Développer une application qui exploite une API REST

Mes Films Préférés

Votre mission, que vous ne pouvez pas refuser

En tant que développeur confirmé travaillant chez l'ESN TopDev, Vous êtes mandaté par un client pour développer une application qui permet à des amis de **partager leurs films préférés et de donner leur avis sur ces films**.

Exigences techniques



- Vous développerez une **application detype client léger** en PHP 12 en vous basant sur le framework **Laravel** et le SGBDR MySQL.
- Vous exploiterez les données de l'**API REST** du site **TMDB**.

https://www.themoviedb.org/?language=fr

Exigences fonctionnelles

- Les utilisateurs auront la possibilité de **créer un compte** pour s'enregistrer.
- Les **utilisateurs** devront **se connecter** afin de pouvoir utiliser le site.
- Les utilisateurs pourront **rechercher des films** dans la base de données de TMDB.
- Les utilisateurs pourront marquer leurs films **favoris**.
- Les utilisateurs pourront donner un **avis** sur chacun de leurs films favoris.
- Les utilisateurs pourront voir les autres utilisateurs du site et les marquer comme **amis** (ou pas)
- Les utilisateurs pourront **partager** leurs favoris avec leurs amis (partage individuel pour chaque ami).
- Les utilisateurs pourront **voir quels films ont été partagés** avec eux par leurs amis.

Elements techniques pour débuter votre projet

Créez un projet Laravel avec composer

```
composer create-project laravel/laravel mesfilmspreferes
```

Exécutez cette commande à l'emplacement où vous voulez créer votre projet !

Après création du projet

Installez la **debug bar** et l'**IDE helper**.

```
composer require barryvdh/laravel-debugbar --dev
composer require --dev barryvdh/laravel-ide-helper
```

Après création du projet

Dans le fichier **.env**

Définissez les paramètres de votre **application**.

```
APP_NAME=mesfilmspreferes
APP_ENV=local
APP_KEY=créée automatiquement
APP_DEBUG=true
APP_URL=http://mesfilmspreferes.fr
```

Après création du projet

Dans le fichier **.env**

Définissez les paramètres de la base de données.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=mesfilmspreferes
DB_USERNAME=root
DB_PASSWORD=
```

Après création du projet

Dans le fichier **config/app.php**

Définissez l'**Application Name**

```
'name' => env('APP_NAME', 'Mes films preferes'),
```

Définissez l'environnement d'application sur '**development**'

```
'env' => env('APP_ENV', 'development'),
```

Mettez le debug à true

```
'debug' => (bool) env('APP_DEBUG', true),
```

Définissez l'URL de l'application

```
 'url' => env('APP_URL',
'http://mesfilmspreferes.fr'),
```

Après création du projet

Créez un VHOST sur votre serveur Apache contenat les lignes suivnates :

```
<VirtualHost *:80>
	ServerName mesfilmspreferes.fr
	DocumentRoot "c:/wamp64/www/mesfilmspreferes/public"
	<Directory "c:/wamp64/www/mesfilmspreferes/public/">
		Options +Indexes +Includes +FollowSymLinks +MultiViews
		AllowOverride All
		Require local
	</Directory>
</VirtualHost>
```

Mettre à jour le fichier "hosts"

Ajoutez ces lignes dans "C:/windows/system32/driver/etc/hosts"

```
127.0.0.1    mesfilmspreferes.fr
::1 mesfilmspreferes.fr
```

Dans votre IDE VSCode

Installez les extensions suivantes :

- **GitHub Pull Request**
  - Pour sauvegarder et partager votre code sur GitHub
- **MYSQL database client**
  - Pour travailler confortablement avec la base de données
- **Laravel officiel**
- **Laravel Blade formatter**
  - Pour travailler confortablement avec Blade
- **PHP Intelephense**
  - Pour vous aider à repérer et corriger les erreurs de syntaxe PHP



Conception

Configuration des données

- Nous n'aurons **pas besoin de créer de modèles**. Laravel va le faire pour nous à partir de la base de données existante.
- Pour cela, nous devons installer le générateur de modèle **Reliese Laravel Model Generator**.

```
composer require reliese/laravel --dev
```

```
php artisan vendor:publish --tag=reliese-models
```

```
php artisan config:clear
```

Configuration des données

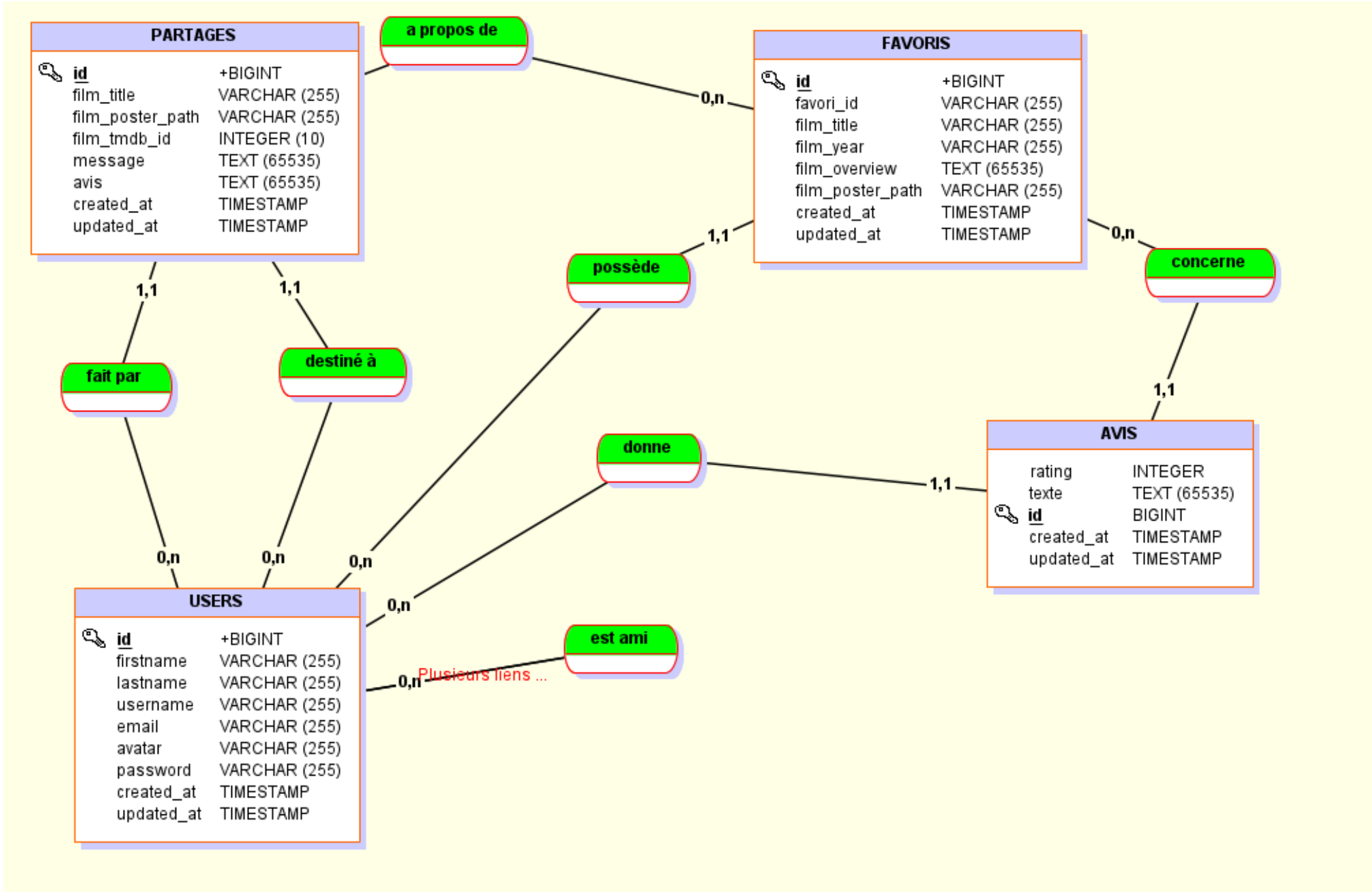
I. Importez la base de données dans MySQL à partir du fichier .sql fourni.

II. Tapez la commande suivante :

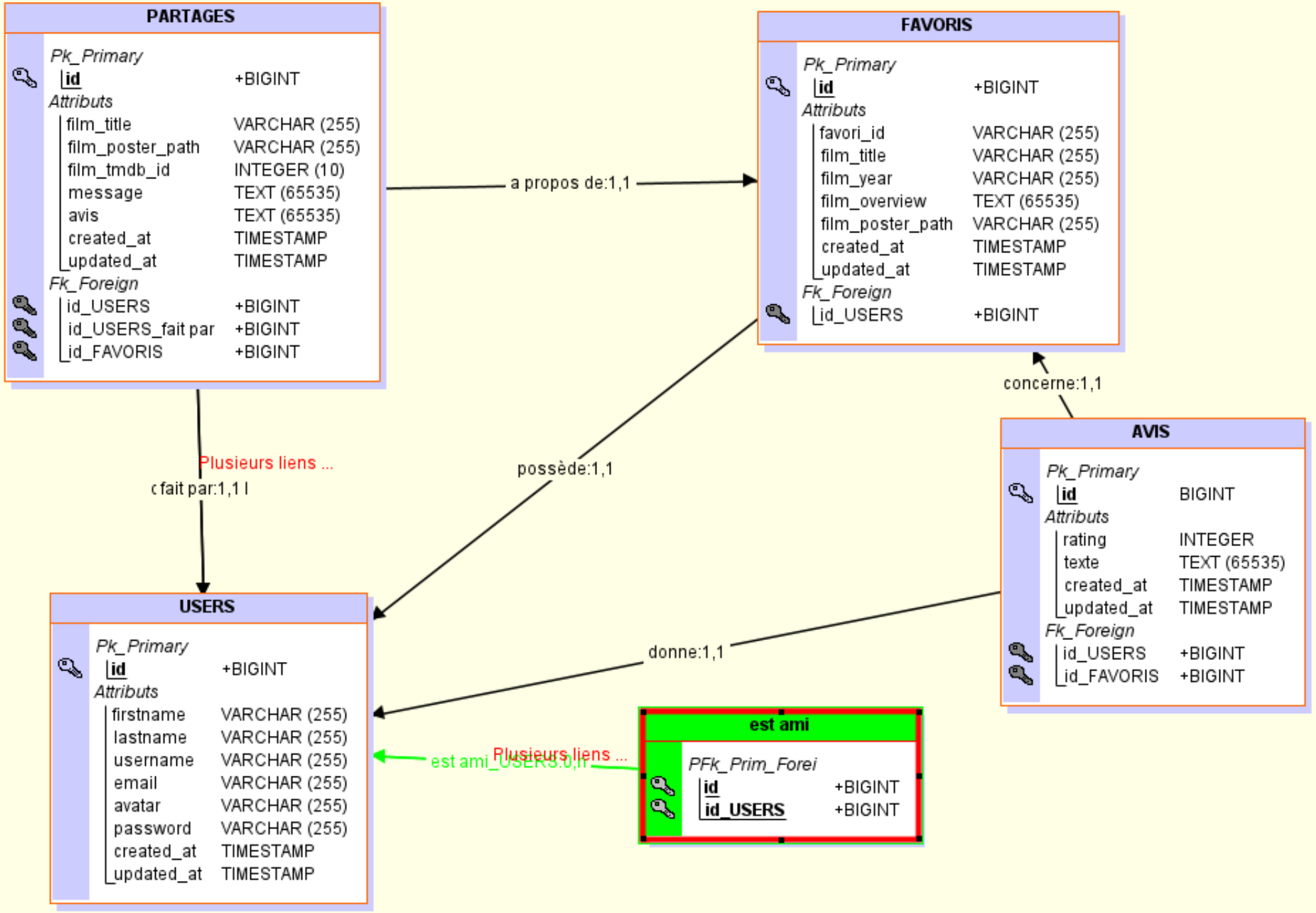
```
php artisan code:models
```

III. Vérifiez le modèle. Les jointures doivent avoir été créées automatiquement. Magique !

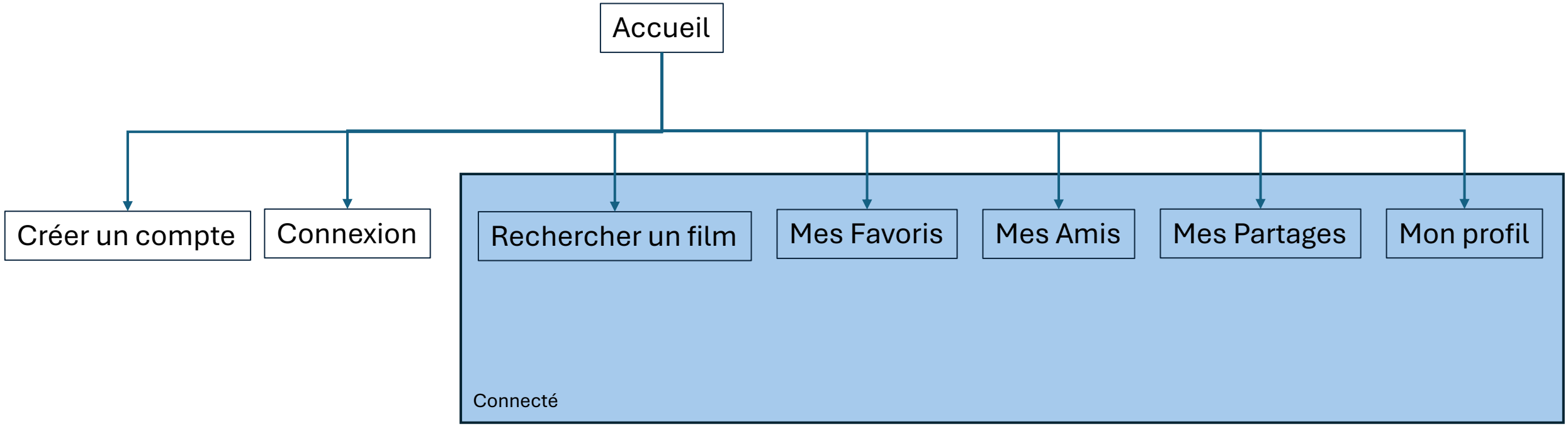
MCD


PARTAGES
id +BIGINT
film_title VARCHAR (255)
film_poster_path VARCHAR (255)
film_tmdb_id INTEGER (10)
message TEXT (65535)
avis TEXT (65535)
created_at TIMESTAMP
updated_at TIMESTAMP
a propos de
0,n
FAVORIS
id +BIGINT
favori_id VARCHAR (255)
film_title VARCHAR (255)
film_year VARCHAR (255)
film_overview TEXT (65535)
film_poster_path VARCHAR (255)
created_at TIMESTAMP
updated_at TIMESTAMP
1,1
possède
0,n
concerne
1,1
1,1
fait par
destiné à
1,1
AVIS
rating INTEGER
texte TEXT (65535)
id BIGINT
created_at TIMESTAMP
updated_at TIMESTAMP
donne
1,1
0,n
0,n
0,n
0,n
USERS
id +BIGINT
firstname VARCHAR (255)
lastname VARCHAR (255)
username VARCHAR (255)
email VARCHAR (255)
avatar VARCHAR (255)
password VARCHAR (255)
created_at TIMESTAMP
updated_at TIMESTAMP
est ami
0,n
Plusieurs liens ...

MLD


PARTAGES
Pk_Primary
id +BIGINT
Attributs
film_title VARCHAR (255)
film_poster_path VARCHAR (255)
film_tmdb_id INTEGER (10)
message TEXT (65535)
avis TEXT (65535)
created_at TIMESTAMP
updated_at TIMESTAMP
Fk_Foreign
id_USERS +BIGINT
id_USERS_fait par +BIGINT
id_FAVORIS +BIGINT
FAVORIS
Pk_Primary
id +BIGINT
Attributs
favori_id VARCHAR (255)
film_title VARCHAR (255)
film_year VARCHAR (255)
film_overview TEXT (65535)
film_poster_path VARCHAR (255)
created_at TIMESTAMP
updated_at TIMESTAMP
Fk_Foreign
id_USERS +BIGINT
AVIS
Pk_Primary
id BIGINT
Attributs
rating INTEGER
texte TEXT (65535)
created_at TIMESTAMP
updated_at TIMESTAMP
Fk_Foreign
id_USERS +BIGINT
id_FAVORIS +BIGINT
USERS
Pk_Primary
id +BIGINT
Attributs
firstname VARCHAR (255)
lastname VARCHAR (255)
username VARCHAR (255)
email VARCHAR (255)
avatar VARCHAR (255)
password VARCHAR (255)
created_at TIMESTAMP
updated_at TIMESTAMP
est ami
PFk_Prim_Forei
id +BIGINT
id_USERS +BIGINT
a propos de:1,1
concerne:1,1
Plusieurs liens ...
fait par:1,1
possède:1,1
donne:1,1
Plusieurs liens ...
est ami_USERS:0,n

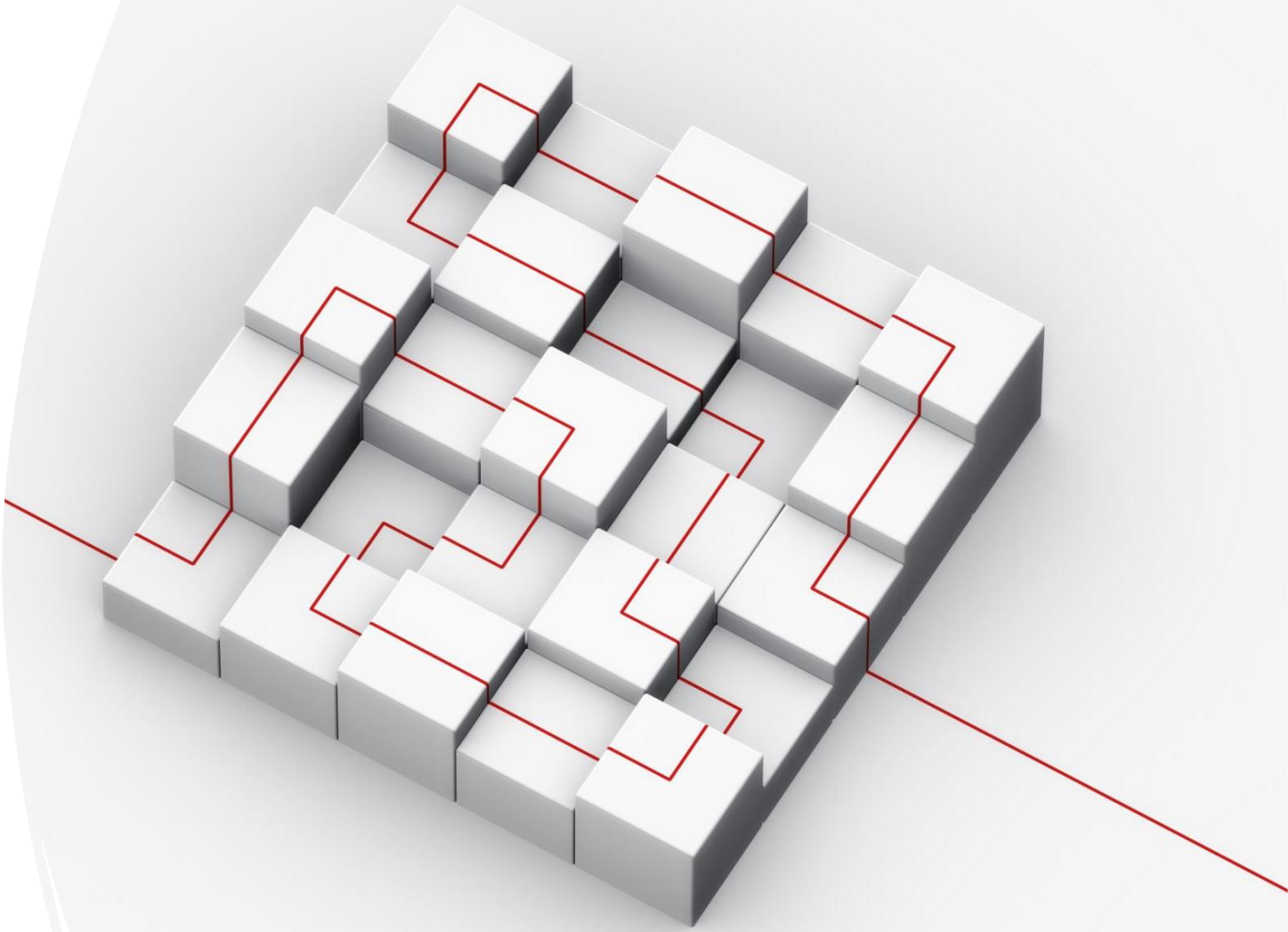
Plan du site


Accueil
Créer un compte
Connexion
Rechercher un film
Mes Favoris
Mes Amis
Mes Partages
Mon profil
Connecté

Conception

Dessinez les **écrans** pour chaque page du site selon vos gouts.

Restez **schématique**. Nous ne sommes pas une formation de webdesign.



Eléments techniques pour développer votre projet

Processus de développement

Travaillez (globalement) dans cet ordre !

I. Les routes

II. Les contrôleurs

III. Les vues

IV. Les appels à l'API REST

V. Les styles

En sachant que vous devrez faire de nombreux allers-retours.

Documentation

Vos sources de documentation :

- La doc officielle
  https://laravel.com/docs/12.x/
- Les IA (On va testez ensemble Copilot dans VSCode, c'est top !)
- Google
- Vos camarades
- Le prof

Avertissement !

Des informations utiles et des astuces sont données dans le reste de ce document, mais vous devrez souvent les compléter par **des rechercher personnelles** pour les mettre en œuvre.

Les routes

Les routes sont dans le fichier **routes/web.php**

Rangez vos routes par catégories.

- Routes **publiques** (accessibles sans connexion)
- Routes pour la gestion des **films favoris**
- Routes pour gérer les **avis**
- Routes pour gérer les **amis**
- Routes pour gérer les **partages**
- ...

Les routes

Rappels

- Toutes les routes doivent renvoyer vers un **contrôleur**.
- Un contrôleur peut traiter plusieurs routes.
  Par exemple, la gestion des avis (**afficher**, **ajouter**, **modifier**, **supprimer**) peut (et doit) se faire dans le même contrôleur (ça fait **4 routes**).

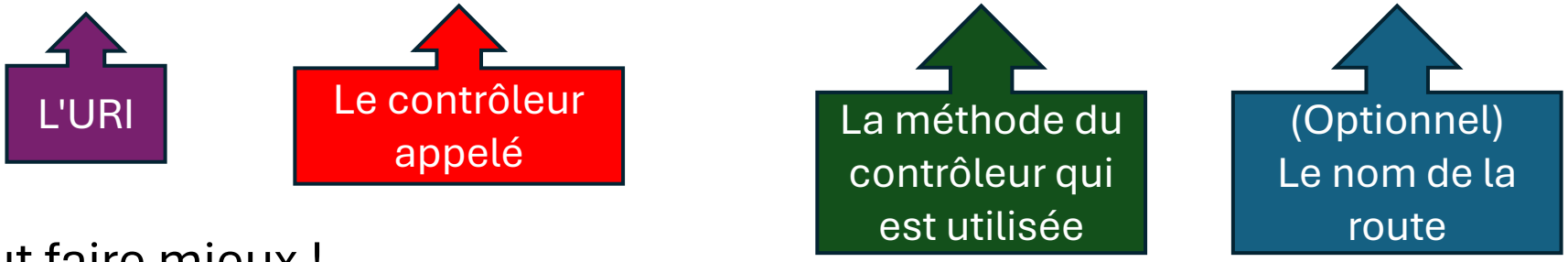
Quelles méthodes HTTP utiliser pour les routes ?

- Une route qui affiche une page web -> **GET**
- Une route qui traite les données d'un formulaire -> **POST**

Les routes

Vous pouvez écrire les routes selon le modèle suivant :

```
//Route qui affiche la page d'accueil
Route::get('/', [HomeController::class, 'index'])->name('home');
```



Mais on peut faire mieux !

Les routes

Groupez vos routes par contrôleur !

Exemple : toutes ces routes font appel au **même contrôleur**. Observez la syntaxe du code.

```
//Syntaxe suggérée pour les routes "favoris"
    Route::controller(FavorisController::class)->group(function () {
        Route::get('/favoris','index')->name('favoris');
        Route::post('/favoris/ajouter','store')->name('favoris.add');
        Route::post('/favoris/{favori}','destroy')->name('favoris.destroy');
        Route::post('/favoris/{favori}/avis','updateAvis')->name('favoris.updateAvis');
    });
```

Les routes

Nommez vos routes !

Ca simplifiera plus tard la navigation dans les vues.

Exemple : toutes ces routes ont été nommées grâce à la méthode name().

```
//Routes qui gèrent les favoris
    Route::controller(FavorisController::class)->group(function () {
        Route::get('/favoris','index')->name('favoris');
        Route::post('/favoris/ajouter','store')->name('favoris.add');
        Route::post('/favoris/{favori}','destroy')->name('favoris.destroy');
        Route::post('/favoris/{favori}/avis','updateAvis')->name('favoris.updateAvis');
    });
```

A vous de jouer !

Créez vos routes

Les contrôleurs

Créez les contrôleurs avec artisan

```
php artisan make:controller UserController
```

A vous de jouer !

Créez vos contrôleurs (sans contenu pour l'instant)

L'authentification

Laravel dispose de fonctionnalités "maison" qui simplifient l'authentification des utilisateurs. Utilisez-les plutôt que de développer une authentification classique car vous gagnerez du temps et de la sécurité.

Vous allez utiliser :

- La classe **Request**
- La classe **Auth**

Ces fonctionnalités sont détaillées dans les diapos suivantes.

Validation des données utilisateur

La classe **Request** sert à gérer les requête HTTP (POST, GET).

Vous pouvez l'exploiter facilement grâce à ses nombreuses méthodes :

`$request->input('nom)` pour obtenir la valeur du champ 'nom'

`$request->session()` pour accéder à la session

`$request->validate([])` pour valider les données (très pratique !)

Validation des données utilisateur

Comment se sert-on de la classe **Request** ?

On la passe en paramètre de la méthode de contrôleur qui traite la requête HTTP.

Exemple : Admettons que le formulaire de connexion de votre site renvoie la requête vers la route POST qui s'appelle "**loginPost**".

```
Route::controller(LoginController::class)->group(function () {
    Route::get('/login', 'index')->name('login');
    Route::post('/login', 'login')->name('loginPost');
    Route::post('/logout', 'logout')->name('logout');
});
```

Validation des données utilisateur

Dans le contrôleur, les données de la requête (donc les variables de la superglobale POST) sont passées en paramètre de la méthode grâce à la classe **Request**.

```
//La méthode du contrôleur qui traite les données de connexion
public function login(Request $request) {
...
```

Validation des donnée utilisateur

Dans la méthode du contrôleur, les données de la requête peuvent être validées grâce à la méthode **validate**()

```
public function login(Request $request) {

      $credentials = $request->validate([
            'email' => ['required', 'email'],
            'password' => ['required'],
        ]);
...
```

Documentation sur la validation des données utilisateur:

https://laravel.com/docs/12.x/validation

Exemple d'utilisation de la classe Auth

Une fois que les données utilisateur sont validées, on peut passer à **l'authentification**. C'est ici qu'entre en jeu la classe **Auth**.

Elle dispose de plein de méthodes très utiles.

`Auth::login($user)` : connecter un utilisateur

`Auth::logout()` : déconnecter un utilisateur

`Auth::check()` : vérifie si l'utilisateur est connecté

`Auth::attempt($credentials)` : vérifie les identifiants de l'utilisateur

Authentification d'un utilisateur avec la classe Auth

C'est très simple :

```
if (Auth::attempt($credentials)) { //si les données d'authentification sont OK
  $request->session()->regenerate(); //pour éviter le vol du cookie de session
  return redirect()->intended('/')->with('success','Connexion réussie !');
}
```

Après l'authentification, les données utilisateur sont sauvegardées dans un objet de la classe **Auth**.

On peut les récupérer comme ça :

```
$user = Auth::user(); //on récupère les données de l'utilisateur
$userId = Auth::id(); //on récupère l'identifiant de l'utilisateur
```

Exemple complet de contrôleur de connexion utilisateur

```
public function login(Request $request) {
    $credentials = $request->validate([
        'email' => ['required', 'email'],
        'password' => ['required'],
   ]);
    if (Auth::attempt($credentials)) {
        $request->session()->regenerate();
        return redirect()->intended('/')->with('success', 'Connexion réussie !');
    }
    return back()->withErrors([
        'email' => 'Les informations de connexion sont incorrectes.',
    ])->onlyInput('email');
}
```

Exemple de méthode de déconnexion d'un contrôleur

```
public function logout(Request $request) {
    Auth::logout();

    // sécuriser la session
    $request->session()->invalidate();
    $request->session()->regenerateToken();

    return redirect('/')->with('success', 'Vous avez été déconnecté.');
}
```

A vous de jouer !

Créez la méthode "login"

Exemple de méthode de contrôleur pour la création d'un compte utilisateur (1/3)

```
public function createUser(Request $request){

    // Validation des données du formulaire
    $request->validate([
        'firstname' => 'required|string|max:255',
        'lastname' => 'required|string|max:255',
        'username' => 'required|string|max:255',
        'email' =>
'required|string|email|max:255|unique:users',
        'password' =>
'required|string|min:8|confirmed'
    ]);
…
```

Exemple de méthode de contrôleur pour la création d'un compte utilisateur (2/3)

```
...
// Création de l'utilisateur avec Eloquent
  $user = User::create([
    'firstname' => $request->firstname,
    'lastname' => $request->lastname,
    'username' => $request->username,
    'email' => $request->email,
    'password' => bcrypt($request-
>password),
  ]);
    $user->save();
...
```

Exemple de méthode de contrôleur pour la création d'un compte utilisateur (3/3)

```
...
    // Connexion automatique de l'utilisateur
    // après création du compte
    Auth::login($user);


    // Redirection vers la page d'accueil
    return redirect('/')->with('success',
'Compte créé avec succès !');
}
```

A vous de jouer !

Créez la méthode "nouveau compte"

Les sessions dans Laravel

A la base, Laravel gère les sessions comme en PHP classique :

Lorsqu'un utilisateur visite le site :

I. Laravel génère un **identifiant de session** unique.
II. Cet identifiant est stocké dans un **cookie** (laravel_session).
III. Les données de session sont stockées côté serveur.
IV. À chaque requête, Laravel lit le cookie et récupère les données associées.

Sécurité des sessions

Mais, en plus, Laravel protège les sessions avec :

- **Cryptage** des cookies,
- **Protection CSRF** (via le token **@csrf** dans les formulaires),
- **Expiration automatique** (configurable le fichier de config **session.php** avec le paramètre "**lifetime**"),
- **Regénération de l'ID de session** après connexion pour éviter le vol de session.

Gestion des données de session

Tout se fait avec la méthode session()

```
// Stocker une donnée
session(['key' => 'value']);

// Récupérer une donnée
$value = session('key');

// Supprimer une donnée
session()->forget('key');

// Vider toute la session
session()->flush();
```

Les vues

Laravel propose des fonctionnalités qui permettent de simplifier la création et la gestion des vues.

- Les **templates**
- Les **directives**
- Les **expressions**

Pour mettre en œuvre ces fonctionnalités, vous devez identifier les fichiers de vues comme étant des fichiers "**blade**".

Nommez vos fichiers de vues avec ".blade" (exemple : profil.**blade**.php)

Quelques exemples d'utilisation sont donnés dans les dispos suivantes.

Quelques @directives Blade

Exemple de directive Blade

@guest

//Ce code ne s'affiche **que pour les utilisateurs non connectés**

@endguest

@auth

//ce code ne s'affiche que pour les **utilisateurs connectés**

@endauth

Exemple de directive Blade

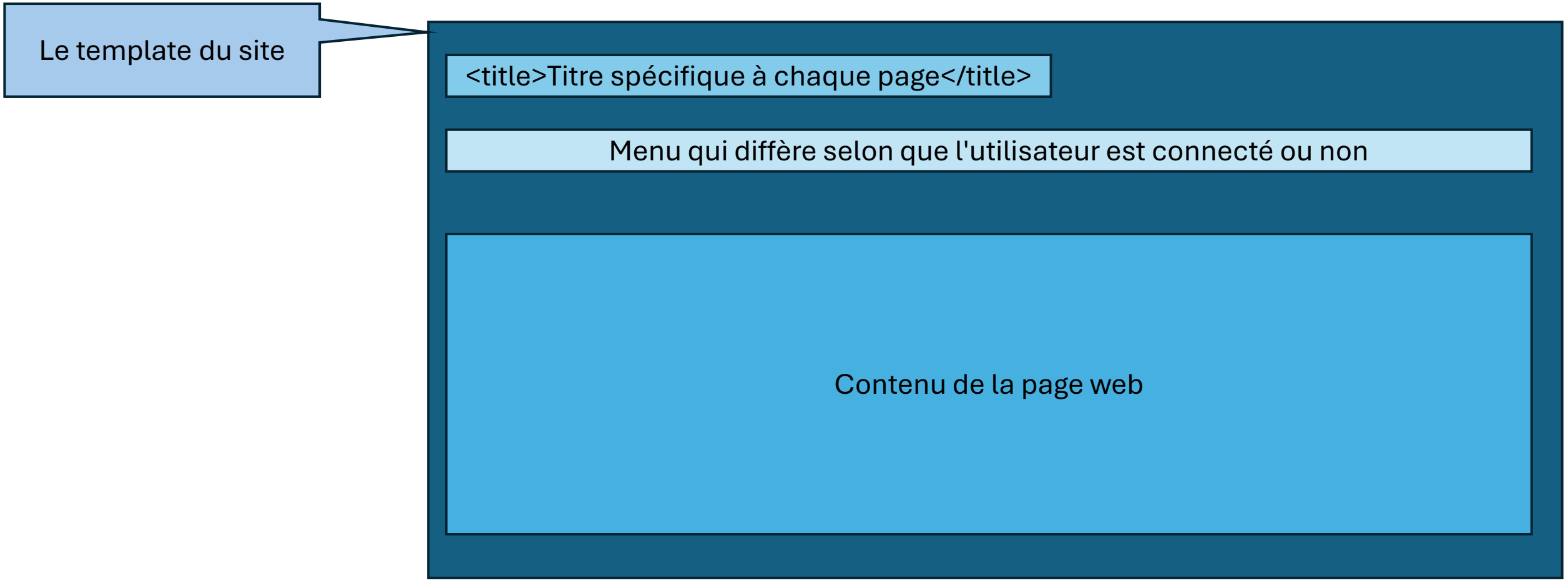
@extends('templates.app')

//Ce code s'affiche à l'intérieur d'une autre page qui s'appelle "template" (et qui est d'ailleurs un template).

@yield('content')

// A cet endroit du code, on affiche la section qui s'appelle 'content'

Exemple d'utilisation d'un template



Un template de page web avec Blade

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>@yield('title', 'Home')</title>
        <link rel="stylesheet" href="/css/app.css">
    </head>
    <body>
        @include('templates.menu')
        @yield('content')
    </body>
</html>
```

Une page web liée à un template Blade

```
//Dans la page de contenu on indique qu'on utilise le template
@extends('templates.app')

//on donne une valeur à la variable 'titre'
@section('title', 'Mes amis')

//on définit le contenu d'une section de page
@section('content')
    <main>
    //ici le contenu de la page web
    </main>
@endsection
```

Exemple de page web liée à un template

```
@extends('templates.app')
@section('title', 'Connexion à mon compte')
@section('content')

//ici le contenu spécifique de la page web

@endsection
```

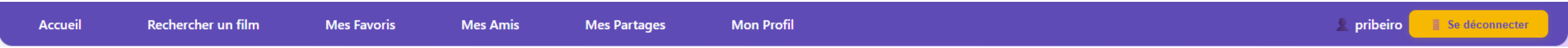
Exemple de menu

Le code des diapos suivantes permet d'obtenir ceci (avec la feuille de styles en plus):

- Pour un utilisateur non connecté



- Pour un utilisateur connecté



Exemple de menu (1/3)

```
<nav class="nav">
    <ul class="nav__list">
        <li class="nav__item"><a href="/" class="nav__link">Accueil</a></li>
       <!--cette partie ne s'affiche que pour les utilisateurs connectés-->
        @auth
            <li class="nav__item"><a href="/films/rechercher" class="nav__link">Rechercher un film</a></li>
            <li class="nav__item"><a href="/favoris" class="nav__link">Mes Favoris</a></li>
            <li class="nav__item"><a href="/amis" class="nav__link">Mes Amis</a></li>
            <li class="nav__item"><a href="/partages" class="nav__link">Mes Partages</a></li>
            <li class="nav__item"><a href="/profil" class="nav__link">Mon Profil</a></li>
        @endauth
    </ul>
  …
```

Exemple de menu (2/3)

```
...
    @auth
        <div class="nav__user">
            <!--Affichage du nom de l'utilisateur connecté-->
            <p class="nav__username">👤 {{ Auth::user()->username }}</p>
            <!--Vers la route de déconnexion-->
            <form method="POST" action="{{ route('logout') }}" class="nav__logout">
                @csrf
                <button type="submit" class="btn btn--logout">Déconnexion</button>
            </form>
        </div>
    @endauth
...
```

Exemple de menu (3/3)

```
...
<!-- Cette partie ne s'affiche que pour les utilisateurs non connectés-->
    @guest
        <div class="nav__user">
            <p class="nav__username">👋 Bonjour !</p>
            <!-- Lien vers la page de connexion-->
            <a href="{{ route('login') }}" class="btn btn--login">🔐
Connexion</a>
        </div>
    @endguest
</nav>
```

A vous de jouer !

Créez la vue "page d'accueil"

Insérer du code PHP

```
@php
  //Pour ajouter un ami à la liste des amis.
  $isFriend = false;
  foreach ($friends as $friend) {
     if ($friend->friend_id == $otherUser->id && $friend->user_id ==
Auth::id()) {
        $isFriend = true;
        break;
      }
  }
@endphp
```

Les conditions

```
@if (session('success'))
    //ici votre code
@endif
```

Les boucles

```
@foreach ($users as $otherUser)
        //ici votre code
@endforeach
```

Quelques {{expressions}} Blade

Les liens hypertexte avec des expressions Blade

Deux façons de gérer les URL :

//en se basant sur la méthode url()

{{ url('/nouveauCompte/creer') }}

//en se basant sur les noms de routes

{{ route('amis.remove') }}

Récupérer les données de session avec une expression Blade

- Récupérer le prénom stocké dans la session.

@auth**{{ Auth::user()->firstname }}**@endauth

- En plus il ne s'affiche que si l'utilisateur est connecté

@auth{{ Auth::user()->firstname }}**@endauth**

Autoremplissage des champs de formulaire

Très pratique pour récupérer l'ancienne valeur du champ "username".

```
value="{{ old('username') }}"
```

Interactions avec une API REST

Appel à l'API REST

Comment appeler l'API REST de TMDB pour récupérer les données des films ?

I. Avec la méthode **`file_get_content()`** qui **récupère les données** envoyées par l'API dans une variable

II. Avec la méthode **`json_decode()`** qui **transforme les données json** en tableau associatif.

Exemple d'appel à l'API REST (1/2)

```
@php
    $results = null;
    $error = null;
    if(request('query')) { //si la requete existe
        try {
            $apiKey = '63905b28b94957ba2d061a85b849243f';//clé obligatoire pour TMDB
            $query = urlencode(request('query'));
            //On forme l'URL de la requête à l'API
            $url =
"https://api.themoviedb.org/3/search/movie?query={$query}&api_key={$apiKey}";
            $response = file_get_contents($url);//on récupère les données
...
```

Appel à l'API REST avec file_get_contents() (2/2)

```
...
    if($response === false) throw new Exception('Erreur lors de la requête
    API');
    //on met les données dans un tableau associatif qu'on peut exploiter
    $data = json_decode($response, true);
        if(isset($data['results'])) {
              $results = $data['results'];
         } else {
              $error = 'Aucun film trouvé.';
         }
    } catch(Exception $e) {
          $error = $e->getMessage();
    }
}
@endphp
```

A vous de jouer !

Créez la vue "recherche de film"

Les styles CSS dans Laravel

Les styles

Deux approches sont possibles :

I. Créer une feuille de styles .css de façon classique

II. Utiliser un framework css du type **Tailwind CSS** ou **Bootstrap**, mais il faut apprendre à les utiliser.

Ma proposition :

Demandez à une IA (par exemple Copilot) de vous concocter une feuille de styles aux petits oignons.