

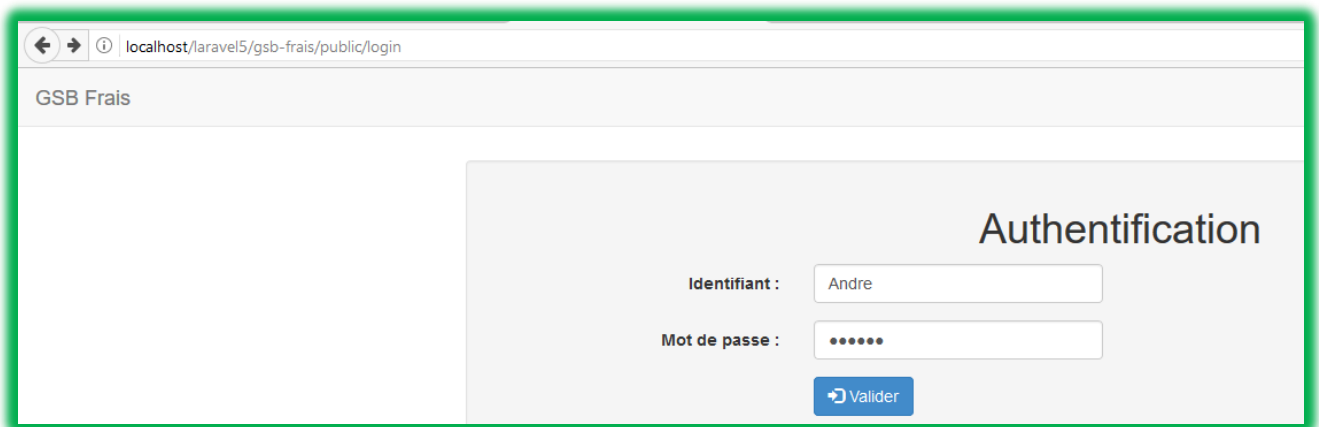
GSB-FRAIS - Login d'un visiteur

Introduction

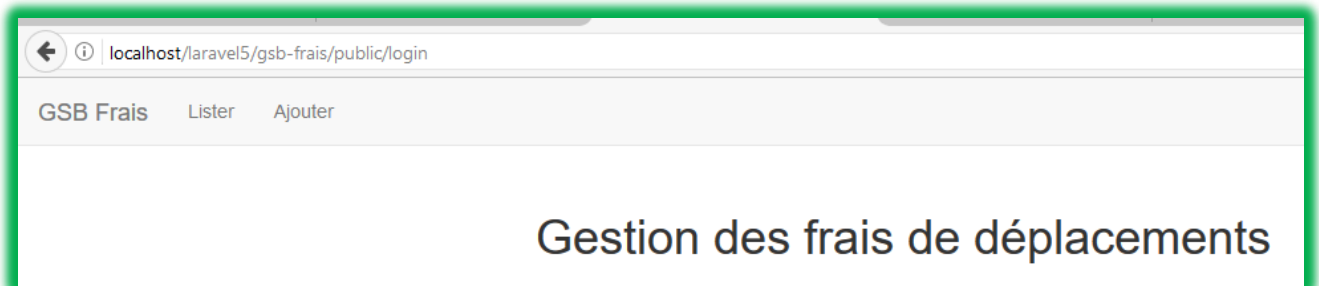
Nous allons développer l'application gsb-frais dans l'environnement Laravel. Cette application permet aux visiteurs médicaux de saisir les différents frais qu'ils ont eus dans le cadre de leurs missions.

Voici quelques écrans de l'application :

Connexion



Menu des frais



Pour mener à bien ce travail vous aurez besoins de trois commandes Laravel : créer un projet, créer un contrôleur mettre à jour le framework (composer.json, les formulaires) et créer une classe métier. Ces commandes ont été vues dans les cours Slam5, recherchez-les et notez-les dans un coin de manière à ne pas avoir à les rechercher à chaque fois. Vous aurez aussi besoin de récupérer des dossiers, fichiers et autres informations, l'ensemble des ressources nécessaires se trouve dans le dossier Ressources\gsb-frais.



Copier le dossier gsb-frais dans votre espace personnel.

Création du projet

Le projet



Décompressez le projet nommé ProjetTemplate et renommez-le sous le nom gsb-frais. Le ProjetTemplate possède dans composer.json la référence à la bibliothèque de formulaires, les sections providers et aliases du fichier config/app.php (voir supports précédents) et la mise à jour le framework (composer update).

Dans le dossier public copier le dossier assets pris dans Ressources.

Ouvrir votre projet gsb-frais sous PhpStorm,



Dans le dossier config, vérifier que le fichier app.php référence bien les bibliothèques HTML et Form.

La base de données



Sous phpmyadmin, créer la base de données gsb à partir des scripts du dossier BdD en les lançant dans l'ordre suivant :

- crt_base.sql,
- crt_tables.sql
- crt_constraints.sql
- insert_content.sql

Penser à se positionner sur cette base avant de lancer les scripts qui se trouvent dans Ressources/bd.

Modifier le fichier .env avec les données suivantes : bdd : gsb, user : usersio, password : sio.



Cryptez les mots de passe par la commande :

- update visiteur set pwd_visiteur = secret ;

L'IHM

C'est la partie visible de l'application.

La page principale


La page principale sera constituée par le fichier master.blade.php (principe étudié en cours). Comme pour les projets précédents, les vues viendront s'y insérer. Cette page a été déjà créée, mais il vous reviendra de la compléter en référencant les fichiers css et js de bootstrap, en définissant la place d'insertion des vues et enfin en ajoutant les url au fur et à mesure des besoins.




Dans le dossier views, créer le dossier layouts, copier dans ce dossier le fichier master.blade.php pris dans Ressources et effectuer les travaux demandés ci-dessus.

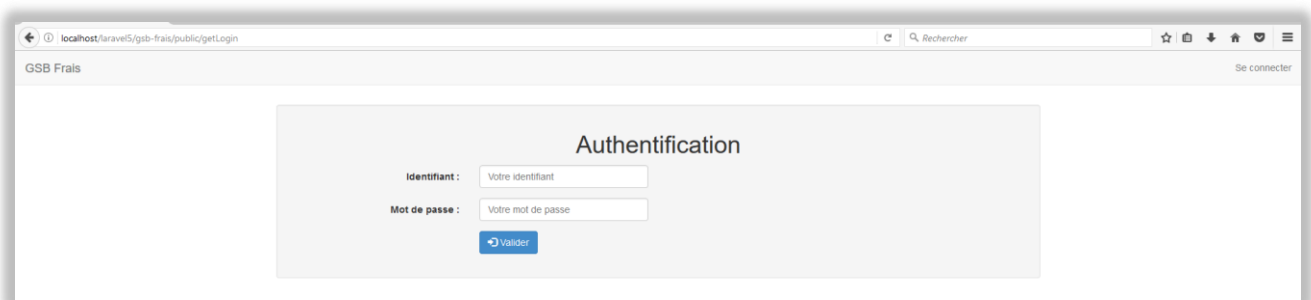
La route par défaut

Lorsqu'on appelle le site par l'url de base (http://localhost/mon_nom/monProjet/public/) on fait appel à la route par défaut qui pour le moment appelle la page welcome.blade.php. Nous allons remplacer cette page par la vue [home.blade.php](#) qui viendra s'insérer à l'emplacement défini par l'instruction @yield().

 Dans le dossier views, supprimer la page welcome, copier la vue [home.blade.php](#) depuis le dossier Ressources, la compléter pour qu'elle s'insère dans la page master. Dans master.blade.php, définir l'url de base sur le lien GSB Frais.

 Dans le fichier web.php, modifier la route par défaut pour qu'elle affiche la vue home au lieu de welcome.

Tester un lancement, vous devriez obtenir quelque chose de proche de l'a copie ci-dessous :



Authentification du visiteur

Avant de pouvoir saisir ses frais, le visiteur doit d'abord s'authentifier à l'aide de son login et de son mot de passe. Cela implique que les items de menu ne doivent être accessibles que s'il s'est authentifié. Nous allons donc modifier la page master pour que les items de menu ne soient visibles que si l'utilisateur s'est authentifié.

Comment sait-on qu'un visiteur s'est authentifié ?

Lorsqu'il saisira son login et son mot de passe on vérifiera que tout est correct et si c'est le cas on placera son id (id_visiteur) dans une variable de session. Variable qui sera donc accessible dans toute l'application y compris dans master. Voici ce à quoi il faut arriver :

```
@if (Session::get('id') == 0)
<div class="collapse navbar-collapse" id="navbar-collapse-target">
  <ul class="nav navbar-nav navbar-right">
    <li><a href="#" data-toggle="collapse" data-target=".navbar-collapse.in">Se connecter</a></li>
  </ul>
</div>
@endif
@if (Session::get('id') > 0)
<div class="collapse navbar-collapse" id="navbar-collapse-target">
  <ul class="nav navbar-nav">
    <li><a href="#" data-toggle="collapse" data-target=".navbar-collapse.in">Lister</a></li>
    <li><a href="#" data-toggle="collapse" data-target=".navbar-collapse.in">Ajouter</a></li>
  </ul>
  <ul class="nav navbar-nav navbar-right">
    <li><a href="#" data-toggle="collapse" data-target=".navbar-collapse.in">Se déconnecter</a></li>
  </ul>
</div>
@endif
```

Dans la Session, on demande à récupérer la variable id : get('id') et on compare son contenu à 0. Si c'est égal à 0 c'est que le visiteur ne s'est pas authentifié dans ce cas tout ce qui est entre le @if() et le @endif sera affiché, par contre si c'est > 0 le contenu du @if() ne sera pas affiché, mais celui du deuxième sera affiché, et évidemment c'est l'inverse si id est > 0.



Modifier master comme indiqué et tester.

GSB Frais

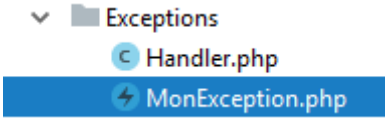
Se connecter

Gestion des frais de déplacements

L'authentification

Pour réaliser l'authentification, nous aurons besoin des composants suivants :

Composant	Valeur	Rôle
Routes	/getLogin	Prise en compte du clic sur l'item Se connecter du menu
	/login	Authentification elle-même
	/getLogout	Suppression de l'authentification
Vues	formLogin.blade.php	Formulaire de saisie du login et MdP
	error.blade.php	Vue d'affichage des messages d'erreur
Classes métier	Visiteur	Stockage des données et requêtes
Contrôleur	VisiteurController	Pilotage des demandes de login et de logout

Méthodes métier	login()	Recherche dans la base de données sur le login saisi et affectation de la variable de session
	logout()	Remise à 0 de la variable de session
Méthodes contrôleur	signIn()	Affichage du formulaire et appel de la méthode métier login()
	signOut()	Appel de la méthode métier logout()
Classe Exception	MonException.php	

Le formulaire formLogin

```

{!! Form::open(['url' => 'login']) !!}
<div class="col-md-12 well well-md">
  <center><h1>Authentification</h1></center>
  <div class="form-horizontal">
    <div class="form-group">
      <label class="col-md-3 control-label">Identifiant : </label>
      <div class="col-md-6 col-md-3">
        <input type="text" name="login" class="form-control" placeholder="Votre identifiant" required autofocus>
      </div>
    </div>
    <div class="form-group">
      <label class="col-md-3 control-label">Mot de passe : </label>
      <div class="col-md-6 col-md-3">
        <input type="password" name="pwd" ng-model="pwd" class="form-control" placeholder="Votre mot de passe" required>
      </div>
    </div>
    <div class="form-group">
      <div class="col-md-6 col-md-offset-3">
        <button type="submit" class="btn btn-default btn-primary"><span class="glyphicon glyphicon-log-in"></span> Valider</button>
      </div>
    </div>
    <div class="col-md-6 col-md-offset-3">
      @include('error')
    </div>
  </div>
</div>

```

Le formulaire est standard, sa seule particularité est d'inclure la vue error à l'aide de l'instruction @include.



Récupérer le formulaire dans les Ressources et le compléter.

Attention : la copie d'écran n'est pas complète, il faut quelque chose en début et en fin de formulaire pour qu'il puisse s'insérer dans la page principale master !

La vue error

Ajoutez l'appel à layouts.master

```

@extends('layouts.master')
@section('content')

```

```
@if ($erreur!= "")
    <div class="alert-danger" role="alert">
        <span class="glyphicon glyphicon-exclamation-sign" aria-hidden="true"></span>{{ $erreur or '' }}
    </div>
@endif
```

Elle est basée sur le même principe que celui utilisé pour les menus si la variable \$erreur est non vide alors le bloc div s'affichera et le contenu de la variable \$erreur avec, sinon rien ne s'affichera.



Récupérer la vue error et la compléter.

Les routes

```
// Afficher le formulaire d'authentification
Route::get('/getLogin', 'VisiteurController@getLogin');

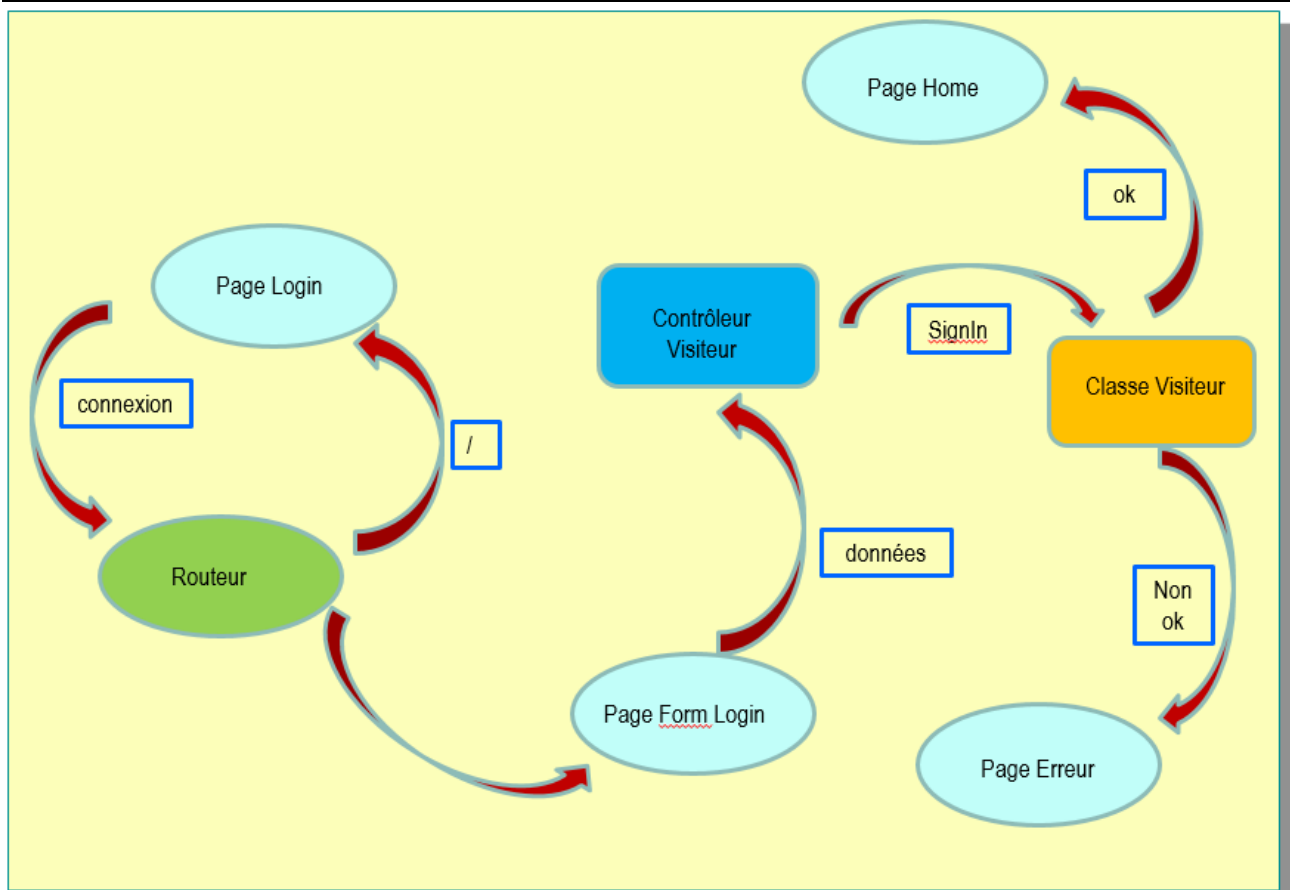
// Authentifie le visiteur à partir du login et mdp saisis
Route::post('/login', 'VisiteurController@signIn');

// Déloguer le visiteur
Route::get('/getLogout', 'VisiteurController@signOut');
```



Saisir les nouvelles routes.

Remarque : enchaînement des écrans.



Le contrôleur VisiteurController



Créer à l'aide de la commande artisan le contrôleur VisiteurController. Son entête ressemblera à ceci :

```
use Request;
use App\metier\Visiteur;
class VisiteurController extends Controller {
```

La méthode getLogin()

```
/**
 * Initialise le formulaire d'authentification
 * @return type Vue formLogin
 */
public function getLogin() {
    try{
        $erreur = "";
        return view ('Vues/formLogin', compact('erreur'));
    }catch (MonException $e){
        $erreur = $e->getMessage();
        return view('Vues/formLogin', compact('erreur'));
    }catch (Exception $e){
        $erreur = $e->getMessage();
        return view('Vues/formLogin', compact('erreur'));
    }
}
```

La variable \$erreur est initialisée à vide, il n'y aura donc pas de message affiché.



Saisir la méthode getLogin().

La méthode *signIn()*

```

public function signIn() {
    try{
        $login = Request::input('login');
        $pwd = Request::input('pwd');
        $unVisiteur = new Visiteur();
        $connected = $unVisiteur->login($login, $pwd);
        if ($connected){
            if (Session::get('type') === 'P'){
                return view('Vues/homePraticien');
            }else{
                return view('Vues/home');
            }
        }
        else{
            $erreur = "Login ou mot de passe inconnu !";
            return view('Vues/formLogin', compact('erreur'));
        }
    }catch (MonException $e){
        $erreur = $e->getMessage();
        return view('Vues/formLogin', compact('erreur'));
    }catch (Exception $e){
        $erreur = $e->getMessage();
        return view('Vues/formLogin', compact('erreur'));
    }
}

```

Si la méthode `login()` retourne `false`, c'est que le login ou le mot de passe ne sont pas corrects, on réaffiche donc le formulaire, mais en ajoutant le message d'erreur.



Saisir la méthode `signIn()`.

La méthode singOut()

```
/**
 * Déconnecte le visiteur authentifié
 * @return type Vue home
 */
public function singOut() {
    $unVisiteur = new Visiteur();
    $unVisiteur->logout();
    return view('home');
}
```

Elle se contente d'appeler la méthode logout() de la classe métier Visiteur.



Saisir la méthode singOut().

La classe métier Visiteur

Créer la classe métier Visiteur à l'aide de la commande artisan, puis la compléter.

```
namespace App\metier;
use Illuminate\Support\Facades\Session;
use Illuminate\Database\Eloquent\Model;
use DB;

class Visiteur extends Model {

    //On déclare la table visiteur
    protected $table = 'visiteur';
    public $timestamps = false;
    protected $fillable = [
        'id_visiteur',
        'id_laboratoire',
        'id_secteur',
        'nom_visiteur',
        'prenom_visiteur',
        'adresse_visiteur',
        'cp_visiteur',
        'ville_visiteur',
        'date_embauche',
        'login_visiteur',
        'pwd_visiteur',
        'type_visiteur'
    ];
}
```

La méthode login()

```

/**
 * Authentifie le visiteur sur son login et Mdp
 * Si c'est OK, son id est enregistré dans la session
 * Cela lui donne accès au menu général (voir page master)
 * @param type $login : Login du visiteur
 * @param type $pwd : Mdp du visiteur
 * @return boolean : True or false
 */
public function login($login, $pwd) {
    $connected = false;
    try
    {
        $visiteur = DB::table('visiteur')
            ->select()
            ->where('login_visiteur', '=', $login)
            ->first();
        if ($visiteur) {
            if ($visiteur->pwd_visiteur == md5($pwd)) {
                Session::put('id', $visiteur->id_visiteur);
                Session::put('type', $visiteur->type_visiteur);
                $connected = true;
            }
        }
    }
    catch (QueryException $e) {
        throw new MonException($e->getMessage(), 5);
    }
    return $connected;
}

```

On lit le Visiteur sur son login qui est unique et on compare le Mdp ramené à celui qui a été saisi. Si c'est le même on enregistre son id dans la variable de session 'id' et on retourne true, sinon on retourne false.



Saisir la méthode login()

La méthode logout()

```
/**
 * Délogue le visiteur en mettant son Id à 0
 * dans la session => le menu n'est plus accessible
 */
public function logout(){
    Session::put('id', 0);
}
```

Elle est simpliste et parle d'elle-même.



Saisir la méthode logout().

Les url des items de menu

Dernière étape, il faut définir les urls des items de menu Se connecter et se déconnecter.

Je vous laisse trouver les valeurs de ces items ainsi que leur emplacement.



Effectuer les ajouts nécessaires et tester une connexion et déconnexion ainsi qu'un login/mdp erroné.