



**[jeremyavid.com](http://jeremyavid.com)**

<https://github.com/doctorgoon/e4>

# SOMMAIRE

## I. Présentation Générale (page 3)

1. Contexte
2. Objectifs
3. Solutions

## II. Technologies Utilisées (page 6)

1. Laravel Framework
2. Materialize
3. MYSQL

## III. Développement (page 10)

1. Base De Données
2. Structure du code
  - a) Les modèles
  - b) Les contrôleurs
  - c) Les vues

## IV. Utilisation (page 18)

1. Gestion clients
2. Gestion appels
3. Gestion stock

## V. Annexes (page 23)

# I. Présentation Générale

## Contexte

Mon entreprise, Lamirau Technologies, est une très petite entreprise (TPE) qui revend des appareils de mesure du souffle nommés spiromètres. Elle est contractuellement associée à la multinationale italienne MIR (Medial International Research) qui fabrique ces appareils. En plus de distribuer les produits MIR, Lamirau Technologies a mis en place un réseau de télémédecine qui permet aux patients de mesurer eux même leur souffle avec le spiromètre, puis d'envoyer à leur médecin les résultats depuis leur ordinateur ou smartphone.

Ce secteur est en pleine expansion, cette croissance est notamment dû à la multiplication des maladies et infections respiratoires chez les français. Mais une augmentation de l'activité implique de repenser le fonctionnement et la structure interne de l'entreprise. En effet, il faut optimiser les différents processus de cette dernière pour s'assurer une productivité à la hauteur des demandes des clients.

S'agissant d'une TPE, il n'existe pas d'organisation structurée et procédurale permettant de faire face à une augmentation rapide de la demande. L'entreprise se retrouve donc face à une nécessité de moderniser, c'est à dire d'automatiser et de numériser, la gestion des clients et des stocks.

Jusqu'à présent cette gestion était faite de manière manuelle et analogique. Il n'existe pas de base de données ni pour les clients, ni pour le stock, ce qui engendre des difficultés dans le suivi. Les données des clients sont éparpillées sur plusieurs supports (emails, post-its, cartes...), entraînant de forts risques de perte de ces données. La relation client devient alors plus compliquée et moins efficace.

La gestion du stock est également gérée manuellement ; les commerciaux se rendent dans les locaux réservés aux stocks et « regardent » les articles encore présents. Cette gestion manuelle est possible du fait des quantités relativement faibles qui transitent, mais une augmentation de l'activité va rendre cette technique compliquée et moins efficace.

## Objectifs

Les objectifs sont donc de moderniser les processus de gestion des stocks et de la relation clients. Il s'agit de mettre en place des solutions qui vont augmenter l'efficacité et la productivité des employés dans le suivi des clients, de leurs appels, ainsi que des stocks, avec les articles qui arrivent, et ceux qui sont expédiés.

Une solution est également nécessaire afin de sauvegarder les ressources informationnelles de l'entreprise. Proposer une solution numérique pour la gestion de la relation client doit permettre de s'affranchir des risques que représente le stockage de ces informations sur des supports

volatiles. Les ressources archivées vont permettre une traçabilité des différentes parties prenantes de l'entreprise. En numérisant la gestion des stocks et des clients, l'entreprise va également réduire le risque d'erreur, notamment en ce qui concerne les stocks.

Cette solution applicative doit aussi permettre de centraliser tous ces processus sur une plateforme commune, qui au final simplifiera ces différentes tâches.

## Solution

Pour faire face à l'augmentation de son activité l'entreprise Lamirau va se doter d'un portail web d'administration qui va lui permettre de gérer ses stocks ainsi que sa relation client.

Chaque employé va disposer d'un compte lui permettant de s'identifier sur la plateforme. Il sera possible par la suite d'ajouter, modifier ou supprimer un utilisateur.

### Gestion des appels :

Une fois connecté il pourra ajouter, grâce à un formulaire, un appel téléphonique reçu en y renseignant :

- l'émetteur et ses informations (nom, entreprise, email, téléphones),
- le destinataire (l'employé à qui l'appel est destiné),
- l'objet de l'appel
- le statut de l'appel (non traité, traité, à rappeler, en attente de rappel),
- l'intervention éventuellement effectuée (prise en main, manipulations, recherche d'informations etc) ainsi que sa durée.

L'appel va alors se classer dans un tableau en fonction de son statut. L'employé aura par ailleurs la possibilité de consulter uniquement la liste des appels qui lui sont destinés.

Il sera également possible de relier l'appel à un client présent dans la base de données ou d'en créer un nouveau le cas échéant, mais aussi de modifier ou de supprimer l'appel.

### Gestion des clients

La plateforme disposera d'un annuaire des clients de l'entreprise qui sera rempli au fur et à mesure par les employés via un formulaire demandant :

- son identité : nom, prénom, entreprise et fonction,
- ses coordonnées : téléphones, fax, email, adresse,

Une fois enregistré, il sera possible d'ajouté au client des appels directement depuis son profil.

L'ensemble des clients seront présentés sous forme de liste et pourront être retrouvés individuellement grâce à une barre de recherche.

### Gestion des stocks

Pour gérer les articles, le site comprendra un formulaire permettant l'ajout de nouveaux appareils (nom de l'article, référence et statut (en stock ou expédié)). Une fois l'article ajouté, nous le

retrouverons soit dans la liste des articles en stock, soit des articles expédiés. Comme pour les clients, il sera également possible de rechercher un article, par son nom ou son numéro de série. L'utilisateur pourra modifier le statut de l'article directement depuis la liste, via un bouton.

L'application proposera également à chaque utilisateur un outils de gestion de tâches lui permettant d'ajouter, modifier et supprimer des activités de sa liste de choses à faire dont il pourra renseigner le pourcentage d'avancement.

## Compétences

- Utilisation d'un framework => c'est un outils puissant mais pas forcément évident à prendre en main au début.
- Utilisation d'un serveur web de développement => le logiciel MAMP sur Mac permet de créer facilement un serveur web local de test.
- Utilisation d'un IDE (Environnement de développement) PHP => j'ai utilisé PHPStorm sur Mac pour développer ce site web. Si je commence à en maîtriser les bases, il dispose d'une quantité impressionnante de fonctionnalités.
- Création et manipulation d'une base de données MySQL => base de données administrée via l'outil PHPMyAdmin.
- Mise en ligne d'un site web sur un hébergement mutualité / Déploiement d'un service => pour faciliter la consultation du site j'ai mis en place un hébergement mutualité sur OVH.
- Création d'un site web en PHP avec utilisation de classes et d'objets => utilisant un framework orienté objet, j'ai du créer les classes qui le constituent (objets, contrôleurs etc).
- Création d'une interface graphique pour site web en HTML et CSS => j'ai dû créer une interface graphique cohérente et convivial répondant au mieux aux attentes des utilisateurs.

## Accès

Site web : <http://jeremyavid.com>

Dépôt Git : <https://github.com/doctorgoone/e4>

Identifiant : johnatest@email.com

Mot de passe : password

## II. Technologies Utilisées

### 1. Laravel Framework

#### Un framework

D'après Wikipedia un framework informatique est un "ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel ».

Un Framework est un outil qui regroupe tout un ensemble de fonctionnalités. Il permet de bénéficier d'une architecture d'application de base et offre ainsi aux développeurs un confort de conception grâce à une facilité et à une rapidité de développement, à des conventions précises et à une décomposition de l'application méthodique et logique.

Un framework, c'est tout simplement un ensemble d'outil venant à faciliter le travail du développeur en regroupant en général les fondations d'un logiciel informatique ou d'une application web. C'est le cas par exemple du système d'authentification, de la gestion des erreurs ou encore la gestions des droits (*administrateurs, visiteurs...*).

Utiliser un framework permet de de s'assurer que l'application construite est mieux structurée et mieux sécurisée, grâce à des composants testés, approuvés et utilisés par d'autres développeur. Aussi, l'utilisation d'un framework permet l'instauration de bonnes pratiques de codage en imposant aux développeurs de respecter une convention particulière. Ces conventions sont en fait des règles à respecter. Par exemple, il peut s'agir de convention de nommage des variables, de fonctions ou même de classes. Ces règles permettent aux développeurs de produire un code propre et compréhensible.

#### Une structure

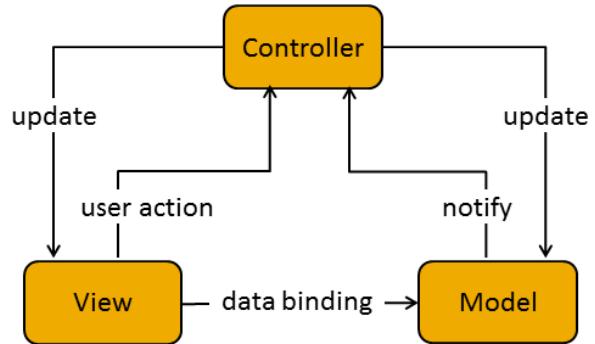
Laravel est un **framework PHP** qui été créé par Taylor Otwell, qui initie une nouvelle façon de concevoir un framework en utilisant ce qui existe de mieux pour chaque fonctionnalité. Il a été, en ce sens, construit en se basant sur Symfony, un autre framework PHP reconnu mondialement pour sa robustesse. C'est un outil supporté par une très large communauté. Il évolue régulièrement et surtout, il est très sécurisé. Son code respecte les standards avancés de la programmation. C'est donc un gage de qualité et de pérennité.

Comme tout framework qui se respecte il est organisé sur la base du patron **MVC** (Modèle-Vue-Contrôleur) , c'est une d'architecture et une méthode de conception qui organise l'interface homme-machine (IHM) d'une application logicielle.

Le principe d'une telle structure est de diviser l'application en 3 parties distinctes :



- Les **modèles** : ils représentent les données de l'application et permettent l'interaction avec la base de données (ou les web services, etc...)
- Les **vues** : ce sont les représentations (les templates) des résultats de la requête que l'utilisateur a effectuée.
- Les **contrôleurs** : ils interceptent toutes les requêtes faites par les utilisateurs.



En bref, le modèle gère la base de données, la vue produit les pages HTML et le contrôleur articule le tout.

Laravel est fondamentalement orienté objet. Avec la Programmation Orienté Objet (**POO**), tout le code est placé dans des classes qui interagissent ensemble. Le fait de programmer avec des classes et de manipuler des objets permet d'éviter la duplication du code et de faciliter sa compréhension. Laravel pousse au maximum cette répartition en utilisant l'injection de dépendance ; elle est destinée à éviter de rendre les classes dépendantes et de privilégier une liaison dynamique plutôt que statique. Le résultat est un code plus lisible.

## Des fonctionnalités

Dans Laravel on retrouve :

- ▶ un système de routage perfectionné :

Il permet de créer des chemins URIs qui sont simples et naturelles pour accéder aux différentes fonctionnalités du site

- ▶ un créateur de requêtes SQL et un **ORM** performants :

Un mapping objet-relationnel (en anglais **ORM**) est une programmation informatique qui définit des correspondances entre une base de données et les objets du langage utilisé.

L'ORM incorporé dans Laravel s'appelle **Eloquent**. Il s'agit de gérer et d'interroger la base de données directement avec Laravel, dans du code PHP. Il s'agit du **QueryBuilder**, un outil pratique de génération de requêtes avec une syntaxe explicite, qui complète les fonctionnalités d'Eloquent.

- ▶ un moteur de template efficace :

**Blade** est simple à utiliser et puissant. Il donne la possibilité d'utiliser des éléments communs à plusieurs pages (header, menu, footer etc.) en agissant sur un fichier uniquement, grâce à un système d'héritage. Cela dans le but de réduire la redondance du code et d'améliorer la visibilité, la compréhension et la maintenabilité du code.

- ▶ un système de validation :

Plus besoin de coder en dur une analyse de conformité des données, qui peut être un processus lourd, rébarbatif, et pas toujours très efficace. Laravel propose des méthodes robustes améliorant et automatisant ce travail.

► **un système de migration pour les bases de données :**

Laravel propose un outil permettant de créer et de mettre à jour un **schéma** de base de données.

► **une gestion des sessions :**

Les **sessions** sont utilisés pour stocker des informations sur l'utilisateur au fur et à mesure de la navigation.

## Documentation

Laravel est un framework très bien documenté et sa communauté est plutôt importante.

- <https://laravel.fr/> => site officiel en français
- <https://laravel.com/> => site officiel en anglais
- <http://laravel.io/> => forum officiel
- <http://laravel-recipes.com> => renseignements et astuces
- <http://laracasts.com/> => tutorials et forum

## 2. Materialize



### Présentation

Materialize est un framework front-end (**CSS**), c'est à dire côté client ; à l'inverse de Laravel qui est un framework back-end, c'est à dire côté serveur. Pour faire simple, le back-end correspond à machinerie derrière un site, et le front-end à la partie interface et cosmétique. Concrètement, c'est un ensemble de fichiers et dossiers livré de façon structurée ; le tout est codé en respectant les standards actuels et fonctionne uniformément sur toutes les plateformes et navigateurs actuels. Ce pack de fichiers fournit un lot de comportements et d'éléments couramment utilisés dans un site web. On y retrouve différents composants habituellement présents sur un site web tel que les cartes, les boutons, les listes, les formulaires, les icônes etc.

Il s'agit d'éléments pré-formatés pour créer une homogénéité visuelle sur le site. Dans le cas de Materialize, il se base sur les codes esthétiques développés par Google avec le **Material Design**. On retrouve ce dernier dans tous les sites Google ainsi que sur bon nombre d'application Android. C'est un design minimaliste, axé sur le flat design, et qui a pour but de créer une interface la plus intuitive possible.

### Documentation

- Site officiel => <http://materializecss.com/>



### 3. MySQL & Apache

#### Présentation

**MySQL** est un système de gestion de bases de données relationnelles (SGBDR). C'est le plus populaire pour les projets de développement web en PHP. Il se base sur un langage de requêtes SQL qui permet de manipuler les bases de données. Le couple PHP/MySQL est très utilisé par les [sites web](#) et proposé par la majorité des hébergeurs Web. Pour administrer MySQL j'ai choisi d'utiliser l'application web [PHPMyAdmin](#). Son interface graphique permet une gestion plus aisée et convivial des bases de données.

Plus de la moitié des sites Web fonctionnent sous **Apache**, qui est le plus souvent utilisé conjointement avec PHP et MySQL. Ce dernier étant un serveur **HTTP** opensource.

Pour configurer les environnement PHP, MySQL et Apache, j'ai utilisé le logiciel **MAMP** sur macOS. Il facilite grandement la gestion des VirtualHosts.

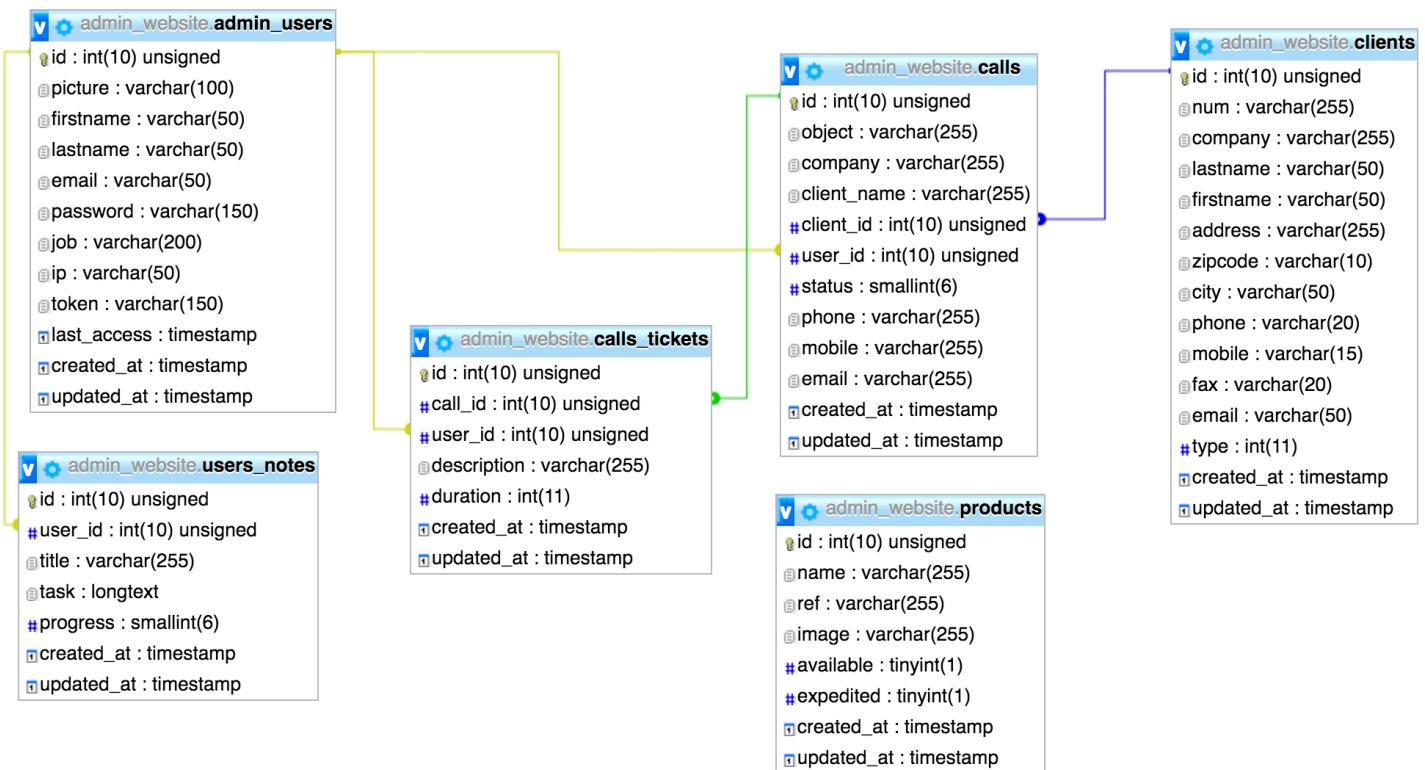
#### Documentation

- Site officiel => <https://www.mamp.info/en/documentation/>

# III. Développement

## 1. La base de données

### Modèle entité-association



### Explications

#### • admin\_users

Correspond à la table des utilisateurs du site. Chacun dispose d'un **password** et d'une adresse **email** qui lui permettront de se connecter au site. Ce dernier enregistrant la dernière adresse IP ainsi que la date de la dernière connexion, on retrouve un champ **ip** et **last\_access**. Le champ **token** est également utilisé pour l'authentification (nous y reviendrons).

Un utilisateur peut être associé à plusieurs entrées dans **users\_notes**, **calls\_tickets** et **calls**.

#### • users\_notes

Table des notes des utilisateurs. Il s'agit des tâches à faire que l'utilisateur ajoute à sa liste. Elle comporte de ce fait la clé étrangère **user\_id** référençant la table **admin\_users**. Le champs **progress** est utilisé pour renseigner l'état d'avancement de la tâche.

#### • clients

Table représentant les clients. On y retrouve toutes les informations de contact, ainsi que un **num**

qui n'est autre qu'une référence client, et un champs `type` qui permet de renseigner le type de client ou son métier / poste (ce champs est VARCHAR et non un INT).

Un client peut être associé à plusieurs entrées dans `calls`.

- **`calls`**

Tables des appels téléphoniques. On y trouve des champs correspondant au informations du client ayant appelé, ils sont prévus pour les cas où le client appelant n'est pas, ou pas encore, présent dans la base de données. Il est cependant possible par la suite de l'associer à un client existant ou de rajouter le client manquant à la base, c'est pourquoi on retrouve un champs `client_id` qui est une clé étrangère faisant référence à la table `clients`. On trouve également un champs `user_id` qui est également une clé étrangère, faisant référence à la table `admin_users` car chaque appel est destiné à un employé.

Un appel peut être associé à plusieurs entrées dans `calls_tickets`.

- **`calls_tickets`**

Table des tickets d'appel. Il s'agit des interventions effectué suite ou pendant un appel téléphonique. Il peut s'agir d'une recherche d'une commande dans l'historique, d'une prise en main TeamViewer pour un dépannage, d'un partage d'informations techniques etc. L'utilisateur doit renseigner la `description` de l'intervention ainsi que le temps que cela lui a pris avec le champs `duration`. On retrouve les clés étrangères `user_id` et `call_id` qui font respectivement référence aux tables `admin_users` et `calls`.

- **`products`**

Table des produits / articles. Elle comporte un champs `ref` correspondant au numéro de série de l'article ; ainsi que des champs `available` et `expedited` qui permettent de définir le statut de l'article, respectivement « disponible » et « expédié ».

## 2. Structure du code PHP

Laravel étant un framework se basant sur la structure Model View Controller, je vais décomposer la présentation du code en 3 parties ; et comme il s'agit également d'un framework orienté objet, je choisis d'expliquer mon code avec l'objet `Calls` qui correspond aux appels reçus par l'entreprise. Les structures de classes et modèles découlant des autres objets étant essentiellement similaires.

### Modèles

Tout d'abord, avant de créer un modèle il faut commencer par créer un fichier de **migration** pour chaque modèle. C'est un fichier que le framework va utiliser pour créer la table dans la base de données SQL correspondante au modèle. Dans ce fichier (`/database/migrations`), on retrouve une classe en rapport avec le modèle (`Create_nomDuModèle_Table`) et qui étend le

classe **Migration** de Laravel qui gère le système de transfère de schéma de la base de données au serveur SQL.

Dans cette classe on retrouve deux fonctions. La fonction **up** qui nous sert à la création de la table et de ses champs, et la fonction **down** qui va permettre de faire l'inverse, c'est à dire supprimer la table. Pour écrire les requêtes SQL nécessaires à la création de la table, nous utilisions la classe **Schema** intégrée à Laravel qui permet de construire ces requêtes en simplifiant leur écriture.

Prenons pour exemple le fichier de migration de la table **calls** :

```
class CreateCallsTable extends Migration
{
    public function up()
    {
        Schema::create('calls', function (Blueprint $table) {
            $table->increments('id');
            $table->string('object')->nullable();
            $table->string('company')->nullable();
            $table->string('client_name');
            $table->integer('client_id')->nullable()->unsigned();
            $table->integer('user_id')->unsigned();
            $table->smallInteger('status');
            $table->string('phone')->nullable();
            $table->string('mobile')->nullable();
            $table->string('email')->nullable();
            $table->timestamps();
            $table->foreign('user_id')->references('id')->on('admin_users')->onDelete('cascade');
            $table->foreign('client_id')->references('id')->on('clients')->onDelete('cascade');
        });
    }

    public function down()
    {
        Schema::drop('calls');
    }
}
```

L'écriture est assez explicite, notamment en ce qui concerne la déclaration de clés étrangères.

A noter également la présence d'un champs **timestamps** ; il s'agit d'une méthode intégrée à Schema qui permet la création en fait de deux champs : **created\_at** et **updated\_at**.

La fonction **up** va exécuter les requêtes SQL ci-dessous :

```
create table `calls` (
    `id` int unsigned not null auto_increment primary key,
    `object` varchar(255) null,
    `company` varchar(255) null,
    `client_name` varchar(255) not null,
    `client_id` int unsigned null,
    `user_id` int unsigned not null,
    `status` smallint not null,
    `phone` varchar(255) null,
    `mobile` varchar(255) null,
    `email` varchar(255) null,
    `created_at` timestamp null,
    `updated_at` timestamp null
)
alter table `calls` add constraint `calls_user_id_foreign` foreign key (`user_id`) references `admin_users` ('id') on delete cascade
alter table `calls` add constraint `calls_client_id_foreign` foreign key (`client_id`) references `clients` ('id') on delete cascade
```

Une fois la migration effectuée et la table en place de la base de données, nous utilisons le modèle pour interagir avec l'objet. Le modèle va permettre d'utiliser, modifier, ajouter ou supprimer des données dans sa table correspondante.

Le fichier du modèle (`/app/`) comprend la classe correspondante à l'objet. On y trouve une variable `$fillable` qui est un tableau rempli avec le nom des champs de la table, et éventuellement des fonctions qui vont permettre de faire des relations avec d'autres modèles. C'est un des principaux atouts d'Eloquent qui permet de manipuler les relations de l'objet (nous verrons cela dans la partie « Vues »).

```
class Calls extends Model
{
    protected $fillable = [
        'id', 'object', 'description', 'company', 'client_name', 'client_id',
        'user_id', 'status', 'phone', 'mobile', 'email'];

    public function tickets() {
        return $this->hasMany('\App\CallsTickets', 'call_id');
    }

    public function client() {
        return $this->belongsTo('\App\Clients', 'client_id');
    }

    public function user() {
        return $this->belongsTo('\App\AdminUsers', 'user_id');
    }
}
```

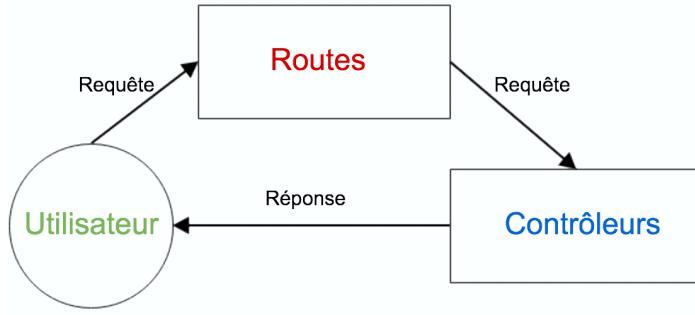
Nous retrouvons ici le modèle `Calls`, avec 3 fonctions permettant de le relier aux modèles `CallsTickets`, `Clients` et `AdminUsers`. Dans ces fonctions nous utilisons les fonctions de relation Eloquent `belongsTo` et `hasMany` :

`belongsTo` : crée une relation 1,1 avec un autre modèle. La lecture est assez simple et claire, dans le cas ci-dessus nous pouvons lire « cet objet appartient à `Clients` avec la clé `client_id` ».

`hasMany` : crée une relation 0,N avec un autre modèle. On peut lire « cet objet appartient à plusieurs `CallsTickets` avec la clé `call_id` ».

## Contrôleurs

La tâche d'un contrôleur est de réceptionner une requête (envoyée via l'URI rentrée dans le navigateur) et de définir la réponse appropriée. Voici une illustration du processus :



Le système de routes est consigné dans le fichier **app/http/routes.php**. Voici un extrait de ce dernier :

```

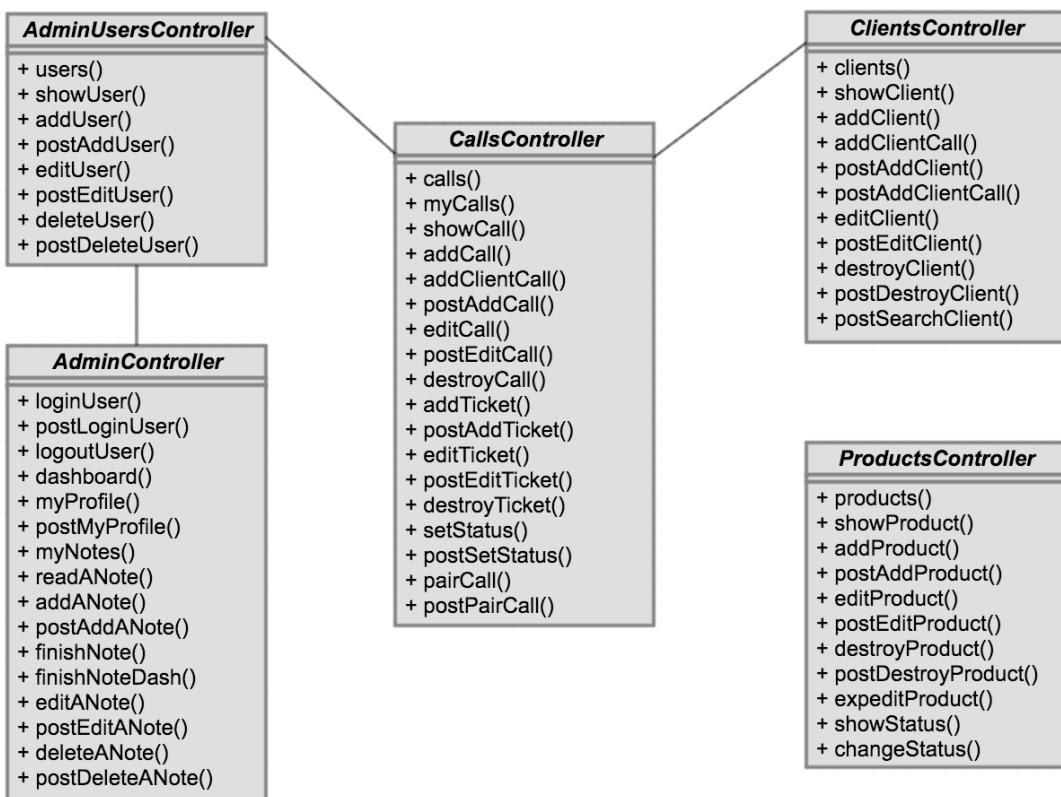
Route::get('/administration/appels', 'CallsController@calls');
Route::get('/administration/mes-appels', 'CallsController@myCalls');
Route::get('/administration/appels/nouveau', 'CallsController@addCall');

```

Le premier argument correspond à l'URI appelée par l'usager du site. Le second argument correspond au contrôleur suivi d'un @ et d'une de ses fonctions.

Dans le cas ci-dessus, l'usager appelant la page *nomDuSite/administration/appels* va faire appel à la fonction **calls()** dans le contrôleur **CallsController** qui permet de faire une requête SQL pour récupérer tous les appels et d'afficher une vue les présentant.

Voici un diagramme récapitulant tous les contrôleurs et les fonctions qu'ils contiennent (**app/http/controllers**) :



Les contrôleurs sont articulés autour des principaux objets : *Calls, Clients, Users, Products*.

Les contrôleurs sont organisés en général autour d'un ensemble de fonctions de bases qui permettent de gérer les objets et les vues qui en dépendent :

- ▶ `calls()` -> afficher l'ensemble des appels
  - ▶ `showCall()` -> afficher un appel en particulier
  - ▶ `addCall()` -> afficher un formulaire permettant de rajouter un appel
  - ▶ `postAddCall()` -> ajouter un appel à la base de données à partir des données récupérées grâce à une requête POST depuis le formulaire
  - ▶ `editCall()` -> afficher un formulaire permettant de modifier un appel
  - ▶ `postEditCall()` -> modifier un appel, présent dans la base de données, à partir des données récupérées grâce à une requête POST depuis le formulaire
  - ▶ `destroyCall()` -> afficher une page demandant la confirmation de suppression d'un appel
  - ▶ `postDestroyCall()`-> supprimer un appel de la base de données
- 

Prenons l'exemple de cette route :

```
Route::get('/administration/utilisateurs/liste-des-utilisateurs', 'AdminUsersController@users');
```

Elle fait donc appel à la fonction `users()` du contrôleur `AdminUsersController` :

```
public function users()
{
    $users = AdminUsers::all();

    // $users = DB::select(DB::raw("SELECT * FROM admin_users"));

    return view('users.users', compact('users'));
}
```

Cette fonction fait une requête SQL pour récupérer l'ensemble des utilisateurs depuis la table `admin_users`. Cette requête retourne une collection d'objets Eloquent que l'on place dans une variable. Enfin, la vue 'users' contenu dans le dossier 'users' est retournée, avec la collection `users` grâce à la fonction `compact()` qui permet de transférer des données vers les vue.

Outre retourner une vue, une fonction de contrôleur peut également envoyer vers une autre fonction du même contrôleur, ou d'un autre :

```
public function postAddCall(Request $request)
{
    /* ... */

    return redirect(action('CallsController@showCall', [ $call->id ]));
}
```

-> Une fois l'appel ajouté à la base de données, on fait appel à la fonction permettant d'afficher l'appel fraîchement ajouté. Grâce à la fonction `redirect()` qui va modifier l'URI actuel et la fonction `action()` qui permet d'appeler une fonction dans un contrôleur avec la même syntaxe que dans le fichier des routes.

Pour revenir enfin sur les routes, il en existe une qui englobe toutes les autres. Elle permet de définir une action à réaliser avant d'exécuter celle associée à l'URI, cette action est appelée un middleware. Il intervient en quelque sorte entre l'appel de la page et son affichage.

```
Route::group(['middleware' => ['admin']], function() {  
  
    // TABLEAU DE BORD  
    Route::get('/administration/tableau-de-bord', 'AdminController@dashboard');
```

On voit ici que la route correspondant à l'affichage du tableau de bord se trouve à l'intérieur d'une route globale faisant appel au middleware « `admin` ».

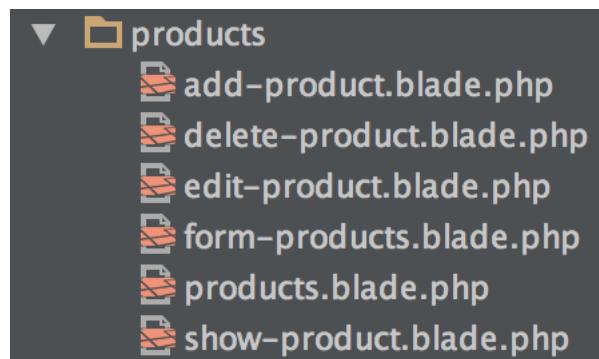
Ce middleware fait référence à la classe `CheckUserAccessToAdmin` (`app/http/middleware`) dont la fonction est de vérifier si l'usager est bien connecté à son compte avant d'afficher la page demandée, et ce pour chaque requête.

Au moment de la connexion de l'usager sur le site, son identifiant (email) ainsi qu'un token généré aléatoirement sont conservé dans une Session PHP, puis supprimer de cette Session à la déconnexion. La fonction `handle()` de `CheckUserAccessToAdmin` va alors vérifier si un token et une adresse email sont bien présents dans la Session avant de rediriger vers la requête suivante (route), dans le cas contraire, elle redirige l'usager vers la vue de connexion.

## Vues

Les vues contiennent le code HTML affichant les différentes pages du site web. On les retrouve dans le dossier **ressources** du projet Laravel.

Elles sont pour la plus part articulées autour des modèles Eloquent ; on retrouve de ce fait une vue pour quasiment chaque fonction du contrôleur lié au modèle en question. Laravel se base sur son système de template (gabarit) nommé **Blade**. C'est un moyen simple et puissant pour créer des vues et en inclure d'autres à l'intérieur. Il permet également d'intégrer plus facilement du code PHP dans les vues.



Puisque la plupart des applications Web maintiennent la même disposition générale sur différentes pages, il est utile de définir cette mise en page sous la forme d'une seule vue Blade.

Ce qui permet de s'affranchir d'une réécriture fastidieuse de la structure HTML de bases dans chacune de nos vues.

Pour ce faire on utilise la balise `@yield('nom_section')` dans le template de base contenant les `stylesheets (CSS)`. Elle permet de définir une section, dont on fera appel dans une vue spécifique avec la balise `@section('nom_section')`. Il suffit pour cela de placer au début de cette vue la balise `@extends(nom_template_de_base)`.

Exemple :

```
<html lang="fr">
<head>
    <title>@yield('title')</title>

    <!-- BEGIN META -->
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="keywords" content="your,keywords">
    <meta name="description" content="Short explanation about this website">

    <!-- BEGIN STYLESHEETS -->
    <link href='http://fonts.googleapis.com/css?family=Roboto:300italic,400italic,300,400,500,700,900' rel='stylesheet' type='text/css' />
    <link type="text/css" rel="stylesheet" href="/assets/css/theme-2/bootstrap.css" />
    <link type="text/css" rel="stylesheet" href="/assets/css/theme-2/materialadmin.css" />
```

Extrait du template de base du site => `layout-admin.blade.php`

```
@extends('templates.layout-admin')

@section('title') Produits @endsection

@section('content')

    <section>
        <div class="section-header">
            <ol class="breadcrumb">
                <li class="active"><h2>Produits</h2></li>
```

Extrait de la vue affichant l'ensemble des produits => `products.blade.php`

## IV. Utilisation

**Portail d'administration**

Identifiant  
goon.jeremy@gmail.com

Mot de passe  
\*\*\*\*\*

**CONNEXION**

*/administration* => page connexion

*/administration/tableau-de-bord* => tableau de bord récapitulant les appels, le nombre

Administration

Tableau de bord

2 / 10 Appels non traités

10 Clients

4 / 2 En stock / Expédiés

A faire

- Appeler Docteur Machin  
Prendre en main pour installation nouveau logiciel + présentation fonctionnalités
- Modifications application Java  
- rendre les boutons disabled si non utilisables - splashscreen démarrage
- Traduction fiche technique produit  
Traduire la documentation d'un

de clients, de produits, ainsi qu'un widget pour gérer la liste de tâches de l'utilisateur.

*/administration/appels => récapitulatif des appels reçus, identifiés par un code couleur permettant d'identifier leur statut plus facilement. L'onglet « Mes appels » permet d'afficher la liste des appels qui concernent l'utilisateur.*

*/administration/appels/{idAppel} => affichage d'un appel et des informations relatives à ce dernier. Possibilité de rajouter un (ou plusieurs) tickets, c'est à dire la description d'une intervention. Possibilité également de relier l'appel à un client présent dans base de données.*

*/administration/appels/nouveau => formulaire pour ajouter un nouveau appel. Nécessité d'entrer au moins le nom du client, un objet, et de définir le statut de l'appel.*

| Contacts  |  |   |  |
|---|--|---|--|
|   |  |   |  |
| 10 client(s)  |  |   | <a href="#">SORT</a>   |
|  | <b>Jonathan Brown</b> Motown<br>0678265779 <a href="mailto:jonathan.brown@motown.com">jonathan.brown@motown.com</a>        |  | <b>Alexandra Capucin</b><br>07 45.66.00.23 <a href="mailto:a.capucin@yahoo.fr">a.capucin@yahoo.fr</a>            |
|  | <b>Michel DUPONT</b> Dupont et fils<br>06.24.42.66.34 <a href="mailto:michel.dupont@gmail.com">michel.dupont@gmail.com</a> |  | <b>Léonard DURAND</b><br>06.87.72.44.99 <a href="mailto:leo.durand@gmail.com">leo.durand@gmail.com</a>           |
|  | <b>Magalie GARNIER</b><br>06.73.89.00.23 <a href="mailto:garnier.m@gmail.com">garnier.m@gmail.com</a>                      |  | <b>Isabelle LEROY</b><br>07.88.70.87.32 <a href="mailto:leroy.isabelle@yahoo.com">leroy.isabelle@yahoo.com</a>   |
|  | <b>Nathalie Lefebre</b> Médic' et vous<br>07.89.00.11.43 <a href="mailto:nathalie.lfb@yahoo.fr">nathalie.lfb@yahoo.fr</a>  |  | <b>Véronique MARTIN</b><br>07.32.11.58.90 <a href="mailto:veroniquemartin@yahoo.fr">veroniquemartin@yahoo.fr</a> |
|  | <b>Patrick MERCIER</b><br>06.32.11.66.44 <a href="mailto:p.mercier@outlook.com">p.mercier@outlook.com</a>                  |  | <b>Bruno Sanchez</b><br>06.34.55.21.10 <a href="mailto:bruno.s@yahoo.fr">bruno.s@yahoo.fr</a>                    |

**/administration/clients =>** affichage de l'ensemble des clients présents dans la base de données.  
Possibilité d'effectuer une recherche par nom, prénom ou entreprise.

**Fiche Client**

|  |                             |
|--|-----------------------------|
| Prénom<br><b>Alexandra</b>                   | Nom<br><b>Capucin</b>       |
| Entreprise                                   | Fonction                    |
| <a href="#">INFOS</a> <a href="#">APPELS</a> |                             |
| Mobile<br><b>07.45.66.00.23</b>              | Téléphone                   |
| Email<br><b>a.capucin@yahoo.fr</b>           | Fax                         |
| Adresse<br><b>10 rue des abeilles</b>        |                             |
| Ville<br><b>Agnognac</b>                     | Code Postal<br><b>24460</b> |
| <a href="#">RETOUR</a>                       |                             |

**/administration/clients/{id\_client} =>** affichage d'un client et de ses informations. On retrouve également un onglet « appels » permettant d'afficher une liste des appels reliés à ce client.

**Ajouter un Client**

|  |             |
|--|-------------|
| Prénom   | Nom         |
| Entreprise   | Function    |
| REFERENCE DU CLIENT                                      |             |
| <a href="#">CONTACT</a>                                  |             |
| Mobile   | Téléphone   |
| Email  | Fax         |
| Adresse  |             |
| Ville  | Code Postal |
| <a href="#">RETOUR</a> <a href="#">AJOUTER LE CLIENT</a> |             |

**/administration/clients/nouveau =>**  
affichage du formulaire d'ajout de client.  
Nécessité de rentrer au moins nom,  
prénom, et téléphone ou email.

## Produits

DISPONIBLES / 4 EXPÉDIÉS / 2

|  |            |   |
|--|------------|---|
|  Spirotel<br>#A23-X.03509   | Expédier → | Supprimer  |
|  Spirodoc<br>#A23-0W.07299  | Expédier → | Supprimer  |
|  Spirobank<br>#A23-0Y.04921 | Expédier → | Supprimer  |
|  Spirodoc<br>#A23-0W.07055  | Expédier → | Supprimer  |

## Produits

DISPONIBLES / 4 EXPÉDIÉS / 2

|   |           |   |
|---|-----------|---|
|  Spirotel<br>#A23-X.03960    | Stocker ↓ | Supprimer    |
|  Spirodoc<br>#A23-0W.07065 | Stocker ↓ | Supprimer  |

*/administration/produits => liste des produits, triés selon leur statut (disponible ou expédiés). Possibilité de changer de statut (ou supprimer) directement dans la liste à l'aide du bouton de droite.*

### Ajouter un nouveau produit

Nom

Référence

URL de l'image

Upload de l'image  
 Aucun fichier choisi

Disponible  Expédié

*/administration/produits/nouveau => formulaire permettant de rajouter un nouveau produit. Nécessité de remplir le champs référence et de choisir un statut.*

| Prénom   | Nom       | Adresse e-mail           | Dernière IP  | Dernière connexion  | Créé le             | Modifier | Supprimer |
|----------|-----------|--------------------------|--------------|---------------------|---------------------|----------|-----------|
| Jeremy   | AVID      | goon.jeremy@gmail.com    | 192.168.1.97 | 2017-04-12 20:45:27 | 2017-04-04 00:00:00 |          |           |
| Laetitia | IACOBOZZI | laetitia.festa@gmail.com | 192.168.1.79 | 2017-04-08 20:32:06 | 2017-04-08 18:49:07 |          |           |

[AJOUTER UN UTILISATEUR](#)

*/administration/utilisateurs/liste-des-utilisateurs => gestion des utilisateurs, possibilité de modifier, ajouter ou supprimer des utilisateurs. Pour l'ajout, il est demandé un nom, prénom et adresse email. Un mot de passe provisoire est alors généré.*

### Mes informations

Prénom  
Jeremy

Nom  
AVID

Email  
goon.jeremy@gmail.com

Dernière connexion  
2017-04-12 21:56:46

Dernière IP connue  
192.168.1.97

### Mon mot de passe

Remplissez ce champ que si vous souhaitez changer votre mot de passe

Mot de passe actuel

---

Nouveau mot de passe

---

Confirmer

---

[VALIDER](#)

*/administration/mon-profil => l'utilisateur peut y voir ses informations et changer son mot de passe.*

### Mes notes

| %     |  |  |
|-------|--|--|
| 0 %   | 12/04/2017 21:01 - Appeler Docteur Machin<br>Prendre en main pour installation nouveau logiciel + présentation fonctionnalités |  |
| 50 %  | 12/04/2017 21:01 - Modifications application Java<br>- rendre les boutons disabled si non utilisables - splashscreen démarrage |  |
| 100 % | 12/04/2017 21:03 - Traduction fiche technique produit<br>Traduire la documentation d'un nouveau produit en anglais             |  |

*/administration/mes-notes => notes de l'utilisateur. Possibilité marqué une note comme terminée (100%) à l'aide du bouton situé à droite.*

## V. Annexes

### Gestion du projet

| Tâches              | Temps |       |       |       |       |       |       |       |
|---------------------|-------|-------|-------|-------|-------|-------|-------|-------|
|                     | 01/01 | 15/01 | 01/02 | 15/02 | 01/03 | 15/03 | 01/04 | 15/04 |
| Structure PHP       |       |       |       |       |       |       |       |       |
| Interface graphique |       |       |       |       |       |       |       |       |
| Base de données     |       |       |       |       |       |       |       |       |
| Rédaction dossier   |       |       |       |       |       |       |       |       |

### Modèle Conceptuel de Données

