

VOOOV

Projets personnels encadrés, dans le cadre du BTS SIO – Option
SLAM.

Année 2023

Sommaire

Table des matières

Préface du projet.....	4
Identification du besoin	5
Préambule	6
Étapes du projet	7
Début du premier projet	9
Le design	9
L'UX	9
Tendances	9
Application des tendances au projet	10
Modélisation des données et structure du projet	12
Modélisations des données	12
Base de données	17
Choix de base de données	17
MPD et Script SQL	17
Fin premier projet	20
Début deuxième projet	20
Implémentation des services.....	20
Création du VPS.....	20
Déploiement et hébergement des différents services	20
Sécurisation du serveur	24
Améliorations possibles.....	24
Mise en place d'un service de versioning	25
Utilité du versioning	25
Utilisation de Git.....	25
Utilisation pour le projet	26
Ticketing et tests	27
Le ticketing	27
Phase de conception et de test	27
Phase de production	28
Déploiement de la base de données	29
Procédures stockées.....	29

Conception, création et implémentation de l'API	30
Conception	30
Créations de l'API	30
Amélioration du code	35
Fin dossier 2	39
Début dossier 3	39
En cours de préparation	39
Conception de l'Application Web	39
Conception de l'application mobile (Android).....	39
Création d'une application de test du CRUD en échange avec l'API.....	39
Bugs.....	39

Préface du projet

Le projet présenté par la suite, suis les étapes d'une solution d'information réalisée dans des conditions réelles en mode projet.

Depuis le commencement de ma formation en BTS SIO Option SLAM. J'ai pu approfondir mes nouvelles connaissances en répondant aux besoins de mon entourage.

Les premières réalisations sont des sites web. Ils m'ont permis de maîtriser les bases des principaux langages du web, à savoir le HTML, le CSS, le JavaScript et le PHP, ainsi que la bibliothèque Symfony.

La demande par des clients réels m'a motivé à m'impliquer davantage. Elle m'a aussi forcé à rendre des outils fonctionnels. Cela m'a obligé à pousser mes recherches, jusqu'à trouver des solutions, aux divers problèmes rencontrés.

Les modifications demandées par les clients, m'ont clairement fait apparaître l'importance d'un code propre, lisible, optimisé et maintenable. Un code bien commenté et bien construit permet de gagner du temps en cas de retour sur celui-ci, mais également dans de nouveaux projets.

Lors de ce projet, j'ai pu mettre en œuvre mes connaissances en Java ainsi qu'en Kotlin.

Ce projet scolaire comporte trois projets, faisant office de projet personnels encadrés. Il mettra en œuvre les compétences variées et nécessaire, pour la création d'une solution d'information, et attendues pour la validation du BTS SIO option SLAM.

Identification du besoin

Le projet présenté est né d'une demande concrète. Une personne de mon entourage, m'a contacté pour la réalisation d'un site web. L'objectif du site était de faire la promotion de sa carrière de Voix-Off.

Le projet naît et se nomme Vooov, pour voice-over : Voix-Off en anglais.

Après la réalisation de plusieurs site web et souhaitant continuer de gagner en compétences durant ma formation, je souhaitais m'essayer à d'autre forme de programmation. Mon but était de réaliser un projet en Java.

L'idée prend forme petit à petit, et après quelques recherches, je n'ai pas pu trouver d'application mobile mettant candidats à la Voix-Off et recruteurs. De plus l'idée m'est apparue que les besoins dans ce domaine pourraient sans doute être étendus et généralisés à d'autres utilisations. Pourquoi pas moins professionnelles, événements familiaux, partage de mémoires, vidéos type Youtube...

Le constat est un certain retour de l'audio, là où le visuel avait pris toute la place. On voit grandir l'utilisation des « Voices » de Facebook prenant une part d'utilisation sur le sms, les appareils tel qu'Alexia de google, répondant à la voix, s'installant dans les foyers et les connectant parfois en domotique. Le nombre d'utilisateurs écoutant, musiques, audiolivres et podcasts, remplaçant la radio, et ce parfois même en voiture, grandit. Simplement, on utilise de plus en plus les technologies récentes pour le contenu audio. Les utilisateurs renouent avec ce sens oublié, en lui prêtant même un côté futuriste et amusant.

Le cœur de cible paraît étendu, puisque l'application peut servir à des besoins professionnels ou ludiques. Les utilisateurs peuvent être de tout âge et venir de tout milieu social.

L'idée finale actuelle est donc de réaliser une application sous forme de réseau sociale de type vocal. Celle-ci devra faciliter la mise en lien entre les candidats Voix-Off et les employeurs. Elle devra également permettre le partage de contenu audio entre personnes non professionnelles.

Cette application a pour objectif de rendre possible un échange direct et privé entre utilisateurs. Il doit également permettre, une forme de contribution entre utilisateurs, interne à l'application, en échange d'un service vocal.

Elle doit être accessible depuis plusieurs plateformes (ordinateurs, tablettes et mobiles) et doit donc permettre de lier les donner entre ces différents médias.

Préambule

Une fois le premier besoin identifié, il est nécessaire de mettre en place une forme de pré architecture abstraite du projet sous forme de brain-storming. Sur ce media en forme nuage, nous plaçons toutes les idées qui peuvent apparaître et qui pourrait paraître cohérentes. Il faut ensuite faire le tri entre les idées essentielles et celle qui pourraient faire perdre de la consistance, ou encore simplement qui pourrait avoir vocation à apparaître ultérieurement.

Le voici :

Doit :	Choix des contraintes	Idées pour le futur:	Idées mises de côté pour le moment:
Pouvoir être enregistrer depuis un Téléphone	Choix d'une durée maximale d'enregistrement	système de gain sur les jetons. Augmentation de 10% du prix d'un jeton à chaque doublement du nombre total de jetons.	ajout d'images pour les enregistrements. Trop de poids supplémentaire.
Mettre en lien candidats voix-off et recruteurs	format d'enregistrement	Moyen de paiement pour l'achat de jetons. Permet de payer directement depuis le site de manière sécurisée.	
Permettre l'échange entre les utilisateurs		Envoyer des enregistrements audio par messages.	
Etre agréable à regarder		Créer des salles : de réunions, de concerts	
Importer des fichiers audios externes		Créer des conversations de groupes	
Donner la possibilité de choisir une type de voix et un type de contenu		Permettre la retouche des enregistrements	
Permettre la recherche d'enregistrement et d'artiste			
		Choix du nom de l'application : Vooov (pour Voice-over, qui signifie voix-off en anglais)	

Par la suite, il faut définir les médias à travers lesquels le projet pourra prendre vie. Il faut définir via quelles méthodes de mise en œuvre cela sera réalisé. A cette étape, on met en lien les compétences de l'équipe et les besoins du projet.

Pour ce projet, dont le but est de permettre un échange entre des utilisateurs, le plus de médias possibles semble être le meilleur moyen d'agrandir la visibilité.

Les compétences disponibles, acquises durant ma formation se compose de langage du web, de Java et de langage C. On peut également ajouter des compétences en gestion de base de données SQL et NoSQL, ainsi que des notions en déploiement, hébergement...

Durant cette expérience, j'ai pu me familiariser avec le langage Kotlin.

Ces compétences ouvrent donc trois champs :

- Application mobile
- Site web
- Application desktop (application dite lourde)

Afin de mettre en lien les données de ces trois réalisations, il faut ajouter au projet une API (Application Programming Interface). Elle permettra de gérer la base de données de manière uniforme depuis les trois médias. (Une première version de l'application : OFFO a été réalisée via l'API Firebase, qui offre une base de données NoSQL stockée dans le cloud. Afin de répondre à des besoins scolaires, le projet a été modifié en créant une API propre, liée à une base de données MySQL)

Étapes du projet

Ce projet global se divise en trois. Une fois les médias et les compétences passées en revue, il est nécessaire de définir un ordre d'approche au projet. Le suivant a donc été défini :

« Début premier dossier »

- **Le design** : Maquettage des différents formats. On crée une identité visuelle commune. Définition de la colorimétrie, de la typographie et du logo. Ici c'est l'outil « Figma » qui est utilisé pour le maquettage de l'ensemble des pages web, application mobile et lourde. Le logo et les icônes sont réalisés via le logiciel « Inkscape ».
- **Modélisation des données** : MCD (Modélisation Conceptuelle des Données) grâce au logiciel « StarUML », MLD (Modélisation Logique des Données) grâce à l'outil « dbdiagram.io ».
- **Script pour la base de données** : MPD (Modélisation Physique des Données) Une fois définis les besoins, il faut retranscrire le script de création de Base de données

« Fin premier dossier »

« Début deuxième dossier »

- **Implémentation des services** : Afin d'héberger les différentes parties du projet, il faut définir un service d'hébergement. L'API, la base de données, les fichiers de l'application lourde et le site web sont hébergés sur serveur VPS dédié, hébergé par « Hostinger », sous Ubuntu. L'outil « CyberPanel » est utilisé afin de faciliter l'administration du serveur. L'application mobile sera hébergée dans un second temps sur le « Playstore d'Android ».
- **Mise en place d'un service de versioning** : Vous pourrez retrouver l'ensemble des versions de mes repositories sur « Github » à cette adresse : <https://github.com/nathan-kedinger?tab=repositories>.
- **Gestion des tickets** : La gestion des tickets en phase de conception et de test sera réalisée via « Github » et Trello. Une gestion des logs d'erreur et de connexion est également enregistrée directement sur serveur. Au quotidien, j'utilise « Trello » afin de lister les différentes tâches à effectuer. Notamment les tickets à résoudre.
- **Déploiement de la base de données** : Une fois le serveur VPS activé et utilisable, il est temps de déployer la base de données et d'y implanter les scripts créés précédemment. Ici l'outil de gestion de base de données est phpMyAdmin qui utilise MySQL.
- **Conception, création et implémentation de l'API** : L'API créée est une API Rest codée en PHP, permettant une gestion CRUD (Create, Read, Update, Delete) des données via des protocoles http sécurisés. On implémente des processus de test aux différents points de liaisons et on teste le fonctionnement grâce à l'outil « Postman ».

« Fin deuxième dossier »

« Début troisième dossier »

- **Conception de l'application mobile** : L'application mobile est créée pour les smartphones et tablettes Android et réalisée via l'IDE d'Android Studio fonctionnant sous l'IDEA IntelliJ . Elle sera déployée sur le PlayStore d'Android.
- **Conception de l'application web** : L'application web est créée via le Framework Symfony 6 utilisant PHP et réalisé via l'IDE Visual Studio Code. Elle sera déployée sur le serveur VPS de l'API.
- **Conception de l'application lourde** : L'application lourde est réalisée en Kotlin et sera téléchargeable depuis le site web.

« Fin dossier 3 »

Mise en place de tests de performances :

Implémentation de services d'intégration continus :

Ouverture et améliorations à venir :

Début du premier projet

Le design

L'UX

Dans tout projet, l'esthétique joue un rôle clé pour l'UX (expérience utilisateur). L'UX ne définit pas uniquement le design, mais également la maniabilité, la facilité d'utilisation, le contenu...

Aujourd'hui, les utilisateurs ont accès à une application pour presque tous les usages de la vie courante. Ils ont l'habitude que cela fonctionne rapidement et ils n'hésiteront pas à quitter l'application s'ils rencontrent des ralentissements ou quelques difficultés. Ce surtout dans les premiers instants d'utilisation.

L'UX tient donc un rôle clé dans le fonctionnement d'une application.

Le rôle du design est d'apporter une identité qui permettra de reconnaître le produit. Il doit également permettre à l'utilisateur un confort visuel. Il doit donc être agréable à regarder, mais également correspondre aux attentes de l'utilisateur, pour le type d'expérience qu'il vient chercher.

Tendances

Ce projet doit répondre à un cœur de cible varié (âge, professionnels/particuliers, sexe...). Il doit donc présenter une esthétique parlant naturellement à tous. Il doit également être dans l'air du temps. Ce projet est basé sur une expérience vocale, auditives et de partages.

Voici une liste, non exhaustive, d'objets de la vie courante répondant à cet imaginaire collectif : Radio, studio d'enregistrement, enceintes, amplis, micros, téléphone, bouches, oreilles, musique, réseaux sociaux, échange, bien-être, écoute... Le groupe d'applications créé plus tard cherchera à s'inspirer de cette liste.

Il est à noter l'aspect psychologique probable vis-à-vis du contenu en création. Par suite d'une recherche d'inspiration, il semble apparaître certains points :

- L'audition peut apporter une certaine forme d'introspection.
- Elle est principalement vue comme liée à la musique et à la parole.
- La musique est source d'émotion. Ces émotions peuvent être fortes, douce, agréable, nostalgiques... (Afin de garder l'attrait des utilisateurs, les aspects positifs sont mis en avant)
- La musique agit sur la concentration.
- La musique est vectrice de calme mais également de forte énergie.
- La parole se rapporte au langage. Elle transporte le savoir et les informations. Elle permet les échanges sociaux et la communication.

- Le résultat nous porte à croire que dans l'imaginaire visuel collectif, le son est souvent apparenté à un esprit vintage.
- Le son est lié à la créativité.

Cela suppose d'accueillir l'œil dans un milieu calme et rassurant, au potentiel énergisant et créatif. Il est nécessaire de faire appel à des images communes et connues.

Les couleurs prédominantes aperçues lors de cette recherche sont, soit, des couleurs sombres, soit, au contraire, des couleurs très brillantes. La couleur bleue ressort régulièrement (souvent associé à l'apaisement et la vérité. Également lié au bien-être et à la spiritualité, au calme et à la communication). Cela correspond bien à notre sujet. Les formes sont le plus souvent douces et arrondies.

Application des tendances au projet

L'étape suivante consiste à créer une planche de tendance. Elle est composée d'un ensemble d'images correspondant aux objets du quotidien identifiés précédemment. Elle permet de déduire une ligne de couleur et de formes.



Les années 90 sont aujourd'hui de retour à la mode. Cependant, il est important de se concentrer sur le point de vue actuel de ces années. Les utilisateurs se souviennent rarement précisément du design d'il y a 30 ans. L'aspect rassurant d'une œuvre, provient principalement, de la sensation d'évoluer dans un milieu, original, mais familier.

Concernant la police d'écriture, l'application sera composée de deux typographies. L'une servira pour le titre de l'application qui rappelle les vinyles : **MONOTON**. La seconde reste sobre et lisible pour ne pas déranger l'utilisateur : Biryani.

Une fois la trame principale fixée, il est possible de passer à la mise en forme concrète du design. Précéder la mise en production du site par une mise en forme visuelle permet de gagner un temps considérable.

L'outil Figma apporte une prévisualisation précise des futurs écrans. Cet outil permet une création et une modification des visuels extrêmement rapide en comparaison à ce que demande le CSS d'une application web, ou le XML d'une application mobile et desktop.

De plus, le vocabulaire utilisé dans le projet peut d'ores et déjà être constitué. La mise en production n'aura plus qu'à suivre pas à pas le fichier sans réflexion supplémentaire.

NB : Pour certains projets, le vocabulaire est une tâche dédiée à une équipe spécialisée.

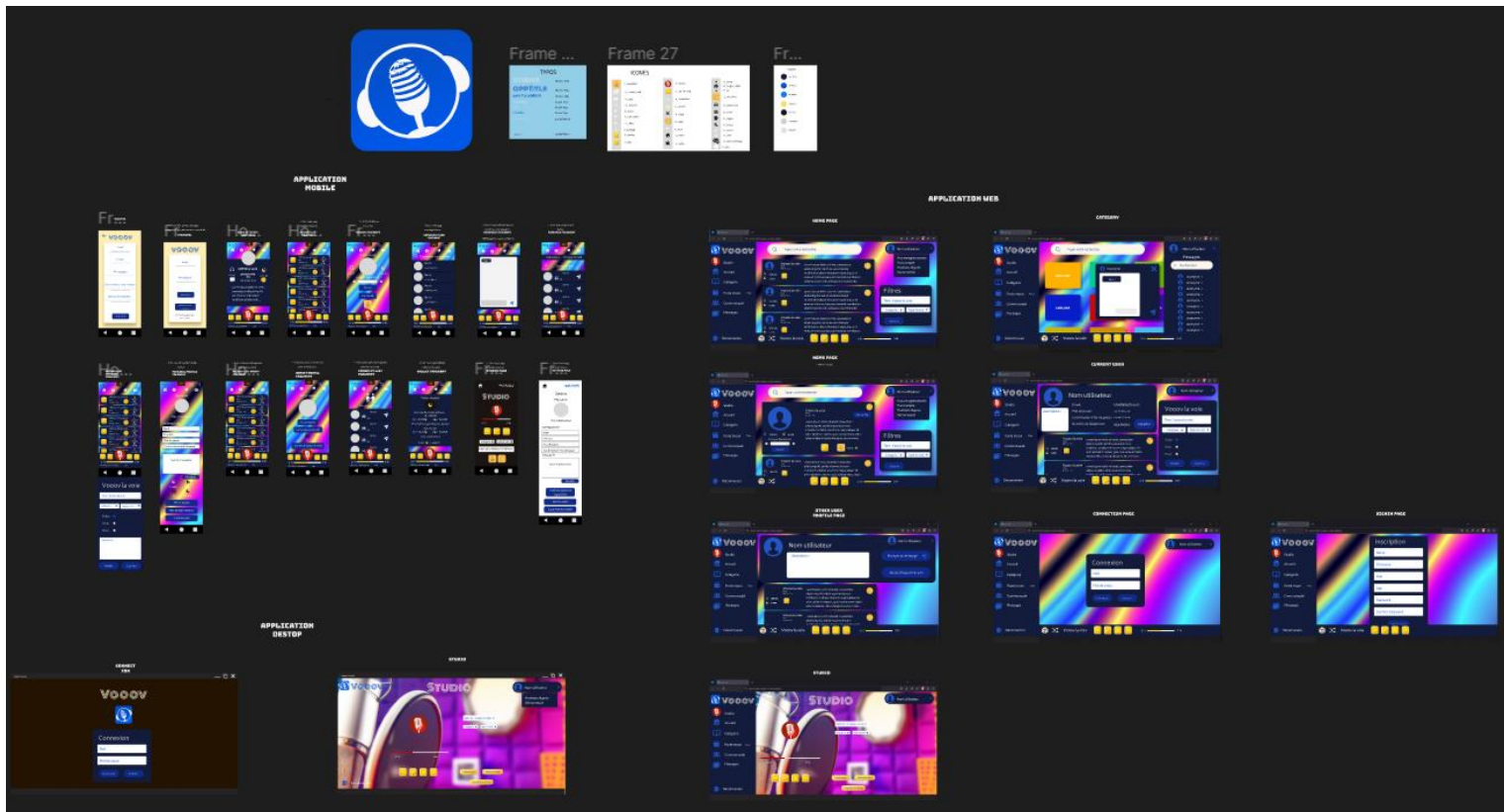
Le visuel général permet également de visualiser plus précisément le projet et les différentes interactions qui auront lieu. Cela nous aide pour la partie suivante : La modélisation des données.

La majorité des icônes sont réalisées via l'outil Inkscape.

Vous pouvez retrouver le fichier complet en cliquant sur ce lien :

<https://www.figma.com/file/G2RugGS4NajZIOXKEV5JHs/OFFO?node-id=0%3A1&t=Kkk0Ub5RggEP4woG-1>

En voici également une capture d'écran :



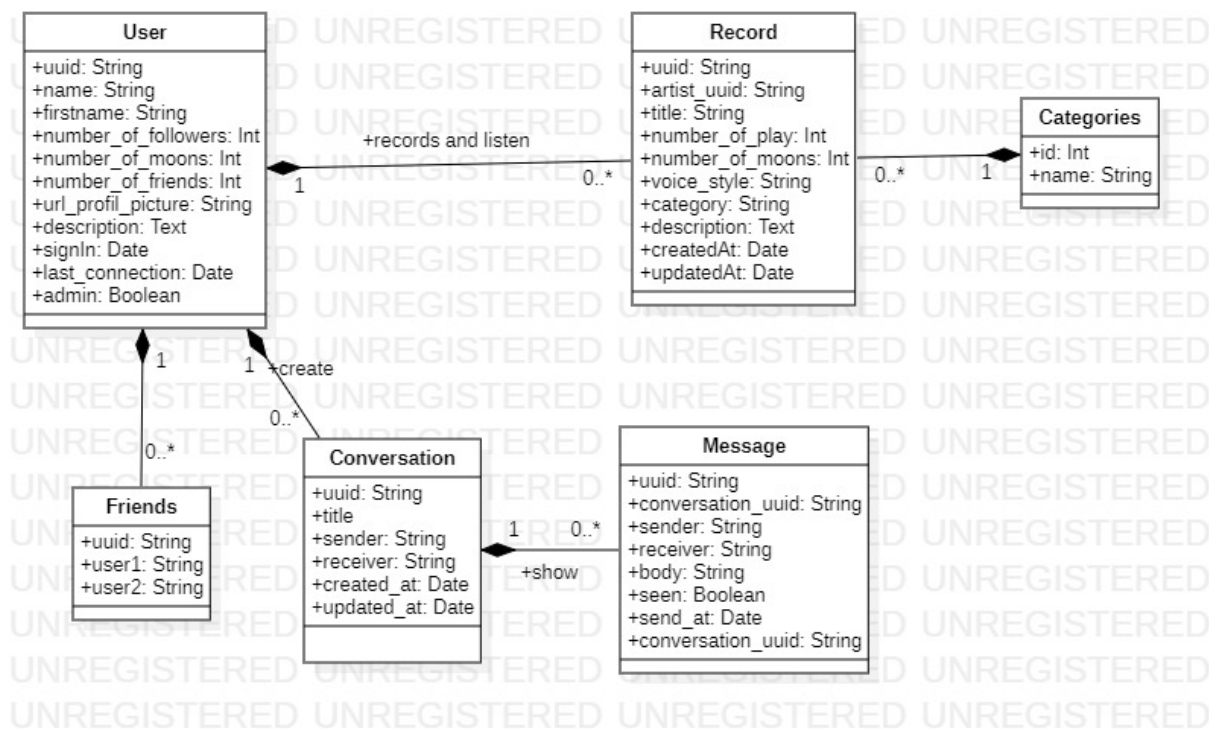
Modélisation des données et structure du projet

Modélisations des données

La modélisation des données consiste en trois phases : MCD (Modélisation Conceptuel des Données), MLD (Modélisation Logique des Données) et Le MPD (Modélisation physique des données).

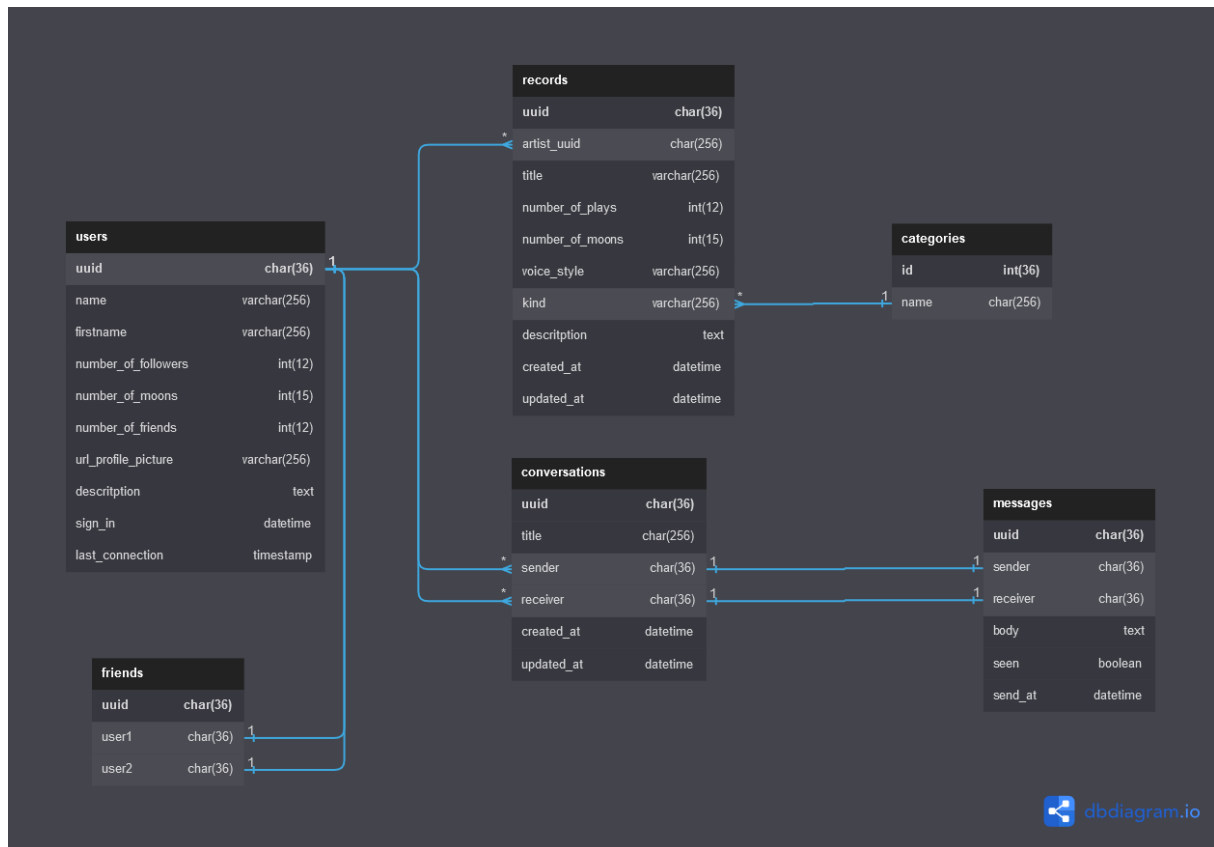
La MCD permet de présenter une vue d'ensemble des données et ainsi, apporter une visibilité sur la finalité de l'application. Elle est réalisée ici via le logiciel StarUML.

La voici :



Vient ensuite la MLD. Celui-ci découle du MCD, en introduisant les relations et les détails contextuels nécessaires à la structure des données. Elle permet de s'approcher de la mise en œuvre finale. Elle met notamment en avant les clés primaires, les clés étrangères le type de données en base de données et le nombre de caractères par ligne de chaque table (cela permet de connaître le poids d'une ligne et ainsi poids de la table selon la quantité de lignes). Elle est mise en œuvre ici grâce à l'outil dbdiagram.io.

Voici le diagramme en résultant (NB : les clés primaires ne sont pas présentes sur la représentation) :



L'étape de la MPD intervient dans la phase suivante de création et met en œuvre les deux étapes précédentes. Elle sera présentée au prochain point.

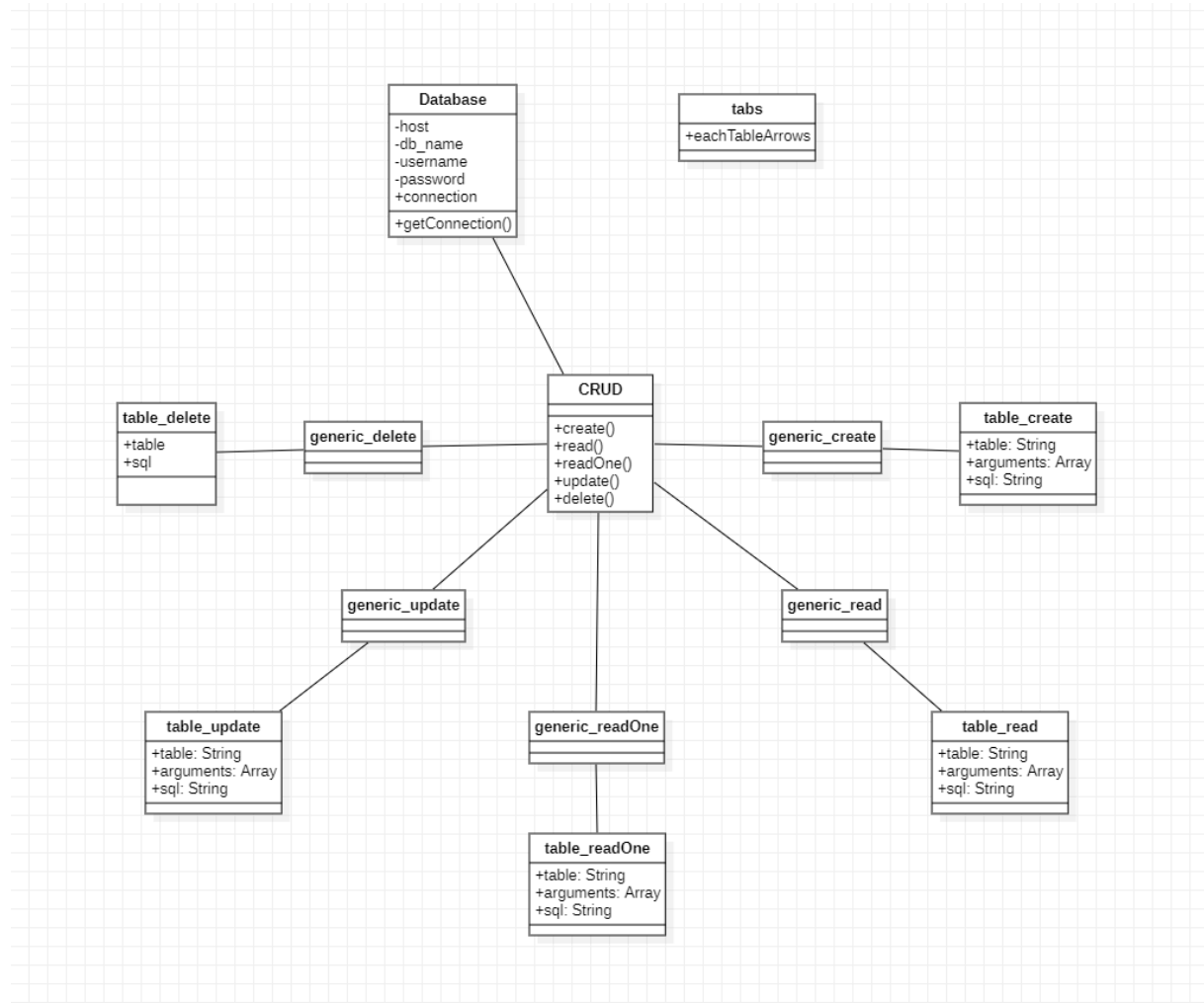
Pour prévoir au mieux les étapes du projet et répartir plus facilement les tâches, il convient de réaliser des diagrammes de classes. Chaque application nécessite sont diagramme, y compris l'API. Ceux-ci sont des bases de travail et sont amenés à évoluer et se préciser selon les besoins. Le design en place permet d'y voir plus clair.

L'architecture du projet est constituée de la modélisation des données et des diagrammes de classes. C'est elle qui permet d'anticiper et d'organiser la structure pour la programmation d'une application. Elle continue de se développer au fil de la création et devient de plus en plus précise à mesure que le projet se développe.

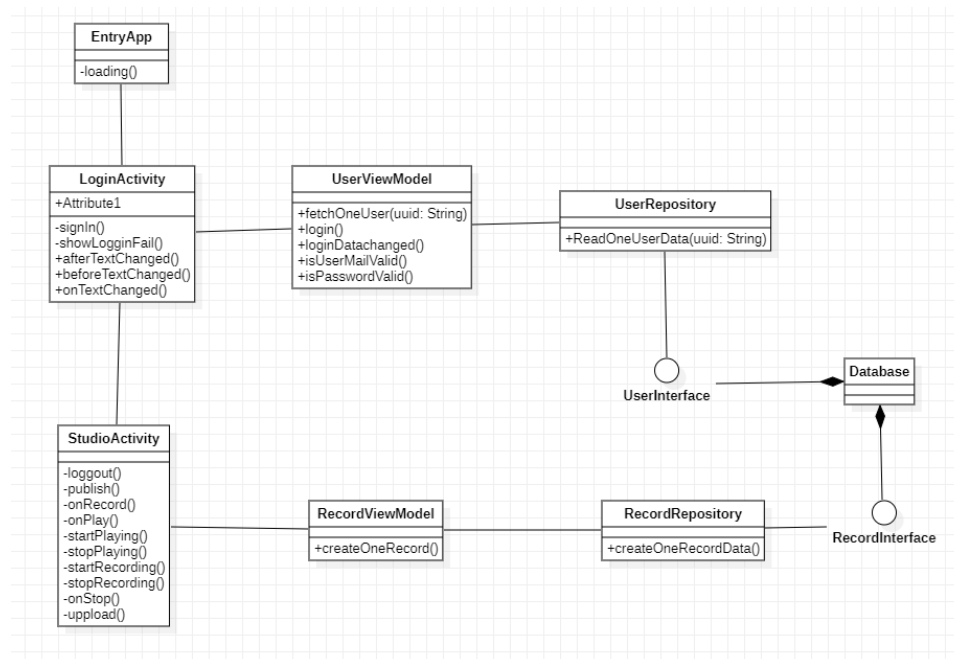
Vous pouvez retrouver les diagrammes en pdf dans le dossier uml, nommés au nom de chaque application.

En voici également des captures d'écran :

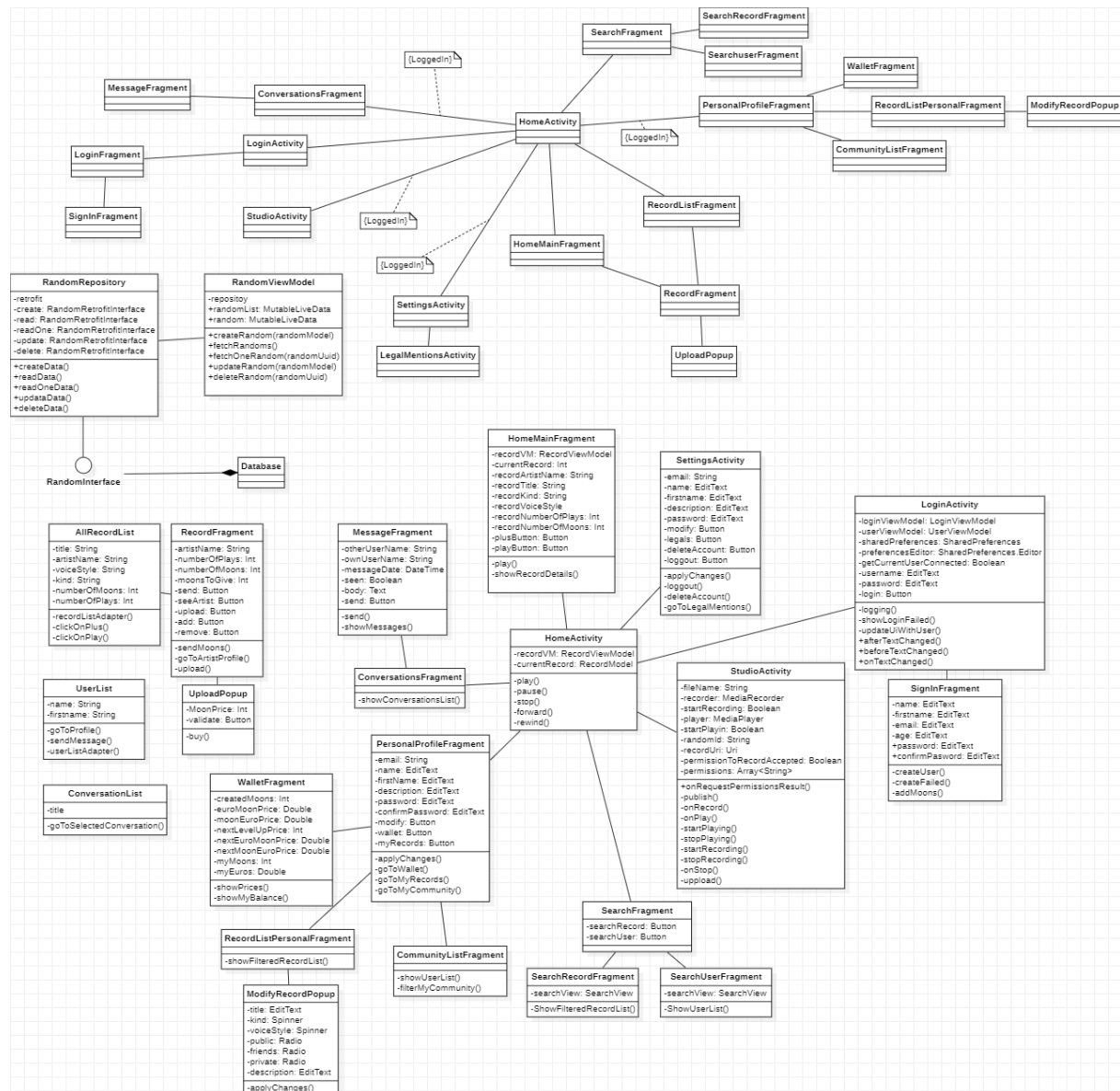
API :



Application lourde :



Application mobile :



16



Base de données

Choix de base de données

Le choix de la base de données doit déjà avoir été pensé en amont. Cependant, nous revenons dessus dans ce chapitre traitant de l'implantation de la base de données. Les bases de données sont des conteneurs d'information stockées sur serveur. Elles désignent une collection d'informations organisées pour être facilement consultables, gérables et mises à jour. Elles peuvent être très structurées dans des tables, comme c'est le cas pour une base de données relationnelles manipulable via le langage SQL. Elles peuvent également être « non structuré » comme c'est le cas du noSQL. Celle-ci sont souvent organisées en JSON au format clé-valeur.

Chaque base de données peut avoir des avantages selon son utilisation. Certaines ont un prix plus élevé que d'autres. Certaines permettent une organisation plus lisible, d'autres permettent de stocker et d'accéder à des données plus volumineuses plus rapidement. On parle d'équilibre entre les trois vecteurs d'une base de données. Selon le théorème de Brewer, une base de données répartie sur plusieurs serveurs ne peut garantir simultanément cohérence, disponibilité et tolérance au partitionnement. Dans la théorie, chaque type de base de données ne peut posséder que deux de ces trois caractéristiques. C'est donc sur ce facteur qu'il faut s'appuyer lors du choix de la base.

Les bases de données relationnelles sont les plus répandues aujourd'hui. Elles restent simples à maintenir et à faire évoluer. De plus elles répondent aux besoins scolaires et utilisent le langage SQL, tout en proposant une rapidité d'exécution très satisfaisante. Elles permettent l'utilisation d'outils tels que MySQL avec phpMyAdmin, gratuits, pour les gérer. C'est donc une base de données relationnelle SQL qui sera utilisée pour ce projet.

Une fois le type de base de données à utiliser confirmée, il est nécessaire de choisir le SGBD (Système de gestion de base de données). MySQL est un SGBD open source et gratuit. L'application web écrite en PHP : PHPMyAdmin, permet de gérer cette SGBD. Cet outil fonctionne parfaitement et est compatible avec la plupart des serveurs compatibles à PHP. C'est donc cet outil qui servira pour l'ensemble du projet.

MPD et Script SQL

Une fois le choix du type de base de données validé, le langage nécessaire pour sa manipulation en découle naturellement. C'est ici que commence l'étape du modèle physique de données (MPD). Elle consiste à transformer le résultat de la MLD en langage SQL.

La première étape consiste à créer la base de données via la ligne de commande suivante :

```
CREATE DATABASE IF NOT EXISTS `voov` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
```

La deuxième étape consiste à créer les différentes tables qui ont été mises en lumière lors de la création de l'architecture du projet. Celles-ci sont illustrées par le diagramme de base de données via l'outil dbdiagram.io.

La première table liste les utilisateurs :

```
CREATE TABLE `users` (  
  `uuid` char(36) NOT NULL PRIMARY KEY, /*Les clés primaires ne sont pas auto-  
incrémentées. Le choix d'avoir des uuid permet d'éviter les collisions lors de  
création de nouvelles lignes, mais ne permet pas l'auto-incrémentations en  
SQL*/  
  `name` varchar(256) NOT NULL,  
  `firstname` varchar(256) NOT NULL,  
  `email` varchar(256) NOT NULL,  
  `password` varchar(256) NOT NULL,  
  `phone` varchar(256) NOT NULL,  
  `description` text NOT NULL,  
  `number_of_followers` int(12) NOT NULL,  
  `number_of_moons` int(15) NOT NULL,  
  `number_of_friends` int(12) NOT NULL,  
  `url_profile_picture` varchar(256) NOT NULL,  
  `sign_in` datetime NOT NULL,  
  `last_connection` datetime NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

La deuxième table liste les enregistrements audios :

```
CREATE TABLE `audio_records` (  
  `uuid` char(36) NOT NULL PRIMARY KEY,  
  `artist_uuid` char(36) NOT NULL,  
  `title` varchar(256) NOT NULL,  
  `number_of_plays` int(12) NOT NULL,  
  `number_of_moons` int(15) NOT NULL,  
  `voice_style` varchar(256) NOT NULL,  
  `kind` varchar(256) NOT NULL,  
  `description` text NOT NULL,  
  `created_at` datetime NOT NULL,  
  `updated_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  FOREIGN KEY (artist_uuid) REFERENCES users(uuid) ON DELETE CASCADE ON  
UPDATE CASCADE,  
  FOREIGN KEY (kind) REFERENCES categories(name)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

La troisième table liste les messages :

```
CREATE TABLE `messages` (  
  `uuid` char(36) NOT NULL PRIMARY KEY,  
  `sender` char(36) NOT NULL,  
  `receiver` char(36) NOT NULL,  
  `body` text NOT NULL,  
  `seen` boolean NOT NULL,  
  `send_at` datetime NOT NULL,
```

```

    FOREIGN KEY (sender) REFERENCES users(uuid) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (receiver) REFERENCES users(uuid) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

La quatrième table créée liste les conversations :

```

CREATE TABLE `conversations` (
  `uuid` char(36) NOT NULL PRIMARY KEY,
  `sender` char(36) NOT NULL,
  `receiver` char(36) NOT NULL,
  `title` text NOT NULL,
  `created_at` datetime NOT NULL,
  `updated_at` datetime NOT NULL,
  FOREIGN KEY (sender) REFERENCES users(uuid) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (receiver) REFERENCES users(uuid) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

La cinquième table créée récupère les liens d'amitié :

```

CREATE TABLE `friends` (
  `uuid` char(36) NOT NULL PRIMARY KEY,
  `user1` char(36) NOT NULL,
  `user2` char(36) NOT NULL,
  FOREIGN KEY (user1) REFERENCES users(uuid) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (user2) REFERENCES users(uuid) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

La sixième table liste les catégories des enregistrements. Cette table n'est pas vouée aux utilisateurs. Elle comporte un nombre de lignes plus restreint :

```

CREATE TABLE `categories` (
  `id` int(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `name` char(36) NOT NULL,
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

Afin de sécuriser l'entrée des données dans la base, nous appliquons des clés étrangères. De cette manière, il est impossible de créer de nouveau lien d'amitié si l'uuid d'un des utilisateurs n'existe pas. De plus la ligne sera supprimée si l'un des utilisateurs de ce lien venait lui-même à l'être.

Fin premier projet

Début deuxième projet

Implémentation des services

Le site web, la base de données et l'API seront stockés sur un serveur VPS et j'utiliserai l'outil « cyber panel » pour la gestion du serveur.

Cependant, pour tout projet, il est préférable de commencer par des tests de mise en place sur un environnement local. Pour se faire, le logiciel XAMPP, permet l'hébergement d'un serveur apache virtuel et l'accès au SGBD MySql via PHPMYAdmin. Chaque étape testée en locale sera détaillée au moment de sa réalisation.

Création du VPS

La création d'un VPS (Virtual Private Server) auprès d'un hébergeur consiste à partitionner un serveur physique en plusieurs serveurs virtuels indépendants. De cette manière, il est possible de dédier une espace de mémoire définit, ainsi qu'une RAM, une bande passante et un espace de stockage.

Lors de la réalisation de mes précédents projets j'avais opté pour la solution plus simple d'un serveur partagé. Cette fois, c'est l'occasion de gagner en compétences sur un déploiement plus conséquent et complet. De plus, cela apporte une meilleure maîtrise sur les capacités d'utilisation, puisqu'elles peuvent être augmentées avec la demande.

Le serveur utilisé pour ce projet est hébergé par Hostinger. Il tourne sous Linux et se situe au Royaume-Uni. Hostinger met à disposition un outil de gestion de serveur appelé « cyber panel ». Cet outil permet de simplifier la gestion des tâches en proposant une interface semblable à une interface proposée par un OS.

Déploiement et hébergement des différents services

Concernant la gestion de la base de données, Cyber panel donne accès automatiquement à PHPMYAdmin. L'utilisation est donc la même que dans un environnement local. Nous y reviendrons dans la partie qui y est consacrée un peu plus bas.

Le déploiement de site internet demande quelques manœuvres supplémentaires. En effet, il est assez simple de créer un nouveau site de test par exemple. En revanche, pour créer un site web pourvu d'un nom de domaine et validé par un certificat SSL, les étapes sont plus complexes. Pour le moment, nous pouvons créer des sites accessibles depuis internet via l'IP du serveur suivit du chemin d'accès vers les fichiers.

Voici les étapes à remplir dans cyber panel dans un premier temps :

WEBSITE DETAILS

Select Package

Default

Select Owner

admin

Test Domain

☐

Domain Name

voooov-api.fr

Email

nathan.kedinger@gmail.com

Select PHP

PHP 8.1

Additional Features

☒ SSL

☒ DKIM Support

☒ open_basedir Protection

☒ Create Mail Domain

Create Website

Cette action aura pour effet de créer un site accessible depuis l'onglet list website. Voici à quoi ressemble sa page :

VOOOV-API.FR - PREVIEW
All functions related to a particular site.

RESOURCE USAGE

STRESS TEST

SET UP SSH/SFTP ACCESS

CLONE/STAGING

MANAGE GIT

Resource	Usage	Allowed
FTP	0	1000
Databases	1	1000
Disk Usage	2 (MB)	0 (MB)
Bandwidth Usage	0 (MB)	0 (MB)

VOOOV-API.FR HAS SELF-SIGNED SSL.
Your SSL will expire in 3635 days.

Disk Usage

Bandwidth Usage

LOGS

Access Logs

Error Logs

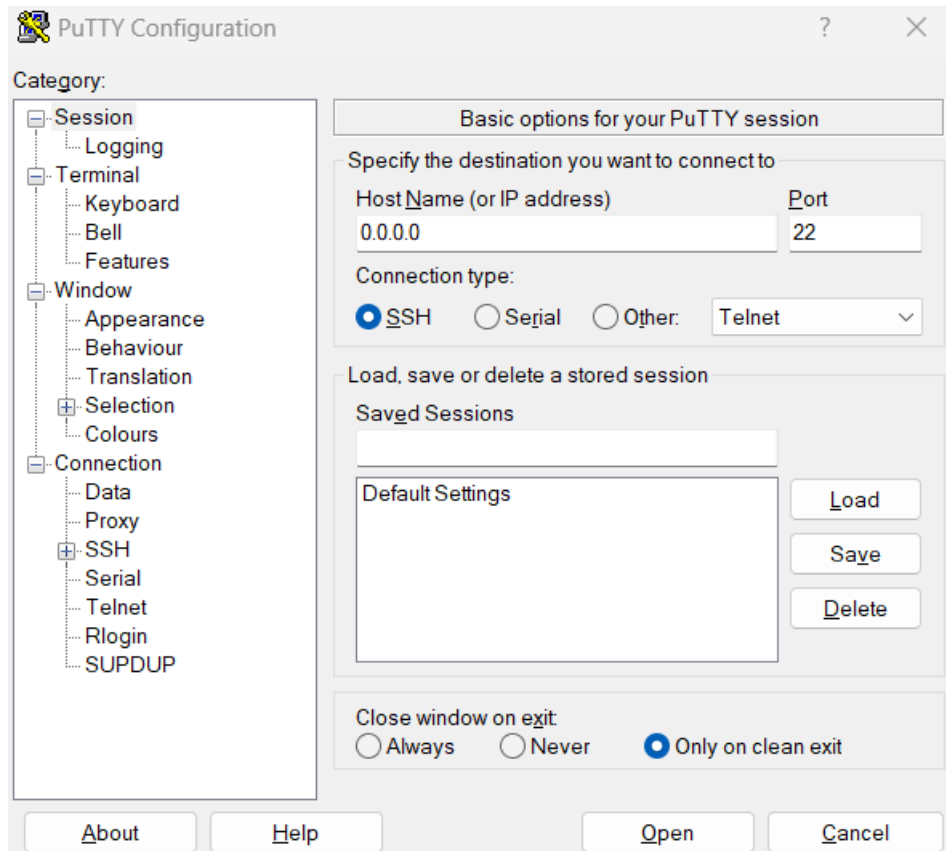
DOMAINS

Le problème survenant à la suite de cette action est un certificat SSL auto-signé. Cela pose des problèmes à plusieurs niveaux. Les sites, ne possédant pas de certificat SSL sécurisés, sont répertoriés par les navigateurs web, comme non sécurisés. Cela affiche une page d'erreur au moment de l'accès à la page.

Un autre problème que cela soulève est qu'Android bloque systématiquement l'accès à des sites non sécurisé, depuis ses applications. Cela nous empêche d'accéder à l'API depuis l'application mobile.

Bien que l'interface Cyberpanel permette un grand nombre d'actions, il reste nécessaire de régler quelques détails directement en ligne de commande. Notamment en ce qui concerne la sécurité. Mais également pour ce détail-ci, la validation du certificat SSL, par exemple. Pour pouvoir y parvenir, il est nécessaire de se connecter au serveur en tant que client. C'est possible grâce à un émulateur de terminal doublé d'un client pour les protocoles SSH. PuTTY convient ici parfaitement.

Afin de se connecter au serveur via le protocole SSH, nous devons entrer l'IP du serveur et son port d'accès :



Un fois la connexion au service effectué, nous devons valider les informations de connexion.

Une fois ceci fait, nous sommes transportés sur ce Terminal de commande :

```
root@vov-server:~  
*****  
Welcome to LiteSpeed One-Click CyberPanel Server.  
To keep this server secure, the firewalld is enabled.  
CyberPanel One-Click Quickstart guide:  
* https://docs.litespeedtech.com/cloud/images/cyberpanel/  
  
In a web browser, you can view:  
* CyberPanel: https://  
* phpMyAdmin: https:// /dataBases/phpMyAdmin  
* Rainloop: https:// /rainloop  
  
On the server:  
* You can get the CyberPanel admin password with the following command:  
  sudo cat .litespeed_password  
* You can get the Mysql cyberpanel user password with the following command:  
  sudo cat .db_password  
  
System Status:  
  Load : 0.00, 0.01, 0.05  
  CPU   : 0.491778%  
  RAM   : 327/2048MB (15.97%)  
  Disk  : 6/40GB (18%)  
  
Your CyberPanel is up to date  
*****  
[root@vov-server ~]#
```

Avant de pouvoir appliquer le certificat à nos sites web, Il est nécessaire de gérer la partie DNS qui n'est pas automatiquement activée. Afin de pouvoir récupérer un certificat SSL fournit par l'hébergeur, les étapes à suivre sont donc les suivantes :

- Acheter un nom de domaine auprès de l'hébergeur. Pour ce projet, ce sera vooov.fr qui sera utilisé. L'API sera également stockée sous ce nom de domaine. Le serveur est de toute façon le même. Cela réduit les coûts de les mutualiser.
- Mettre à jour les enregistrement DNS et de supprimer celui attribué par défaut par l'hébergeur, pour le remplacer par celui pointant vers l'IP du serveur VPS.
- Mettre à jour le serveur : `sh <(curl https://raw.githubusercontent.com/usmannasir/cyberpanel/stable/preUpgrade.sh || wget -O -`
- Mettre à jour le client SSL ACME via cette commande : `wget -O - https://get.acme.sh | sh`
- Imprimer un journal de log d'erreurs supplémentaires qui pourrait encore subsister : `cat /home/cyberpanel/error-logs.txt` (Pas d'erreurs retournées)
- Dernière étape à effectuer dans la console pour appliquer le certificat :
`/root/.acme.sh/acme.sh --issue -d yourdomain.com -d www.yourdomain.com --cert-file /etc/letsencrypt/live/www.rmronsol.com/cert.pem --key-file /etc/letsencrypt/live/yourdomain.com/privkey.pem --fullchain-file /etc/letsencrypt/live/yourdomain.com/fullchain.pem -w /home/yourdomain.com/public_html --server letsencrypt --force --debug`
- Retourner dans CyberPanel dans la section SSL et cliquer sur le issue SSL du domaine concerné.

Problème rencontré

Impossible de débloquent le certificat SSL pour l'application ou le site web. Il fallait d'abord acheter un nom de domaine et le faire pointer vers le serveur.

Sécurisation du serveur

Le serveur est fonctionnel, le nom de domaine qui hébergera le site web ainsi que l'API est en ligne et disponible. En revanche, j'ai pu remarquer grâce à la connexion via PuTTY, que des centaines de tentatives de connexion au serveur avait lieu entre chacune de mes connexions. Mon accès root semblait être la cible parfaite pour des attaques par force brut depuis des bots lancés au hasard d'internet.

En premier lieu, j'ai doublé la longueur de mon mot de passe.

Dans un second temps, j'ai modifié mon port d'accès 22 qui est le port standard d'accès SSH. De cette façon, les bots automatisés réglé sur le port 22 n'y accèdent plus automatiquement. Pour se faire, Il faut se connecter au SSH via PuTTY puis insérer la ligne de commande suivante :

```
sudo nano /etc/ssh/sshd_config
#Entrer en bas de la liste, la ligne :
Port #le port choisi
#Puis par un redémarrage du serveur :
Service sshd restart
```

NB : Bien penser à modifier l'accès au firewall s'il y en a un. Dans le cas contraire, plus possible de se connecter.

Améliorations possibles

La sécurité d'accès au serveur peut encore être améliorée. Pour se faire, il faudrait commencer par activer les firewalls qui contrôle les adresses IP de connexion. On pourrait ensuite changer le nom de l'utilisateur root par un autre possédant des droits limités selon utilisation. Il faudrait ensuite changer l'accès de connexion de l'utilisateur par clé rsa.

Le déploiement de l'application mobile se fera sur le serveur d'Android. En attendant, elle est déployée directement sur les appareils mobiles en les connectant au pc sur lequel est stockée l'application par un simple build dans l'application Android studio.

Mise en place d'un service de versioning

Utilité du versioning

Les services de versioning possèdent plusieurs objectifs :

- Sécuriser les sauvegardes. Ils permettent l'application de la règle 3-2-1 : Trois copies des données, une originale et deux copies, sur deux supports différents, dont une hors site (c'est ici qu'intervient le service de versioning).
- Garder une trace des modifications apportées au code. Il est possible d'accéder aux anciennes versions et d'effectuer un retour en arrière si besoin.
- Permettre une simplification des mises à jour de déploiement.
- Faciliter le travail en équipe en fournissant une version commune à l'ensemble des collaborateurs.
- Git permet le partage et la réutilisation de code entre différents partis (autres développeurs, étudiants, correcteurs d'épreuves de BTS...)

L'outil le plus utilisé pour le versioning est Git. Le versioning consiste à figer le code d'un projet à un instant donné. Pour ce faire, il faut inclure ce projet dans un répertoire (ou repository) Git. A chaque nouvel enregistrement du projet dans Git, cela crée une nouvelle version. Cette version enregistre uniquement les modifications ayant eu lieu depuis la version précédente.

Git a été développé par Linus Torvald, le créateur du noyau Linux. C'est un outil libre et disponible sur tous les systèmes d'exploitation.

Utilisation de Git

Pour utiliser Git, il est d'abord nécessaire de l'installer. Le plus simple étant généralement d'installer un plugin dans l'IDE utilisé. Aujourd'hui, la plupart en fournissent.

Les étapes de dépôt d'une nouvelle version sur Git consiste à écrire en ligne de commande :

```
git init # On initialise git
git add . # Cette commande ajoute tous les fichiers. C'est pourquoi, si l'on
souhaite empêcher certains fichiers d'être ajoutés au repository il est
judicieux d'ajouter un fichier .gitignore qui contiendra la liste de tous les
fichiers à ne pas transmettre dans un commit. Par ailleurs, il est possible
d'ajouter les fichiers un par un. Pour se faire, remplacer le point par le nom
des fichiers à ajouter.
git commit -m "commentaire obligatoire qui comporte une liste rapide des
modifications apportées"
git push # Cette commande est la commande finale qui envoie la capture du code
à l'instant du commit.
```

Aujourd'hui, il existe principalement deux outils plus larges simplifiant l'usage de Git. Ils proposent grâce à celui-ci, d'autres services facilitant le travail du code, et des développeurs. Ce sont Github, proposé par Microsoft, et GitLab, un logiciel open source.

Chacun de ces outils possède ses qualités propres. Toutefois, la majeure différence réside dans le fait, que GitLab intègre des flux de travail d'intégration continue/ livraison continue et de DevOps. Ce sont des atouts majeurs pour des projets d'équipe.

GitLab semble particulièrement destiné aux équipes de développeurs. Github quant à lui semble suffisant pour un développeur seul. L'avantage de Github vient également de sa notoriété et du grand nombre d'utilisateurs en faisant usage. Les recruteurs sont plus actifs, à ma connaissance, sur Github que sur GitLab.

Utilisation pour le projet

J'utilise Github depuis le début de ma formation. J'utiliserai Github pour le versioning de ce projet. Cependant, il est tout à fait possible de lier ces deux outils ensemble, et même de migrer de l'un à l'autre selon les besoins.

L'un des points fort du versioning avec Git, est qu'il est possible de lier les répertoires de fichier des IDE vers le service Git choisi.

Dans l'autre sens, cela permet de lier le répertoire Git au répertoire de fichier hébergé sur un serveur. De cette façon l'intégration des nouvelles versions sur serveur est très rapide et simple. Nul besoin de copier-coller le code, au risque de produire des erreurs.

CyberPanel propose ce service. Pour se faire, il est nécessaire de se rendre dans la liste des site web, sélectionner le site puis cliquer sur l'onglet « manage Git ». Choisir, soit d'utiliser un repo existant, ou d'en créer un. Dans tous les cas, il est nécessaire de relier le dépôt à Github par la suite, à l'aide d'une clé de déploiement, à communiquer au site Github. Une fois les vérifications effectuées, il est possible d'importer le dépôt présent sur Github d'un simple clic, sur l'onglet « pull ».

Un détail à noter, toutefois, il est déconseillé de modifier les fichiers directement sur le serveur une fois un répertoire lié. Cela a pour effet de créer un conflit d'écrasement de données et bloque le pull.

Vous pouvez retrouver l'ensembles des réalisations de ces projets cette adresse :

<https://github.com/nathan-kedinger?tab=repositories>

Ticketing et tests

Le ticketing

Le ticketing est mis en place via des outils de ticketing. On pense par exemple au très notoire GLPI. Un outil de ticketing sert principalement dans un projet passé en phase de production. Cependant, un relevé de logs d'erreur est très important pour un projet en phase de test, également en phase de conception.

En phase de production, il permet directement aux utilisateurs, de faire un retour au service client. Il permet de mémoriser l'ensemble des réclamations, problèmes, erreurs et incidents remontés par les utilisateurs. Il permet d'effectuer un archivage et un suivi de chaque problème. Il permet d'évaluer la priorité d'un ticket, son degré d'avancement ou encore l'état d'avancement du problème.

Un système de ticketing permet d'archiver de manière centralisée tous les problèmes rencontrés. De cette façon, l'équipe technique peut visualiser simplement l'ensemble des problèmes à résoudre.

Phase de conception et de test

Lors de la phase de conception, un outil de ticketing destinés à des équipes de services clients n'a pas de sens.

En revanche, il existe bien des outils permettant de remonter et lister les problèmes rencontrés au cours de cette phase. Le premier outil à utiliser par les développeurs découle de la partie précédente : Déclaration d'"Issues" sur le site github. En projet d'équipe, mais également seul, les issues permettent de garder un suivi des problèmes à résoudre.

L'outil utilisé pour ce projet est également un outil collaboratif. Cet outil se nomme Trello. Il offre une visibilité sur la liste des problèmes rencontrés. Son avantage est son nombre d'utilisation possibles. Il peut également servir à stocker les tâches à effectuer, celles en cours, les idées... Il peut être partagé entre plusieurs acteurs et les tâches effectuées ne sont pas supprimées mais simplement rayées. Cela laisse donc une trace. Il est possible de retourner en arrière si nécessaire...

Les autres outils utiles pour la gestion des erreurs sont directement implantées dans le serveur. Ce sont les fichiers de logs. Dans CyberPanel, deux fichiers de logs sont enregistrés. Le premier est un fichier de logs de connexion. Il répertorie l'ensemble des connexions ayant eu lieu sur le serveur. Le deuxième est un fichier de logs d'erreur, ayant eu lieu au cours de son utilisation.

Cependant, les erreurs relevées par ces logs restent limitées (principalement des erreurs internes au serveur). Afin de récupérer un maximum d'erreurs dans le code et son utilisation, il est intéressant d'implanter un maximum de « try catch » dans le code source des diverses applications.

Il est possible par la suite de lier ce fichier de logs à des outils ticketing de manière automatisée, en envoyant un ticket par mail à chaque erreur relevée. Cette approche fait sens, puisque les utilisateurs ne remonteront pas systématiquement une erreur survenue lors de l'utilisation de l'application. Ils auront plutôt tendance à stopper son utilisation.

Une gestion de logs automatisée est également un gros avantage lors de la phase de création et de test. Les « try catch » permettent de remonter des erreurs détaillées. Ils permettent également de cibler la recherche d'une erreur particulière lors d'une tentative de débogue du code.

Phase de production

Lors de la phase de production, l'ensemble des outils de relevé d'erreurs mis en place pour la phase de création et de test restent en place. (Pour les try catch et autres relevés d'erreurs uniquement. Dans un projet Symfony, il est important de le passer le projet en mode production. Cela permet de cacher les erreurs aux utilisateurs). Un formulaire de contact est ajouté dans chaque application, afin de permettre la remontée des erreurs ou des réclamations par les utilisateurs. Ces retours d'erreurs sont gérés par l'outil trac, un logiciel libre et open source.

En phase de production, le code passe sur GitLab afin de permettre l'intégration continue

Déploiement de la base de données

Le déploiement de la base de données est la première implémentation concrète du projet. Celle-ci stockera l'ensemble des données des différentes applications et les mettra en lien. C'est donc nécessairement le point d'entrée logique.

Dans un premier temps, bien que ce ne soit pas obligatoire, il est judicieux de tester l'implantation de la base de données dans un environnement local. Pour se faire, nous utilisons l'outil XAMPP présenté précédemment. L'outil PHPMyAdmin qu'il propose permet d'entrer des requêtes SQL en direct.

Une fois le script précédemment construit intégré à la base, et fonctionnel, nous testons l'implantation de données dans la base. Les contraintes ajoutées dans les requêtes de créations de table nous obligent à suivre un certain ordre. La première donnée à entrer doit forcément être un utilisateur. L'ensemble des autres tables (excepté la table categories) ont besoin d'un uuid utilisateur pour créer des données.

Voici le script d'ajout d'un utilisateur :

```
INSERT INTO `users` (`uuid`, `name`, `firstname`, `email`, `password`,  
`phone`, `description`, `number_of_followers`, `number_of_friends`,  
`number_of_moons`, `url_profile_picture`, `sign_in`, `last_connection`) VALUES  
('1', 'John', 'Doe', 'hasard@gm.com', 'password', '0606060606', 'a young  
person working in IT', 0, 0, 0, 'randomUrl', '2019-08-30 15:34:33', '2021-08-  
30 15:36:33')
```

Procédures stockées

L'utilisation de procédures stockées consiste à créer des fonctions qui viendront être appliquées directement en base de données. Seul l'appel d'exécution de la procédure est envoyé. Cela permet de traiter le code en un seul lot. De cette manière, on réduit considérablement le trafic réseau entre le serveur et le client.

Dans cette application, certaines fonctionnalités devront être appelées plus régulièrement que d'autres. Par exemple, la lecture d'un utilisateur ou d'un ou plusieurs enregistrements audios. Pour cette raison, des procédures stockées seront appliquées à ces requête directement sur serveur. Voici comment en présenter une pour retrouver un utilisateur par son nom :

```
DELIMITER $$  
CREATE PROCEDURE find_user_by_name  
(IN user_name CHAR(256))  
BEGIN  
    SELECT * FROM users WHERE name = user_name;  
END $$  
DELIMITER ;
```

Il ne reste plus qu'à implanter l'appel de cette procédure depuis l'API :

```
"CALL find_user_by_id(" . $userName . ")";
```

Conception, création et implémentation de l'API

Conception

Afin de concevoir une API (Application Programming Interface), il est nécessaire d'identifier les besoins auxquels celle-ci doit répondre.

Pour quelle raison utiliser une API ? Dans un projet tel que celui-ci, il est essentiel de pouvoir accéder à la base de données depuis les différents supports utilisés. Une Api permet d'encapsuler le code communiquant avec la base de données. Dans un projet reparté sur plusieurs serveurs différents, c'est la méthode la plus simple et efficace pour permettre la communication des différents acteurs.

Un standard a été défini concernant l'architecture des API. C'est le standard REST, l'abréviation du terme anglais « Representational State Transfer ». Il résulte d'un style d'architecture créé en 2000 par Roy Fielding. Bien qu'il en existe encore d'autres, cohérent pour certaines utilisations, c'est le standard le plus répandu aujourd'hui. Pour qu'une API soit dite REST, elle doit répondre à certaines contraintes. A savoir :

- L'architecture client-serveur : Les serveurs et les clients ne doivent connaître que les URL des ressources, sans dépendances.
- L'absence d'état : Le serveur ne doit pas faire de relations entre les différents appels des clients, ni même d'un même client. Il n'a pas connaissance de l'état du client entre ces transactions.
- La mise en cache des ressources : Le client doit être capable de garder les données en cache pour optimiser les transactions.
- Une interface uniforme : Tous composant qui comprend le protocole HTTP doit pouvoir communiquer avec l'API.
- Un système de couches : Permet d'organiser les différents types de serveurs. Cela permet une répartition des charges et de rendre l'application plus flexible.
- Le code à la demande : l'API peut envoyer du code exécutable au client.

Cette API tachera donc de respecter le mieux possible ce standard.

Le prochain point consiste à déterminer les données à gérer pour notre application. Nous pouvons ici nous servir du travail réalisé dans les différents diagrammes présentés plus haut.

Créations de l'API

L'API est codée en PHP via Visual Studio Code avec un suivi du versioning sur GitHub, visible à cette adresse : https://github.com/nathan-kedinger/api_vooov. Vous trouverez à suivre les différentes étapes suivies lors de la réalisation de cette API.

Dans un objectif de test, l'API est d'abord déployée en local, grâce à l'outil XAMPP. J'entame mes tests sur la table « users ».

La première étape consiste à créer un fichier permettant la connexion à la base de données. Créé un fichier spécifique à la connexion permet à la fois d'améliorer la portabilité et d'éviter la redondance de code à chaque appel de la base. Cela sécurise également les identifiants de connexion. En effet, cela

permet de stocker ce fichier à la racine du serveur (en amont du dossier « public_html »), et donc d'éviter qu'il soit accessible à des utilisateurs externes.

Ce fichier comporte une classe « Database » dans laquelle sont établis des identifiants de connexion. Il comporte également une méthode getConnection() qui permet la connexion via un objet PDO. L'objet PDO permet la sécurisation des données importées dans la base.

La méthode getConnection() peut ensuite être appelée depuis le reste de l'API.

Voici le contenu du fichier « Database » pour une utilisation locale :

```
<?php

class Database{
    private $host = "localhost";
    private $db_name = "api_vooov";
    private $username = "root";
    private $password = "";
    public $connection;

    //getter for connection
    public function getConnection(){
        //closing the connection if exists
        $this->connection = null;

        // try to connect
        try{
            $this->connection = new PDO("mysql:host=" . $this->host . ";
dbname=" . $this->db_name, $this->username, $this->password);
            $this->connection->exec("set names utf8"); // force transaction
in <UTF-8></UTF-8>
        }catch(PDOException $exception){
            echo "Erreur de connexion : " . $exception->getMessage();
        }

        return $this->connection;
    }
}
```

L'étape suivante consiste à créer un fichier CRUD (Create, Read, Update, Delete). Dans un premier temps, les CRUD que j'ai réalisé étaient spécifique à une table. Ils comprenaient chacun 5 méthodes, à savoir :

```
create()
read()
readOne()
update()
delete()
```

Vous pouvez retrouver le fichier précédent ici : https://github.com/nathan-kedinger/api_vooov/blob/main/models/UsersExampleCrud.php

L'exemple de la méthode utilisée à l'origine est stocké dans le dossier « exemple ». **Il a été modifié par la suite dans un but d'amélioration. Ceci sera expliqué un peu plus bas.**

Une fois le fichier CRUD créé, il est nécessaire de créer des fichiers permettant l'interaction entre l'API et la base de données. Pour qu'une API soit REST, chaque méthode de requête est appelée par une unique méthode (GET, POST, PUT, DELETE...). En ce sens, il est nécessaire de créer un fichier spécifique pour chaque méthode.

Ces fichiers, permettant l'appel du protocole http, doivent posséder des entêtes (headers). Voici la liste des entêtes utilisées ici (en l'occurrence avec la méthode POST) :

```
// Headers
// Access from any site or device
header("Access-Control-Allow-Origin: *");

// Data format
header("Content-Type: application/json; charset=UTF-8");

// Authorised method
header("Access-Control-Allow-Methods: POST");

// Request lifetime
header("Access-Control-Max-Age: 3600");

// Authorised headers
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
```

Il est ensuite nécessaire de vérifier que la méthode utilisée lors de l'appel de l'API est correcte :

```
// Verification that used method is correct
if($_SERVER['REQUEST_METHOD'] == 'POST'){
```

Nous appelons les fichiers de connexion et de CRUD (Le fichier CRUD est ici Users). Nous instancions ensuite la Base de données, l'objet CRUD et nous récupérons les datas envoyées par l'utilisateur :

```
// DDB instantiation
$database = new Database();
$db = $database->getConnection();

// Users instantiation
$user = new Users($db);

// Get back sended informations
$datas = json_decode(file_get_contents("php://input"));
```


La partie suivante du fichier permet de récupérer les données transmises par l'utilisateur et de les renvoyer au serveur. Dans un premier temps, le code se présentait sous cette forme :

```
if(!empty($datas->name) && !empty($datas->firstname) && !empty($datas->email) && !empty($datas->phone) && !empty($datas->number_of_followers) && !empty($datas->number_of_moons) && !empty($datas->number_of_friends) && !empty($datas->url_profile_picture) && !empty($datas->description) && !empty($datas->sign_in) && !empty($datas->last_connection)){

    //here we receive datas, we hydrate our object
    $user->name = $datas->name;
    $user->firstname = $datas->firstname;
    $user->email = $datas->email;
    $user->phone = $datas->phone;
    $user->number_of_followers = $datas->number_of_followers;
    $user->number_of_moons = $datas->number_of_moons;
    $user->number_of_friends = $datas->number_of_friends;
    $user->url_profile_picture = $datas->url_profile_picture;
    $user->description = $datas->description;
    $user->sign_in = $datas->sign_in;
    $user->last_connection = $datas->last_connection;

    if($user->create()){
        // Here it worked => code 201
        http_response_code(201);
        echo json_encode(["message" => "The add have been done"]);
    }else{
        // Here it didn't worked => code 503
        http_response_code(503);
        echo json_encode(["message" => "The add haven't been done"]);
    }
}
}else{
    // We catch the error
    http_response_code(405);
    echo json_encode(["message" => "This method isn't authorised"]);
}
```

Les erreurs éventuelles étaient recueillies par un retour http en JSON.

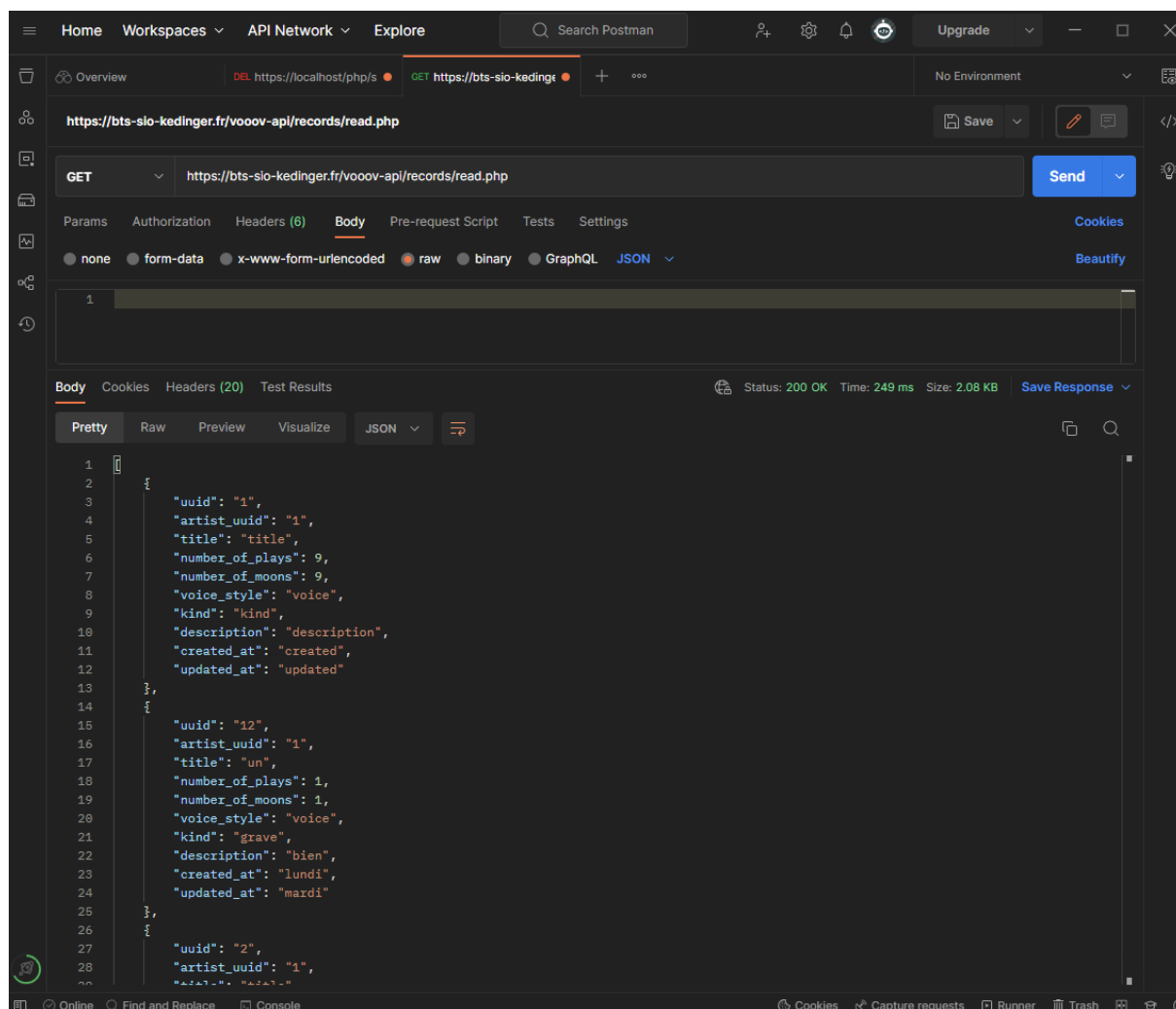
La récupération des erreurs nous permet de les situer plus précisément lors des tests.

Voici donc la méthode fonctionnelle REST pour transmettre des données à un serveur depuis une API. Nous transformons les données de l'utilisateur en format JSON. Le format JSON est un format très portable qui permet de récupérer les données sous forme d'objet.

Cette méthode POST est un exemple de code. Vous pouvez retrouver l'ensemble des méthodes sur le dépôt Github.

Afin de tester le bon fonctionnement de l'API j'utilise l'outil « Postman ». Celui-ci permet de tester l'envoi de requêtes et les réponses du serveur.

Exemple d'une requête GET sur la table « records » :



Erreurs corrigées

Lors des premiers tests, je récupérais des erreurs sans code. J'avais omis la récupération d'une erreur en laissant un `if` sans `else`. J'ai également pu récupérer les erreurs de syntaxe SQL (qui ne sont pas relevées par l'IDE) grâce à des `try catch` et l'objet PDO, dans le fichier CRUD. Par la suite, j'utilise systématiquement les « `try catch` » qui sont renvoyés dans un fichier de logs sur le serveur.

Une fois chaque méthodes testées et fonctionnelles, je peux lancer mon code en test sur serveur.

Sur serveur, il faut implémenter le code produit et penser à modifier les identifiants de connexion à la base. Mis à part ces étapes, le déploiement est relativement similaire à celui en local.

Pour faciliter les mises à jour des évolutions, j'ai lié mon dossier d'API sur serveur à mon repository Github. Cela me permet de travailler sur VS Code et de n'avoir qu'un pull à faire pour récupérer le code côté serveur.

Amélioration du code

Une fois le CRUD de la table « users » implanté et testé sur serveur, j'ai commencé à écrire celui faisant référence à la table « records ». Au fil de l'adaptation des fichiers, deux points problématiques sont relevés. Le fait de devoir recopier le code dans son ensemble est chronophage et peut amener des erreurs. Le deuxième point apparaît au moment du test du code, à l'apparition d'une erreur dans le fichier Users.php. Elle doit également être corrigée dans le fichier Records.php. Cette API possède plusieurs tables, l'optimisation est donc nécessaire.

Pour optimiser le code, je réfléchis au code qu'il est possible de supprimer.

Tout d'abord, il semble important de n'avoir qu'un seul fichier CRUD. Pour ce faire, je dois supprimer toutes les références à des variables définies et les rendre génériques. En ce sens, tous les appels de variables redondantes sont itérés en boucle. Elles sont ensuite appelées depuis le fichier propre de chaque méthode, à chaque table.

Je crée un fichier « tabs » dans lequel seront passés les arguments pour chaque table, sous forme de tableau. Cela me permet de n'avoir qu'un seul fichier à modifier en cas de modifications dans une table de la base de données. Notamment pour l'ajout ou la suppression d'une colonne.

Exemple de la méthode create() dans le fichier CRUD:

```
/**
 * Creating
 *
 * @param array $arguments the columns to insert in the table
 * @param string $sql the sql query to prepare
 * @return boolean return true if the insertion is successfull, false
 otherwise
 */
public function create($arguments, $sql){

    try{
        // Request preparation
        $query = $this->connection->prepare($sql);

        // Protection from injections
        foreach($arguments as $argument){
            $this->$argument=htmlspecialchars(strip_tags($this-
>$argument));
        }

        // Adding protected datas
        foreach($arguments as $argument){
            $query->bindParam(":". $argument, $this->$argument);
        }

        // Request's execution
        $query->execute();
        // If there are no exceptions, return true
        return true;
    }
```

```

    } catch (PDOException $e) {
        // If there is an exception, print exception's message and return
false
        echo $e->getMessage();
        return false;
    }
}

```

Une fois le fichier CRUD modifié, il est nécessaire de modifier les fichiers contenant les méthodes destinées aux appels http.

Pour améliorer encore la maintenabilité du code, ces fichiers sont séparés en deux parties. Une partie commune à toutes les tables est créée, nommée « generic ».

Toujours dans un souci de réduire le code à modifier (pour une meilleure portabilité) le tableau est appelé et une boucle est lancée sur celui-ci, lui passant ainsi les arguments.

De cette façon, les fichiers generic ne changent jamais. S'ils devaient être modifiés, ils ne le seraient qu'une seule fois. Si une erreur est relevée elle sera modifiée à un seul endroit.

Un autre fichier est ensuite créé pour chaque méthode et chaque table avec les informations de la table. On y passe la table visée, le tableau des titres de colonne et sa requête SQL.

Voici un exemple du fichier generic_create.php :

```

try{
    // Verification that used method is correct
    if($_SERVER['REQUEST_METHOD'] != 'POST'){ // Change with good method
        throw new InvalidArgumentException("Invalid request method. Only POST
is allowed", 405);
    }

    // Including files for config and data access
    include_once '../Database.php';
    include_once '../models/CRUD.php';

    // DDB instantiation
    $database = new Database();
    $db = $database->getConnection();

    // Records instantiation
    $crudObject = new CRUD($db);

    // Get input data
    $input = file_get_contents("php://input");
    if (!$input = json_decode($input)) {
        throw new InvalidArgumentException("Invalid input data. Must be valid
JSON", 405);
    }

    foreach($arguments as $argument){
        if(isset($datas->$argument)){
            //here we receive datas, we hydrate our object

```

```

        $crudObject->$argument = $datas->$argument;
    }else{
        // We catch the error
        http_response_code(400);
        echo json_encode(["message" => "Arguments doesn't match"]);
    }
}
if($crudObject->create($arguments, $sql)){
    // Here it worked => code 201
    http_response_code(201);
    echo json_encode(["message" => "The add have been done"]);
}else{
    // Here it didn't worked => code 503
    http_response_code(503);
    echo json_encode(["message" => "The add haven't been done"]);
}
} catch (Exception $e){
    http_response_code($e->getCode());
    echo json_encode(["Message" => $e->getMessage()]);
    error_log($e->getMessage());
}
}

```

Et voici le fichier create.php pour la table users :

```

<?php
    include_once '../tabs/tabs.php';

    $table = "friends"; // Change with the good BDD table name

    // Datas
    $arguments = $tabFriends; // Replace with the good tab

    // SQL request
    $sql = "INSERT INTO " . $table . " SET ". implode(', ',
array_map(function($argument)
    { return $argument . '=: ' . $argument; }, $arguments));

    include_once '../generic_cruds/generic_create.php';

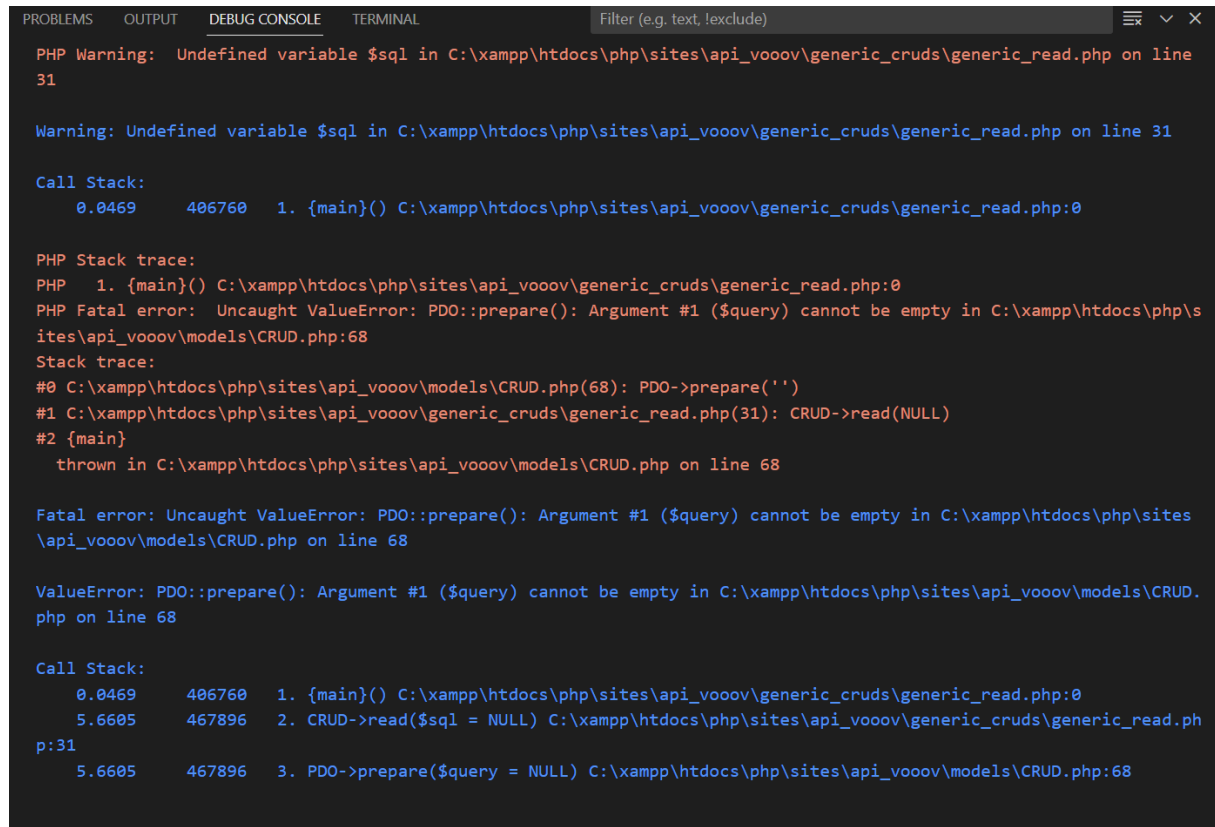
```

Vous pouvez retrouver le code complet de l'API à cette adresse : https://github.com/nathan-kedinger/api_vooov

Tests

Afin de valider le fonctionnement de l'API, j'utilise ici l'outil de test unitaire PHP Unit, via Xdebug, dans l'IDE VSCode.

Voici un exemple de résultat de test unitaire comportant des erreurs :



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Filter (e.g. text, !exclude)

PHP Warning: Undefined variable $sql in C:\xampp\htdocs\php\sites\api_vooov\generic_cruds\generic_read.php on line 31

Warning: Undefined variable $sql in C:\xampp\htdocs\php\sites\api_vooov\generic_cruds\generic_read.php on line 31

Call Stack:
  0.0469    406760    1. {main}() C:\xampp\htdocs\php\sites\api_vooov\generic_cruds\generic_read.php:0

PHP Stack trace:
PHP    1. {main}() C:\xampp\htdocs\php\sites\api_vooov\generic_cruds\generic_read.php:0
PHP Fatal error: Uncaught ValueError: PDO::prepare(): Argument #1 ($query) cannot be empty in C:\xampp\htdocs\php\sites\api_vooov\models\CRUD.php:68
Stack trace:
#0 C:\xampp\htdocs\php\sites\api_vooov\models\CRUD.php(68): PDO->prepare('')
#1 C:\xampp\htdocs\php\sites\api_vooov\generic_cruds\generic_read.php(31): CRUD->read(NULL)
#2 {main}
   thrown in C:\xampp\htdocs\php\sites\api_vooov\models\CRUD.php on line 68

Fatal error: Uncaught ValueError: PDO::prepare(): Argument #1 ($query) cannot be empty in C:\xampp\htdocs\php\sites\api_vooov\models\CRUD.php on line 68

ValueError: PDO::prepare(): Argument #1 ($query) cannot be empty in C:\xampp\htdocs\php\sites\api_vooov\models\CRUD.php on line 68

Call Stack:
  0.0469    406760    1. {main}() C:\xampp\htdocs\php\sites\api_vooov\generic_cruds\generic_read.php:0
  5.6605    467896    2. CRUD->read($sql = NULL) C:\xampp\htdocs\php\sites\api_vooov\generic_cruds\generic_read.php:31
  5.6605    467896    3. PDO->prepare($query = NULL) C:\xampp\htdocs\php\sites\api_vooov\models\CRUD.php:68
```

Les erreurs sont causées car le test était effectué sur un script local donc sans passage de paramètres à la méthode. Lorsque Xdebug tourne et que Postman est exécuté avec les bons paramètres, les erreurs disparaissent.

Les tests unitaires permettent de récupérer des informations essentiels et détaillées en cas de problèmes au lancement du code.

Fin dossier 2

Début dossier 3

En cours de préparation

Conception de l'Application Web

Conception de l'application mobile (Android)

Création d'une application de test du CRUD en échange avec l'API

Liste des bogues rencontrés