

# RAPPORT DE PROJET

## MANGATECH

### SOMMAIRE:

Liste des compétences du référentiel.....	
Résumé du projet.....	2
Expression des besoins.....	3
Modélisation de l'application.....	4
1.UML.....	
2.MERISE.....	
3.MAQUETTAGE.....	
* Zoning.....	
* Wireframe.....	
Langages utilisés.....	
Environnement de travail.....	
1. VsCode.....	
2. phpMyAdmin.....	
Structure du projet.....	
1. Décomposition des fichiers.....	
2. Architecture MVC.....	
Réalisation de l'application.....	
1. Connexion à la base de donnée.....	
2. Création d'une migration.....	
3. Création d'un model.....	
4. Création d'un controller.....	
5. Création de la vue avec Blade.....	
6. Création de la route.....	
Site final.....	

## **LISTE DES COMPÉTENCES DU REFERENTIEL**

- MAQUETTER UNE APPLICATION
  - RÉALISER UNE INTERFACE UTILISATEUR WEB STATIQUE ET ADAPTABLE
  - DÉVELOPPER UNE INTERFACE UTILISATEUR WEB DYNAMIQUE
  - CRÉER UNE BASE DE DONNÉE
  - DÉVELOPPER LES COMPOSANTS D'ACCÈS AUX DONNÉES
  - DÉVELOPPER LA PARTIE BACK-END D'UNE APPLICATION WEB OU WEB MOBILE
- COMPÉTENCE TRANSVERSALE DE L'EMPLOI :
- UTILISER L'ANGLAIS DANS SON ACTIVITÉ PROFESSIONNELLE EN DÉVELOPPEMENT WEB ET WEB MOBILE

## **RÉSUMÉ**

L'application Mangatech est une bibliothèque en ligne qui permet aux visiteurs de naviguer entre une liste de mangas et animés, ainsi qu'un forum avec plusieurs sujets de discussions. Elle possède nombre de fonctionnalités permettant par exemple de filtrer les œuvres par tag, ou d'en rechercher avec la barre de recherche, voir le nombre de j'aime sur une œuvre, écrire des commentaires etc...

L'application contient 2 type de rôle : l'admin et l'utilisateur.

En créant un compte et en se connectant, l'utilisateur s'ouvre plus de possibilités. Il peut écrire des commentaires sous les œuvres, lancer de nouvelles discussions dans les forum, et accéder à leur espace d'utilisateur qui contiendra leur bibliothèque avec les la liste des œuvres qu'ils ont ajoutés en tant que Favori, lu ou à voir plus tard.

Une fois connecté avec son compte admin, il peut accéder a son panneau d'administration et gérer l'entièreté des données de l'application ici. Il aura le contrôle sur les utilisateurs, les discussions, les commentaires, les œuvres, les sujets et les tags.

## **EXPRESSION DES BESOINS**

Les visiteurs de l'application pourront naviguer à travers le site avec des limites au niveau de l'interactivité avec les autres utilisateurs et des items. Ils pourront :

- Voir la liste des items.
- Filtrer un item par tags.
- Chercher un item avec une barre de recherche.
- Naviguer dans le forum et filtrer les discussion par sujet.
- S'inscrire.

Les utilisateurs authentifiés bénéficieront d'autres fonctionnalités

- Ecrire un commentaire sous les items.
- Ecrire une nouvelle discussion dans le forum.
- Ecrire des réponses dans le forum.
- Supprimer / éditer leurs propre messages.
- Ajouter un item à leur liste de favori / Fini / à regarder plus tard.
- Accéder a leur espace utilisateur où ils pourront consulter leurs listes.

Les utilisateurs connectés en tant qu'administrateur pourront eux gérer :

- Les utilisateurs (supprimer / modifier / ajouter).
- Les discussions (supprimer).

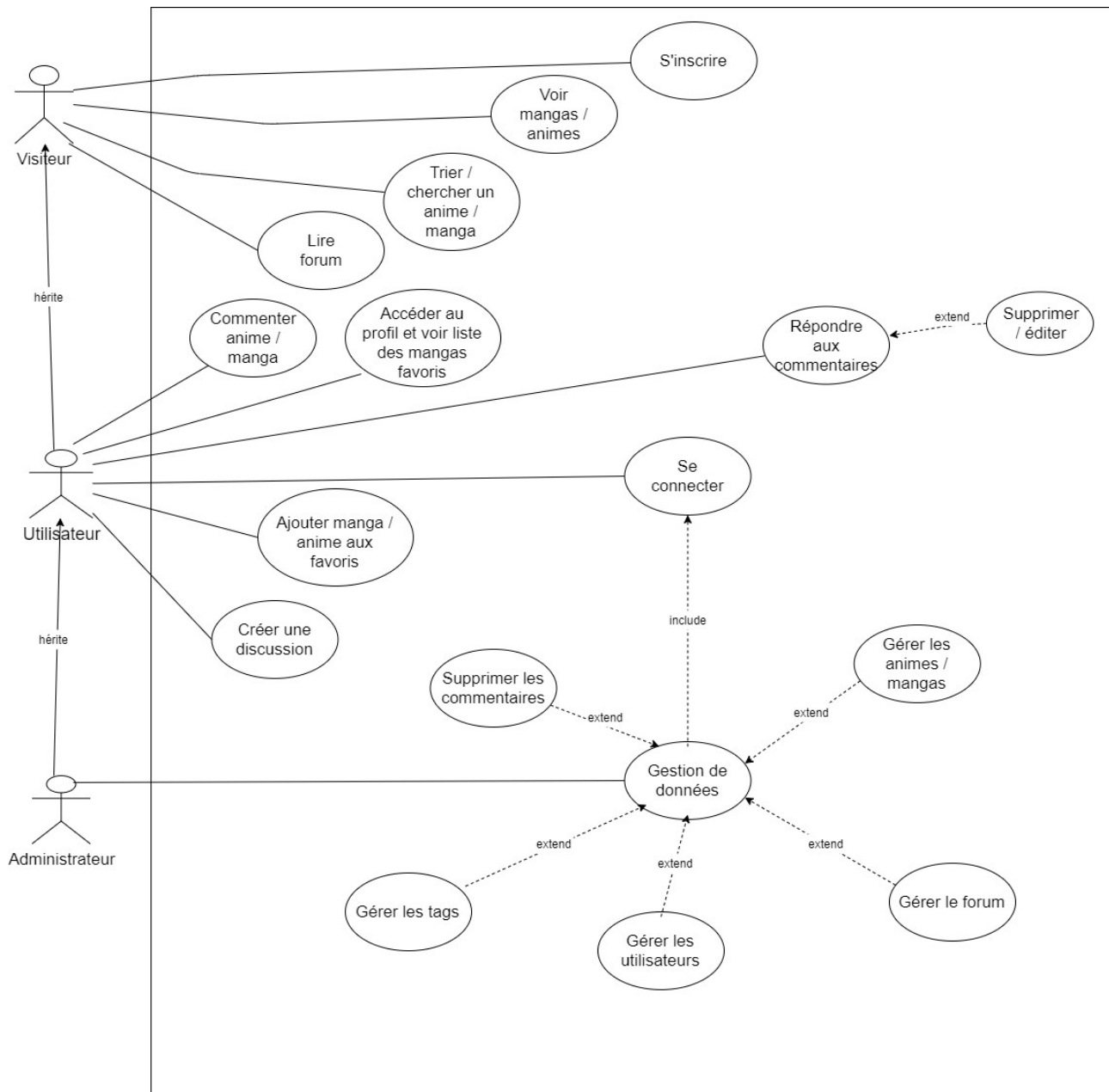
- Les commentaires (supprimer).
- les items (supprimer / modifier / ajouter).
- les sujets (supprimer / modifier / ajouter).
- les tags (supprimer / modifier / ajouter).

## **Modélisation de l'application**

### **UML**

J'ai commencé par faire le diagramme UML : le diagramme de cas d'utilisation qui schématise les possibilités et limites d'actions des acteurs sur le site.

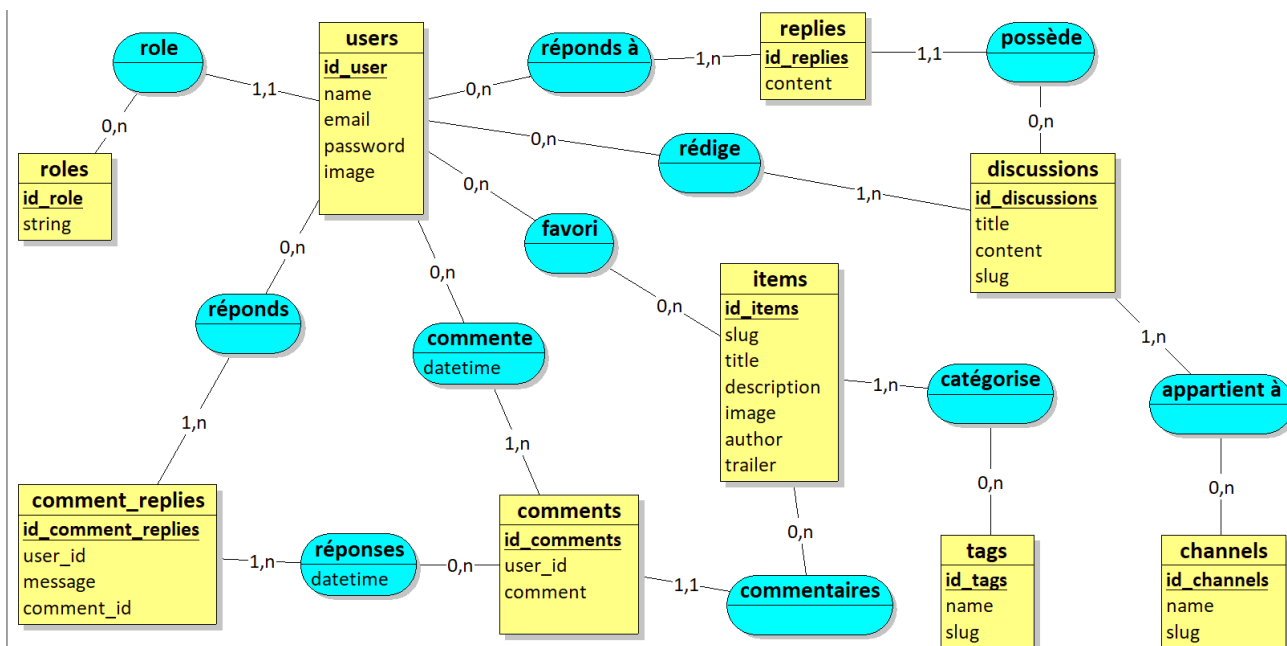
### **Diagramme de cas d'utilisation**



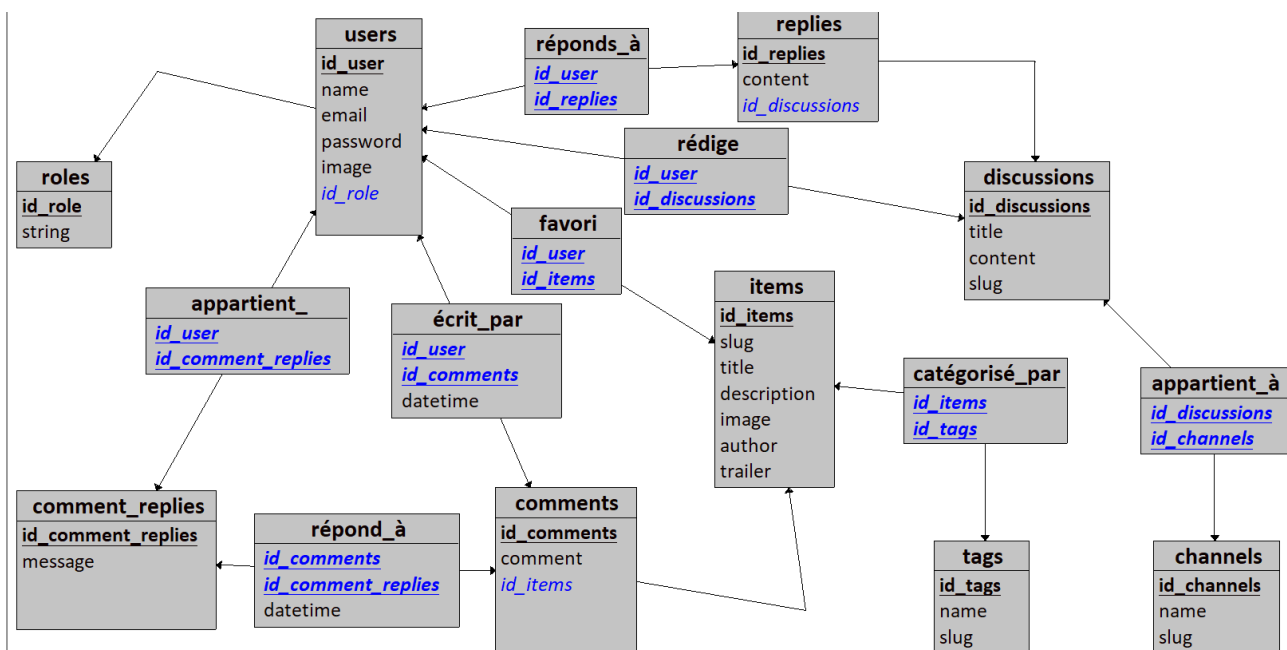
## MERISE

La méthode MERISE présente l'analyse du côté base de donnée contrairement au langage de modélisation UML qui présente la partie développement. J'ai donc réalisé le MCD, qui va permettre de représenter les données qui vont être utilisées, et le MLD, qui va décrire la structure des données utilisées, sur l'application Looping.

## MCD



## MLD



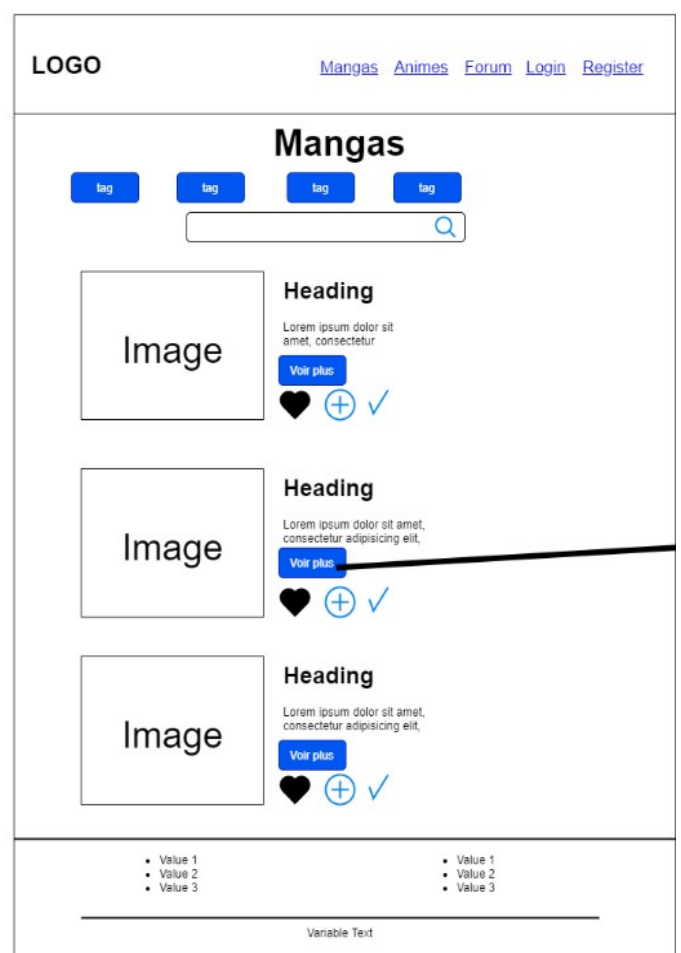
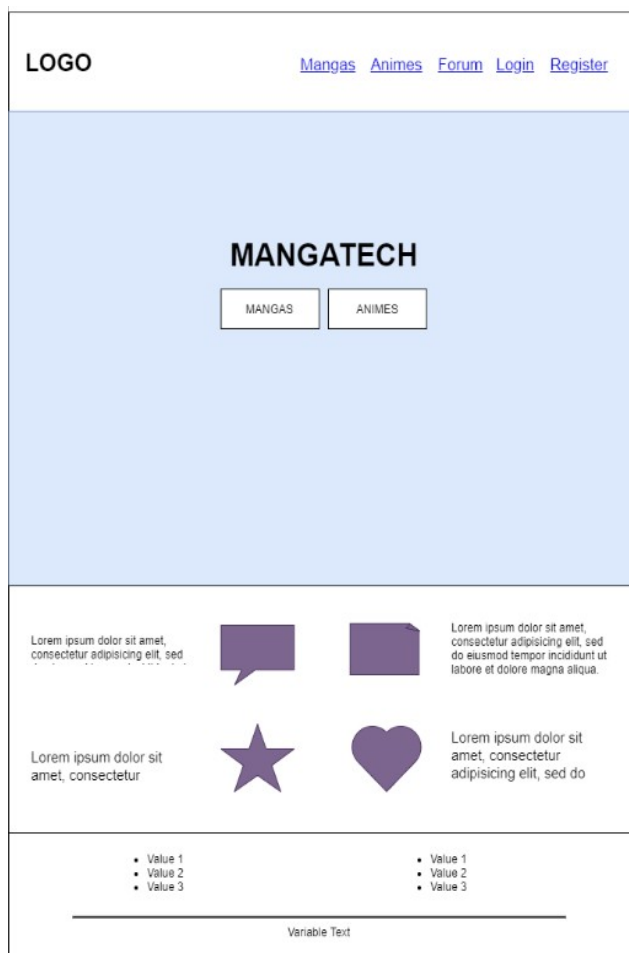
## MAQUETTAGE

J'ai ensuite maqueté mon application pour organiser l'interface du projet, ce qui va faciliter l'intégration du code pour créer le front-end. Et aussi pour donner au client(si j'en avait un) une idée du futur produit. J'ai donc effectué le Zoning, qui sera le

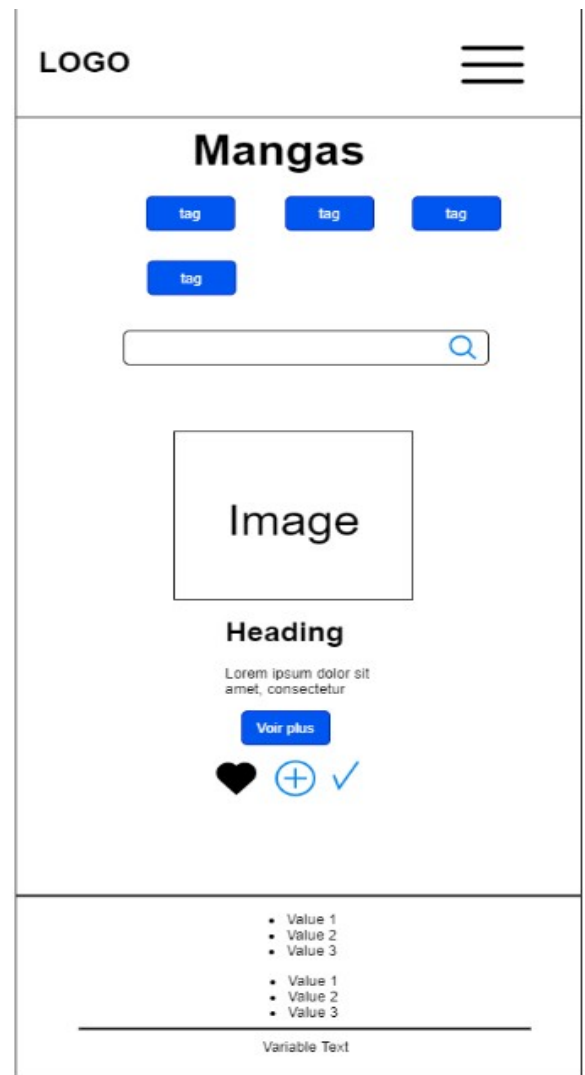
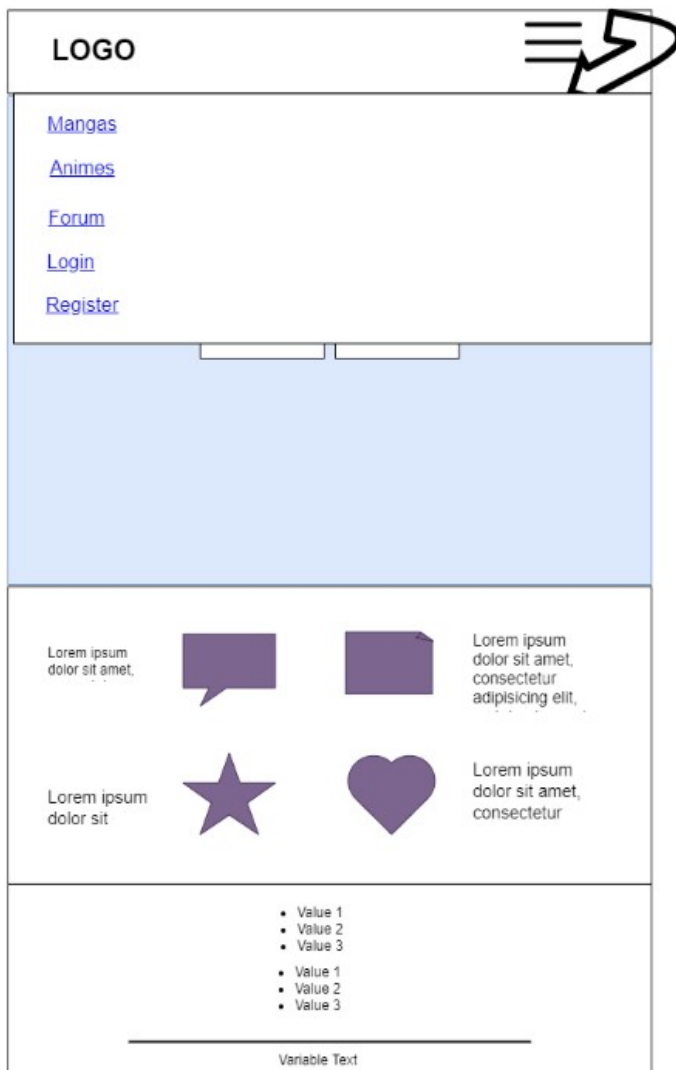
schéma des pages web pour pour montrer les emplacement des différents éléments sur la page web. Et le Wireframe, qui va être une schématisation plus aboutie qui va montrer les fonctionnalités et le contenu présent dans chaque bloc de la page.

## ZONING

## WIREFRAME



Avec les versions mobiles :



## LANGAGES UTILISÉS

HTML5

CSS3

PHP(7.3.21)

MySQL(5.7.31)

JAVASCRIPT

## FRAMEWORKS

TAILWIND CSS

jQuery

LARAVEL 8



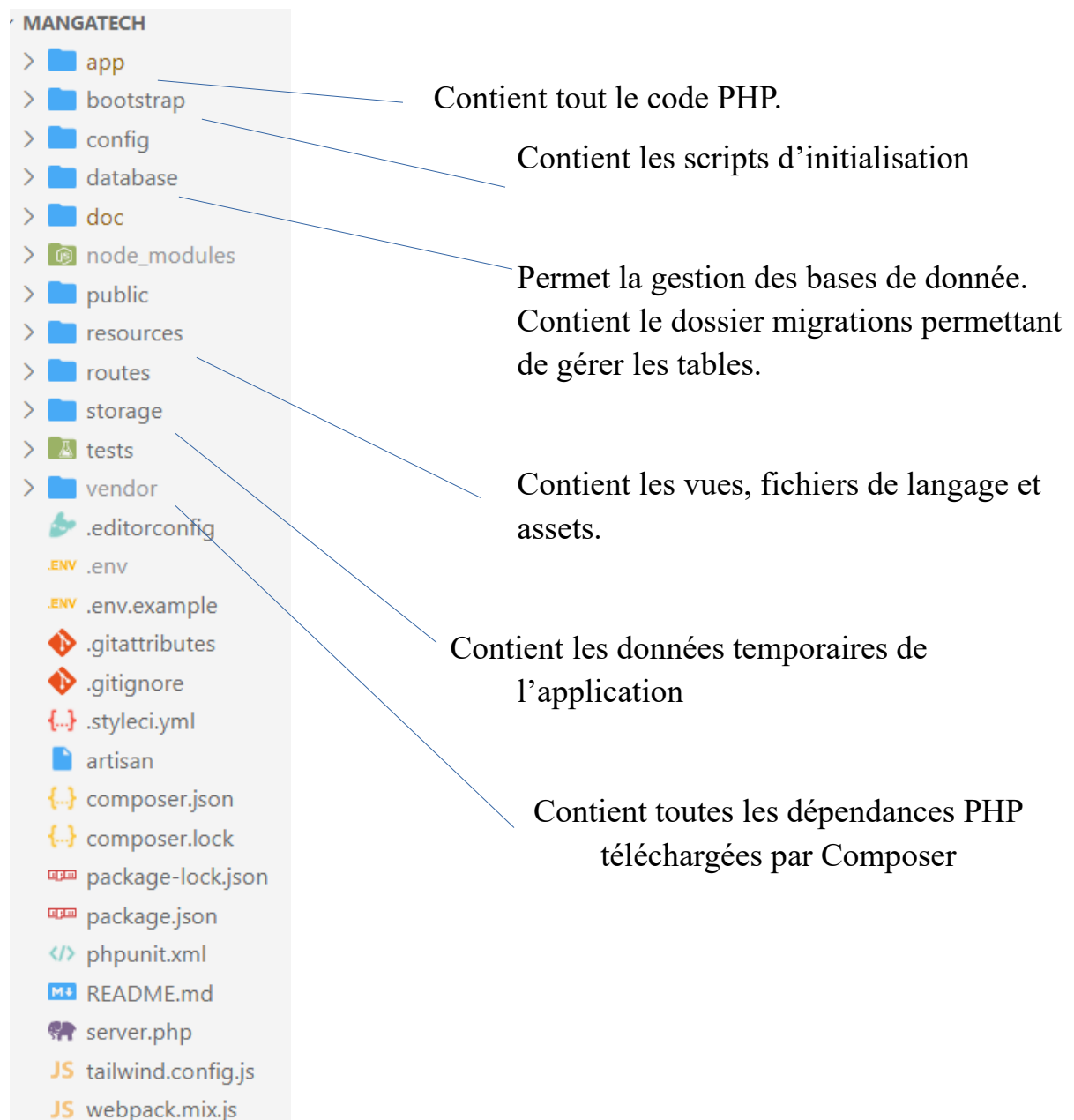
## ENVIRONNEMENT DE TRAVAIL

J'utilise l'éditeur de texte VsCode où j'y ai installé des extensions pour améliorer l'ergonomie du code.

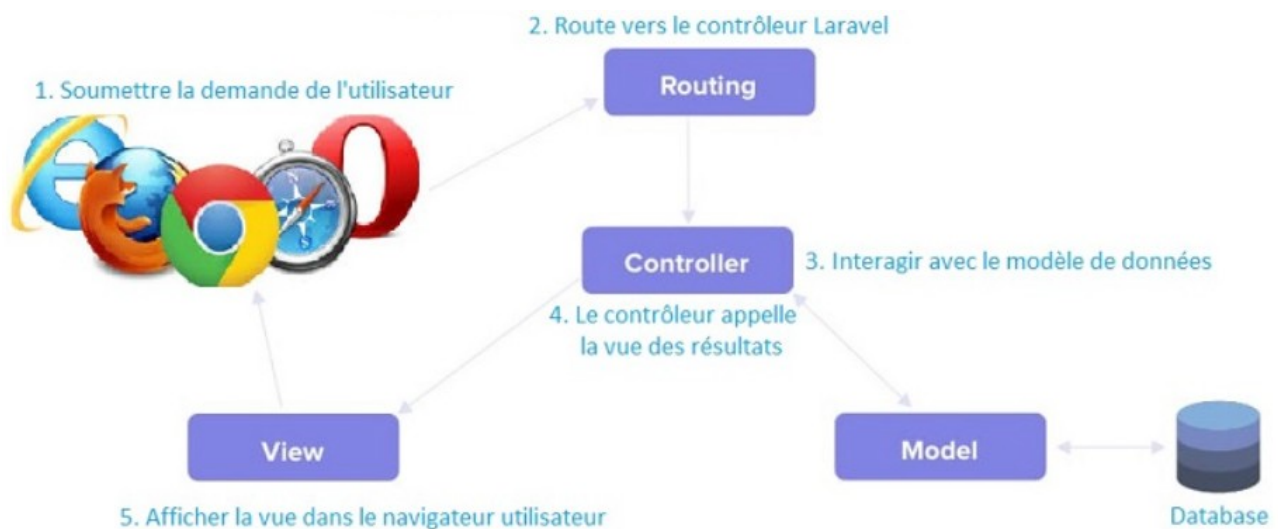
Pour le serveur local et la base de donnée j'utilise la plateforme de développement web WampServer qui utilise le serveur Apache2, et l'interface de gestion phpMyAdmin qui permet de gérer la base de donnée MySQL.

## STRUCTURE DU PROJET

### Décomposition des fichiers



## Architecture MVC



Laravel utilise le modèle architectural MVC. Qui utilise

- le modèle (app/Models) pour interagir avec la base de donnée.

- la vue (ressources/views) qui contient la représentation visuelle à renvoyer à l'utilisateur.

- le contrôleur (app/Http/Controllers) est le cerveau qui synchronise le modèle et la vue

Le router (routes/web.php) est aussi une entité cruciale de cette architecture car c'est lui qui va relier la requête de l'utilisateur et l'action du contrôleur.

## INSTALLATION

J'ai d'abord téléchargé le gestionnaire de dépendances Composer puis j'ai entré la commande `composer global require laravel/installer` dans mon invite de commande.

J'ai ensuite utilisé `laravel new Mangatech` qui crée un nouveau projet appelé «Mangatech».

## RÉALISATION DE L'APPLICATION

J'utilise la console de VsCode pour écrire les commandes de Artisan qui permettent de créer des fichiers, dossiers, classes, vérifier les routes etc.

J'utilise donc la commande `'php artisan serve'` qui permet de visualiser l'application web dans un localhost sur n'importe quel navigateur.

### Base de donnée

La première étape a été de créer une nouvelle base de donnée appelée «Mangatech» via phpMyAdmin.

J'ai ensuite effectuée la liaison entre la base de donnée et l'application dans le fichier `.env`

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=mangatech
DB_USERNAME=root
DB_PASSWORD=
```

Cette connexion est l'équivalent de la ligne 4 de cette fonction faite en php brut.



```

1 public static function getPdo()
2     {
3         try {
4             $pdo = new PDO('mysql:host=localhost;dbname=mangatech;charset=utf8'
5 , 'root', '', [
6                 PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
7                 PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
8             ]);
9             // echo"ok";
10        }catch(PDOException $e) {
11            echo"failed" . $e->getMessage();
12        }
13
14        return $pdo;
15    }

```

*Équivalent de la connexion Bdd du .env en php brut*

## Création d'une migration

Je vais vous montrer les étapes du développement de la partie forum du site avec ici la table 'discussions'.

La deuxième étape est de créer une migration. Elle va permettre de créer un 'schéma' de base de donnée, autrement dit:une table avec ses colonnes.

On utilise la ligne de commande 'php artisan make:migration create\_discussions\_table'. Cela va créer un fichier du même nom dans le dossier 'database/migrations'.

```
2021_04_22_080846_create_discussions_table.php X
database > migrations > 2021_04_22_080846_create_discussions_table.php > ...
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  class CreateDiscussionsTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('discussions', function (Blueprint $table) {
17             $table->id();
18             $table->integer('user_id');
19             $table->string('title');
20             $table->text('content');
21             $table->string('slug');
22             $table->integer('channel_id');
23             $table->timestamps();
24         });
25     }
26
27     /**
28      * Reverse the migrations.
29      *
30      * @return void
31      */
32     public function down()
33     {
34         Schema::dropIfExists('discussions');
35     }
36 }
37
```

*Contenu du fichier créé*

```
CREATE TABLE discussions(
    id_discussions INT NOT NULL AUTO_INCREMENT,
    title VARCHAR(250),
    slug VARCHAR(250),
    content TEXT,
    user_id INT,
    channel_id INT,
    PRIMARY KEY(id_discussions),
    FOREIGN KEY(user_id) REFERENCES users(id),
    FOREIGN KEY(channel_id) REFERENCES channels(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

*création de la table si j'avais utilisé le langage SQL*

## Création du modèle

On utilise la commande 'php artisan make:model Discussion' qui va créer le fichier 'Discussion' dans 'app/models'.

```
public function user()
{
    return $this->belongsTo(User::class);
}

public function channel()
{
    return $this->belongsTo(Channel::class);
}

public function replies()
{
    return $this->hasMany(Reply::class);
}
```

Ici on crée les relations entre les tables. Chaque fonction correspond à une 'cardinalité'.

-> Contenu du modèle Discussion

Voici une relation  
présente dans le  
modèle 'User' liée à

```
public function discussions()  
{  
    return $this->hasMany(Discussion::class);  
}
```

'Discussion'. On peut voir qu'une discussion appartient à un user et qu'un user peut avoir plusieurs discussions. C'est donc une relation One-To-Many.

Eloquent se charge automatiquement de définir les clés étrangères. Ici, par exemple, Eloquent va ajouter la colonne 'user\_id'(il prend le nom du parent en snake\_case et ajoute le suffixe '\_id') au modèle 'Discussion'.

Toujours dans le modèle on créer une fonction pour filtrer les discussions par channels.

```
//Filtre les discussions par channels  
public function scopeFilterByChannels($builder)  
{  
    //Si la requete est faite  
    if (request()->query('channel')) {  
        //on cherche la channel spécifique que l'on veut filtrer  
        $channel = Channel::where('slug', request()->query('channel'))->first();  
        //Si la channel existe  
        if ($channel) {  
            // on filtre le builder dont l'id correspond a celui de la database  
            return $builder->where('channel_id', $channel->id);  
        }  
        return $builder;  
    }  
    return $builder;  
}
```

*On créer la méthode qui va sélectionner les éléments dans la base de donnée.*

On utilise la collection d'Eloquent avec ses méthodes comme 'first' qui permettent d'interagir avec la base de donnée. Ici cette méthode va chercher le premier item.

*La même requête en langage SQL*

```
SELECT * FROM `discussions`  
INNER JOIN channels ON channels.id =  
discussions.channel_id  
where channels.slug = ':slug'
```

## **Création du controller**

J'utilise ensuite la commande 'php artisan make:controller Forum/DiscussionController'.

Elle crée le fichier  
app/Http/Controllers/Forum/DiscussionController  
Laravel va automatiquement créer le namespace Forum.

Dans la class DiscussionController j'ai fait cette méthode index qui va retourner la vue que nous allons voir après.

On va placer la variable 'discussions' à la vue avec la fonction laravel 'with', cette variable contient Toutes les discussions et la fonction pour les filtrer par channels que nous avons vu dans le modèle.



```
public function index()
{
    return view('forum.index', with('discussions', Discussion::filterByChannels()->get()));
}
```

## Création de la vue

Laravel utilise le moteur de template Blade pour la vue.

Pour utiliser Blade il suffit d'ajouter `.blade.php` en suffixe d'un fichier.

```

1 @foreach ($discussions as $discussion)
2
3     <!--Card-->
4     <div class="rounded w-full overflow-hidden shadow-lg text-center">
5         <div class="px-6 py-4">
6             <div class="font-bold text-xl mb-2 text-left">{{ $discussion->user->name }}
7         </div>
8             <span class="block text-right">{{ $discussion->created_at }}</span>
9             <div class="font-bold text-xl mb-2"><a href="{{ route('discussions.show'
10 , $discussion->slug) }}">{{ $discussion->title }}</a></div>
11             <p class="text-gray-700 text-base">
12                 {{ $discussion->content }}
13             </p>
14         </div>
15         <div class="px-6 pt-4 pb-2 flex justify-end">
16             <span class="inline-block bg-blue-500 text-gray-100 rounded-full px-3
17                 py-1 text-sm font-semibold mr-2 mb-2">{{ $discussion->channel->name }}</span>
18         </div>
19     </div>
20 @endforeach

```

### *Code du contenu d'un bloc de discussion*

Blade utilise les crochets pour intégrer les variables php et les @ pour intégrer les structures de contrôle.

Ici on fait un foreach, pour passer en défilé un tableau, sur la variable \$discussions envoyée par le controller précédemment.

La variable correspond au modèle on peut donc y extraire toutes les données de ses colonnes car on a utiliser la méthode 'get' dans son controller.

J'utilise un dossier 'Layouts' où je place des parties de code de la vue qui doivent être présents sur plusieurs pages.



```
1 <head>
2   <meta charset="utf-8">
3   <meta name="viewport" content="width=device-width, initial-scale=1">
4
5   <!-- CSRF Token -->
6   <meta name="csrf-token" content="{{ csrf_token() }}">
7
8   <title>{{ config('app.name', 'Mangatech') }}</title>
9
10  <script src="https://kit.fontawesome.com/efbe1e177e.js" crossorigin="anonymous"
    ></script>
11
12  <!-- Styles -->
13  <link href="{{ mix('css/app.css') }}" rel="stylesheet">
14  <link rel="stylesheet" href=
    "http://cdn.bootcss.com/toastr.js/latest/css/toastr.min.css">
15 </head>
```

*Ici la partie header*

```

1 <header class="bg-blue-900 py-6">
2     <div class=
"container mx-auto flex justify-between items-center px-6 flex-wrap">
3
4         <a href="{{ url('/') }}" class=
"text-lg font-semibold text-gray-100 no-underline inline-flex">
5             {{ config('app.name', 'Mangatech') }}
6         </a>
7
8         <button class=
"text-white inline-flex p-3 hover:bg-gray-900 rounded lg:hidden ml-auto hover:text-
white outline-none nav-toggler"
9
10             data-target="#navigation">
11                 <i class="fas fa-bars"></i>
12             </button>
13             <nav class=
"space-x-4 text-gray-300 text-sm sm:text-base hidden top-navbar w-full lg:inline-fl
ex lg:flex-grow lg:w-auto"
14                 id="navigation">
15                     <div class=
"lg:inline-flex lg:flex-row lg:ml-auto lg:w-auto w-full lg:items-center items-star
t flex flex-col lg:h-auto"
16                         <a class=
"no-underline hover:underline lg:inline-flex lg:w-auto w-full px-3 py-2 rounded tex
t-gray-400 items-center justify-center hover:bg-gray-900 hover:text-white"
17                             href="{{ route('manga.index') }}">{{ __('Mangas')
18                         </a>
19                         <a class=
"no-underline hover:underline lg:inline-flex lg:w-auto w-full px-3 py-2 rounded tex
t-gray-400 items-center justify-center hover:bg-gray-900 hover:text-white"
20                             href="{{ route('anime.index') }}">{{ __('Animes')
21                         </a>
22                         <a class=
"no-underline hover:underline lg:inline-flex lg:w-auto w-full px-3 py-2 rounded tex
t-gray-400 items-center justify-center hover:bg-gray-900 hover:text-white"
23                             href="{{ route('discussions.index') }}">{{ __('
24                             'Forum') }}
25                         </a>
26                         @guest
27                         <a class=
"no-underline hover:underline lg:inline-flex lg:w-auto w-full px-3 py-2 rounded tex
t-gray-400 items-center justify-center hover:bg-gray-900 hover:text-white"
28                             href="{{ route('login') }}">{{ __('Login') }}
29                         </a>
30                         @if (Route::has('register'))
31                         <a class=
"no-underline hover:underline lg:inline-flex lg:w-auto w-full px-3 py-2 rounded tex
t-gray-400 items-center justify-center hover:bg-gray-900 hover:text-white"
32                             href="{{ route('register') }}">{{ __('Register'
33                             ) }}
34                         </a>
35                         @endif
36                         @else
37                         <a href="{{ route('user.dashboard') }}">{{
38                             Auth::user()->name }}</a>
39                         <a href="{{ route('logout') }}"
40                             class=
"no-underline hover:underline lg:inline-flex lg:w-auto w-full px-3 py-2 rounded tex
t-gray-400 items-center justify-center hover:bg-gray-900 hover:text-white"
41                             onclick="event.preventDefault();
42                                 document.getElementById('logout-form')
43                                 .submit();">{{ __('Logout') }}
44                             </a>
45                             <form id="logout-form" action="{{ route('logout') }}"
46                                 method="POST" class="hidden">
47                                 {{ csrf_field() }}
48                             </form>
49                         @endguest
50                     </div>
51                 </nav>
52             </div>
53         </div>
54     </header>

```

*Ici la navbar*

*J'utilise @extends('layouts.app) sur mes autres pages*

*et @section('content') pour intégrer ce layout et écrire le code dans le main .*



```
1 <main>
2         @yield('content')
3     </main>
```

*Ici la partie main du layout*

## **Création de la route**

La dernière étape du développement de la fonctionnalité est de créer la route.

La route est dans le fichier routes/web.php.



```
1 Route::resource('discussions', DiscussionController::class);
```

J'utilise la méthode statique 'ressource' de Laravel qui transforme automatiquement toutes méthodes du CRUD du controller choisi en paramètre en routes.

*Le premier paramètre, ici 'discussions', représente l'url qui va être utilisé pour appeler la route.*

*Le deuxième paramètre représente la class du controller.*

*Si on avait fait une route sans utiliser la ressource, on aurait indiqué l'action du controller et le nom simplifié de la route comme ceci :*



```
1 Route::post('comment-reply/{comment}', [CommentReplyController::class, 'store']  
2   )->name('comment-reply.store');
```

*C'est Laravel, grâce au fichier des 'Façades', qui fournit une interface statique aux classes pour simplifier la syntaxe.*

## Utiliser l'anglais dans son activité professionnelle

Pour arriver à mes fins, j'ai du faire des recherche en anglais notamment pour Laravel qui a une communauté majoritairement anglophone.

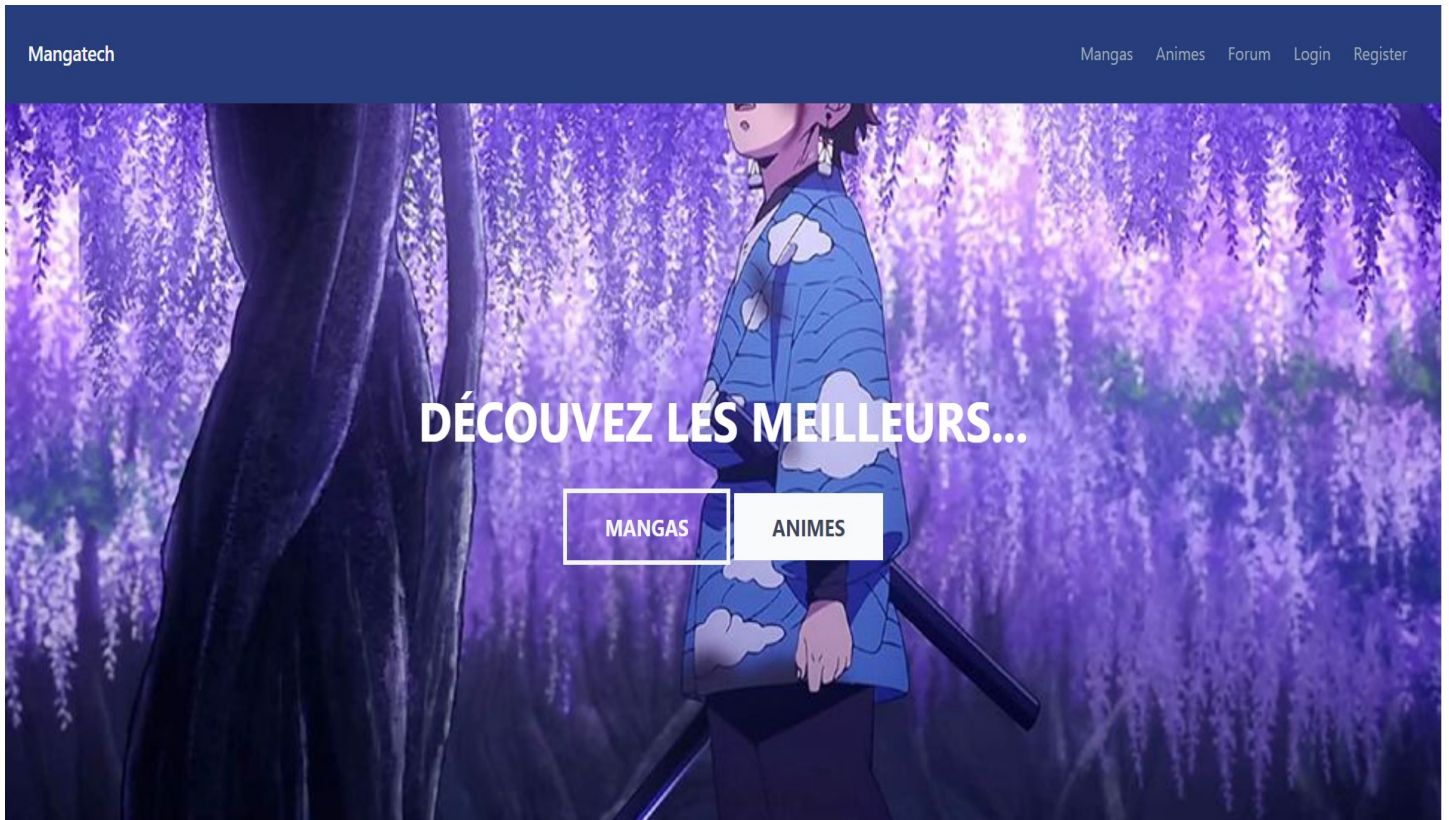
J'ai par exemple été sur le forum de Laracasts.com pour poser mes questions relatives à des problèmes qui m'ont bloqués pendant quelques temps.

The screenshot shows the Laracasts forum interface. On the left is a sidebar with navigation links: 'NEW DISCUSSION' (highlighted in blue), 'All Threads', 'My Questions' (highlighted in blue), 'My Participation', 'My Best Answers', 'Following', 'Popular This Week', 'Popular All Time', 'Solved', and 'Unsolved'. The main area at the top has filters for 'My Questions' and 'All', a search bar with the text 'Whatcha Looking For?', and a hamburger menu icon. Below the filters, three question cards are displayed:

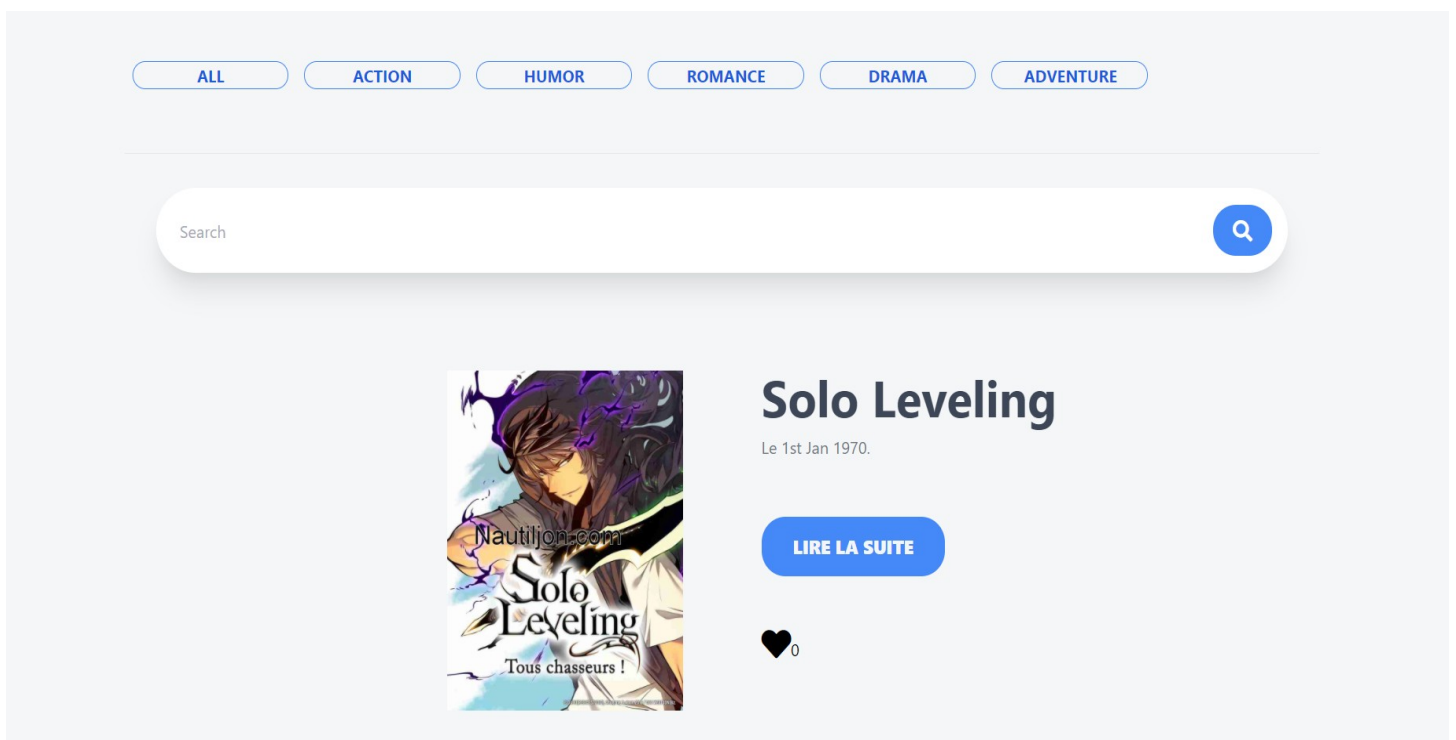
- Login form filled**: 120 views, 2 replies. The user 'JIEL' replied 4 weeks ago. The question text is: 'Hello i used laravel/ui auth method and my problem is that when i click on the login the page with the form is already filled by a data i did put weeks ago. I tried to clear the cache on chrome and see on the view and au...'
- Many controller in one view**: 234 views, 6 replies. The user 'JIEL' replied 4 weeks ago. The question text is: 'Hi ! I would like to put data from differents controller in a view. here are my models class Manga extends Model { use HasFactory; public function readed\_to\_user() { return \$this->belongsToMany(User::class,...'
- Problem storing data to wrong table**: 507 views, 14 replies. The user 'SR57' replied 1 month ago. The question text is: 'Hello, i'm trying to make a system for adding manga to different lists (favorites and planning). My 'favorite' method works but my 'planning' method is doing the same and storing in the favorite table. Here is my cod...'



## Résultat final du site



### Accueil du site





## Page mangas

Mangatech

MangasAnimesForumLoginRegister

ADD DISCUSSION

Debates

Reviews

Recommandation

Other

User

dgfgs

msg content

2021-04-25 15:12:16

Debates

User

fzefzeef

je sais pas

2021-04-25 15:45:21

Debates

User

je sais pas

grgerg

2021-04-25 15:45:47

Debates

## Page Forum