UNIVERSITÉ MOHAMMED V de Rabat

Faculté des Sciences

Département d'Informatique

Filière Licence fondamentale
en Sciences Mathématiques et Informatique

PROJET DE FIN D'ÉTUDES

intitulé :

Conception et réalisation d'un
Système d'information des étudiants du
département informatique

Gestion des inscriptions aux projets tutorés

Présenté par : Ilyas Chaoua , Mohammed Amine Achalhi

soutenu le 4 Juin 2015 devant le Jury

M. El Mamoun SOUIDI	Professeur à la Faculté des Sciences - Rabat	*Président*
M. Oussama Mohamed REDA	Professeur à la Faculté des Sciences - Rabat	*Encadrant*
M. El Amrani Younès	Professeur à la Faculté des Sciences - Rabat	*Examinateur*

Année universitaire 2014-2015

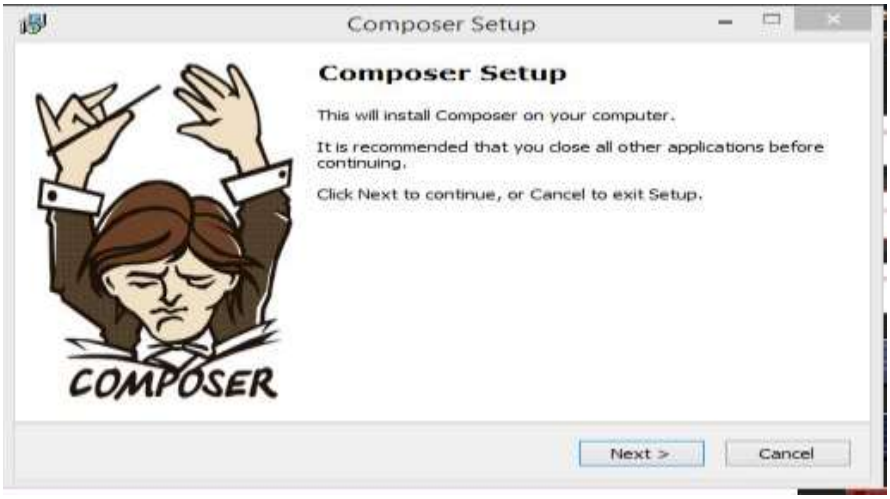
Dossier d’implémentation du code source

Table des matières

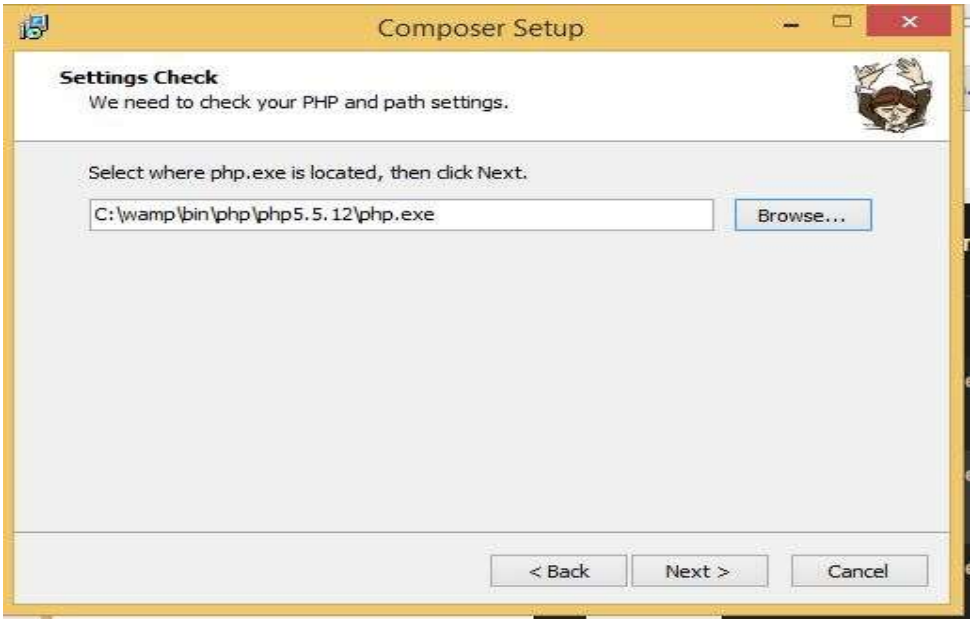
Structure du projet

Installation et structures de fichiers :

Tout d'abord avant de commencer l'installation de Laravel il faut commencer par installer composer qui est un gestionnaire de dépendance qui permet d'en le téléchargeant à partir de leurs sites officiels www.getcomposer.org

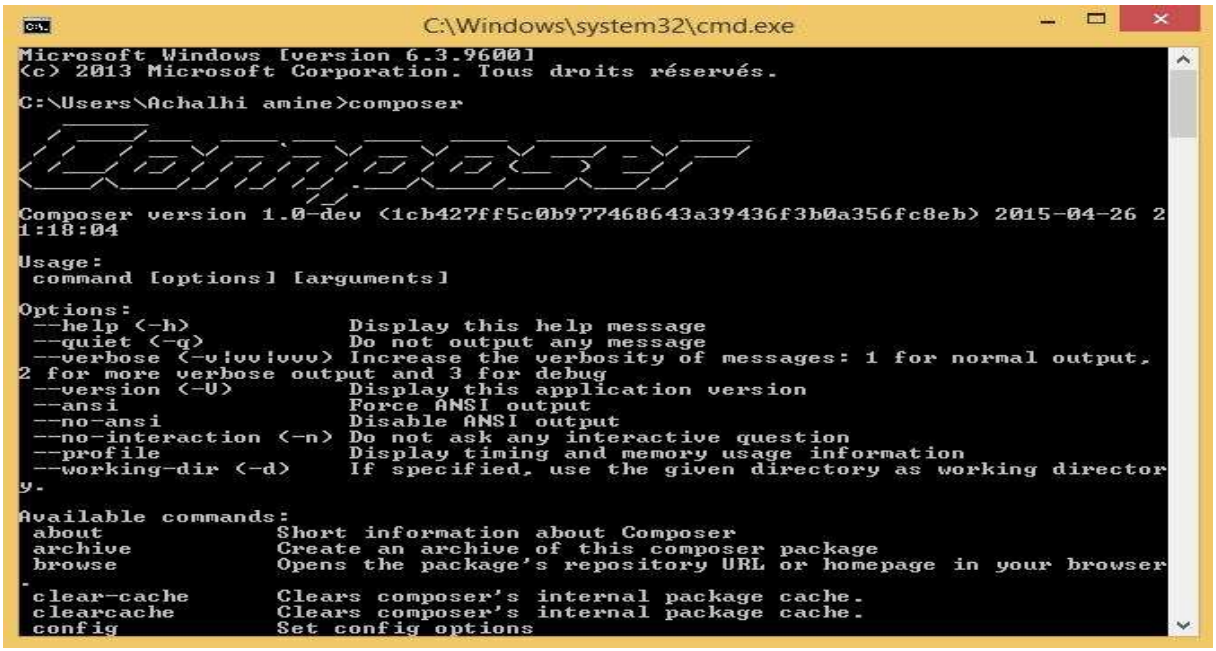


Ensuite il faut cliquer sur Next et indiquer le chemin où se trouve le fichier php.exe :



Et terminer les étapes de l'installation en appuyant sur le bouton Next et install

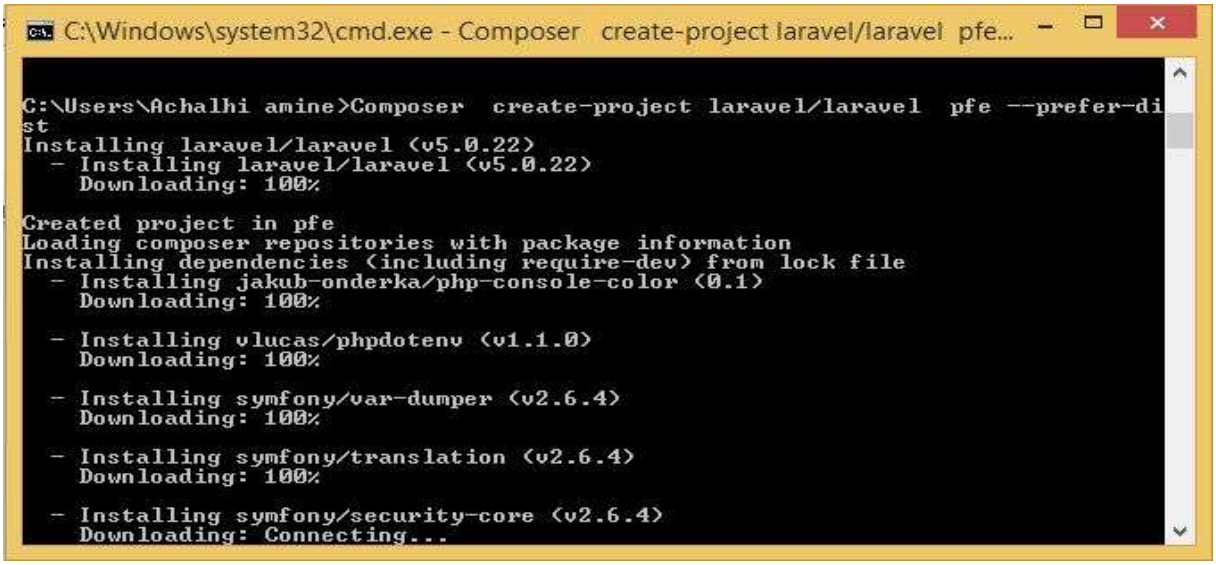
Pour s'assurer que Composer a bien été installé il suffit d'entrer dans l'invité de commande et taper la commande **composer** :



Une fois composer installer on peut démarrer l'installation du Framework Laravel :

Pour installer laravel il faut taper la commande :

```
composer create-project laravel/laravel laravel --prefer-dist
```



On peux vérifier que tout fonctionne bien avec l'URL http://localhost/tutov5/public.

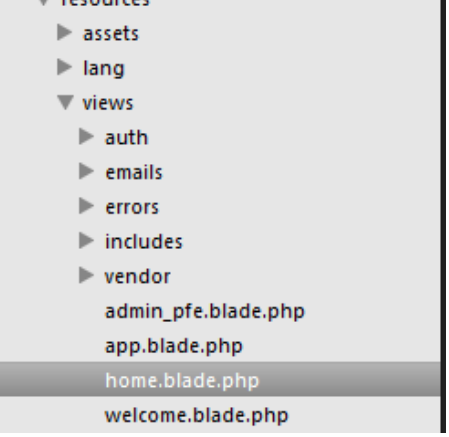
Après que l'installation soit achevé on aura comme résultat une architecture comme celle-ci :

Le dossier App : Ce dossier contient les éléments essentiels de l'application

- **Commands :** ici on mettra les commandes concernant les tâches que doit effectuer l'application. C'est une nouveauté de la version 5.0

- **Console/Commands** : ici on mettra toutes les commandes en mode console, il y a au départ une commande `InspireCommand` qui sert d'exemple,

- **Events :** ici on va avoir les événements nécessaires pour l'application,



- **Handlers** : ici on récupère les commandes et événements,

- **Http** : ici on va trouver tout ce qui concerne la communication : contrôleurs, routes, middlewares (il y a 2 middlewares de base) et requêtes,

- **Providers** : ici on va mettre tous les providers, il y en a déjà 5 au départ. Les providers servent à initialiser les composants,

- **Services** : ici on va placer les services nécessaires à l'application, il en existe déjà un au départ (`Registrar`) destiné en l'enregistrement des nouveaux utilisateurs.

Il existe d'autres dossiers :

Voici une description du contenu des autres dossiers :

- **bootstrap** : scripts d'initialisation de Laravel pour le chargement automatique des classes, la fixation de l'environnement et des chemins, et pour le démarrage de l'application,

- **public** : tout ce qui doit apparaître dans le dossier public du site : images, CSS, scripts...

- **vendor** : tous les composants de Laravel et de ses dépendances,

- **config** : toutes les configurations : application, authentification, cache, base de données, espaces de noms, emails, systèmes de fichier, session...

- **database** : ici on aura les migrations et les populations,

resources : ici on aura les vues, les fichiers de langage et les assets (par exemple les fichiers LESS), concernant les vues nours avons ajoutés des dossiers dans lesquels on mets des vues pour mieux structurés notre application le dossier auth contient les vues concernant l'authentification , le dossier emails pour la vues de la verification des mails , le dossier errors pour la vue des erreurs, le dossier includes contient la majorité des vues de l'application, les autres fichier sont les fichiers principaux, que tous les autres fichier héritent d'eux .

- **storage** : pour stocker les données temporaires de l'application : vues compilées, cache, clés de session...

- **tests** : pour les fichiers de tests unitaires.

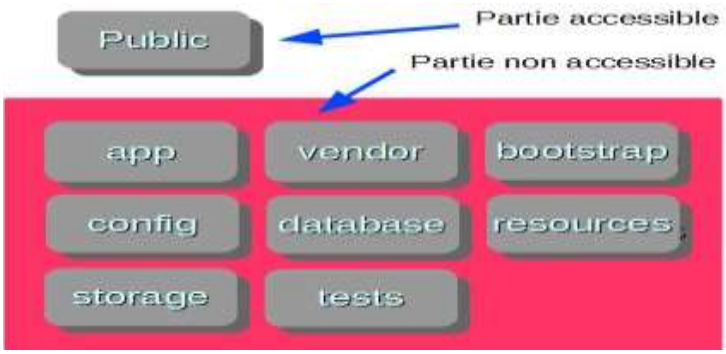
Fichiers de la racine

Il y a un certain nombre de fichiers dans la racine dont voici les principaux :

- **artisan** : outil en ligne de Laravel pour des tâches de gestion,
- **composer.json** : fichier de référence de Composer,
- **phpunit.xml** : fichier de configuration de phpunit (pour les tests unitaires),
- **.env.example** : fichier d'exemple pour spécifier l'environnement d'excécution.

Accessibilité

Pour des raisons de sécurité sur le serveur seul le dossier `public` doit être accessible :



.

Technologie utilisée

Package « DOMPDF Wrapper » :

C'est un outil qui permet de générer des pages pdf :

- Il faut d'abord ajouter ce packages dans **Composer .jason** et mettre à jour composer. Ceci va permettre d'installer les librairies et le **DOMPDF :**

```
"barryvdh/laravel-dompdf": "0.6.*"
```

- la maj de Composer ce fait avec la commande **« composer update »**
- après la maj de composer il faut ajouter le ServiceProvider dans le tableau providers du dossier « config/app.php »

```
'Barryvdh\DomPDF\ServiceProvider',
```

- On peut créer une instance du **DOMPDF** et chargé la vue et l'afficher dans le navigateur

```
$pdf = App::make('dompdf.wrapper');
$pdf->loadHTML('la vue ');
return $pdf->stream();
```

Package « HTML-INPUT » :

Dans la version 4 de Laravel il y avait directement le composant `Html` qui permet de créer facilement des formulaires et qui offre un lot d'helpers pour l'écriture du HTML. Dans la version 5 ce composant n'est pas chargé par défaut. Comme nous en aurons besoin dans notre application, on doit aller modifier ainsi le fichier `composer.json`

```
"require": {
    "php": ">=5.5.9",
    "laravel/framework": "5.1.*",
    "laravelcollective/html": "5.1.*"
},
```

On demande ainsi à Composer de charger le composant laravelcollective/html. Et on Lance alors une mise à jour de composer. Il faut ensuite modifier ainsi le fichier `config/app.php` en ajoutant cette ligne dans le tableau des providers :

```
 <?php
    Collective\Html\HtmlServiceProvider::class,
```

Il faut aussi ajouter ces deux lignes dans le tableau des alias :

```
<?php
'Form'  => Collective\Html\FormFacade::class,
'Html'  => Collective\Html\HtmlFacade::class,
```

Implémentation de la base de données

Une migration permet de créer et de mettre à jour un schéma de base de données. Autrement dit elle permet de créer des tables, des colonnes dans ces tables, en supprimer, créer des index... Tout ce qui concerne la maintenance des tables peut être prise en charge par cet outil.

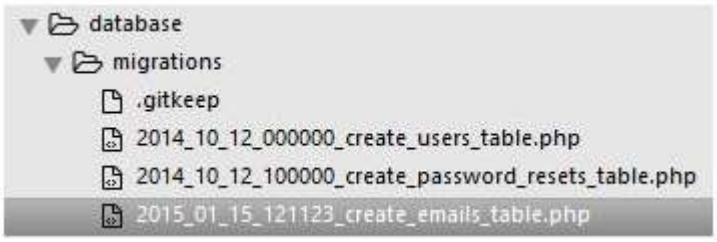
Il faut indiquer où se trouve la base, son nom, le nom de l'utilisateur, le mot de passe dans le fichier de configuration `.env` :

```
DB_HOST=localhost
DB_DATABASE=sie_data
DB_USERNAME=root
DB_PASSWORD=
```

La deuxième étape consiste à créer la migration pour notre table :

```
php artisan make:migration create_emails_tabl .
```

Si vous regardez maintenant dans le dossier `database/migrations` vous trouvez un fichier du genre `2015_07_15_121123_create_emails_table.php` (la partie numérique représente la date)



Voici le contenu de la migration que nous venons de créer :

```php
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateEmailsTable extends Migration {

    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        //
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        //
    }

}
```

On dispose dans cette classe de deux fonctions :

- **up** : ici on mettra le code de création
- **down** : ici on mettra le code de suppression

Supposons qu'on veut créer une table "emails" avec un id auto-incrémenté et un champ "email" de type texte, et de longueur 100. Voilà le code correspondant :

```php
<?php
public function up()
{
    Schema::create('emails', function(Blueprint $table) {
        $table->increments('id');
        $table->string('email', 100);
    });
}
```

On demande au constructeur de schéma (`Schema`) de créer (`create`) la table "emails". Dans la fonction anonyme on définit ce qu'on veut pour la table :

- une colonne "id" auto-incrémentée qui sera ainsi la clé primaire de la table,
- une colonne "email" de type string et de longueur 100.

Pour la méthode `down` on va juste supprimer la table avec un `drop` :

```php
<?php
public function down()
{
    Schema::drop('emails');
}
```

On va maintenant lancer la migration (utilisation de la méthode `up` de la migration) :

```
php artisan migrate
```

Si on regarde maintenant dans la base on trouve la table "emails" avec ces deux colonnes :

	#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
☐	1	id	int(10)		UNSIGNED	Non	*Aucune*	AUTO_INCREMENT
☐	2	email	varchar(100)	utf8_unicode_ci		Non	*Aucune*	

Création de la maquette

AdminLTE est un template de back-end qui a été créé avec bootstrap 3. Le template est très complet et contient des fonctionnalités divers et variés (Widgets, formulaire, calendrier, charts, etc…).

Pour pouvoir l'integrer il faut :

- Télécharger la template AdminLTE.
- Copier les dossiers bootstrap, dist et plugins de la template dans le dossier public de laravel 5.
- Modifier la vue de base : resources\views\home.blade.php(toutes les vue du projet vont hériter de cette vue, elle contient les éléments utilisés dans la plus part des vues : inclusion du CSS et JavaScript, header et sidebar) en ajoutant les liens vers les fichiers CSS et JavaScript du template .La

gestion du CSS et JavaScript se fait par des assets(Asset fournit une manière simple de gérer le CSS et le Javascript utilisés par l'application) comme nous allons voir dans l'exemple ci-dessous :

```
<meta content='width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no' name='viewport'>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
<script src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js"></script>

<!-- Bootstrap 3.3.2 -->
<link href="{{ asset('/bootstrap/css/bootstrap.min.css') }}" rel="stylesheet" type="text/css" />
<!-- FontAwesome 4.3.0 -->
<link href="https://maxcdn.bootstrapcdn.com/font-awesome/4.3.0/css/font-awesome.min.css" rel="stylesheet" type="text/css" />
<!-- Ionicons 2.0.0 -->
<link href="http://code.ionicframework.com/ionicons/2.0.0/css/ionicons.min.css" rel="stylesheet" type="text/css" />
<!-- Theme style -->
<link href="{{ asset('/dist/css/AdminLTE.min.css') }}" rel="stylesheet" type="text/css" />
<!-- AdminLTE Skins. Choose a skin from the css/skins
     folder instead of downloading all of them to reduce the load. -->
<link href="{{ asset('/dist/css/skins/_all-skins.min.css') }}" rel="stylesheet" type="text/css" />
<!-- iCheck -->
<link href="{{ asset('/plugins/iCheck/flat/blue.css') }}" rel="stylesheet" type="text/css" />
<!-- Morris chart -->
<link href="{{ asset('/plugins/morris/morris.css') }}" rel="stylesheet" type="text/css" />
<!-- jvectormap -->
<link href="{{ asset('/plugins/jvectormap/jquery-jvectormap-1.2.2.css') }}" rel="stylesheet" type="text/css" />
<!-- Date Picker -->
<link href="{{ asset('/plugins/datepicker/datepicker3.css') }}" rel="stylesheet" type="text/css" />
<!-- Daterange picker -->
<link href="{{ asset('/plugins/daterangepicker/daterangepicker-bs3.css') }}" rel="stylesheet" type="text/css" />
<!-- bootstrap wysihtml5 - text editor -->
<link href="{{ asset('/plugins/bootstrap-wysihtml5/bootstrap3-wysihtml5.min.css') }}" rel="stylesheet" type="text/css" />


<link href="{{ asset('/css/app.css') }}" rel="stylesheet">
<!-- Fonts -->
<link href='//fonts.googleapis.com/css?family=Roboto:400,300' rel='stylesheet' type='text/css'>
```

- On doit maintenant créer un dossier includes dans lequel on va ajouter deux fichier un sidebar.blade et un header.blade
- Ensuite il faut inclure ces deux fichier dans [home.blade.php]
- Pour créer une nouvelle vue il suffit d'hériter de la vue de base home.blade.php en utilisant la commande @extends('home) puis placer le contenue entre @section('content') et @endsection
- Le contenue entre @section('content') et @endsection est ajouter dans la partie @yield('content') de la vue de base home.blad.php :

Partie technique

Description du côté technique de l'application

L'application site du système vise à mettre en contact des étudiants et leur département universitaire avec des conditions, sous la supervision du département informatique de la faculté des sciences.

Pour ce faire il sera créé une base de données en ligne supportera un nombre assez des étudiants et des enseignants du département informatiques, administrée par un responsable. Cette base de données sera accessible par les étudiants de départements dûment identifiés via leur compte fournis à l'inscription et validé par un e-mail de vérification, à ce stade 'utilisateur peut se connecter et bénéficier des fonctionnalités offertes. L'utilisateur doit remplir certaines contraintes dont avoir un accès internet, un e-mail et un code valides, un navigateur web supportant l'application, un statut réel au sein du département (l'application ne traite pas les personnes morales).

L'application sera techniquement décomposée en plusieurs logiciels :

* un serveur web qui recevra les requêtes des navigateurs des utilisateurs relayant la requête au serveur d'application.

* un serveur d'application qui exécutera le code source de l'application et recevra les requêtes relayées par le serveur web, il construira ses réponses à l'aide d'informations extraites de la base de données avec des algorithmes moins complexes qui fournies un temps minimum de réponses.

* un serveur de base de données qui conservera les données entrées par les utilisateurs et permet une taille élevé des fichiers d'entrée.

Réalisation des cas d'utilisations :

Acteur : Etudiant

Cas d'utlisation : S'enregistrer – S'authentifier – Se déconnecter

Routes

Ce ligne de code résume cinq routes fournis par le Framework Laravel 5. Le deuxième ligne présente le système de vérification des emails.

```
Route::controllers([
    'auth' => 'Auth\AuthController',
```

```
    'password' => 'Auth\PasswordController',
]);
Route::get('register/verify/{code}', [
    'as' => 'confirmation_code',
    'uses' => 'WelcomeController@confirm'
]);
```

Controller : AuthController et Registrar du Services

Modèle : User

Vue :

Les vues concernées par ce cas d'utilisation sont toutes les vues du dossier « auth »

Cas d'utilisation : Télécharger

Routes

A l'instant où l'étudiant clique sur le bouton « Télécharger », le système fait appelle à cette route qui exécute le Controller : DashboardController.

```
Route::get('/telecharger', 'DashboardController@lister');
```

Controller : DashboardController

La méthode « Lister() » récupère tous les fichiers visible c'est-à-dire différent de 0 depuis la base des données de la table fichiers, cette dernière utilise le modèle « Fichier » et génère une vue dans le dossier « includes/télécharger » en passant les fichiers come des entités.

Modèle : Fichier

Aucune méthode utilisée

Vue

La vue télécharger hérite de la vue '' home ''

Cas d'utlisation : Visualiser Cours

Routes

Quand l'étudiant ouvre la page d'accueil, le système fait appelle à cette route qui exécute le Controller : DashboardController

```
Route::get('/', 'DashboardController@courParSemestre');
```

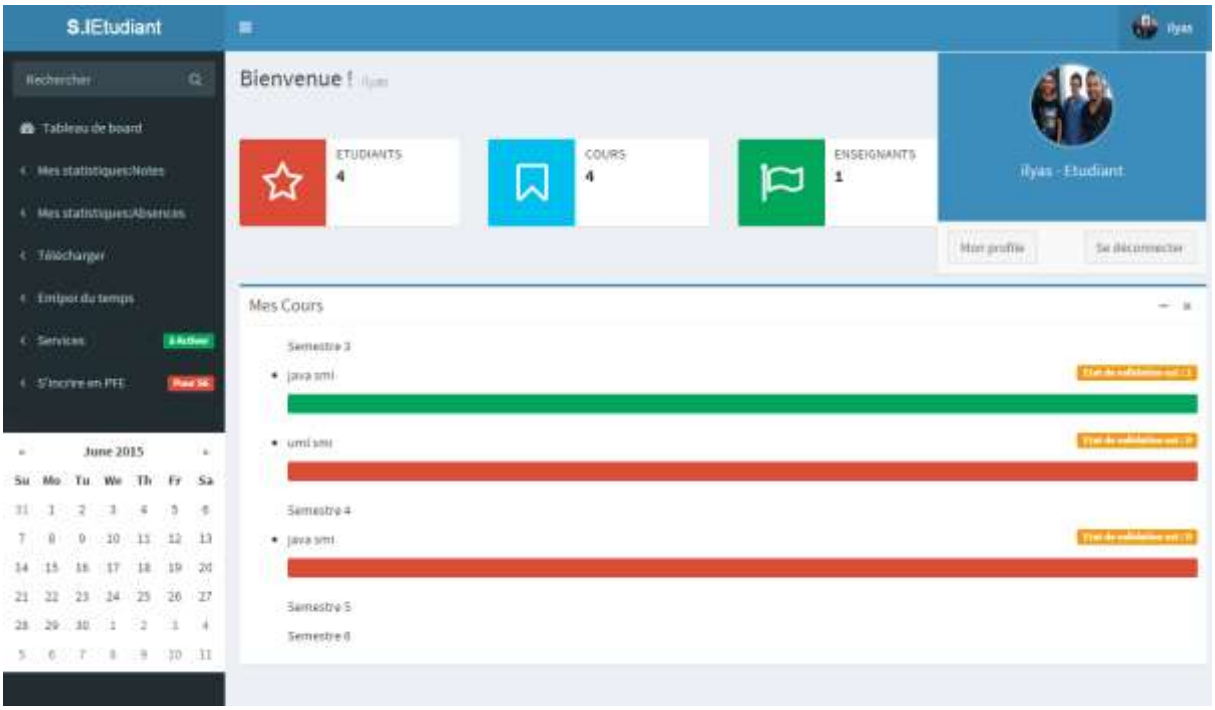
Controller : DashboardController

La méthode « courParSemestre() » récupère tous les cours où l'étudiant authentifié et inscrit depuis la base des données de la table Inscription, cette dernière utilise les modèles « User » et « Module » pour génère une vue dans le dossier «welcome » en passant les cours come un tableau des tableaux avec chaque sous tableau est un semestre. D'autres données représentant les statistiques des étudiants, enseignants, cours, fichiers seront ajoutés.

Modèles : User, Module, Fichier et Enseignant

La principales méthode dans le modèle User qui calcule les cours inscrit et leurs états est : « courParSemestre () » de la classe User. Cependant la table Inscription doit être remplie à l'avance.

Vue



Cas d'utlisation : Rechercher

Routes

L'étudiant entre un mot-clé dans le formulaire est valide. La fonction rechercher() sera exécuter.

```
Route::post('/rechercher', 'DashboardController@rechercher') ;
```

Controller : DashboardController

Le système recherche tous les fichiers associés à ce mot-clé et les affiche dans la vue « includes/recherche » leur noms et lien de téléchargement.

Modèles : Fichier/Tag

Cas d'utlisation : S'inscrire en PFE

Routes

Lorsque l'étudiant clique sur le bouton « S'inscrire en PFE » le contrôleur suivant sera exécuté :

```
Route::get('/pfe', 'DashboardController@pfe');
```

Sinon, s'il remplit le formulaire ce contrôleur sera exécuté :

```
Route::post('/pfe',[
    'as' => 'pfe_post', 'uses' =>
'DashboardController@pfe_post']);
```

Controller : DashboardController

La méthode « pfe() » génère un formulaire d'inscription en module PFE.

La méthode « pfe_post() » génère un fichier '.PDF' à imprimer dans le cas normale.

Les contraintes : + les emails et les codes doivent figurées dans la table users.

Modèle : Users

Des méthodes statiques sont satisfaisantes afin de réaliser ce cas d'utilisation.

Vue

Il éxiste deux vues, une présente un formulaire à remplir et l'autre le fichier PDF à imprimer.

Elles sont dans le dossier includes sous les noms « pfe_form » et « print ».

Cas d'utlisation : Demander des services

Routes

A la demande su services un formulaires sera généré grâce à :

```
Route::get('/services', 'DashboardController@services');
```

Ensuite, l'étudiant rempli le formulaire et valide à cette instant ce contrôleur sera exécuté :

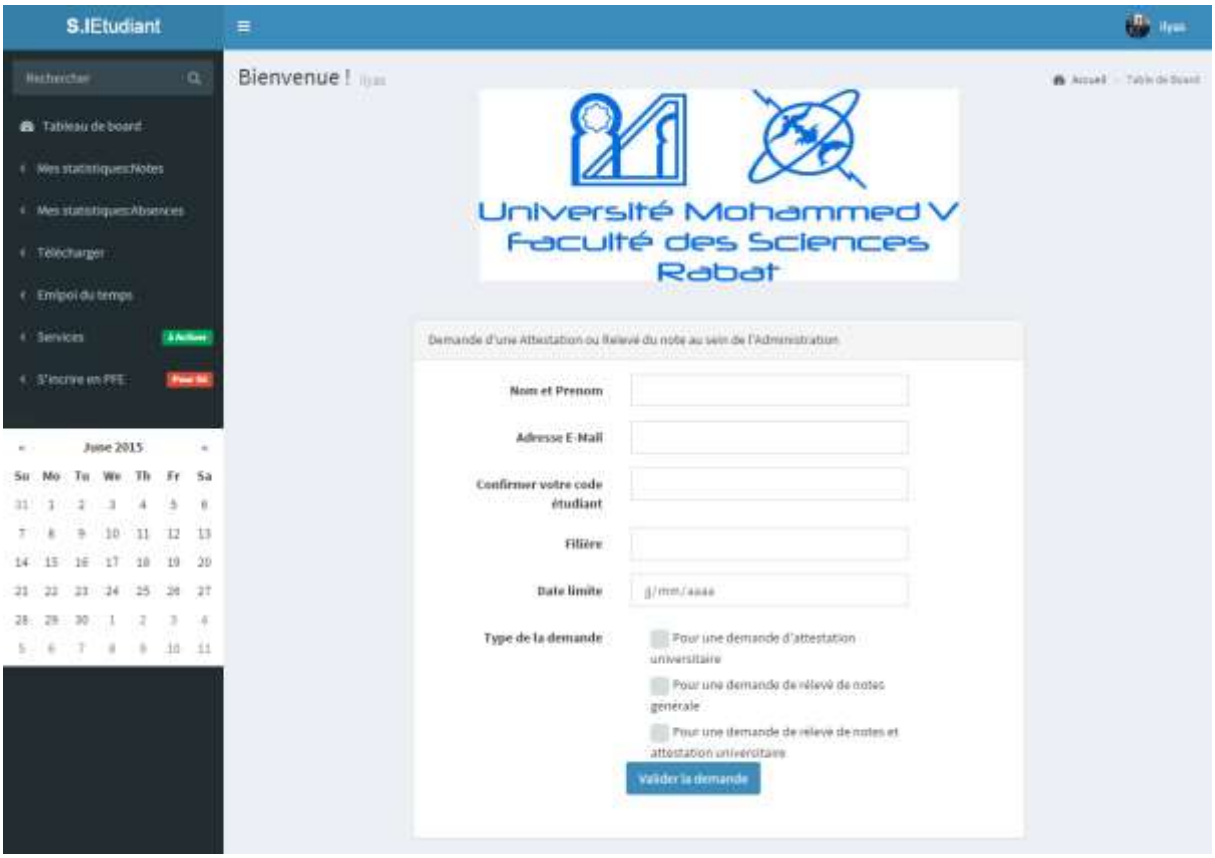
```
Route::post('/services',[
    'as' => 'services', 'uses' =>
'DashboardController@services_store']);
```

Controller : DashboardController

La première méthode retourne un formulaire dans la vue « includes/attestaion_form » et la deuxième enregistre la demande.

Modèle : Service

Vue



Cas d'utlisation : Visualiser notes

Routes

L'utilisateur est authentifié et déclenche un événement en cliquant sur le champ "Mes Statistiques" qui mène à la route « prog/ » celle-ci exécute le contrôleur DashboardController@prog qui exécute la méthode de calcule des moyennes par semestre et les chargent dans la vue « View/includes/prog » où l'utilisateur visionne des graphs des différents semestre depuis la table « evaluations » dans la base des données.

```
Route::get('/prog', 'DashboardController@prog');
```

Controller : DashboardController

La méthode principale est « moyenneParSemestre() » de la classe User. Cette méthode retourne un tableau key => value contenant semestre => moyenne de l'étudiant authentifié.

Modèle : User

Vue

La vue générer est « includes/prog » :

Cas d'utlisation : Visualiser Absences

Routes

L'étudiant clique sur le bouton « Mes statistiques Absences », cette dernière fait appelle à :

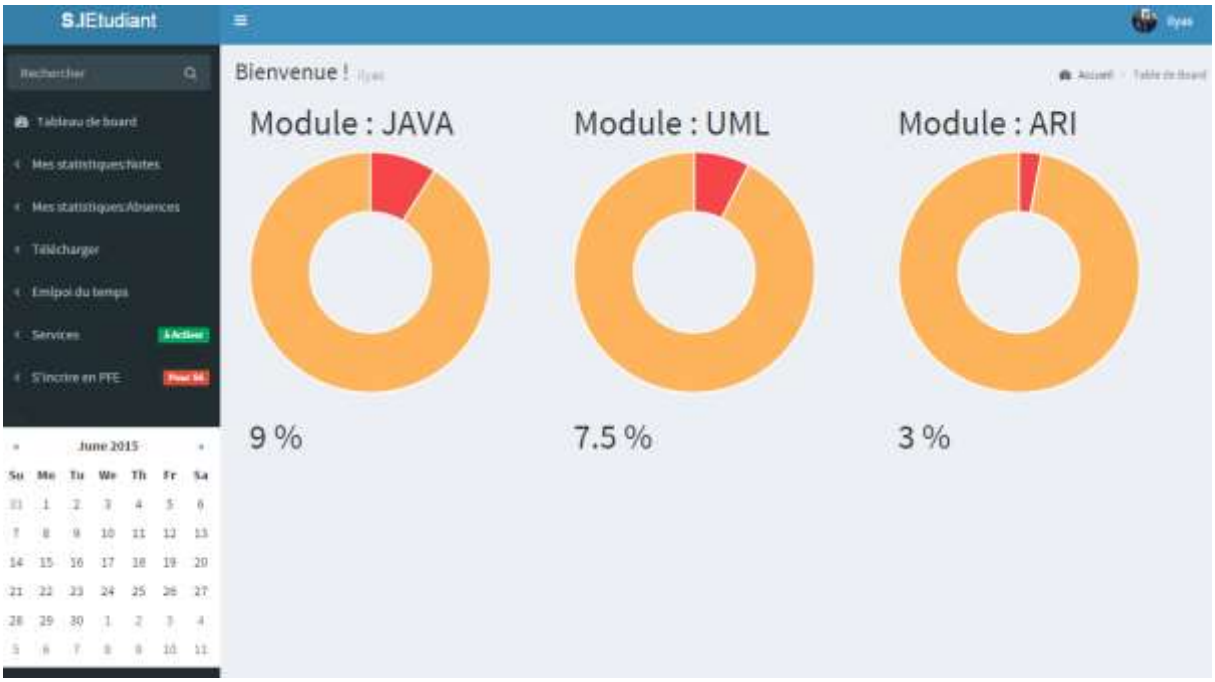
```
Route::get('/absences', 'DashboardController@absences');
```

Controller : DashboardController

Modèle : User

Principale méthode : « getAbsences() » de la classe User.

Vue



Cas d'utlisations : Gérer son profile et visualiser les emplois

Ces case d'utilisations sont similaires à d'autres cas donc la documentation du code sera suffisante .