les données au format xml, celles-ci étant nombreuses. Cependant, au sein même de ces données les attributs changeant parfois de nom, ou la structure changeant, cette idée a été abandonnée en fin de projet. Le choix de ne créer qu'un import de données via des csv bruts ici du Search-Engine d'alegoria a été fait.

5.5 Logiciels

5.5.1 GeoServer

Le manque de documentation sur la base de données a été le principal obstacle à l'utilisation de GeoServer. La documentation de GeoServer est assez complète, en plus que le logiciel soit simple d'utilisation. Il faut tout de même un minimum de temps d'apprentissage et d'adaptation pour pouvoir se servir des outils (comme tout finalement).

Il faut également souligner que la réalisation de cartes de chaleurs reste une situation bien spécifique ou un cas particulier d'utilisation de GeoServer, par conséquent la recherche devient longue et chronophage.

5.5.2 PostgreSQL

PostgreSQL est le logiciel utilisé pour la base de données de notre projet. Un des reproches qu'on peut faire à ce logiciel est le manque de documentation, et notamment d'exemple.

Ceci est d'autant plus vrai avec l'utilisation de la librairie plpython3 permettant d'intégrer le langage python aux scripts SQL. Il y a peu de documentation sur le sujet. L'utilisation de la

librairie est donc complexifiée sur certains points (récupération d'une requête SQL dans le script python etc.).

5.5.3 Versions

Pour les différentes installations nous avons rencontrés des problèmes de versions. Par exemple, la construction d'un des projets sur les dépôts GitHub nécessite une version de NodeJS supérieure à 14. Un autre exemple est l'installation de la bibliothèque plpython3 qui nécessite une version de PostgreSQL supérieure ou égale à 12. Cependant plpython3 n'est pas directement intégré pour les versions de PostgreSQL supérieure à 13 et nécessite l'installation d'un connecteur.

5.6 Gestion de projet

Pour finir avec les difficultés, nous allons parler de la gestion de projet. Au début les User Story étaient parfois beaucoup trop volumineuses et les points attribués à celles-ci n'étaient pas correct : nous nous sommes ainsi retrouvés avec des grosses User Story, ou alors avec des User Story sous notés. Cependant, cela s'est affiné au fur et à mesure du projet.

Au début du projet, il y a eu également quelques petits soucis sur la rédaction des User Story qui n'étaient pas assez détaillés. Mais comme pour le reste, cela s'est également affinés au fur et à mesure que nous pratiquions et avec la prise d'action lors de la rétrospective.

6 Réalisation

De nombreuses missions ont été réalisées lors de notre projet, et ce à tous les niveaux : de la documentation, la création de la base de données, de l'API ou encore du GeoServer. L'application cliente a également été modifiée.

6.1 Documentation

Un travail de documentation conséquent a été réalisé dans le projet. En effet, celle-ci étant inexistante à la base malgré la réalisation de nombreuses briques de l'application, nous avons dû prendre du temps pour la réaliser. Nous avons ainsi réalisé au cours de notre projet :

- Un guide d'installation de chaque brique de l'application, de la partie cliente à la partie suivante, en passant par celle ce l'API ;
- Une documentation claire des attributs de la base de données, disponible en annexe ;
- Une documentation sur la structure de la base de données et de la partie cliente ;
- Une documentation sur les différents scripts d'entretien de la base de données et comment les utiliser.

Il faut noter que toute la plupart de la documentation a été écrite en anglais, en accord avec le souhait du commanditaire.

6.2 Base de données

De nouveaux scripts pour créer la base de données et l'entretenir ont été créé :

- Un script python de création des tables et attributs de la base de données. Ce script s'inspire de certains fichiers déjà présent pour créer une ancienne base ;
- Un script d'ajout d'un lot de données ;
- Un script SQL intégrant du python pour supprimer une donnée ou un lot de données, ou encore pour modifier une donnée ;
- Un script python pour l'ajout de données csv récupéré à partir d'une requête SPARQL sur le virtuoso d'Alegoria.

Mise en place de la base de données

Avec le guide d'installation, l'utilisateur a auparavant créé une base de données nommée alegoria, avec les nom d'utilisateur et mot de passe indiqué. Une fois le script lancé en ligne de commande avec les bons paramètres, les différentes tables et attributs vont s'ajouter à la base de données ainsi que les extensions nécessaires (si ceux-ci ne sont pas déjà créé). Le projet utilisant des données géographiques, il est nécessaire d'ajouter l'extension PostGIS à notre base de données.

Ajout d'un lot de données

L'ajout d'un lot de données, quelque soit le format, nécessite une réflexion avant sa mise en place. En effet, les différentes tables de la base de données doivent être ajoutées dans un ordre précis en fonction des différentes dépendances :

- Avant toute chose, la source de la donnée doit être ajoutée, si elle n'existe pas déjà dans la base de données ;
- Dans un second temps, les images doivent être ajoutées dans la base de données

— Pour ajouter un géoréférencement par défaut à une image, il faut auparavant ajouter les tables interne, externe et transfo2d, correspondant aux différents paramètres de la caméra. Il faut donc les instancier avant de créer un géoréférencement
— Enfin, les géoréférencements, s'ils n'existent pas déjà, peuvent être ajoutés.

Ajout d'un lot de données au format xml

Des fichiers aux formats xml nous ont été fournis au début du projet. Ces fichiers sont des fichiers de métadonnées sur des images en majorité issues du fond IGN. Ils permettent d'identifier les images mais également de leur donner une première orientation calculée par photogrammétrie.

Le volume de fichiers étant important nous avons écarté la possibilité d'importer les données manuellement. Ces fichiers couvrant une large période dans le temps (1919-2005) certains xml ne disposent pas de toutes les informations nécessaires ou ne respectent pas le même format. Nous avons donc développé une fonction assez générale pour permettre leur import.

Un travail a été réalisé, mais la fonctionnalité n'a pas aboutit, dû à la complexité des structures des xml qui nécessiterait un temps important de travail.

Ajout d'un lot de données au format csv

Des fichiers au format csv sont récupérables à l'aide de requêtes SPARQL, dont nous parlerons plus loin. Ces fichiers, grâce à un script python, peuvent être ajoutés à la base de données en suivant la logique dont nous avons parlé précédemment. Le fichier a été codé de tel sorte qu'il est impossible de réimporté des données déjà présentes dans la base.

Cependant, il faut préciser que l'ensemble des métadonnées nécessaires ne sont pas présentes dans les données csv, notamment concernant les paramètres de la caméra. Ainsi, il manque :
— Les paramètres internes de la caméra : ceux-ci ont été définit par défaut par le commanditaire ;
— Les paramètres externes de la caméra : le point central de la caméra doit être géocodé à partir du code insee. Le reste des paramètres par défaut doit être calculé par une autre alternative. MicMac reste une solution envisagée
— La matrice de transformation de la caméra. L'utilisation de MicMac est aussi envisagée pour la calculer.

Suppression d'une donnée ou d'un lot de données

Une fois le script lancé sur l'interface pgAdmin, la base de donnée intègre une nouvelle fonction PostgreSQL dans la base de données, permettant la suppression d'une donnée avec les bons paramètres d'entrées. La même action doit être effectuée pour ajouter la fonction de suppression d'un lot de données.

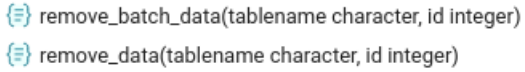


FIGURE 11 – Fonctions de suppression dans pgAdmin

Les deux fonctions prennent en entrée l'id de la donnée à supprimer. Cependant chacune des fonctions effectuent un type de suppression différent. La fonction remove_data permet de supprimer n'importe quelle donnée dans la base, en prenant soin de supprimer les dépendances associées. Par exemple lorsqu'une image est supprimée, il ne faut pas oublier de supprimer ses géoréférencements. Cependant cette fonction ne permet pas de supprimer une source, qui est ici considérée comme un lot de données.

Pour supprimer un lot de données (cf une source), remove_batch_data est la bonne fonction à utiliser. Le choix a été fait de demander en entrée la table à utiliser pour être sûre que l'administrateur n'opère pas une suppression par erreur sur une autre table. De plus, la suppression d'une lot de données nécessite la suppression d'absolument toutes les données associées : les images, les géoréférencements, les paramètres de la caméra, les points d'appuis etc. Sachant que ces suppressions doivent s'opérer dans un ordre précis pour éviter toute erreur (suppression des clés étrangères...).

Modification d'une donnée

Pour modifier une donnée, il faut également le lancer dans le client pgAdmin pour ajouter les différentes fonctions de modification de la base de données.

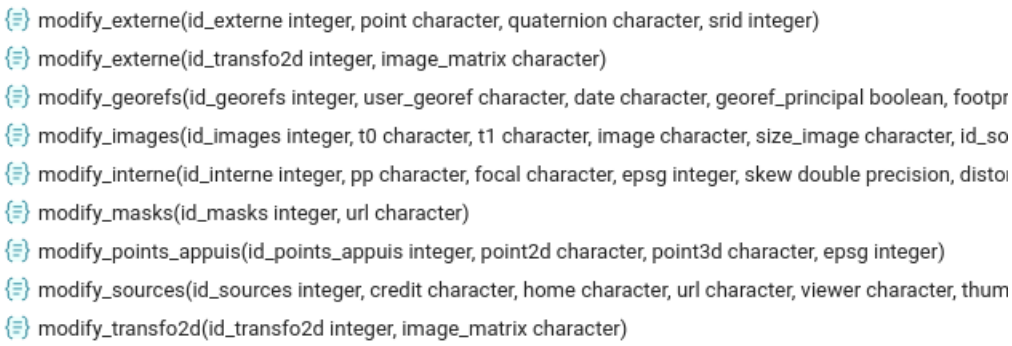


FIGURE 12 – Fonctions de modification d'une donnée dans pgAdmin

Pour chaque table, une fonction de modification de la donnée a directement été créé. Il suffit à l'administrateur de renseigner quelle variable il souhaite modifier pour que celle-ci s'effectue. Par exemple, en tant qu'administrateur, je souhaite modifier l'identifiant unique d'une image qui s'avère être erroné. Je dois donc utiliser la fonction modify_images telle que :

```
SELECT modify_images(id_images => 4, image => '''UnAutreIdentifiant''');
```

FIGURE 13 – Exemple de modification d'une donnée

En tant qu'administrateur, je dois donc obligatoirement indiquer l'id de l'image que je souhaite modifier, et indiquer que c'est à l'attribut image que je souhaite affecter une nouvelle valeur. Dans ce cas seule cet attribut sera mis à jour. En tant qu'administrateur, on peut également modifier l'ensemble des attributs d'une donnée. Ces fonctions permettent de moduler les modifications en fonction du besoin administrateur.

6.3 Requête SPARQL sur Search Engine

Les métadonnées des images du projets Alegoria sont accessibles sur un virtuoso où des requêtes peuvent directement être exécutées (http://data.alegoria-project.fr/SPARQL) . Afin de récupérer, les métadonnées nécessaires pour compléter la base de données, nous avons écrit une requête qui permet de récupérer : l'URI de la photographie, le nom du fichier, le titre, la date, le toponyme et le code insee de la commune où a été pris le cliché, ainsi que le WKT.

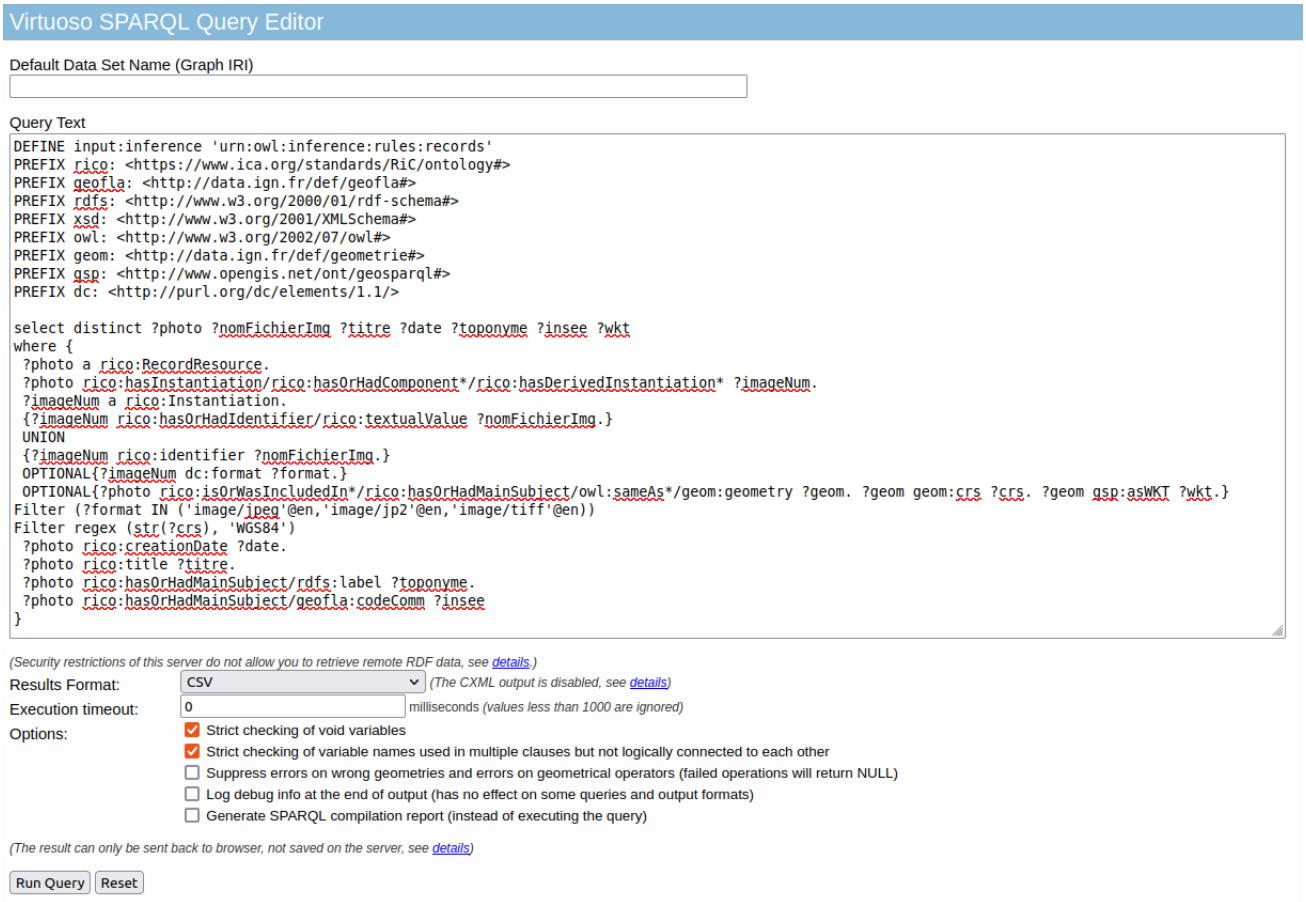


FIGURE 14 – Requête SPARQL pour obtenir les métadonnées

6.4 GeoServer

Afin d'afficher la carte de chaleur des clichés, nous avons utilisé GeoServer pour créer et diffuser la couche correspondante. Dans un premier temps la base de données PostGIS a été ajoutée au GeoServer. Puis nous avons créé une vue qui représente les centroïdes de toutes les emprises des clichés présents dans la base. Cette vue a été publiée afin d'être diffusée par flux WMS. De plus, pour obtenir une carte de chaleur à partir du semis de points, nous avons ajouté un style au GeoServer que nous avons appliqué à la couche.

Nous avons ensuite adapté le code de l'application web pour pouvoir afficher la carte de chaleur sur le globe iTowns. Par défaut la carte n'est pas visible et l'opacité de la couche est fixé à 0,6. Ces paramètres peuvent être modifiés dans le menu de contrôle.

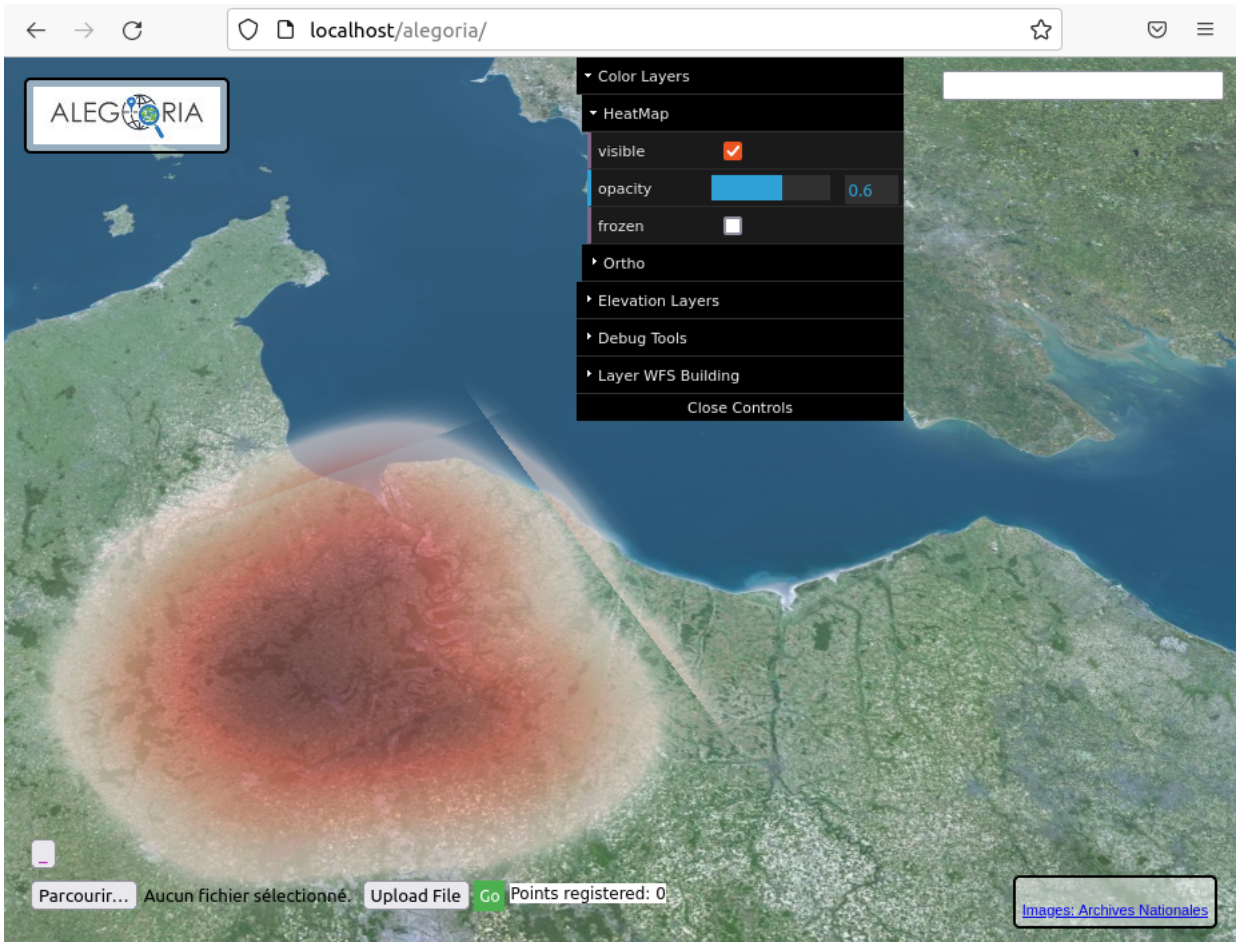


FIGURE 15 – Exemple de carte de chaleur

6.5 API

Une API pour effectuer les traitements MicMac et créer les fichiers nécessaires à ces derniers a été développée. L'objectif était d'avoir une API généraliste qui puisse être utilisée par d'autre applications. Nous avons choisi de faire une application Express sous Node.

Les routes *calib*, *point2d* et *point3d* permettent respectivement d'obtenir le xml de calibration, des points d'appuis 2d et 3d. La route aspro, quant à elle, lance la commande aspro qui permet de calculer l'orientation de l'image. Les xml nécessaires à ce calcul sont fournis dans le corps de la requête. Cette route renvoie le xml du fichier orientation calculé.

FIGURE 16 – Routes implémentées pour l'API MicMac

6.6 Docker

Dans ce projet, Docker sert à déployer les différentes briques décrites précédemment. Pour le fonctionnement de l'application, nous avons prévu une architecture en 4 conteneurs communiquants les uns avec les autres : *le client*, *la base de données*, *l'API* et *le GeoServer*

Pour mettre cela en place, il a fallut créer des images personnalisées pour certaines briques, notamment celle de l'API qui doit contenir un serveur NodeJS et l'application MicMac pour lancer les calculs. Pour effectuer cela, nous sommes partis d'une image MicMac a laquelle nous avons ajouté NodeJS pour faire fonctionner l'API. Ainsi lorsque l'on déploie un conteneur avec cette image, on peut avoir accès à l'API qui écoute sur le port 3000 du conteneur, celle-ci pouvant lancer MicMac lorsque cela est nécessaire.

Pour ce qui est de la partie cliente et du GeoServer, la mise en place est plus simple étant donné qu'une image docker pour le GeoServer existe déjà et qu'il suffit de la configurer comme on le souhaite. Et que pour la partie client, une image Apache avec le code source du site en volume suffit a la faire fonctionner.

Pour la dernière brique, n'ayant pas eu le temps de nous plonger précisément dessus, nous n'avons pas d'image fonctionnelle à proposer. Cependant l'idée était de partir d'une image PostgreSQL existante et de la modifier pour créer les tables et la autorisations relatives au projet alegoria pour avoir un conteneur déjà fonctionnel au lancement de l'application.

6.7 Partie cliente

Bien que nous n'ayons pas eu le temps de nous focaliser sur la partie cliente autant que sur les autres composantes de l'application, nous avons commencé à travailler sur l'application cliente. Nous avons notamment supprimé des fonctionnalités que le commanditaire ne souhaite plus utiliser dans le futur. Un début de travail de refactoring a été également entamé.

7 Perspective d'évolution

Dans cette partie, nous allons discuter des différentes évolutions possibles de notre application.

7.1 Base de données

Différentes évolutions peuvent être envisagées pour l'évolution de la base de données. Un premier, pourrait être de finir de transcrire les scripts python en script SQL intégrant du python. Une autre pourrait être de créer certains scripts d'automatisation pour la base de données.

Une autre évolution envisageable pour la base de données serait pour l'ajout des données au sein de la base. En effet, dans la partie difficultés, nous avons évoqué le fait que chaque xml possède ses propres métadonnées, toutes nommées différemment. Nous avons également précisé plus haut que cette fonctionnalité a finalement été abandonné dû à la complexité des différences entre fichiers. Une solution serait soit d'écrire des scripts pour les autres sources de données en remplaçant les différents noms de chaque balise, ou alors de créer un script qui saurait chercher les métadonnées selon les différents noms existants.

Par ailleurs, nous pouvons également envisager des évolutions pour l'import des fichiers csv. Nous pouvons notamment évoquer les paramètres par défaut :

- Le centre de la caméra par défaut devrait être calculé grâce au géocodage ;
- Ils manquent une partie de ceux-ci qui ne sont pas encore calculé (matrice de transformation, quaternion...). Le calcul de ceux-ci pourrait être envisageable via l'utilisation de MicMac.

Une autre évolution envisageable pour l'import des données csv serait de réfactoré le code, ainsi que de l'optimiser pour réduire au maximum les temps de chargement des données.

7.2 GeoServer

Plusieurs améliorations pourrait être réalisées pour la diffusion des cartes de chaleur. Étant donné que les cartes sont calculées à la volée, il serait nécessaire de vérifier qu'avec un plus grand nombre de clichés le temps de chargement des couches reste acceptable. De plus, afin d'améliorer le temps de chargement, il est envisageable de mettre les couches en cache. Cependant si la base de donnée est modifiée, il faudra alors faire attention à invalider le cache.

7.3 API

Actuellement notre API possède quatre routes et permet de faire tourner une commande MicMac. Une évolution envisageable serait d'étendre notre API afin de permettre l'utilisation de nouvelles commandes MicMac.

7.4 Client

Au niveau du client, plusieurs amélioration peuvent être apportées.

Pour commencer, notre application n'est actuellement pas directement reliée à la base de données et à l'API. Il faudrait donc tout relier ensemble pour une première évolution.

Nous pouvons parler dans un premier temps du chargement des tuiles du fond de cartes d'iTowns, qui est relativement long. Une première idée serait de pouvoir les mettre en cache une fois téléchargée pour éviter des temps d'attente assez long à chaque rafraîchissement de page.

Un autre problème dont nous pouvons parler est la sécurisation de l'ensemble des interactions de la partie cliente. Bien que l'API ajoute un nouveau de sécurité, la sécurité n'est pas le point fort de notre application. Celle-ci ne possède pas, par exemple, de système d'authentification avec une gestion de droits. Différents rôles auraient ainsi pu être définis. Pour commencer un rôle utilisateur, qui a la possibilité d'interagir avec le client, d'enregistrer des points d'appuis et de créer des géoréférencements. Un second rôle, cette fois-ci administrateur qui aurait pu permettre de modifier directement la base de données depuis l'interface client, avec l'accès à une page dédiée sur le client.

Un autre soucis qui pourrait être réglé dans le futur est l'ajout d'image dans l'application, sans forcément l'ajouter dans la base de données, pour que l'utilisateur puisse visualiser ses images sans passer par le système d'url. En effet, actuellement l'utilisateur doit préciser le nom de l'image dans l'url car c'est elle qui est découpée pour récupérer les différents paramètres.

Nous pouvons également évoquer le fait que la page d'index de l'application cliente peut être largement optimisée. Bien que nous ayons commencé à effectuer un travail de nettoyage, celui doit être bien plus poussé pour retirer le code mort, les commentaires inutiles. Il doit également y avoir un refactoring poussé, certaines lignes étant dupliquer un certain nombres de fois.

Enfin le gros point d'évolution qui pourrait être abordé dans le futur serait la réalisation d'une architecture complète pour la partie cliente, en intégrant peut-être un modèle MVC. Cela pourrait permettre de renforcer la sécurité de l'application etc., ou encore de rendre l'application cliente bien plus simple d'évolution. Par ailleurs, elle pourrait être mieux optimisée. On pourrait également diminuer la force de couplage des fichiers par rapport au fichier index.html, car aujourd'hui l'utilisation tous les fichiers clients sont appelés à travers lui. C'est également lui qui fait le lien entre chaque dépôt. Une autre évolution pourrait égalent être la fusion des trois dépôts en un seul et unique, pour diminuer la complexité d'installation pour l'utilisateur.

7.5 Dépôts GitHub

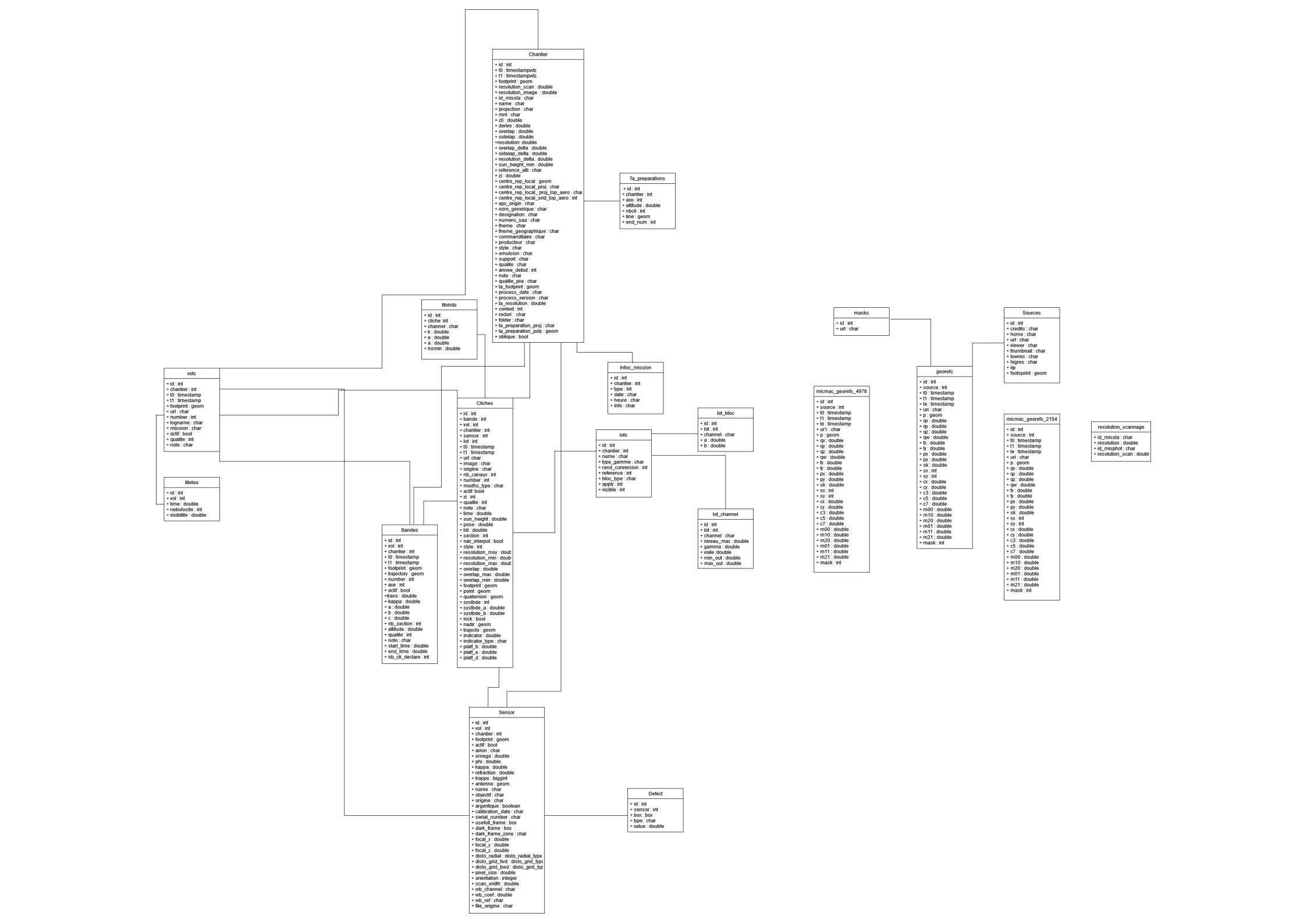
Une autre évolution possible dans le temps serait de reprendre l'ensemble des dépôts GitHub du commanditaire et de les nettoyer. Il s'agirait de nettoyer l'ensemble des branches sur les dépôts pour apporter de la clarté au projet.

7.6 Déploiement

Concernant le déploiement, nous pouvons évoquer deux axes d'évolutions :
— Bien qu'un début de Dockerisation ai été effectué, l'ensemble de l'application n'est pas déployable actuellement ;
— De plus, celle-ci n'est pas non plus présente sur les VM dédiée du projet Alegoria.

8 Annexes

Ci-dessous est présenté le diagramme de classe de la base de données existantes à notre arrivée.


Chantier
+ id : int
+ t0 : timestampwtz
+ t1 : timestampwtz
+ footprint : geom
+ resolution_scan : double
+ resolution_image : double
+ id_missta : char
+ name : char
+ projection : char
+ mnt : char
+ z0 : double
+ derive : double
+ overlap : double
+ sidelap : double
+resolution: double
+ overlap_delta : double
+ sidelap_delta : double
+ resolution_delta : double
+ sun_height_min : double
+ reference_alti : char
+ zi : double
+ centre_rep_local : geom
+ centre_rep_local_proj : char
+ centre_rep_local_proj_top_aero : char
+ centre_rep_local_srid_top_aero : int
+ apx_origin : char
+ nom_generique : char
+ designation : char
+ numero_saa : char
+ theme : char
+ theme_geographique : char
+ commanditaire : char
+ producteur : char
+ style : char
+ emulsion : char
+ support : char
+ qualite : char
+ annee_debut : int
+ note : char
+ qualite_pva : char
+ ta_footprint : geom
+ process_date : char
+ process_version : char
+ ta_resolution : double
+ context : int
+ redori : char
+ folder : char
+ ta_preparation_proj : char
+ ta_preparation_poly : geom
+ oblique : bool
Ta_preparations
+ id : int
+ chantier : int
+ axe : int
+ altitude : double
+ nbcli : int
+ line : geom
+ end_num : int
Mohds
+ id : int
+ cliche :int
+ channel : char
+ k : double
+ e : double
+ a : double
+ hsmin : double
vols
+ id : int
+ chantier : int
+ t0 : timestamp
+ t1 : timestamp
+ footprint : geom
+ url : char
+ number : int
+ logname : char
+ mission : char
+ actif : bool
+ qualite : int
+ note : char
Infos_mission
+ id : int
+ chantier : int
+ type : int
+ date : char
+ heure : char
+ info : char
Cliches
+ id : int
+ bande : int
+ vol : int
+ chantier : int
+ sensor : int
+ lot : int
+ t0 : timestamp
+ t1 : timestamp
+ url :char
+ image : char
+ origine : char
+ nb_canaux : int
+ number : int
+ modhs_type : char
+ actif :bool
+ zi : int
+ qualite : int
+ note : char
+ time : double
+ sun_height : double
+ pose : double
+ tdi : double
+ section : int
+ nav_interpol : bool
+ style : int
+ resolution_moy : doub
+ resolution_min : doub
+ resolution_max : doub
+ overlap : double
+ overlap_max : double
+ overlap_min : double
+ footprint : geom
+ point : geom
+ quaternion : geom
+ systbde : int
+ systbde_a : double
+ systbde_b : double
+ lock : bool
+ nadir : geom
+ trajecto : geom
+ indicator : double
+ indicator_type : char
+ platf_b : double
+ platf_e : double
+ platf_d : double
lots
+ id : int
+ chantier : int
+ name : char
+ type_gamme : char
+ rand_conversion : int
+ reference : int
+ bloc_type : char
+ apply : int
+ visible : int
lot_bloc
+ id : int
+ lot : int
+ channel : char
+ a : double
+ b : double
lot_channel
+ id : int
+ lot : int
+ channel : char
+ niveau_max : double
+ gamma : double
+ voile :double
+ min_out : double
+ max_out : double
Meteo
+ id : int
+ vol : int
+ time : double
+ nebulosite : int
+ visibilite : double
Bandes
+ id : int
+ vol : int
+ chantier : int
+ t0 : timestamp
+ t1 : timestamp
+ footprint : geom
+ trajectory : geom
+ number : int
+ axe : int
+ actif : bool
+trans : double
+ kappa : double
+ a : double
+ b : double
+ c : double
+ nb_section : int
+ altitude : double
+ qualite : int
+ note : char
+ start_time : double
+ end_time : double
+ nb_cli_declare : int
Sensor
+ id : int
+ vol : int
+ chantier : int
+ footprint : geom
+ actif : bool
+ avion : char
+ omega : double
+ phi : double
+ kappa : double
+ retraction : double
+ trappe : bigint
+ antenne : geom
+ name : char
+ objectif : char
+ origine : char
+ argentique : boolean
+ calibration_date : char
+ serial_number : char
+ usefull_frame : box
+ dark_frame : box
+ dark_frame_zone : char
+ focal_x : double
+ focal_y : double
+ focal_z : double
+ disto_radial : disto_radial_type
+ disto_grid_fwd : disto_grid_typ
+ disto_grid_bwd : disto_grid_typ
+ pixel_size : double
+ orientation : integer
+ scan_width : double
+ wb_channel : char
+ wb_coef : double
+ wb_ref : char
+ file_origine : char
Defect
+ id : int
+ sensor : int
+ box : box
+ type : char
+ value : double
masks
+ id : int
+ url : char
georefs
+ id : int
+ source : int
+ t0 : timestamp
+ t1 : timestamp
+ te : timestamp
+ uri : char
+ p : geom
+ qx : double
+ qy : double
+ qz : double
+ qw : double
+ tx : double
+ ty : double
+ px : double
+ py : double
+ sk : double
+ sx : int
+ sy : int
+ cx : double
+ cy : double
+ c3 : double
+ c5 : double
+ c7 : double
+ m00 : double
+ m10 : double
+ m20 : double
+ m01 : double
+ m11 : double
+ m21 : double
+ mask : int
micmac_georefs_4978
+ id : int
+ source : int
+ t0 : timestamp
+ t1 : timestamp
+ te : timestamp
+ uri : char
+ p : geom
+ qx : double
+ qy : double
+ qz : double
+ qw : double
+ tx : double
+ ty : double
+ px : double
+ py : double
+ sk : double
+ sx : int
+ sy : int
+ cx : double
+ cy : double
+ c3 : double
+ c5 : double
+ c7 : double
+ m00 : double
+ m10 : double
+ m20 : double
+ m01 : double
+ m11 : double
+ m21 : double
+ mask : int
Sources
+ id : int
+ credits : char
+ home : char
+ url : char
+ viewer : char
+ thumbnail : char
+ lowres : char
+ higres : char
+ iip
+ footsprint : geom
micmac_georefs_2154
+ id : int
+ source : int
+ t0 : timestamp
+ t1 : timestamp
+ te : timestamp
+ uri : char
+ p : geom
+ qx : double
+ qy : double
+ qz : double
+ qw : double
+ tx : double
+ ty : double
+ px : double
+ py : double
+ sk : double
+ sx : int
+ sy : int
+ cx : double
+ cy : double
+ c3 : double
+ c5 : double
+ c7 : double
+ m00 : double
+ m10 : double
+ m20 : double
+ m01 : double
+ m11 : double
+ m21 : double
+ mask : int
resolution_scannage
+ id_missta : char
+ resolution : double
+ id_misphot : char
+ resolution_scan : doubl

Ci-dessous se situe la documentation des attributs de la base de données. Après cela, nous trouvons les différents documents relatif à notre diagramme de Gantt ainsi que celui-ci.

Documentation of database attributes

Amaryllis Vignaud, Melodia Mohad, Axelle Gaige, Thomas De Beaumont

March 2022

1 images table

The images table is used to model the important information in an image.

Attribute	Type	Description	NULL	UNIQUE
id_images	SERIAL	Image's id	NOT NULL	UNIQUE
t0	timestamp	Start date of image capture	NOT NULL	-
t1	timestamp	End date of image capture	NOT NULL	-
image	VARCHAR	Name of the image	NOT NULL	-
size_image	Point	Image size	NOT NULL	-

2 sources table

An image has a source, and a source can be used to find batches of images. This table contains all the information such as access urls to the resource, to the view or to the credits of the image.

Attribute	Type	Description	NULL	UNIQUE
id_sources	SERIAL	Source's id	NOT NULL	UNIQUE
credit	VARCHAR	1uthor of the image	NOT NULL	-
home	VARCHAR	Home page of the source site	NOT NULL	-
url	VARCHAR	Link to the source of the image	NOT NULL	UNIQUE
viewer	VARCHAR	Link to image view	-	
thumbnail	INT	?	NOT NULL	-
lowres	FLOAT	?	-	-
highres	FLOAT	?	-	-
iip	FLOAT	?	-	-
footprint	Multipolygon	Footprint of a set of images	-	-

3 masks table

An image has a single mask. However, a mask can be used for several images. The masks table identifies each mask with an associated url.

Attribute	Type	Description	NULL	UNIQUE
id_masks	SERIAL	masks's id	NOT NULL	UNIQUE
url	VARCHAR	url of the image mask	NOT NULL	UNIQUE

4 point_appuis table

An image will have support points as it is used. An image can have several support points. The aim of the support point table is to store both 2D and 3D support points.

Attribute	Type	Description	NULL	UNIQUE
id_points	SERIAL	id of support points	NOT NULL	UNIQUE
point_2D	POINT*	support points of the imported image	-	-
point_3D	POINTZ*	support points on the georeferenced map	-	-

*POINT is used to designate a point in two dimensions. For a point in three dimensions, the designation POINTZ is used

5 Table georefs

Each image will be associated with a georeferencing, however an image can have several georeferencing. The georeferencing table will allow access to the user who georeferenced the image, the date of its creation and to determine if this georeferencing is the main one of an image.

Attribute	Type	Description	NULL	UNIQUE
id_georefs	SERIAL	georeferencing's id	NOT NULL	UNIQUE
user_georef	VARCHAR	user creating the georeferencing	NOT NULL	
date	timestamp	creation date	NOT NULL	-
georef_principal	BOOL	main georeferencing of the image	NOT NULL	-
footprint	Multipolygon	Footprint of the image	NOT NULL	-
near	Polygon	Closest polygon to the camera	NOT NULL	-
far	Polygon	Farthest point to the camera	NOT NULL	-

6 externe table

The externe table stores the external georeferencing parameters of an image. These are ?, the quaternion and the SRID of the image.

Attribute	Type	Description	NULL	UNIQUE
id_externe	SERIAL	id of the external georeferencing parameters	NOT NULL	UNIQUE
point	POINTZ	designates the centre of the camera (position)	NOT NULL	-
quaternion	POINTZ	point for rotation	NOT NULL	-
SRID	INT	SRID of the georeferencing	NOT NULL	-

7 interne table

The interne table stores the internal georeferencing parameters of an image. These include the camera fulcrum, the focal, the skew and the image distortion.

Attribute	Type	Description	NULL	UNIQUE
id_interne	SERIAL	id of the internal georeferencing parameters	NOT NULL	UNIQUE
pp	POINTZ	point of support of the camera	NOT NULL	-
focal	POINTZ	focal point of the sensor	NOT NULL	-
skew	FLOAT	deviation	NOT NULL	-
distortion	ARRAY	distortion matrix	NOT NULL	-

8 transfo2D table

The transfo2D table stores the information related to the georeferencing of the image, i.e. the image matrix from the 2D georeferencing.

Attribute	Type	Description	NULL	UNIQUE
id_transfo2D	SERIAL	id of the transformation	NOT NULL	UNIQUE
image_matrix	ARRAY	image matrix	NOT NULL	-

Projet de fin d'année

13 avr. 2022

http://

Chef de projet	
Dates du projet	9 mars 2022 - 20 avr. 2022
Avancée	0%
Tâches	37
Ressources	0

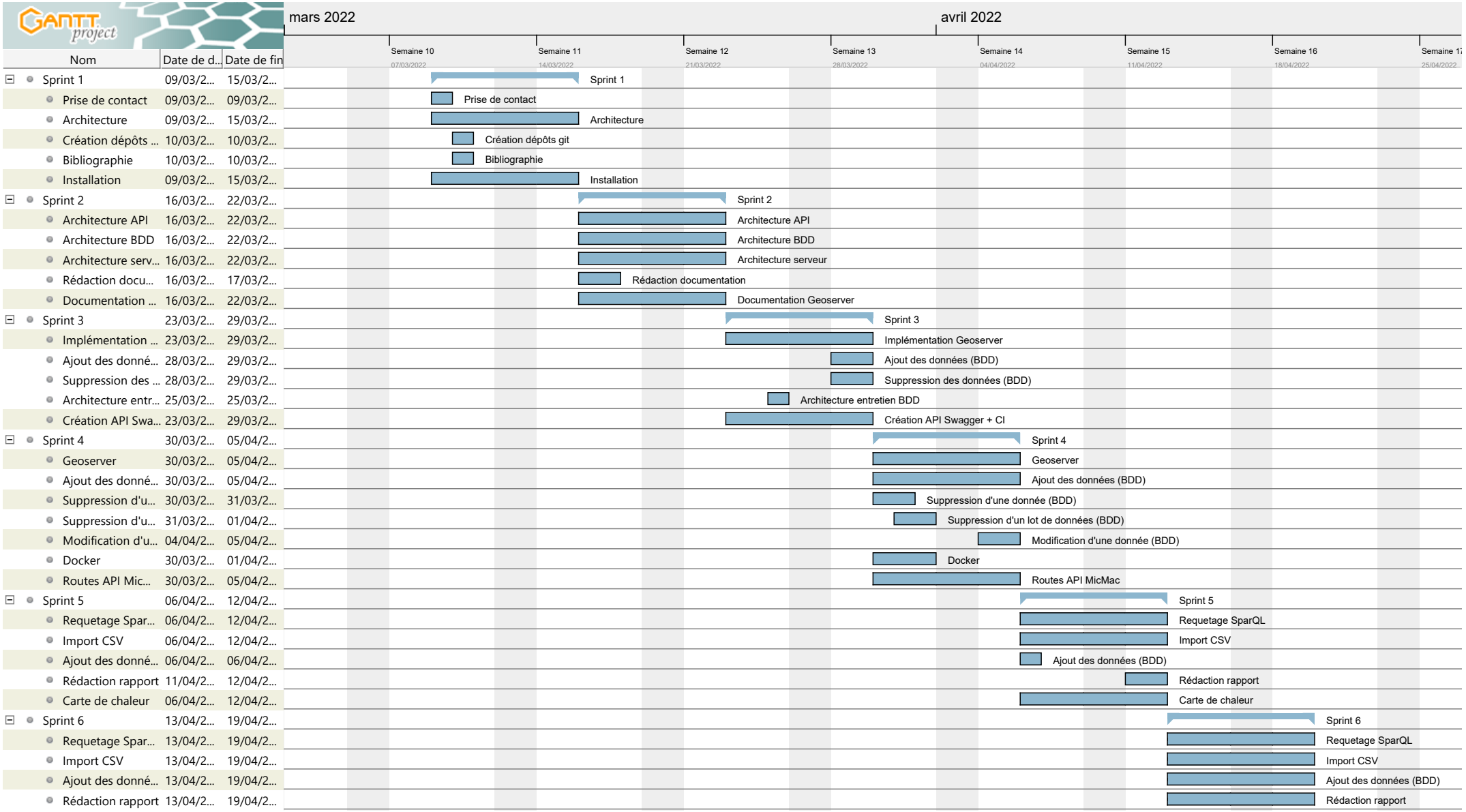
Tâches

Nom	Date de début	Date de fin
Sprint 1	09/03/2022	15/03/2022
Prise de contact	09/03/2022	09/03/2022
Architecture	09/03/2022	15/03/2022
Création dépôts git	10/03/2022	10/03/2022
Bibliographie	10/03/2022	10/03/2022
Installation	09/03/2022	15/03/2022
Sprint 2	16/03/2022	22/03/2022
Architecture API	16/03/2022	22/03/2022
Architecture BDD	16/03/2022	22/03/2022
Architecture serveur	16/03/2022	22/03/2022
Rédaction documentation	16/03/2022	17/03/2022
Documentation Geoserver	16/03/2022	22/03/2022
Sprint 3	23/03/2022	29/03/2022
Implémentation Geoserver	23/03/2022	29/03/2022
Ajout des données (BDD)	28/03/2022	29/03/2022
Suppression des données (BDD)	28/03/2022	29/03/2022
Architecture entretien BDD	25/03/2022	25/03/2022
Création API Swagger + CI	23/03/2022	29/03/2022
Sprint 4	30/03/2022	05/04/2022
Geoserver	30/03/2022	05/04/2022
Ajout des données (BDD)	30/03/2022	05/04/2022
Suppression d'une donnée (BDD)	30/03/2022	31/03/2022
Suppression d'un lot de données (BDD)	31/03/2022	01/04/2022
Modification d'une donnée (BDD)	04/04/2022	05/04/2022
Docker	30/03/2022	01/04/2022
Routes API MicMac	30/03/2022	05/04/2022

Tâches

Nom	Date de début	Date de fin
Sprint 5	06/04/2022	12/04/2022
Requetage SparQL	06/04/2022	12/04/2022
Import CSV	06/04/2022	12/04/2022
Ajout des données (BDD)	06/04/2022	06/04/2022
Rédaction rapport	11/04/2022	12/04/2022
Carte de chaleur	06/04/2022	12/04/2022
Sprint 6	13/04/2022	19/04/2022
Requetage SparQL	13/04/2022	19/04/2022
Import CSV	13/04/2022	19/04/2022
Ajout des données (BDD)	13/04/2022	19/04/2022
Rédaction rapport	13/04/2022	19/04/2022

Diagramme de Gantt



Nom	Date de d...	Date de fin
Sprint 1	09/03/2...	15/03/2...
Prise de contact	09/03/2...	09/03/2...
Architecture	09/03/2...	15/03/2...
Création dépôts ...	10/03/2...	10/03/2...
Bibliographie	10/03/2...	10/03/2...
Installation	09/03/2...	15/03/2...
Sprint 2	16/03/2...	22/03/2...
Architecture API	16/03/2...	22/03/2...
Architecture BDD	16/03/2...	22/03/2...
Architecture serv...	16/03/2...	22/03/2...
Rédaction docu...	16/03/2...	17/03/2...
Documentation ...	16/03/2...	22/03/2...
Sprint 3	23/03/2...	29/03/2...
Implémentation ...	23/03/2...	29/03/2...
Ajout des donné...	28/03/2...	29/03/2...
Suppression des ...	28/03/2...	29/03/2...
Architecture entr...	25/03/2...	25/03/2...
Création API Swa...	23/03/2...	29/03/2...
Sprint 4	30/03/2...	05/04/2...
Geoserver	30/03/2...	05/04/2...
Ajout des donné...	30/03/2...	05/04/2...
Suppression d'u...	30/03/2...	31/03/2...
Suppression d'u...	31/03/2...	01/04/2...
Modification d'u...	04/04/2...	05/04/2...
Docker	30/03/2...	01/04/2...
Routes API Mic...	30/03/2...	05/04/2...
Sprint 5	06/04/2...	12/04/2...
Requetage Spar...	06/04/2...	12/04/2...
Import CSV	06/04/2...	12/04/2...
Ajout des donné...	06/04/2...	06/04/2...
Rédaction rapport	11/04/2...	12/04/2...
Carte de chaleur	06/04/2...	12/04/2...
Sprint 6	13/04/2...	19/04/2...
Requetage Spar...	13/04/2...	19/04/2...
Import CSV	13/04/2...	19/04/2...
Ajout des donné...	13/04/2...	19/04/2...
Rédaction rapport	13/04/2...	19/04/2...