**Laravel**

**الوظيفة**

**1**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
|  |  |
|  |  |

**ملاحظة:**

**المقدمة**

**1**

1. **Laravel the php framework**
2. **بواسطة لارفيل يمكن إنشاء تطبيقات و صفحات ويب**
3. **بواسطة لارفيل يمكن إنشاء تطبيقات هواتف ذكية أو تطبيق ويب من صفحة واحدة و ذلك من خلال اسخدام api**

**المتطلبات**

**2**

1. **تثبيت visual studio code**
2. **تثبيت فيه الأضافات التالية material theme icons / laravel blade snippets / laravel extension pack**
3. **تثبيت XAMPP**
4. **تثبيت Git**

**التثبيت**

**3**

1. **نثبت أولا composer**
2. **ثم نتأكد انه تم تثبيته من خلال كتابة composer في git bash**
3. **ثم نقوم بتحميل اللارفيل في ملف إختياري من خلال كتابة الأمر**
4. **composer create-project laravel/laravel example-app**
5. **laravel new example-app**
6. **ثم ندخل إلى المشروع من خلال cd app**
7. **ثم نقوم يتشغيل السرفير من خلال كتابة الأمر**
8. **php artisan serve**
9. **ثم نفتح المتصفح وندخل إلى الرابط من أجل فتح المشروع**
10. [**http://127.0.0.1:8000**](http://127.0.0.1:8000)
11. **الأن نقوم بتجهيز قاعدة البيانات من خلال تشغيل xampp وتشغيل الأباتشي و المايسكيول**
12. **ندخل على php myadmin في xampp ونقوم بإنشاء قاعدة بيانات جديدة باسم التطبيق**
13. **ثم نكبس على الخادم و ثم على حسابات المستخدمين ونقوم بإنشاء مستخدم جديد من خلال الضغط على add user account**
14. **ثم ندخل أسم المستخدم و ال host يكون محلي (localhost) وندخل كلمة السر نفس أسم المستخدم مرتين ثم نعمل شك للكل الخيارات ثم نضغط أنطلق**
15. **ثم نضغط على كبسة قاعدة البيانات يلي حد global ثم نحدد أسم قاعدة البيانات التي سوف نتحكم فيها ثم أنطلق ثم نعمل تشك للكل ثم أنطلق وهكذا تم أنشاء قاعدة بيانات باسم التطبيق و أنشاء أسم مستخدم وكلمة مرور وحددنا المستخدم الذي سوف يتحكم بالقاعدة البيانات الأن ندخل إلى المشروع ثم إلى .env ثم نغيير DB\_DATABASE / DB\_USERNAME/ DB\_PASSWORD**
16. **علما أنه ملف .env يتحكم بإعدادات المشروع لهذا غيرنا إعدادات mysql منه**
17. **ثم ندخل إلى المشروع عن طريق git bash ونضغط الأمر التالي**
18. **php artisan migrate**
19. **نحدث قاعدة البيانات التي باسم التطبيق لنشاهد وجود جداول جديدة**
20. **داخل ملف .envيوجد متغيير أسمه APP\_DEBUG=true يقوم بشرح الأخطاء ومكانها عند تطوير المشروع لذلك نقوم بإقافه قبل رفع المشروع ونحوله لل false**

**التعرف على الملفات المشروع**

**4**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| يحتوي على أغلب مكونات التطبيق مثل controllers & model   1. console    1. kernel : يحتوي على وحدة التحكم في جميع الأوامر المخصصة لل php artisan ويمكن إضافة عليها في الكوميدلاين من خلال الأمر php artisan make:command يمكنك الأطلاع أكثر بالبحث عن Task Scheduling 2. exceptions    1. handler : يحتوي دليل الاستثناءات على معالج استثناءات التطبيق الخاص بك وهو أيضًا مكان جيد لوضع أي استثناءات يتم طرحها بواسطة التطبيق الخاص بك و ذلك من خلال تعديل على فئة معالج في handler 3. http    1. controllers       1. controller.php :يحتوي على ملفات التحكم الخاصة بالمشروع والتي تأتيها الروابط من route    2. middleware : يحتوي على ملفات لحماية ال route فهو يقع قبله بجدار الحمايه    3. kernel.php 4. models : وهو الوسيط بين controller و قاعدة البيانات database فهو يقوم بإنشاء و تتعديل وتحديث وحذف المعلومات في قاعدة البيانات 5. providers : وهو مزود الخدمة للمشروع حيث يتم ربطة مع حاوية الخدمات مثل event & route | **app** |
| وهو الذي يقوم بتشغيل التطبيق في ذاكرة التخزين المؤقتة لتحسين أداء التطبيق ولا تحتاج إلى التعديل عليه مع المستقبل | **bootstrap** |
| يحتوي على ملف أعدادات التطبيق مثل إعدادات الإتصال و معلومات قاعدة البيانات و خادم الإتصال بإضافة إلى العديد من قيم التكوين الأساسية مثل المنطقة الزمنية لتطبيق و مفتاح التشفير  ويتم التحكم بهذه الملفات من خلال مكتبة DotEnv التي توفر لنا ملف .env الذي يحتوي على متغييرات تتحكم بملفات ال config | **confing** |
| يحتوي على قاعدة البيانات و على عمليات الترحيل في قاعدة البيانات ومصانع النماذج و الجداول والبيانات | **database** |
| وهو نقطة الدخول الأولى عند طلب الركويست فيقوم بفحص للبحث عن الصفحة التي يريد عرضها فأذا وجدها قام بتحميلها مباشرة أما إذا لم يجدها يقوم بالتحويل إلى ال route وتم إنشائه لتحميل الصفحات الستاتيكية مباشرة دون تشغيل معالجات المشروع وتكون الصفحات التي بداخله لا تحتوي على أمور تحتاج لقاعدة بيانات أو مستخدم مسجل دخوله | **public** |
| يحتوي على ملفات views & css & js & lang وهذه الملفات تكون مرتبطة مع المودل و قاعدة البيانات | **resources** |
| يحتوي على مسارات التطبيق والتي تأتي على شكل ركويست ويأتي قبله middleware و public   1. api : يحتوي الملف على التوجيهات التي يقوم RouteServiceProvider من middleware group التي تهدف هذه المسارات إلى أن تكون عديمة الحالة ، لذا فإن الطلبات التي تدخل التطبيق من خلال هذه المسارات تهدف إلى المصادقة عليها عبر الرموز ولن يكون لها حق الوصول إلى حالة الجلسة. 2. channels : هو الذي يحتوي على جميع الأحداث التي يدعمها المشروع 3. console : يحتوي على الأوامر التحكم القائمة على الأغلاق بحيث كل أمر أغلاق له أمر مثيل له يتم المقارنة فيه عند الأدخال و الأخراج المعنى أنه يحدد نقاط الدخول الى route في التطبيق 4. web : يحتوي الملف على التوجيهات التي يقوم RouteServiceProvider من middleware group الذي يوفر حالة الجلسه عن طريق الحماية csrf وتشفير ملفات الأرتباط ال cookie إذا كان تطبيقك لا يقدم واجهة مستخدم تعتمد على rest api فمن الممكن أن تكون جميع المسارات داخل web.php | **routes** |
| يحتوي دليل التخزين على سجلاتك ، وقوالب Blade المترجمة ، والجلسات القائمة على الملفات ، وذاكرة التخزين المؤقت للملفات ، والملفات الأخرى التي تم إنشاؤها بواسطة إطار العمل يمكن استخدام دليل التطبيق لتخزين أي ملفات تم إنشاؤها بواسطة التطبيق الخاص بك أو ملفات التخزين المؤقت أو السجلات التي أنشاؤها بواسطة التطبيق الخاص بك | **storage** |
| يحتوي على أوامر و واحدات تختبر التطبيق يمكن أن تختبر النطبيق من خلال كتابة الأمرين التالين في الكوميندلاين  php artisan test  php vendor/bin/phpunit | **tests** |
| يحتوي على تبعيات ال composer الخاص بك ويحتوي على ملفات مطور المشروع | **vendor** |

**نشر المشروع على السيرفر**

**5**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
|  |  |
|  |  |

**تحميل المشاريع من github**

**5**

1. **git clone link**
2. **composer install**
3. **php artisan key:generate**
4. **php artisan serve**

**Configuration**

**6**

|  |  |
| --- | --- |
| اسم الوظيفة | الرمز |
| لنتعرف على ملفات المشروع |  |
| تخزن جميع ملفات الضبط الخاصة بالمشروع في config مثل إعدادات الإتصال و معلومات قاعدة البيانات و خادم الإتصال بإضافة إلى العديد من قيم التكوين الأساسية مثل المنطقة الزمنية لتطبيق و مفتاح التشفير  ويتم التحكم بهذه الملفات من خلال مكتبة DotEnv التي توفر لنا ملف .env الذي يحتوي على متغييرات تتحكم بملفات ال config | **config** |
|  |  |
|  |  |
| .env يحتوي على متغييرات تتحكم بملفات ال config  .env.example يحتوي على عينات لإعدادات ملف .env الأصلي التي يشاهدها فريق التطوير  ملاحظة :   1. لا نقوم ببرمجة دوال في ال route تتحكم بإعدادات .env لان في حال تم الوصول للملف .env من قبل المتسللين يتم التحكم بالتطبيق وتهكيره لذلك نبرمج دوال فقط في config والدوال هي مثل env() 2. عادة ما يتم تحليل المتغييرات في ملف .env إلى سلاسل بشكل تلقائي لذلك إذا الكلمة تتالف من كلمتين تكتبها بدون فراغات مثل myapp وفي حال أردنا إضافة متغييرات نضعها بين علامتي إقتباس مزدوج مثل "my app" | **.env**  **.env.example**  **environment** |
| المتغييرات في ملف .env يتم الوصول إليها من ملفات ال config عن طريق الدالة  'debug' => env('APP\_DEBUG', false), حيث القيمة الأولى تعبر عن قيمة المتغيير الذي في ملف .env والقيمة الثانية تعبر عن القيمة الأفتراضية في حال القيمة الأولى فارغة  ملاحظة : لانستدعي هذه الدالة ألا فقط في ملفات ال config من أجل حماية ملف .env | **env('','')** |
| ننشئ متغيير أو عدة متغييرات في ملف .env وذلك بشكل إختياري مهمة المتغييرات تخزين القيمة التي سترحل إلى ملف ال config الجديد ثم ننشئ ملف في ملف ال config مثل ammar.php ثم نعيد داخله مصفوفة يحتوي العنصر داخلها على مفتاح و قيمة 'debug' => env('APP\_DEBUG', false), مثل  return [  'ammararrya' => env('APP\_AMMAR', 'hello'),  'مفتاح' => env('قيمة', 'قيمة افتراضية'),  ]; | **أنشاء ملف**  **config** |
| تستخدم للوصول للعناصر المصفوفات التي بداخل ملفات ال config مثلا في المثال السابق يتم الحصول على قيمة المفتاح الذي في المصفوفة عن طريق الدالة $val = config('ammar.ammararrya'); المعنى  config('أسم الملف داخل الكونفيغ . اسم المفتاح العنصر داخل المصفوفة')  القيمة الثانية في الدالة تمثل القيمة الأفتراضية  ملاحظة : لا نستدعي ال config() داخل ال route بل نستدعيها في controllers | **config('','')** |
| يتم دمج جميع ملفات الأعدادات بعد نشر المشروع على السرفير من خلال الأمر  php artisan config:cache  يقوم الأمر بدمج جميع ملفات الأعدادات في ملف واحد وتخزينه في الذاكرة المؤقتة لتسريع من وتيرة تحميل ملفات الأعدادات ويقوم بقفل ملف .env لكي لا يتمكن أحد من الدخول إليه | **دمج جميع ملفات الأعدادات بعد نشر المشروع على السرفير** |
| وهي خاصية التي تسمح بإظهار الأخطاء ومكانها ويتم التحكم فيها من خلال المتغيير APP\_DEBUG في ملف .env فنقوم يتشغيل الميزة فقط في حال التطوير للمشروع من خلال وضع المتغيير APP\_DEBUG = true أما في حال نشرنا المشروع على السرفير نقوم بإقافها من أجل تبقى الأخطاء والمتغيرات مخفيه عن المستخدمين APP\_DEBUG=false  و ستظهر رسالة في المتصفح 500 error | **DEBUG mode**  **نمط الأخطاء** |
| يفعل هذا النمط أثناء صيانة الموقع حيث ستظهر رسالة 503 SERVICE UNAVAILABLE عند الدخول على كل روابط الموقع  يفعل أمر الصيانة من خلال الأمر php artisan down  يتم إيقاف صفحة الصيانة من خلال الأمر php artisan up | **Maintenance Mode**  **نمط الصيانه** |

**mvc**

**7**

**شرح طريقة عمله بشكل مبسط :**

**يعتمد إطار العمل laravel في عمله على طريقة mvc = model views controller و هي طريقة لترتيب ملفات المشروع فقط حيث عندما يرسل المستخدم إلى السرفير ريكويست فيدخل الركويست أولا إلى ملف public/index.php للبحث عن صفحة المطلوبة حيث تكون الصفحات في هذا الملف من النوع ستاتيكي معناها لا تحتاج إلى قاعدة بيانات وتشغيل يرنامج التطبيق إذا لم يجد الصفحة يدخل إلى طبقة الحماية الأولى وهي middleware تعمل على التأكد من الريكويست إذا كان خاطئ ترفضه و إذا كان صحيح تقوم بتحويله إلى route الذي يحتوي على مسارات التطبيق وبدوره يقوم بتحويل إلى controller حسب المسار المرسل من المستخدم يقوم controller بفحص المسار هل يحتاج إلى قاعدة بيانات إذا لم يحتاج يطلب ال views ويحمل الصفحة المطلوبة أما إذا إحتاج إلى قاعدة بيانات يطلب model الذي بدوره ينشئ ويقرأ ويحدث ويحذف في قاعدة البيانات مع العلم أن يتم إنشاء الجداول والحقول لأول مرة بإستخدام migration ثم يقوم بتحويل إلى ال views مع المعلومات من قاعدة البيانات**

**public -> middleware -> route -> controller 1-> views 2-> model -> database -> modle -> views**

**Routing**

**8**

|  |
| --- |
| اسم الوظيفة |
| route : وهي مسارات تكتب في المتصفح نستطيع من خلالها التحويل إلى ال controller الذي يقوم بتحويل إلى ال views أو model  علما أنه يأتي قبل ال route يأتي middleware وهو الذي يحمي ال route من روابط غريبة ويتم إنشاء الراوت من خلال كتابة Route:: |
| ملفات الأساسية لل route هي :   1. web : يحتوي الملف على التوجيهات التي يقوم RouteServiceProvider من middleware group الذي يوفر حالة الجلسه عن طريق الحماية csrf وتشفير ملفات الأرتباط ال cookie إذا كان تطبيقك لا يقدم واجهة مستخدم تعتمد على rest api فمن الممكن أن تكون جميع المسارات داخل web.php فهو مسارات المخصصة للموقع الويب و المتصفحات 2. api : يحتوي الملف على التوجيهات التي يقوم RouteServiceProvider من middleware group التي تهدف هذه المسارات إلى أن تكون عديمة الحالة ، لذا فإن الطلبات التي تدخل التطبيق من خلال هذه المسارات تهدف إلى المصادقة عليها عبر الرموز ولن يكون لها حق الوصول إلى حالة الجلسة فهو المسارات المخصصة للتطبيقات أو الأدوات التي تعتمد على api 3. console : يحتوي على الأوامر التحكم القائمة على الأغلاق بحيث كل أمر أغلاق له أمر مثيل له يتم المقارنة فيه عند الأدخال و الأخراج المعنى أنه يحدد نقاط الدخول الى route في التطبيق 4. channels : هو الذي يحتوي على جميع الأحداث التي يدعمها المشروع |
| طريقة كتابة ال route هي :  Route::get('/المسار',function () {} أو controller); |
| وظيفة ال route :  Route::get('/link', function () {  return 'Hello World';  });// تنفيذ داله  أو  Route::get('/link', [UserController::class, 'index']);// التحويل إلى كونترولير  حيث link هو المسار الذي يكنب بالمتصفح \ UserController هو أسم الكونترولير \ index هو الدالة التي داخل الكلاس في الكونترولير |
| أنواع طرق التوجيه ال route :  Route::get($uri, $callback);// خاص بالبيانات غير الحساسة مثل التحويل لصفحة أو كونترولير التي لا تطلب مودل  Route::post($uri, $callback);// خاص بالبيانات الحساسة مثل التحويل لمودل أو تسجيل دخول في قاعدة البيانات  Route::put($uri, $callback);// خاص بالتعديل على البيانات  Route::patch($uri, $callback);  Route::delete($uri, $callback);// خاص بحذف البيانات  Route::options($uri, $callback); |
| Route::match(['get', 'post'], '/', function () {  //  });  تستخدم match لجمع بين نوعين روت أو أكثر  Route::any('/', function () {  //  });  معناها أنها تشمل كل أنواع ال route السابقة |
| use Illuminate\Http\Request;  Route::get('/users', function (Request $request) {  // ...  });  يستخدم لتوقع المسار سندرسها لاحقا |
| حقول ال <form> تضمن رمز سري يتم حقنه بالمسار لتأكد من الفورم  <form method="POST" action="/profile">  @csrf  ...  </form>  سندرسه بقسمه لاحقا |
| Route::redirect('/here', '/there');  عند طلب المسار الأول يتم التحويل إلى المسار الثاني مع رمز الحالة 302  Route::permanentRedirect('/here', '/there');  عند طلب المسار الأول يتم التحويل إلى المسار الثاني مع رمز الحالة 301 |
| Route::view('/welcome', 'html');  التحويل إلى صفحات ال view مباشرة  Route::view('/welcome', ' html ', ['verbview' => 'Taylor']);  يمكننا تحويل متغيير إلى صفحة ال view و يكون أسم المتغيير في صفحة ال view هو verbview وقيمته Taylor  Route::view('/welcome', ' html ', ['verbview' => $verb]);  يمكننا تحويل متغيير إلى صفحة ال view و يكون أسم المتغيير في صفحة ال view هو verbview وقيمته المتغيير $verb أو متغييرن مثل  Route::view('/welcome', ' html ', ['verbview' => $verb, 'verbview2' => $verb2]); |
| في حال كان لدينا مسار متغيير نقوم بوضح قوسين وبداخلهم متغيير بدون إشارة $ والقوسين لدلالة على متغيير  Route::get('/user/{id}', function ($id) {  return 'User '. $id;  });// <http://127.0.0.1:8000/user/33> >> user 33  وظيفة النقطة . هي الجمع بين المغيرات  Route::get('/posts/{post}/comments/{comment}', function ($postId, $commentId) {  //  });  في حال كان لدينا أكثر من مسار متغيير أو نريد نحول إلى كونترولير  Route::get('/verb/{verb1}/verb2/{verb2}', [ammar::class, 'verb', 'verb1', 'verb2']);  في الكونترولير  public function verb($verb1, $verb2)      {          return $verb1 . $verb2;      }  في حال لم يدخل المتغييرات يا أما مننشأ مسار جديد بدون متغييرات أو نضيف ? مع null مثل  Route::get('/verb/{verb1?}/verb2/{verb2?}', [ammar::class, 'verb', 'verb1', 'verb2']);  في الكونترولير  public function verb($verb1 = null, $verb2 = null)      {          return $verb1 . $verb2;      } |
| عندما يكون لدينا متغير في المسار ونريد من المستخدم ان يدخل نوع معين من الأحرف أو الأرقام نستخدم ->where('id', '[0-9]+') مثل  Route::get('/user/{name}', function ($name) {  //  })->where('name', '[A-Za-z]+');  Route::get('/user/{id}', function ($id) {  //  })->where('id', '[0-9]+');  Route::get('/user/{id}/{name}', function ($id, $name) {  //  })->where(['id' => '[0-9]+', 'name' => '[a-z]+']);  تستخدم هذه الطريقة للمسار المحدد أما اذا نريد تطبيق على جميع المسارات التي تحمل أسم المتغيير id أو أي متغيير نحدده نذهب إلى  App\Providers\RouteServiceProvider  ثم ندخل إلى function boot() {} ونكتب هذه الجملة  public function boot()  {  Route::pattern('id', '[0-9]+');// هذه هي الجملة  }  والمسار الأساسي يكون  Route::get('/user/{id}', function ($id) {  // Only executed if {id} is numeric...  });  أو أي مسار اخر يحمل نفس المتغيير |
| named routes : يمكن تسمية المسار من أجل تميزه و استدعائه من أماكن أخرى مثل view يتم التسمية كالتالي  Route::get('/user/profile', function () {  //  })->name('profile');  من اجل كونترولير  Route::get('/user/profile', [UserProfileController::class, 'show'])->name('profile');  الاستدعاء عن طريق الدالة route('name') مثل  {{route('profile')}} // in view  $url = route('profile');  return redirect()->route('profile'); // لتوليد إعادة توجيه للمسار  في حال كان للمسار متغييرات نمررها في الدالة عن طريق مصفوفة لكن المتغير الأول في المسار و الثاني في الصفحة نفسها  route('name', ['verb1' => $name])  اذا متغييرين  route('name', ['verb1' => $name, 'verb2' => $name2])  اذا كنت ترغب في تحديد ما إذا كان الطلب الحالي قد تم توجيهه إلى مسار مسمى معين ، فيمكنك استخدام التابع المسمى في نسخة المسار. على سبيل المثال ، يمكنك التحقق من اسم المسار الحالي من برمجية وسيطة للمسار:  public function handle($request, Closure $next)  {  if ($request->route()->named('profile')) {  //  }  return $next($request);  }  سندرس لاحقا ال middleware |