Department of Computer Science and Software Engineering

Advanced Programming Practices
SOEN 6441

Project

Submitted By:

Pankaj Deep Sahota 40225528

Harinder Singh Rana 40230961

Professor:

Dr. C. Constantinides, P.Eng.

Index

Overview

Implemented Languages / Softwares :

1. Laravel (Implemented on PHP with MVC)
2. MySQL Database
3. SQL
4. DataTables(Plugin for Jquery Javascript Library)
5. XAMPP (LocalHost)
6. HTML, CSS, Bootstrap, Javascript
7. AJAX
8. PHPUnit (For Testing)

This Project is built on MVC (Model, View & Controller) architecture. MVC is used for creating complex applications, and is an architectural/design pattern that separates an application into three main logical components **Model**, **View**, and **Controller**.

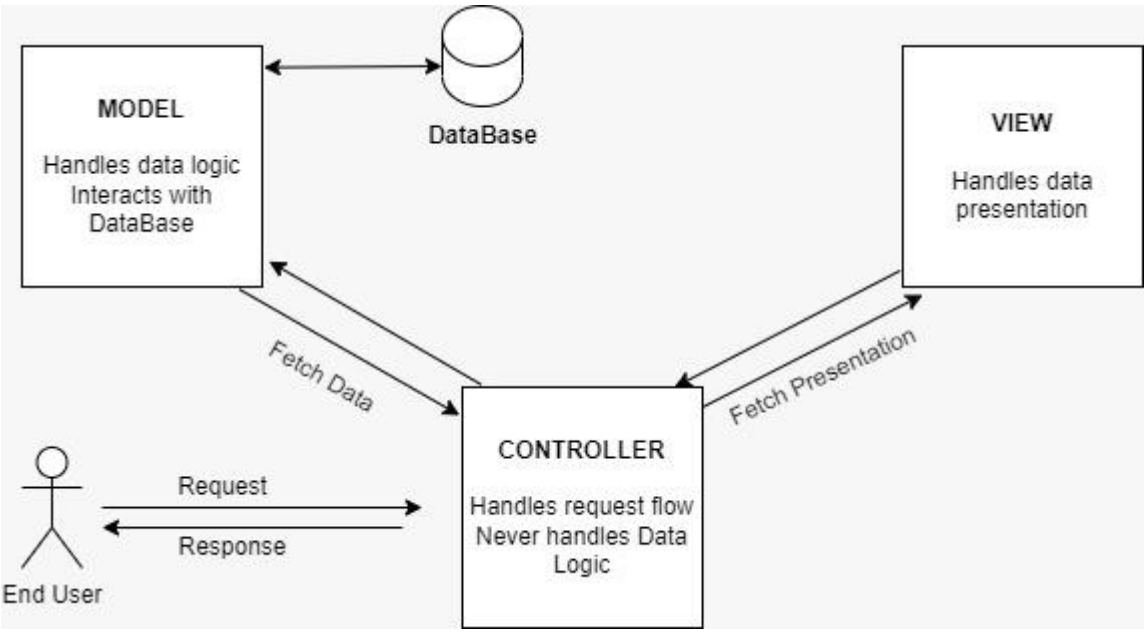
Laravel is a web application structure based on PHP language with expressive, elegant syntax. Laravel strives to provide an amazing developer experience while providing powerful features such as thorough dependency injection, an expressive database abstraction layer, queues and scheduled jobs, unit and integration testing, and more. Laravel is the best choice for building modern, full-stack web applications.

MySQL Database Service is a fully managed Oracle Cloud Infrastructure native service, which automates tasks such as backup and recovery, and database and operating system patching.

SQL database is connected with the frontend through Laravel MVC and displayed on the UI.

MVC Architecture

Model-View-Controller (MVC) is both a design pattern and architecture pattern. It's seen as more of an architectural pattern, as it tries to solve some problems in the application and affects the application entirely. Design patterns are limited to solving a specific technical problem.



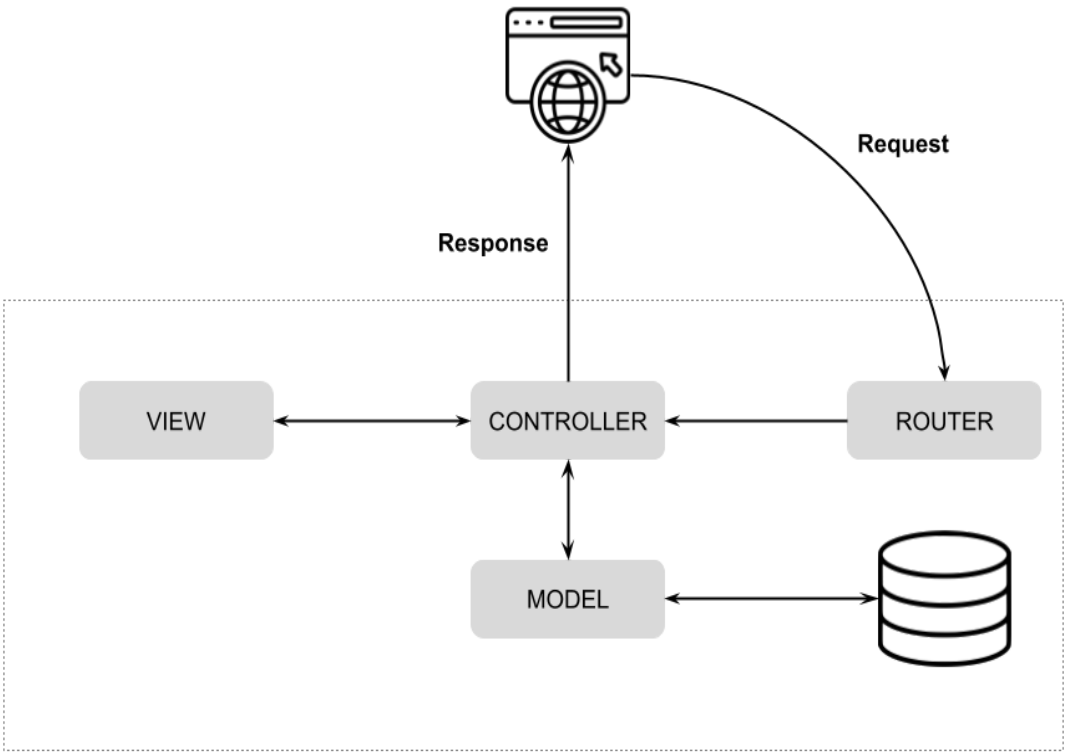
MVC divides an application into three major logical sections:

- Model
- View
- Controller

The Model component governs and controls the application database(s). It's the only component in MVC that can interact with the database, execute queries, retrieve, update, delete, and create data. Not only that, but it's also responsible for guaranteeing the evolution of the database

structure from one stage to another by maintaining a set of database migrations. The Model responds to instructions coming from the Controller to perform certain actions in the database.

The View component generates and renders the user interface (UI) of the application. It's made up of HTML/CSS and possibly JavaScript. It receives the data from the Controller, which has received the data from the Model. It merges the data with the HTML structure to generate the UI.



Refactoring

Before :

```
post::updateOrCreate(
   ['id' =>$data[0]['id']],
   [
      'id' => $data[0]['id'],
      'userId' => $data[0]['userId'],
      'title' => $data[0]['title'],
      'body' => $data[0]['body'],
   ]
);
post::updateOrCreate(
   ['id' =>$data[1]['id']],
   [
      'id' => $data[1]['id'],
      'userId' => $data[1]['userId'],
      'title' => $data[1]['title'],
      'body' => $data[1]['body'],
   ]
);
post::updateOrCreate(
   ['id' =>$data[2]['id']],
   [
      'id' => $data[2]['id'],
      'userId' => $data[2]['userId'],
      'title' => $data[2]['title'],
      'body' => $data[2]['body'],
   ]
```

After :

```
public function index()
{
    $api_url = 'https://jsonplaceholder.typicode.com/posts';
    $response = Http::get($api_url);
    $data = json_decode($response->body());

   echo "<pre>";

   foreach($data as $post)
   {
    $post = (array)$post;
    post::updateOrCreate(
        ['id' =>$post['id']],
        [
            'id' => $post['id'],
            'userId' => $post['userId'],
            'title' => $post['title'],
            'body' => $post['body'],
        ]
    );

   }
   dd("data stored");
}
```

Design Pattern

Design patterns are typical solutions to common problems in software design. Each pattern is like a blueprint that you can customize to solve a particular design problem in your code.

Design patterns differ by their complexity, level of detail and scale of applicability to the entire system being designed.

There are three, most used, types of Design Pattern:

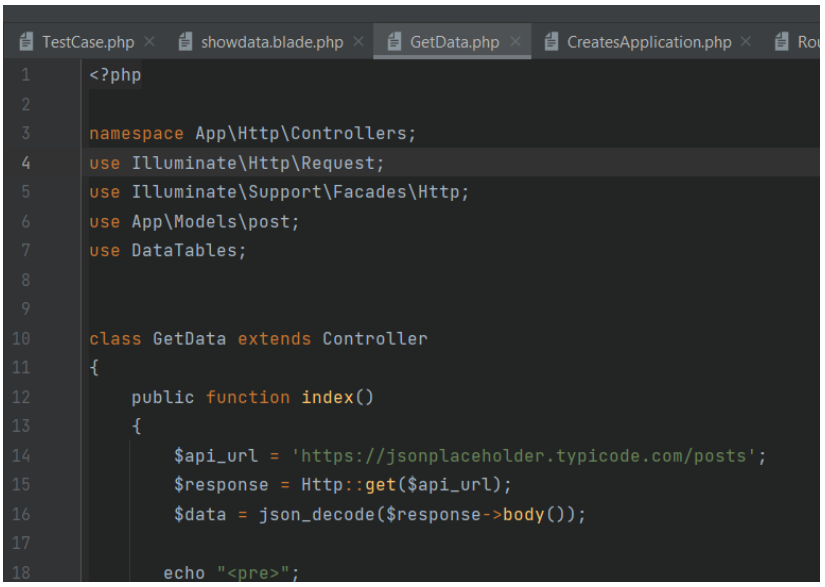
- Creational patterns provide object creation mechanisms that increase flexibility and reuse of existing code.
- Structural patterns explain how to assemble objects and classes into larger structures, while keeping these structures flexible and efficient.
- Behavioral patterns take care of effective communication and the assignment of responsibilities between objects.

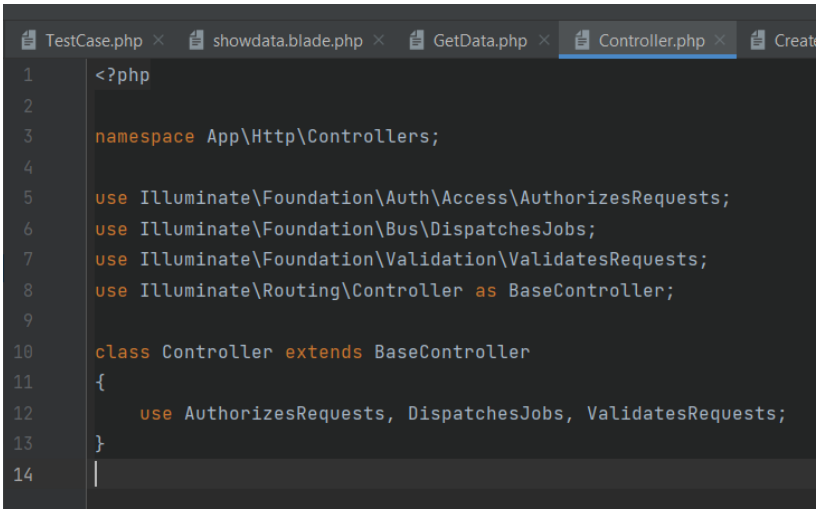
We are using Bridge Pattern in our project, which is a structural design pattern that lets you split a large class or a set of closely related classes into two separate hierarchies—abstraction and implementation—which can be developed independently of each other.

Bridge Pattern is basically the implementation of abstract classes within the project and using the abstract methods within the sub-class which extends that particular abstract class.

Implementation of bridge pattern in the project is shown below :
The GetData.php file contains a class “GetData” which is extending the class Controller, which is the super class and has some methods implementation in it, which is further extending BaseController, which is

an abstract class. It does not provide implementation to some members which the controller does and so on.



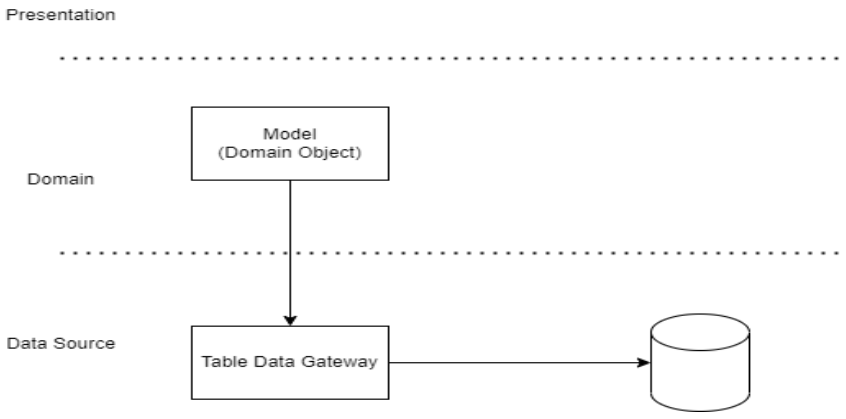


Object Relational Structural Pattern

Object-oriented programming for business logic and relational databases for data storage.
Object-relational mapping (ORM) is a bridge between the two that allows applications to access relational data in an object-oriented way.
Model, View, Controller interactions is the implementation of ORM in this project.
According to our project:
Firstly, the viewer will interact with the UI, the framework, and the UI interactions will be sent to the controller first and the controller will then connect with the Model.

After, the model sends the data requested by the controller, it will call View and the View will be responsible for displaying the data accordingly to the frontend to the viewer.

Model, view and controller do not interact directly with each other. Controller is the mid-way point to interact with Model and View. Each communication happens through the controller.

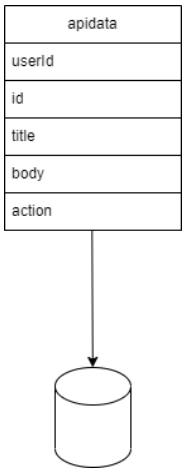


Data Source Architectural Pattern

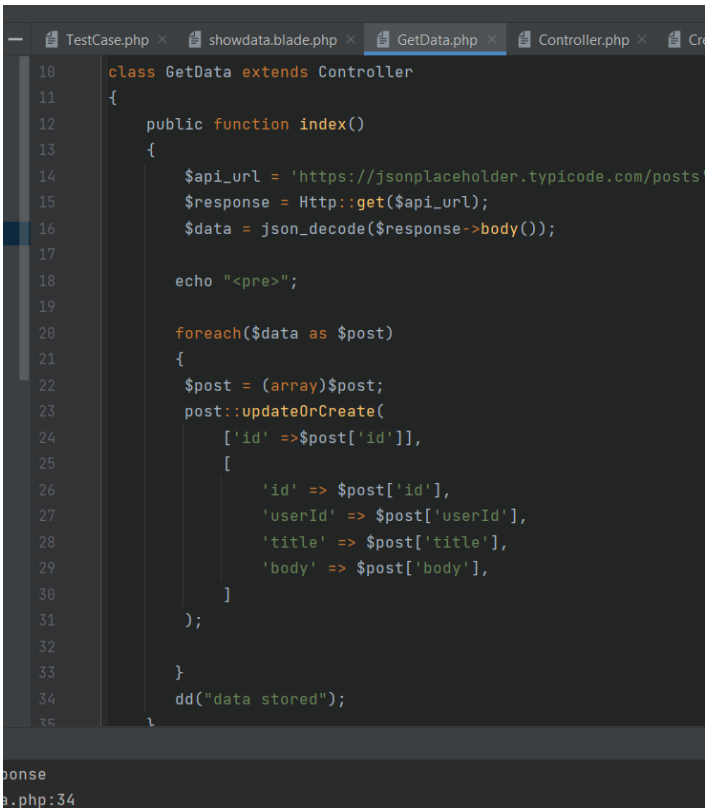
There are some commonly accepted patterns which have been established for accessing the data service layer of an application namely 1. Table Data Gateway 2. Active Record 3.Data Mapper

In this project we have implemented Table Data Gateway.

In the controller, the first function Index is hitting an API and fetching a response in JSON format and then decoding that json data which can be used in our application. In the foreach loop we are taking every data set and mapping it to the corresponding fields in the table post using the “post” model where the model is connected with the database and this relationship shows the mapping of data in the model with the data of the table. Every attribute of the class in the model is connected with the table and that is how the Model and table interact without any issues and errors. After finishing filling up the data in the local database, it is going to print output "Data Stored".



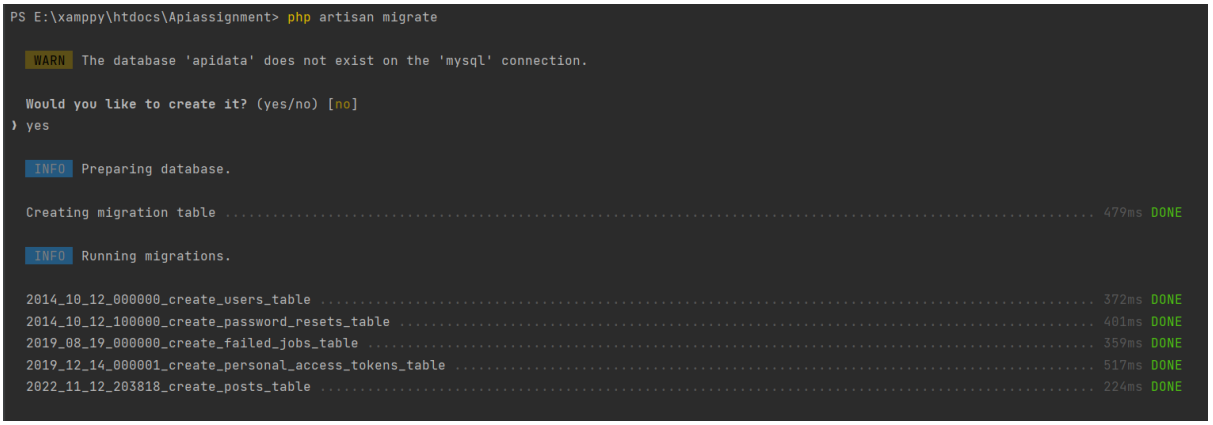
UNIVERSITÉ Concordia UNIVERSITY

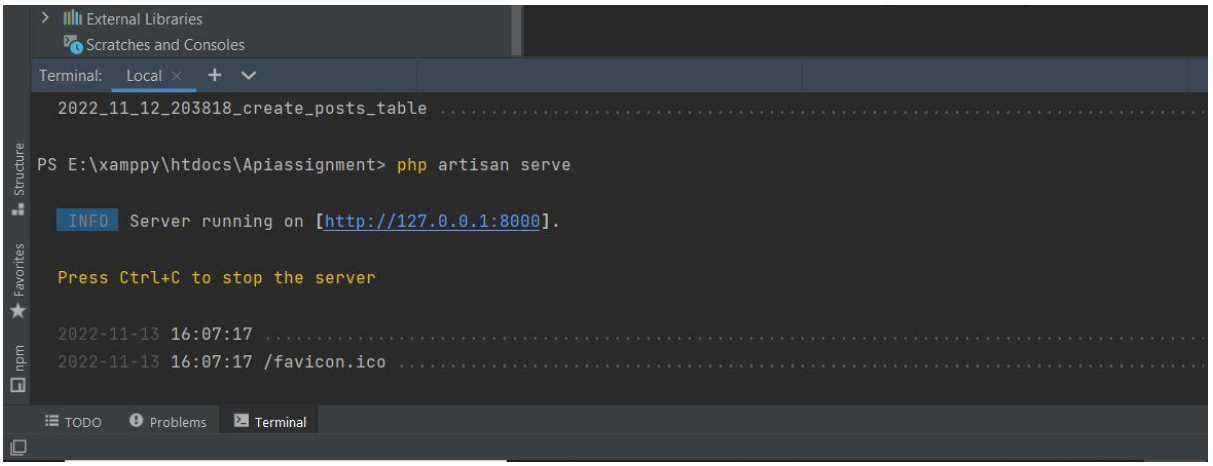



```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class post extends Model
{
    protected $table = "posts";
    protected $fillable =
    [
        'userId',
        'title',
        'body',
    ];
}
```

Implementation

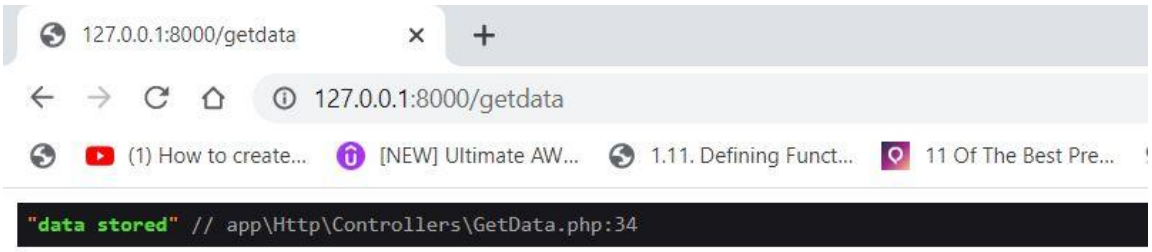
Simple execution of laravel:
First of all, we have to run a command prompt and run a simple query - **“php artisan migrate”** which will migrate the database and connect it to the MVC. Migrate helps the database to be initialized and is typically paired with Laravel's schema builder to build the application's database schema
After the above command, we execute the command **“php artisan serve”** which will give us the local url where the web app will be opened. If we go to the server url, we will not see the implementation of the project. It requires a few more steps to be finally able to see the implementation which is mentioned below.
These two commands are the base of the project, and without these commands executed on the command line, the project could not be executed on the personal desktop. Example screenshot is attached below:





Note: It is recommended to open the project folder in Visual Studio IDE.

After running the two commands mentioned above, just copy the url created from the last command and paste it to the browser and add **"/getdata"** after the url. It will call **getdata.php** and the data will be stored in the database and the connection will be made by the laravel with the database. It will display the message **"data stored"** followed by the file called during execution of the command. Screenshot is attached below for more clarity:



Now, the final command will take us to the UI of the project, which is to remove the "/getdata" and replace it with **"/showdata"**. As the name suggests, it will now show the data whatever is there in the UI, as it is.

The execution of the program will start from the frontend, which is what will be displayed on the first screen when a user hits that web url.
The **Routes** folder will have a web.php file which has a base url that will automatically open up the webpage defined in the Controller. It returns **View** which is within **resources** folder which then calls "**showdata.blade.php",** which in turns displays the content on the frontend. In our case, it shows the table created.

Model**:**
The model is located within the **App** folder and inside the **Models** folder there is a file named as **Post.php** which will be responsible for creating a class name post which extends Model and created two **attributes** which are protected to the class namely, **table** and **fillable**, which has **userId**, **title** and **body** attributes.

Controller:
The controller is present in the **Http** folder within **app** folder which is further present within Controllers where **Controller.php** is the one which is linked to our model and view.
The work of the controller is to pass the data to the frontend/interface after fetching the data from the model. Everything is related to the controller, the model and view will interact everytime with the controller to get and display the messages every time they are called. Controller acts as an intermediary between these two.

View:
Within the **resources** folder is the **views** folder which has the file called **showdata.blade.php.** This file is the view of our program which will help in displaying the data which will be implemented in the program as to how the data should be presented.

.env file:

Database connection is mentioned in the .env file which is the environmental file, which includes all the things related to the application and some other user defined services.

The Database attributes:
Name = apidata
Username = root
Password =

The connection will be read by laravel and simply displayed to the frontend. This file is mostly used in the development environment and while pushing the code to the github.

It is not committed each time due to the property of the file as it stores the local data which does not frequently change and hence, it does not get committed every time we commit the whole project folder.

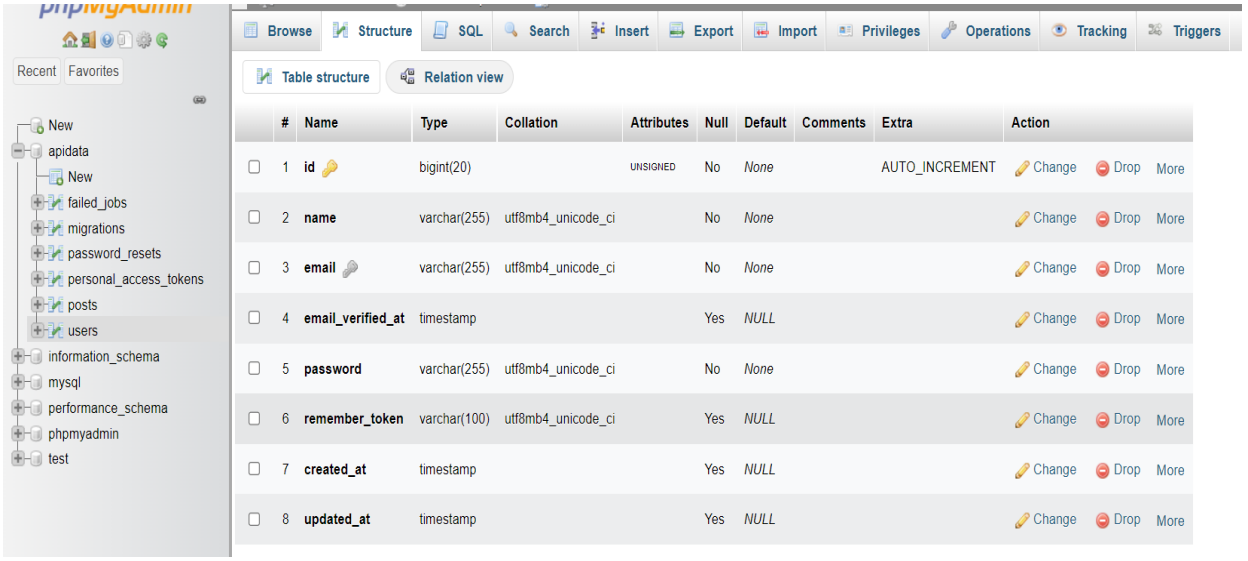
Migrations:
It means the blueprint or schema of the local database before actually creating the database. It is just a structure of the table and creates it before we put the data inside it.

According to the project, Table created in the migration database is named as **migrations.** It keeps the track of all the database tables created within the project. It fetches the data while the commands are executed, such as "**php artisan migrate**".
It is useful to Laravel which keeps the track of all the DB's and it saves the time everytime when the changes are made to the already created

database. DBs created and saves time when we again run the command, it doesn't always create the database.
If changes are made to the migrations, only then the migrations will be changed. But, before that, the migrations need to be cleared or reverted to execute and save the new migrations. Then only the mirations will be allowed in the DB.

UNIVERSITÉ Concordia UNIVERSITY

```php
public function viewdata(Request $request)
{
    if ($request->ajax()) {
        $data = post::select('*');
        return Datatables::of($data)
                ->addIndexColumn()
                ->addColumn('action', function($row){

                      $btn = '<a href="javascript:void(0)" class="edit btn btn-primary btn-sm">View</a>';

                       return $btn;
                })
                ->rawColumns(['action'])
                ->make(true);
    }

    return view('showdata');
}
```

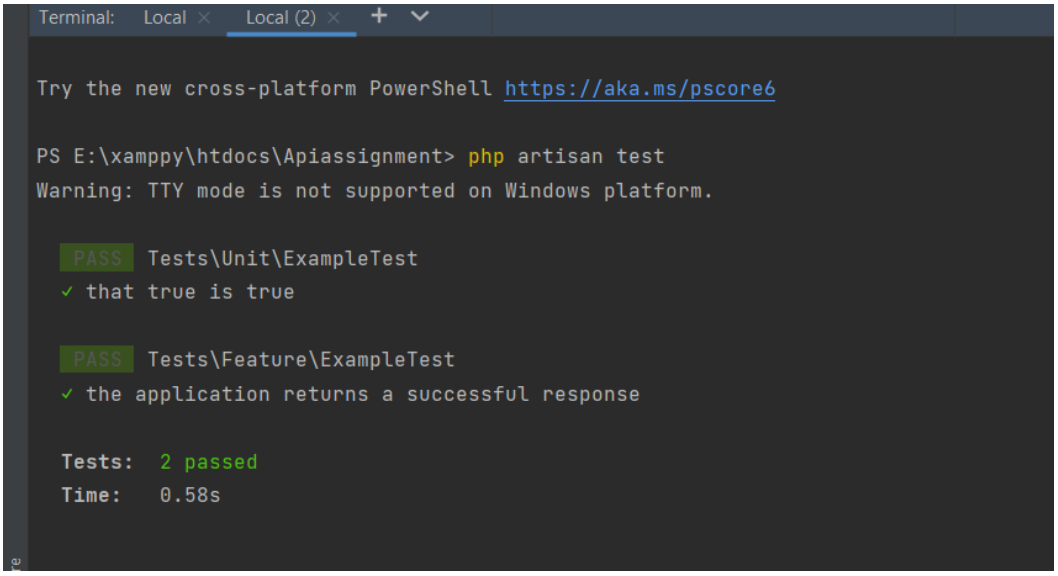


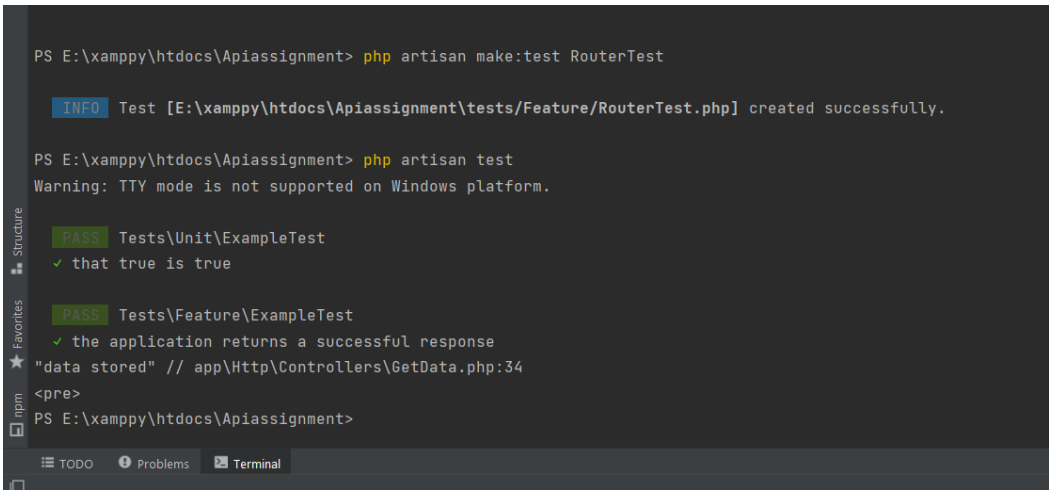
Testing

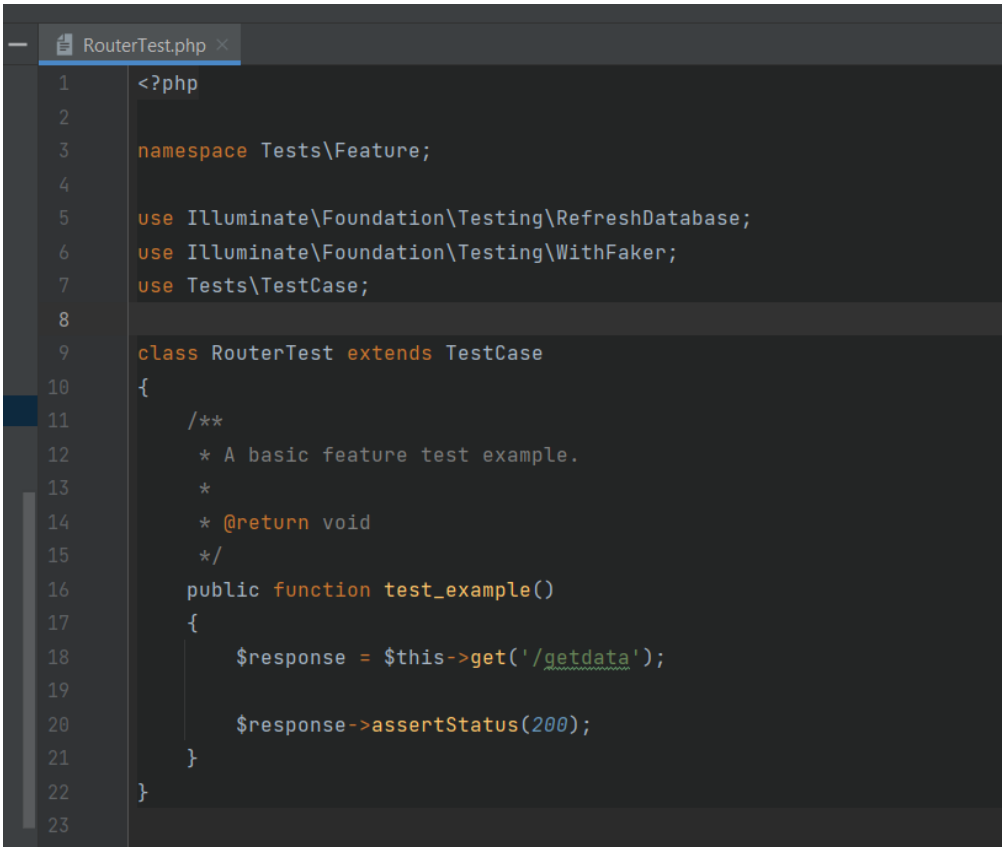
PHPUnit is a unit testing framework for the PHP programming language. It is an instance of the xUnit architecture for unit testing frameworks that originated with SUnit and became popular with JUnit. The support for testing with PHPUnit is included inside the laravel and a phpunit.xml file is set up for the application.

In order to make your own Unit Test you can run the Command - “php artisan make:test RouterTest”.
In this project, For the testing we have created 2 tests. The first one will basically check the base test case of creating a new application in Laravel, to make sure there is no compile error and the second test will check for the route and its internal components ... If the route exists and the controller function is executing its functionality as per the expectations.

Testing screenshot is attached below:




PS E:\xamppy\htdocs\Apiassignment> php artisan make:test RouterTest
INFO Test [E:\xamppy\htdocs\Apiassignment\tests/Feature/RouterTest.php] created successfully.
PS E:\xamppy\htdocs\Apiassignment> php artisan test
Warning: TTY mode is not supported on Windows platform.
PASS Tests\Unit\ExampleTest
✓ that true is true
PASS Tests\Feature\ExampleTest
✓ the application returns a successful response
"data stored" // app\Http\Controllers\GetData.php:34
<pre>
PS E:\xamppy\htdocs\Apiassignment>
Structure
Favorites
npm
TODO
Problems
Terminal


RouterTest.php
<?php
namespace Tests\Feature;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\WithFaker;
use Tests\TestCase;
class RouterTest extends TestCase
{
/**
* A basic feature test example.
*
* @return void
*/
public function test_example()
{
$response = $this->get('/getdata');
$response->assertStatus(200);
}
}

Software Architecture Diagram