HugOvatars

Barbé Loïc - Sallebert--Menaut Hugo - Soubrevilla Basil

Sommaire

[**I- Origine du projet**](#_hkc094junamx) **4**

[**II- Fonctionnalités du projet**](#_jjam7344w57f) **4**

[1°) Fonctionnalités de l’application web](#_pxy1a8sw9tm1) 4

[2°) Fonctionnalités de l’API](#_kja1abqramko) 5

[3°) Fonctionnalité de l’application android](#_vvy83htg56k8) 6

[**III- Maquette**](#_65ddwldmsbiy) **6**

[**IV- Solutions techniques**](#_7ccqldc5n7km) **8**

[**V. Base de données**](#_tgmp7rilq6w4) **9**

[**VI. Organisation du code**](#_l4o7sw46dxeb) **10**

[1°) Les vues](#_slfltxwyp7dt) 10

[2°) Les routes](#_54idcsgd1n2j) 10

[3°) Les controllers](#_j42oli28g5o5) 10

[4°) La gestion des données](#_ye68347prnbw) 11

[5°) Fonctionnement de l’API](#_t6a7i1ms1991) 11

[6°) Les informations utiles](#_a8jq09f416d8) 11

# I- Origine du projet

L'idée de notre projet est de construire une application web permettant de créer son compte et d’enregistrer ses avatars. Cette application web expose une API de manière publique et nous avons développé une application android qui permet de tester cette API.

# II- Fonctionnalités du projet

Le projet repose sur trois parties distinctes, chacune ayant ses propres fonctionnalités.

## 1°) Fonctionnalités de l’application web

Afin de lister les différentes fonctionnalités disponibles au travers de l’application web, nous avons réalisé le diagramme des cas d’utilisations suivant :

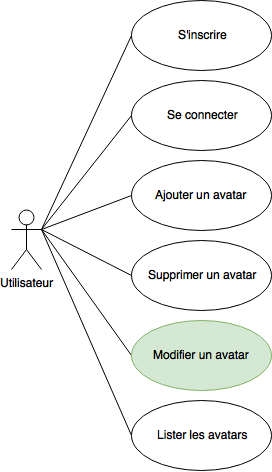


Diagramme des cas d’utilisations de l’application web

L’acteur principal qui utilisera l’application est un utilisateur. Grâce à l’application il pourra :

* **S’inscrire** : En donnant son adresse mail, un nom d’utilisateur et un mot de passe, il pourra créer un compte avec lequel il accédera au site.
* **Se connecter** : En s'identifiant à l’aide de l’adresse mail et du mot de passe saisis lors de l’inscription, l’utilisateur pourra accéder à toutes les fonctionnalités du site.
* **Ajouter un avatar** : En remplissant un formulaire, l’utilisateur pourra lier un avatar à une de ses adresses mail.
* **Supprimer un avatar** : En sélectionnant un/des avatar(s), l’utilisateur pourra choisir de le(s) supprimer ainsi que la/les adresse(s) mail de la base de données du site.
* **Modifier un avatar** : En choisissant une adresse mail, l’utilisateur pourra choisir de changer l’avatar associé à celle-ci. Cette fonctionnalité est d’une couleur différente dans le diagramme, car elle n’était pas demandée dans le cahier des charges initial.
* **Lister les avatars** : En accédant à son profil, l’utilisateur pourra visualiser sous forme d’un tableau toutes les adresses mail associées à son compte ainsi que l’avatar associé à chaque adresse.

## 2°) Fonctionnalités de l’API

Afin de lister les différentes fonctionnalités proposées par l’API, nous avons réalisé le diagramme des cas d’utilisations suivant :

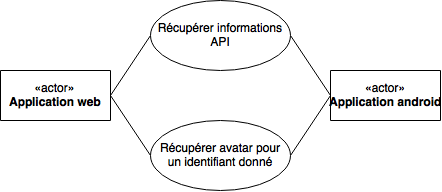


Diagramme des cas d’utilisations de l’API

L’API peut être utilisée par deux acteurs non-humains :

* L’application web que nous avons évoquée plus tôt ;
* L’application android dont nous parlerons dans le prochain point.

Elle permet à ces deux acteurs de réaliser les mêmes actions :

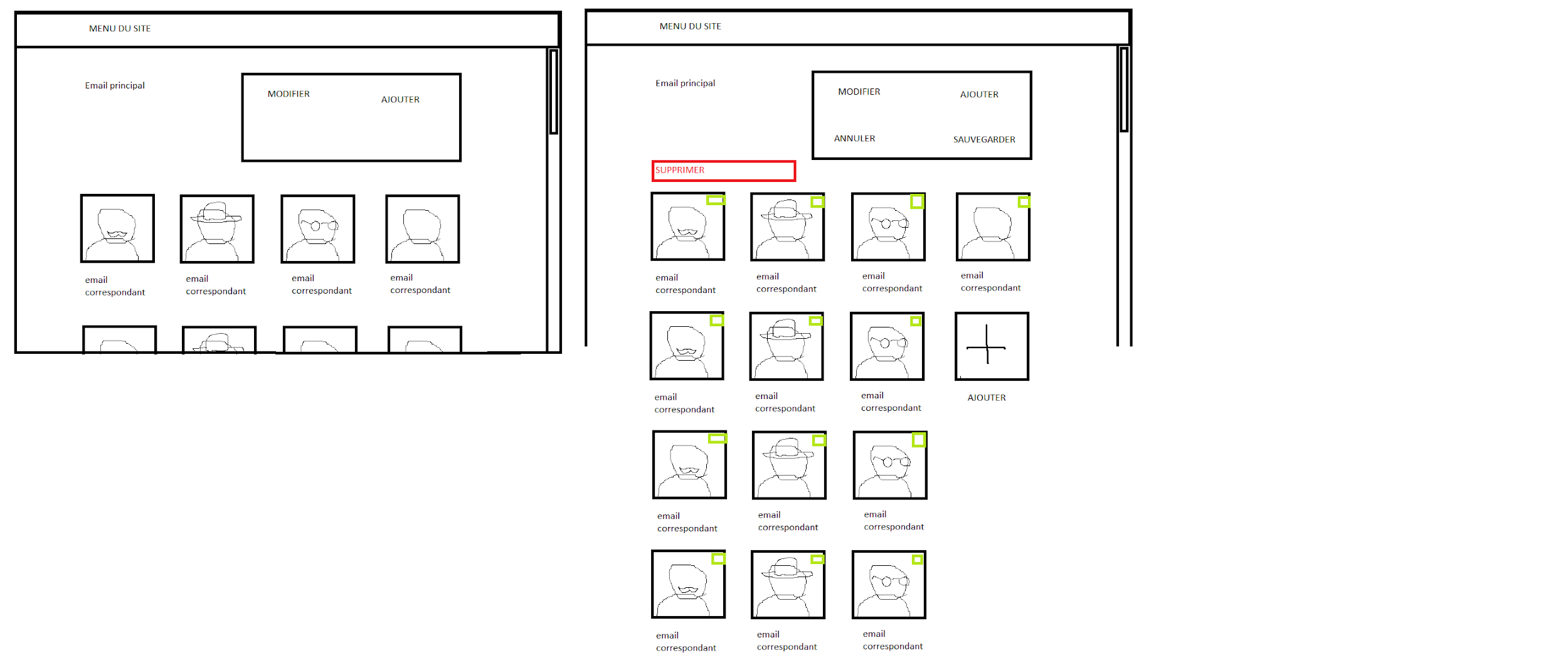
* **Récupérer les informations de l’API** : En appelant cette fonctionnalité de l’API, celle-ci retour la version de l'API, les tailles d'avatar disponibles, la taille d'avatar par défaut et les formats d'image supportés sous forme structurée
* **Récupérer l’avatar pour un identifiant donné** : En appelant cette fonctionnalité de l’API et en lui passant un identifiant en paramètre, on récupère l’image associée à cet identifiant. Si aucune image n’est associée, l’API renvoie une erreur 404.

## 3°) Fonctionnalité de l’application android

L’application android ne propose qu’une seule fonctionnalité : elle permet à l’utilisateur de saisir une adresse mail et l’application fait un appel à l’API pour lui renvoyer l’image associée à cette adresse mail.

# III- Maquette

Afin de visualiser ce que donnera l’application, nous avons réalisé des maquettes (Loïc l’a fait), qui, par la suite, ont été amenées à évoluer, comme le montre la différence entre les maquettes ci-dessous et l’application finale.



Maquette initiale de l’application

# 

# IV- Solutions techniques

Afin de développer le produit demandé, nous avons utilisé plusieurs solutions techniques.

En effet, ce produit a demandé deux types de développement distincts : un développement web et un développement mobile. C’est pour cela que nous avons utilisé 2 IDE (Environnement de Développement Intégré) différents :

* Cloud9 sur lequel était installé Laravel 5.6 ainsi que PhpMyAdmin pour le développement web et celui de l’API. La mise en forme a été réalisée à l’aide du framework Bootstrap 3.3.6. Le versionning de l’application a été réalisé à l’aide l’outil Git;
* Android Studio pour le développement mobile

Utiliser Cloud9 nous a permis de travailler en collaboration sur le code de l’application web et sur l’API. Ce travail a été facilité avec l’utilisation de PhpMyAdmin qui nous permettait une visualisation en direct de la base de données. Enfin, toute la mise en forme et l’aspect graphique du site a été réalisé en utilisant les classes CSS proposées par Bootstrap. Git quant à lui nous a permis de sauvegarder l’application à un instant T, nous permettant en cas de problème un retour en arrière, à une version de l’application opérationnelle.

# 

# V. Base de données

La base de données sur laquelle se repose notre application respecte de diagramme de classes suivant :

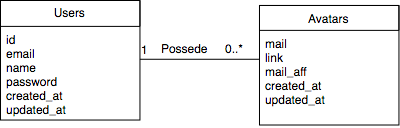


Diagramme de classe

Les données de la table Users représentent :

* **id** : clé primaire de la table, il est généré automatiquement lors de l’inscription.
* **email** : email saisi par l’utilisateur lors de son inscription ;
* **name** : nom saisi par l’utilisateur lors de son inscription ;
* **password** : mot de passe saisi par l’utilisateur lors de son inscription. Il est enregistré sous forme hashée ;
* **created\_at** : date de création de l’enregistrement dans la table. Ce champ est généré automatiquement ;
* **updated\_at** : date de modification de l’enregistrement. Ce champ est généré automatiquement.

A cette classe Users sont associés plusieurs Avatars :

* **mail** : clé primaire de la table. C’est un hash de l’adresse mail à laquelle l’utilisateur veut associer un avatar ;
* **link** : lien vers l’image associée à l’adresse mail ;
* **mail\_aff** : champ mail non hashé ;
* created\_at : date de création de l’enregistrement dans la table. Ce champ est généré automatiquement ;
* **updated\_at** : date de modification de l’enregistrement. Ce champ est généré automatiquement.

Nous utilisons dans la classe Avatars deux fois l’adresse mail. En effet, de cette façon, nous pouvons utiliser le champ “mail” afin d’effectuer des traitements, évitant de cette façon l’utilisation des adresses mails brutes dans les URL. Le champ “mail\_aff” nous permet, une fois que l’enregistrement a été déterminé avec le champ “mail” d’afficher l’adresse mail saisie par l’utilisateur lors de l’ajout.

# VI. Organisation du code

Laravel repose sur un modèle Modèle-Vue-Controller. Nous nous sommes concentrés sur la partie Vue-Controller.

## 1°) Les vues

Les vues sont stockées par défaut dans le répertoire resources/views de l’application. Ces vues utilisent toutes le layout “master” (contenu dans resources/views/layouts) et pour la vue userprofile, le layout “masterProfile” en plus.

Ces vues sont écrites en Blade, permettant d’intégrer certaines fonctionnalités de programmation dans le code HTML, telles que les boucles de répétition ou les structures conditionnelles.

## 2°) Les routes

Toutes les routes sont définies dans le fichier routes.php contenu dans le répertoire app/Http/. Une grande partie des routes concernant la gestion de l’utilisation a été réalisée à l’aide du gestionnaire d’utilisateurs proposé par Laravel 5.6.

Les routes ayant pour but l’ajout et la modification des avatars sont gérées par une ressource.

## 3°) Les controllers

Notre application dispose de trois controllers principaux qui contiennent la partie métier de l’application.

* Le premier controller, AvatarsController, permet de gérer toutes les interactions avec la gestion des avatars (leur création, leur modification, leur suppression…).
* Le second controller, ProfileController, permet d’afficher la page de profil de l’utilisateur.
* Le troisième, HomeController, s’occupe de réaliser les redirections vers les différentes pages.

Notre application utilise d’autres controllers, générés par Laravel pour la gestion des utilisateurs.

## 4°) La gestion des données

La gestion des données repose sur l’utilisation de l’ORM (Object-Relational Mapping) de Laravel, Eloquent. Cette utilisation permet de représenter les données sous forme d’objet. De plus, la création des différentes tables de la base de données a été réalisée avec artisan. On peut retrouver les différentes migrations réalisées dans le répertoire database/migrations.

## 5°) Fonctionnement de l’API

Pour obtenir l’avatar d’une adresse mail il faut utiliser le lien suivant, en remplaçant email par l’email de l’utilisateur hashé :

[bhlprojet-lbarbe.c9users.io/BHLprojet/public/avatars/email](http://bhlprojet-lbarbe.c9users.io/BHLprojet/public/avatars/email)

L’API retournera l’avatar correspondant sous forme d’image.

Pour obtenir les informations concernant la version il faut utiliser le lien suivant :

[bhlprojet-lbarbe.c9users.io/BHLprojet/public/API](http://bhlprojet-lbarbe.c9users.io/BHLprojet/public/API)

L’API retournera les informations sous forme d’un json .

## 6°) Les informations utiles

Les informations de connexion à la base de données sont disponibles dans le fichier “.env”. De cette façon, ces informations sont cachées aux utilisateurs.

Il existe un fichier CSS contenu dans le répertoire public/css dans lequel est stocké le code CSS ajouté à l’application.