Internet Applications and Techniques

Create your first Laravel project

In this lab class, you will firstly install Composer and then create your first Laravel project. You will get to know the Laravel directory structure and understand the Laravel MVC framework. A simple banking project is developed using basic routing, database migration and Eloquent ORM etc. This banking system allows a user to list all the banking accounts and show one single account.
This tutorial will use the new Laravel version 7.0 which needs PHP version 7.2.5 or newer.

Task 1. Install Composer

To install Composer, you could just follow the instructions from the web page: `https://getcomposer.org/download/`.

For Window users, it is quite handy to use the Composer installer to install the composer globally.

For Linux / Unix / OSX users, you could install the Composer using command line by following the instructions in `https://getcomposer.org/download/`. If successful, you will see the downloaded **composer.phar** in your directory. This file is the Composer binary in PHAR (PHP archive) format which can be run on the command line.

You choose to make the Composer installation globally, you could run the Composer commands anywhere. If you install locally, you could only run the Composer commands in the installed directory Therefore for local installation, you should install the composer within in your document root folder.

To test the composer, you could run the command: `php composer.phar -V`

Or `composer -V` or `PHP composer -V` depending on your installation. This command will display the Composer version information.

Note, you could find the more information about Composer commands from:
`https://getcomposer.org/doc/01-basic-usage.md`

Task 2 Create a basic Laravel project

1) Create a default project

Firstly, in your document root folder, run the following *`create-project`* command of Composer to create a Laravel project called `larabank`.

```
php composer.phar create-project --prefer-dist laravel/laravel larabank
```

This command may need a couple minutes to finish. Basically, the **`laravel/laravel`** is the package for the composer to install and **larabank** is the directory name of your new project.

Note, we only use one method to create Laravel project here. You could find more information from http://laravel.com.

2) View the default project page using Laravel's development server

After the installation, let us check the default project web page using the Laveral's development web server.

Laravel framework comes with a nice built-in command line interface, the **Artisan CLI**, which provides you with a bunch of useful commands to help you build your application. More info about artisan could be found in https://laravel.com/docs/7.x/artisan.

In the terminal and make your **larabank** folder as the current directory, run the command:

```
php artisan serve
```

This `artisan serve` command will start the built-in Laravel development server at **http://localhost:8000**. Check it in a browser, the default Laravel welcome page will show like:

*Note, **ctrl+c** can be used to terminate the built-in web server.*

3) View the default project page using your Apache/Mysql server

Since we will use an existing MySQL database and we have put this project into the **document root** folder, we need to use the Apache/Mysql server installed on your computer. Now start you Apache and MySQL. You will be able to see the same default page:

http://localhost/larabank/public/

Important: *You may need to change the permission allowing to read and write the larabank **storage** folders since some logging file are in the sub-folders.*

4) Routing to show a Hello World page

Open the project root folder (**larabank)** using your preferred IDE/editor (**Atom** is recommended here)**.** The **routes/web.php** file defines routes for your web interface. There is already one statement there for the default web page.

Add the following code to this **web.php**

```
Route::get('/hello',function(){
    return 'Hello World!';
});
```

This statement responds to the GET method of HTTP for the URI **hello**. function(…) defines an anonymous function that does the actual work for the requested URI. Here this function simply returns 'Hello World!' to the requesting browser.

After saving the code, you could see the Hello World page at: http://127.0.0.1:8000/hello on the Laravel development server. Or *http://localhost/larabank/public/hello* on the Apache server.

Task 3. Laravel directory structure and MVC

To explore the directory structure of the **larabank** project, you could use command lines or file browser but some files are hidden in default. The best approach is to use an IDE/Editor (e.g. Atom) to check the project files. You could see many folders and files have been created for this project. The following table shows the key directories in a standard Laravel project.

	DIRECTORY	DESCRIPTION
1	**/app**	contains all of your core application code
2	/app/Console	contains all of your artisan commands
3	/app/Events	contains event classes
4	/app/Exceptions	contains exception handling classes
5	/app/Http	contains **controllers**, middleware, and requests
6	/bootstrap	contains **app.php** file required by the bootstrap framework
7	/config	contains the application configuration files
8	**/database**	Contains database migrations and seeds. Also used to store the database for SQLite
9	/public	contains **index.php** as entry point. Holds assets such as images, CSS, JavaScript etc.
10	**/resources**	contains the **views** and also un-compiled resources and languages etc.
11	**/routes**	contains all of the route definitions for your applications. The **web.php** file contains routes in the web middleware group
12	/storage	contains compiled Blade templates, file based sessions, file caches, and other files generated by the framework.
13	/tests	contains automated unit tests
14	/vendor	contains composer dependencies

Check the contents of the **composer.json** and **composer.lock** files in the project root. C*an you find the matching packages in the **vendor** folder of your project?*

Laravel follows the model-view-controller (MVC) architectural pattern as shown in the diagram below. A browser sends a request, which is received by a web server. This request is given to the Laravel routing engine. Based on the routing pattern Laravel router sends it to the controller. Sometimes, the controller will immediately render a view, which is a template that gets converted to HTML and sent back to the browser. More commonly for dynamic sites, the controller interacts with a model and communicates with the database. After invoking the model, the controller then renders the final view (HTML, CSS, and images) and returns the complete web page to the user's browser.

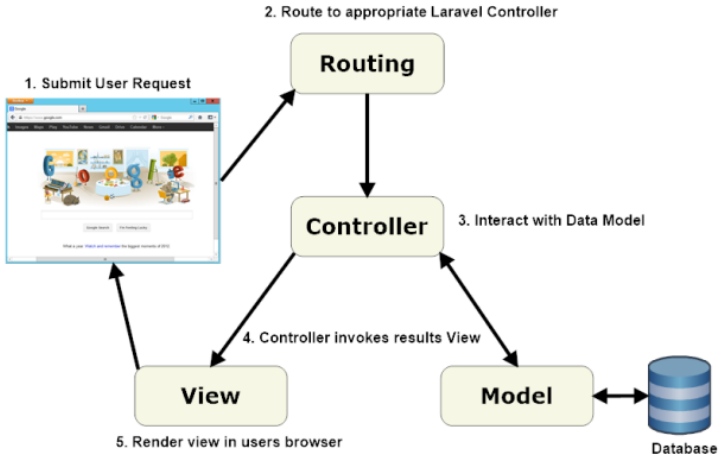

Diagram from: https://anaghar1996.wordpress.com/2016/09/16/laravel/

Task 4. Configure Database

We need to use a database for our larabank project so we will configure our database first. Laravel supports a group of databases including MySQL, Postgres, SQLite etc.

For this tutorial, we will use MySQL which you should have installed in your computers. For your larabank project, you need to configure accordingly. Laravel project comes with ***.env*** file where you can configure your database and other environment variables and we will work with ***.env*** file.

Open the ***.env*** file located at your project root folder. In the file you will find code like below:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=~~xxxxxxx~~banking
DB_USERNAME=~~xxxxxxx~~root  (your DB username here)
DB_PASSWORD=~~xxxxxxx~~  (your DB password here)
```

Edit this **.env** file and change the database to **banking** and change username as **root** and leave the password empty which is the default setting for WAMP, XAMPP.

We will use the Apache and MySQL for the following tasks. You should have a **banking** database produced from last tutorial. If not, just use **phpMyAdmin** to create a database called **banking**.

Task 5. Create s simple banking project

This task aims to create this larabank system which allows a user to list all the banking accounts and show the information of one single account.

1) The Model

Firstly, we'll start by creating a model, to represent an account for banking.

In Terminal, move into your project root directory, we will **php artisan** command line for this task. Run the following command to generate a new **Account** model:

```
php artisan make:model Account -m
```

We would like to generate a database migration thus use the **-m** (or **–migration)** option. Migrations are like version control for your database, allowing your team to easily modify and share the application's database schema. For more info: https://laravel.com/docs/7.x/migrations

Models are all stored in the **app** directory, so this command will generate an **app/Account.php** model file.

```
<?php
        namespace App;
        use Illuminate\Database\Eloquent\Model;
        class Account extends Model
        {
            //
        }
```

Note: the code above imports the **Eloquent's model** class namespace. Eloquent models are models that extend the **Illuminate\Database\Eloquent\Model** class. Eloquent ORM is an Object Relational Mapper and implements the active record pattern. It is used to interact with relational databases inside a Laravel project.

By supplying the **-m** option when generating the model, Laravel also generated a **database migration** file for creating the **accounts** database table. The migration file **[timestamp]_create_accounts_table.php** is located at **database/migration** folder.

Note, we only want to create one table in our **banking** database for this lab exercise therefore need to move the **xxx_create_users_table.php** and **xxx_create_password_table.php** two files in the

database/migration to another folder (e.g. the **database** folder). When we want to use them in the future, we can move them back.

The migration file **[timestamp]_create_accounts_table.php** contains the following code:

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateAccountsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('accounts', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
    }
    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('accounts');
    }
}
```

The **up** method is used to add new tables, columns, or indexes to your database, while the **down** method should reverse the operations performed by the **up** method.

By convention, the "snake case", plural name of the class will be used as the table name unless another name is explicitly specified. So, in this case, Eloquent will assume the **Account** model stores records in the **accounts** table.

Add a few columns in the **accounts** table by adding the code below in the **up** function:

```
$table->string('accountno');
$table->string('type');
$table->float('balance');
$table->float('interest');
```

Secondly, use migration to create the accounts table

Run the following Artisan command:

```
php artisan migrate
```

You should be able to see the **accounts** table created in your MySQL **banking** database. If you want to make any change, you could easily rollback by running `php artisan migrate:rollback`.

Thirdly, add a couple of records in the accounts table.

Of course, you could simply add some records in the accounts table through the phpMyAdmin GUI. Another alternative is Artisan Tinker, which can be used to easily interact with a Laravel application.

Let's try to use **artisan tinker** to add a few account records for our application. From the Larabank project root, run the following command:

```
php artisan tinker
```

This command opens a Read–Eval–Print Loop (repl) for interacting with your application. From the repl, we can create a new account. You should note that we interact with this repl just like we would write code for a Laravel application. So to create a new account, we would just type:

```
$account = new App\Account;
$account->accountno = "111111";
$account->type = "saving";
$account->balance= 1000;
$account->interest = 0.01;
$account ->save();
```

Here, we could use new App\Account since we have created the Account class specified in the Account.php file in the APP folder. Now we can type **$account** to the repl and show the account added:

```
App\Account {#2928
accountno: "111111",
type: "saving",
balance: 1000,
interest: 0.01,
updated_at: "2020-03-03 10:32:16",
created_at: "2020-03-03 10:32:16",
id: 1, }
```

Just adding a couple more records, you could type **q** to quit the Tinker.

2) The Controller

Next, let us work on the controller. A controller is a part that delegate tasks. Controller is the manager for our app which tells the model what to compute and tells Blade (view) what to display. It is the middleman between view and model.

Open up your Terminal and run the command below to create an **AccountController**:

```
php artisan make:controller AccountController
```

This will generate an **app/Http/Controllers/AccountController.php** controller file with the following code:

```
<?php
    namespace App\Http\Controllers;
    use Illuminate\Http\Request;

    class AccountController extends Controller
    {
    //
    }
```

Then we just need to define some controller actions and later on we will create routes to associate URLs with these controller actions.

Since we will need to use Account object in the **AccountController.php**, add following code before the class definition.

```
use App\Account;
```

Now we add two controller actions in the **AccountController.php**: **show** for showing one account and **list** for listing all accounts. Adding the following code in the class body:

```
    public function show($id){
      $account = Account::find($id);
      return view('/show', array('account' => $account));
}
```

```
    public function list(){
      return view('/list', array('accounts'=>Account::all()));
  }
```

In the above code:
For retrieving an account object to show, we just call **Account::find** method and pass in the **$id**.
For listing all accounts, we just use **Account::all()** method to return all the account objects
The view is then loaded using the **view** function and passing the name of the view and an array with the data to be supplied to the view.

3) Routing

We have done a simple routing in Task 2. Now we can simply add the following code in the **web.php** file and assign the **/list** and **/show** URIs to the AccountController with **list** and **show** actions:

```
Route::get('list', 'AccountController@list');
Route::get('show/{id}', 'AccountController@show');
```

4) The View

Laravel view files are all stored in the **resources/views** folder. In the previous step, we passed the **view** function the view name **show**. That tells Laravel to look for a view file located in the main **resources/views** directory named **show.blade.php**. Laravel view files utilize the **Blade templating engine**, and hence are named **.blade.php**.

To finish the implementation of this show account page, we can create a simple view named as **show.blade.php** in **resources/views folder** with the following code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Account {{ $account->id }}</title>
  </head>
  <body>
    <h1>Account no {{ $account->accountno }}</h1>
    <ul>
      <li>Balance: {{ $account->balance }}</li>
      <li>Interest: {{ $account->interest }}</li>
      <li>Created: {{ $account->created_on }}</li>
    </ul>
  </body>
</html>
```

Since we passed the **Account** object to the view - back in the **show** action of the controller - with the array key **account**, we can access it in the view via a variable of the same name, **$account**.

And, as you can see, the **Account** object's values can be accessed using the same names as its associated **accounts** database table's column names.

Finally, you see the use of the Blade syntax for printing out information. The following, for example, prints out the account's balance:

```
{{ $account->balance }}
```

That statement is simply translated into a pure PHP **echo** statement in the background:

```
<?php echo $account->balance; ?>
```

But the Blade syntax makes writing views much faster and more enjoyable, and reading them much easier. You will not go deep about the Blade in this exercise. You could find more info from *https://laravel.com/docs/7.x/blade*.

Similarly to finish the implementation of this list account page, we can create the **resources/views/list.blade.php** file with the following code:

```
<table>
    <thead>
        <tr>
            <th> id</th>
            <th> Account no</th>
            <th> Type  </th>
            <th> balance </th>
            <th> Interest</th>
        </tr>
    </thead>
    <tbody>
        @foreach($accounts as $account)
         <tr>
            <td> {{$account->id}} </td>
            <td> {{$account->accountno }} </td>
            <td> {{$account->type }} </td>
            <td> {{$account->balance}} </td>
            <td> {{$account->interest}} </td>
         </tr>
        @endforeach
   </tbody>
</table>
```

Now go to the web page: *http://localhost/larabank/public/show/1*

You will see the account information with ID as '1' in your accounts database table.

Go to: *http://localhost/larabank/public/list*

You will see the list of all accounts in your accounts database table.

This is a simple re-implementation of the MVC book example we introduced in the previous Unit. Review what you have done in this lab to get a better understanding of the MVC architecture pattern and its implementation in Laravel.