**LARAVEL**

Vidéo 1 :

Lorsque nous avons utilisés **laravel**  nous avons besoin d’installer un serveur qui repend aux exigences suivantes :

* Extension PHP open ssl
* PHP 7.0.0
* Extension PHP PDO
* Extension PHP XML

Vous allez trouver dans Laravel :

* un système de routage (RESTFul et ressources),
* un créateur de requêtes SQL et un ORM,
* un moteur de template,
* un système d’authentification pour les connexions,
* un système de validation,
* un système de pagination,
* un système de migration pour les bases de données,
* un système d’envoi d’emails,
* un système de cache,
* un système de gestion des événements,
* un système d’autorisations,
* une gestion des sessions,
* un système de localisation,
* un système de notifications

Vidéo 2 :

**Laravel** utilise le composer pour gérer ses dépendance. Apres l’installation de composer on peut créer un projet **laravel** avec la commande suivante :

* Composer create – project –prefer- dist laravel/laravel nom\_prjoet

Vidéo 3 :

On utilise la commande strat pour démarrer notre projet **Laravel.** Apres la création du projet on doit vérifier le fichier de  l'environnement **.env**

* Dans ce fichier on peut modifier les paramètres de base de données.
* On peut changer les paramètres de serveur SMTP (Email).

Vidéo 4 :

La commande **php artisan serve** permet de démarrer le serveur.

Vidéo 5 :

Dans projet Laravel crée :

* Un le dossier app/http/  contient:

les modeles : le modèle correspond à une table d'une base de données. C'est une classe qui étend la classe Model qui permet une gestion simple et efficace des manipulations de données et l'établissement automatisé de relations entre tables.

* un dossier app/http/controller

Les controllers : le contrôleur se décline en plusieurs catégories : contrôleur classique, contrôleur RESTfull et contrôleur de ressource.

* un dossier database :

Les migration.

* un dossier public :

Les fichiers javscript, jquery, css et les images

* un dossier resources:

Les vues : la vue est soit un simple fichier avec du code HTML, soit un fichier utilisant le système de template Blade de Laravel.

* un dossier route :

Web.php : contient les routes collection du vues et controllers

Vidéo 6 :

La commande **php artisan make :auth** : permet de créer les vues Login et registre

On trouve deux migrations déjà créé

* Users : Contient les champs de table users
* Migration : contient tous la migration déjà créé
* Chaque migration associe à une table dans la base de données

Vidéo 7 :

Chaque migration contient tous les champs d’une table dans la base de données

Vidéo 8🡺13 :

Pour créer une nouvelle migration :

* **Php artisan make : migration create\_nomtable\_table**

Après la création on peut ajouter les champs des tables dans la migration qui correspond

Dans le fichier database/migration

La méthode up () permet la création d’une nouvelle table

|  |
| --- |
| Public function up ()  {  Schema ::create(‘nom\_table’,function (Blueprint $table)  $table->increments (‘ cle\_primaire ’) ; [\\id\\](file:///\\id\\)  $table ->String (‘champ2’); \\pour les chaines de caractère\\  $table->integer (‘champ3’)->unsignd(); [\\pour](file:///\\pour) les entiers positifs\\  $table->foreing (‘nom\_key’)->references(‘champ’)->on(‘nom\_table’) ; [\\les](file:///\\les) clés étrangers//  $table->timestamps();\\ pour definir le temp de creation\\  }) ; |

La méthode down () permet la suppression de la table.

|  |
| --- |
| Public function down ()  {  Schema::*dropIfExists*(‘nom\_table’); } |

* Pour ajouter la migration a la base de données:

**Php artisan : migrate**

Video 16+17:

Le framework Laravel a quelques exigences système. Toutes ces exigences sont satisfaites par la machine virtuelle Laravel Homestead, il est donc fortement recommandé d'utiliser Homestead comme environnement de développement Laravel local.

Laravel basée sur l’architecture client/serveur (MVC)



***Architecture MVC***

Pour créer un Model

* **Php artisan make:model nom\_model**

Pour chaque migration a un modèle. On va créer dans le model tous les champs de la migration qui correspond

Dans la classe du model :

|  |
| --- |
| Public $fillable=[  ‘Champ1’,  ‘Champ2’,  ‘champn’  ]; |

Apres on va créer les méthodes de liaison de table :

|  |
| --- |
| Public class nom\_table()  {  Return $this->belongto(‘App\Model’);  } |

Les méthodes sont utilisées pour les relations des tables :

* La méthode **belongTo**: définir l'inverse d'une relation hasOne en utilisant cette méthode
* La méthode **OneToMany** : utilisée pour définir les relations où un seul modèle possède une quantité quelconque d'autres modèles.
* La méthode **ManyToMany :** Les relations many-to-many sont définies en écrivant une méthode qui renvoie le résultat de la méthode belongsToMany. Par exemple, définissons la méthode des rôles sur notre modèle utilisateur
* La méthode **HasManyThrough**: un raccourci pratique pour accéder aux relations distantes via une relation intermédiaire.

Vidéo 19+20 :

Pour créer un Controller

* **Php artisan make :controller nom\_contreller –ressources**

Ou bien

* **Php artisan make :controller nom\_controller –ressources --model associer**

Lorsque vous créez un contrôleur vous devez déterminer le nom des méthodes, les créer puis les coder.

On se retrouve avec 7 méthodes correspondant aux tâches classiques :

* **index** : afficher la liste des utilisateurs
* **create** : afficher le formulaire pour la création d’un utilisateur
* **store** : créer un utilisateur
* **show** : afficher l’utilisateur
* **edit** : afficher le formulaire pour la modification d’un utilisateur
* **update** : modifier un utilisateur
* **destroy** : supprimer un utilisateur

Vidéo 21+22 :

Dans le fichier route/web.php on définit le route collection des controllers avec le code suivant

|  |
| --- |
| Route :: get(‘/nom\_viw’,’nom\_controller@nom\_methode’)>name(‘nom\_view’) |

Ou bien Pour générer les routes vers les 7 méthodes du contrôleur de ressource c’est aussi très simple. Entrez ce code dans le fichier **routes/web.php** :

|  |
| --- |
| Route :: resource(‘nom-dossier’,’nom\_controller’) |

Pour afficher un contenu d’une table

|  |
| --- |
| Dans la methode index ()  «$var=nom\_model:: all ()🡺permet de sélectionner le contenu du table  Return view (‘view’, [‘table’->$var]); |

Vidéo 23:

Pour afficher une table dans le view :

|  |
| --- |
| @foreach($var as $var1)  {{$var1->nom\_champ,..}}  @endforeach |

Vidéo 24 :

On peut afficher le contenu d’une table selon ID :

|  |
| --- |
| $var=nom\_model ::where(‘id’,$var->id)->first();  Return view (‘view’,[‘table’->$var]); |

Vidéo 30:

Pour la modification des tables

Dans la méthode edit (): Sélectionner le champ a modifié selon ID

|  |
| --- |
| $var=model ::find($var->id) ;  Return view (‘view.edit’,[‘table->$var’]) ; |

Video 31:

Pour la méthode Update choisir l’élément à modifier

|  |
| --- |
| $var=model ::where(‘id’,$var->id) ;  Return back()->withinput() ; |

Video 32:

Pour afficher un message d’erreur

|  |
| --- |
| @include(‘errors’); |

Errors c’est une méthode qui permet d’afficher un message d’erreur

Pour afficher un message succès

|  |
| --- |
| @include(‘success’); |

Success c’est une méthode qui permet d’afficher un message succès

Vidéo 35 :

Pour la suppression d’un élément dans la base de données

|  |
| --- |
| $var  $var1=model::fint(‘$var->id’);  If($var1->delete())  {  Return redirect()->route(‘view’)->with(‘succss’,’message’);  }  Return back()->withinput()->with(‘errors’,’message’); |

Dans le vue qui correspond on doit supprimer la forme :

|  |
| --- |
| @if(var)  {  Event.preventDefault();  Document.getElementByID(‘delete-form’).submit();  }  @endif |

Vidéo 38 :

Pour créer un élément

* On doit remplir les formulaires d’ajout
* Dans la méthode store () du Controller on va faire une vérification pour les champs avec le code suivant :

|  |
| --- |
| Public functoin create(Request $request)  {  $var=model::create([  ‘name’=>request->input(‘name’),  ‘description’=>request->input(‘description’),  User\_id=>Auth::user()->id;  ]);  If($var)  {  Return redirect()->route(‘view’,[‘table’=>$var->id])-> with(‘success’,’message’);  }  Return bach()->withinput->with(‘errors’,’message’);  } |