Practical 3 : Working with Databases and Object-Relational Mapping

In this lab, you will use previously developed web application to learn expound concept of database and object-relational mapping (ORM).

1. Database Configuration and Fetch Data using Controller.

Database configuration file reflects the configurations in .env. General specification such as database name, host, username and password can be further specified through .env file in the Laravel web application to match with the server that it intends to connect as shown in Figure 1 and Figure 2.

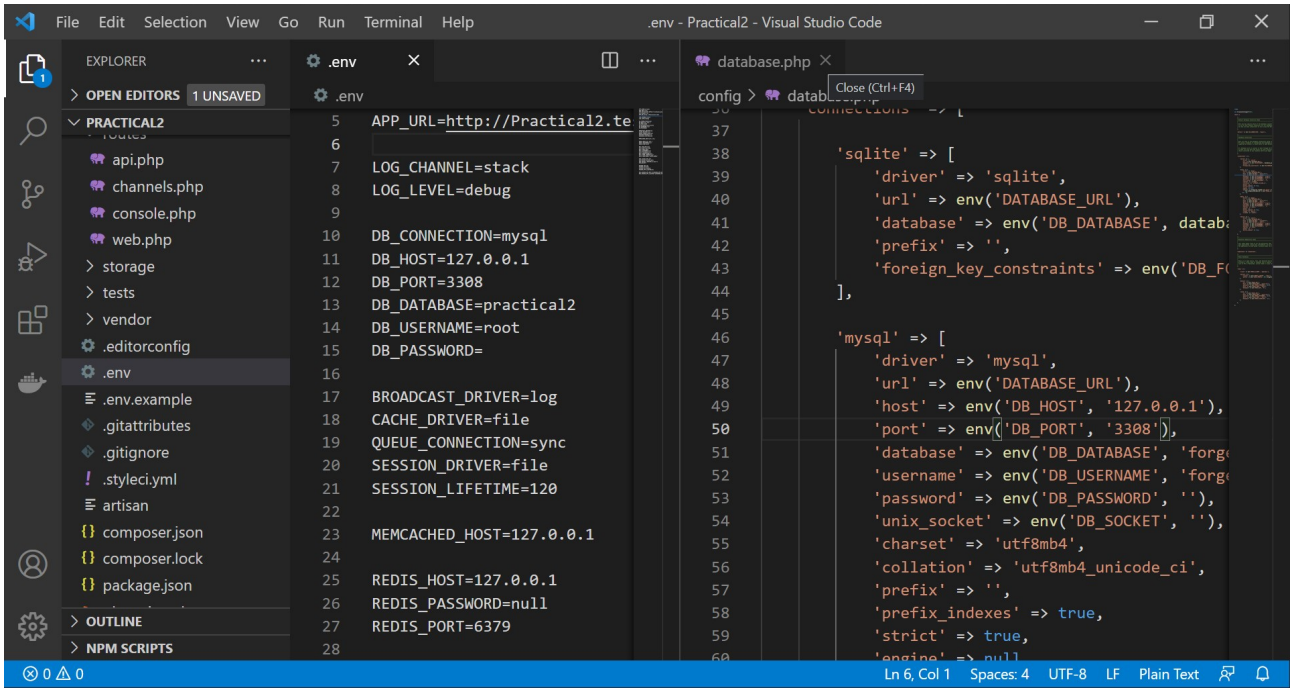


Figure 1: Database configuration file and lists for Laravel Web Application's database.

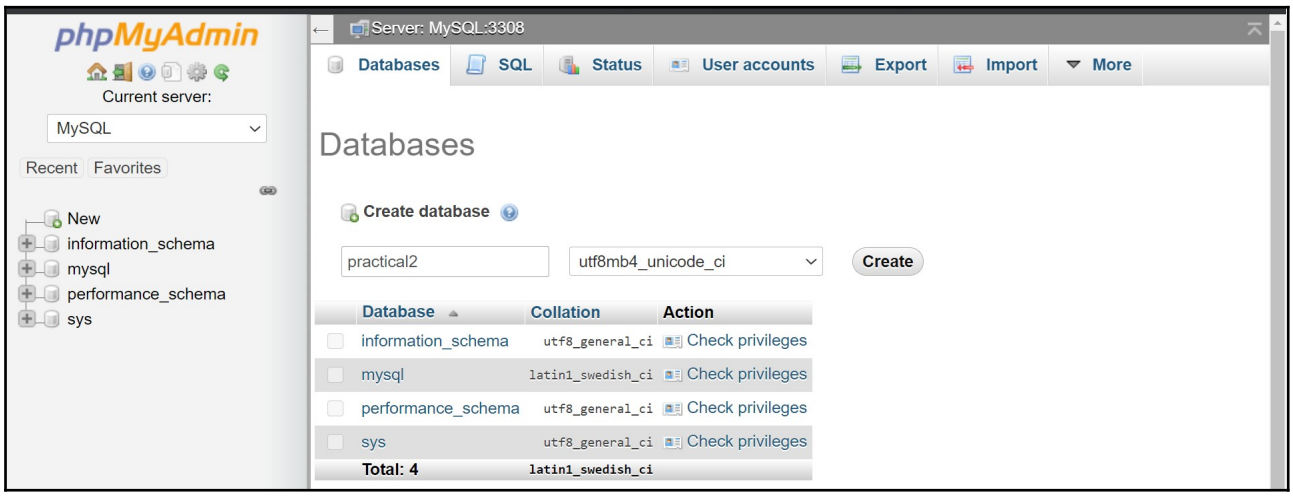


Figure 2: MySQL database to be used by Laravel Web Application.

In order to expound interacts with the web application's database, create a new database as illustrated in above Figure 2. Then, create a users table and insert some data into the table for testing, as shown in Figure 3.

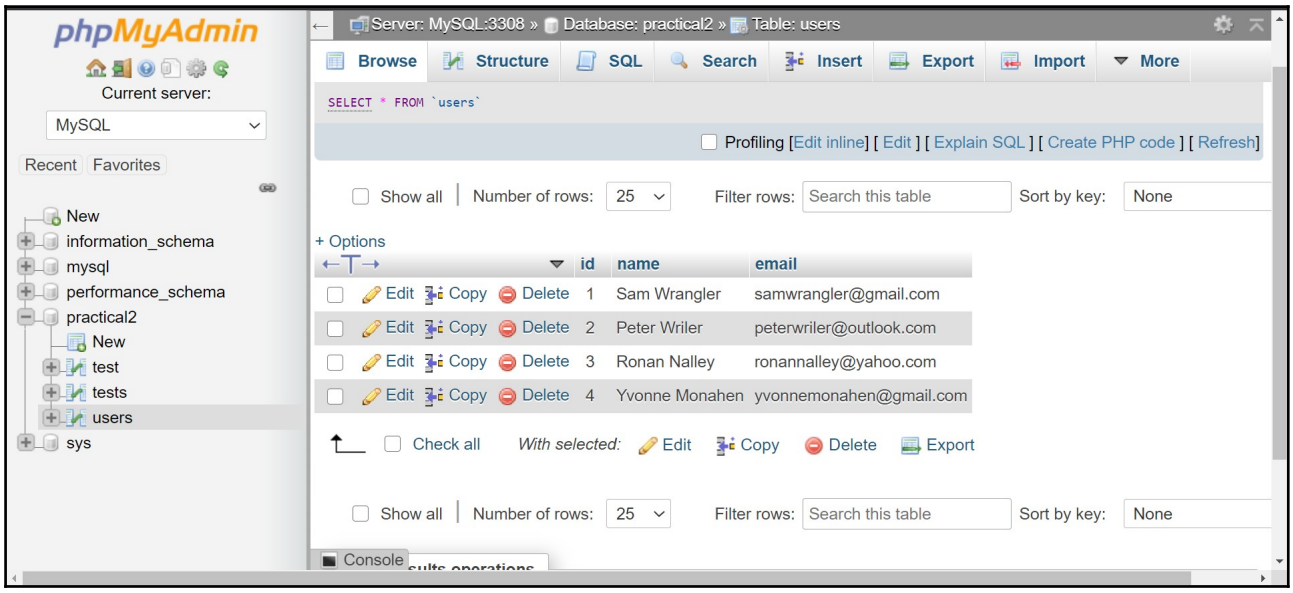


Figure 3: A users database table.

Next, we explore concept of raw SQL queries in Laravel Framework using Controller. Create a simple data fetch function and output with raw SQL queries within the UserController created previously as shown in Figure 4.

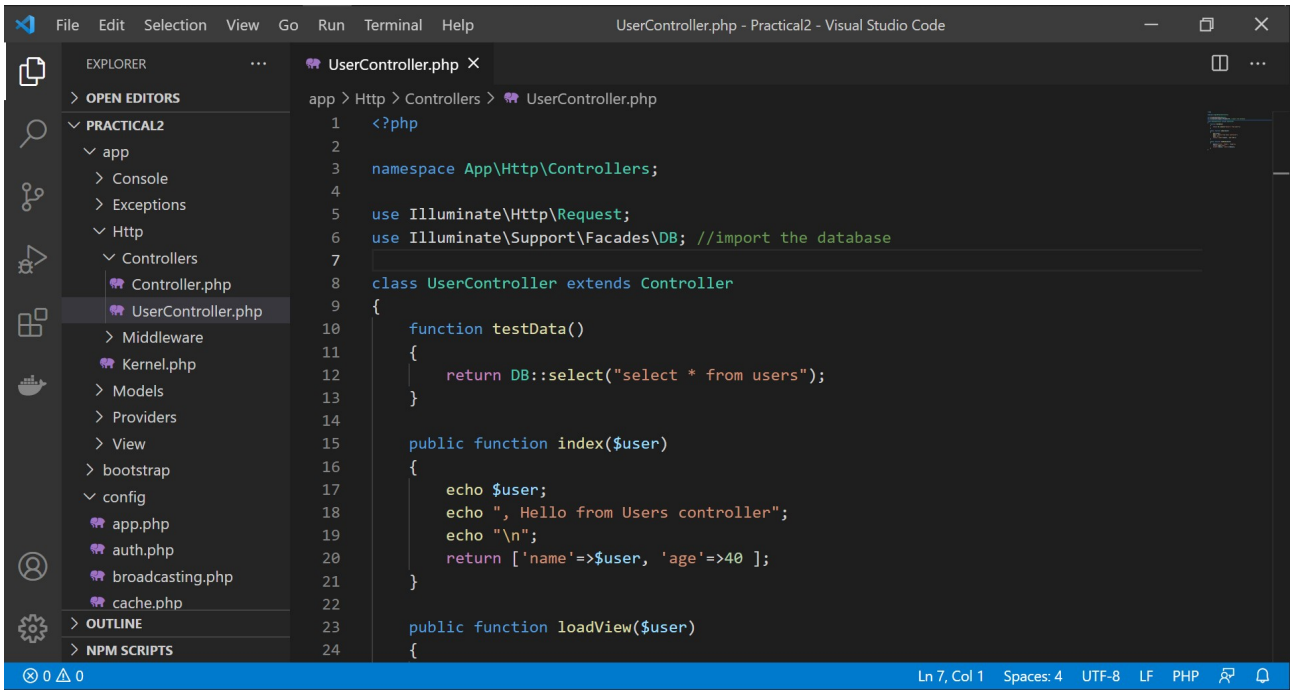


Figure 4: Raw SQL queries for data fetch using a controller.

After having the simple data fetch function and output in UserController, create a route to it so that the output can be seen in view result of the web application as shown in Figure 5.

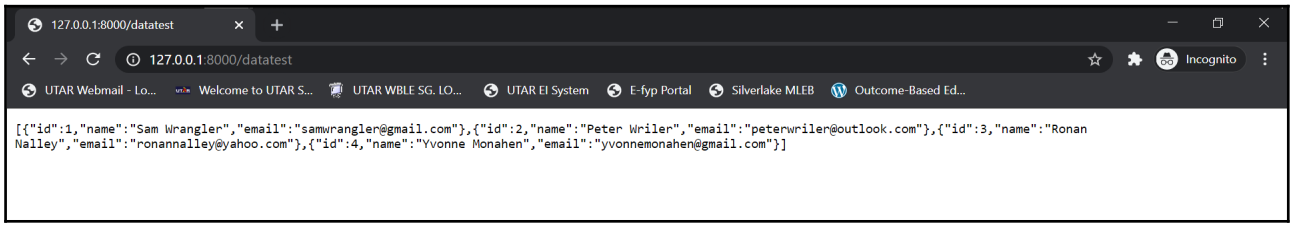


Figure 5: Fetch data from practical2 database users table using controller.

2. Model.

Model is the interface for database of any MVC architecture. Model basically fetch requirement from Controller and model the required data by fetching data from database. Laravel Model contains connection of Laravel web application with database, eloquent object-relational mapping (ORM), database structure and application logics.

Eloquent ORM feature in Laravel Framework enables Laravel web application to map database table with class name. A general rule of class naming for eloquent ORM to be done is: plural name for database table, while singular name for model class name. For instance, database name "users" will imply that the model name is "user" and if database name "employees" will imply that the model name is "employee". In case if a web developer insist on same name for database table and model class, a further configuration is required.

In order to explore the model component in Laravel application, let's create a model in Laravel web application to interact with users database table.

How to create a model in Laravel web application? There are two ways for doing so.

(a) Through the Artisan CLI "php artisan make:model". Figure 6 shows the example of creating "User" model.

Figure 6: Using Artisan CLI to create User model.

(b) Through manual "New File" creation within web application project folder.

Once the model is created, the model file can be found within the Models folder in app/Models directory. ***Take note: older Laravel versions placed Models folder in app/http/models directory*. Within the newly created User model, create a simple data fetching function from UserController as illustrated in Figure 7.

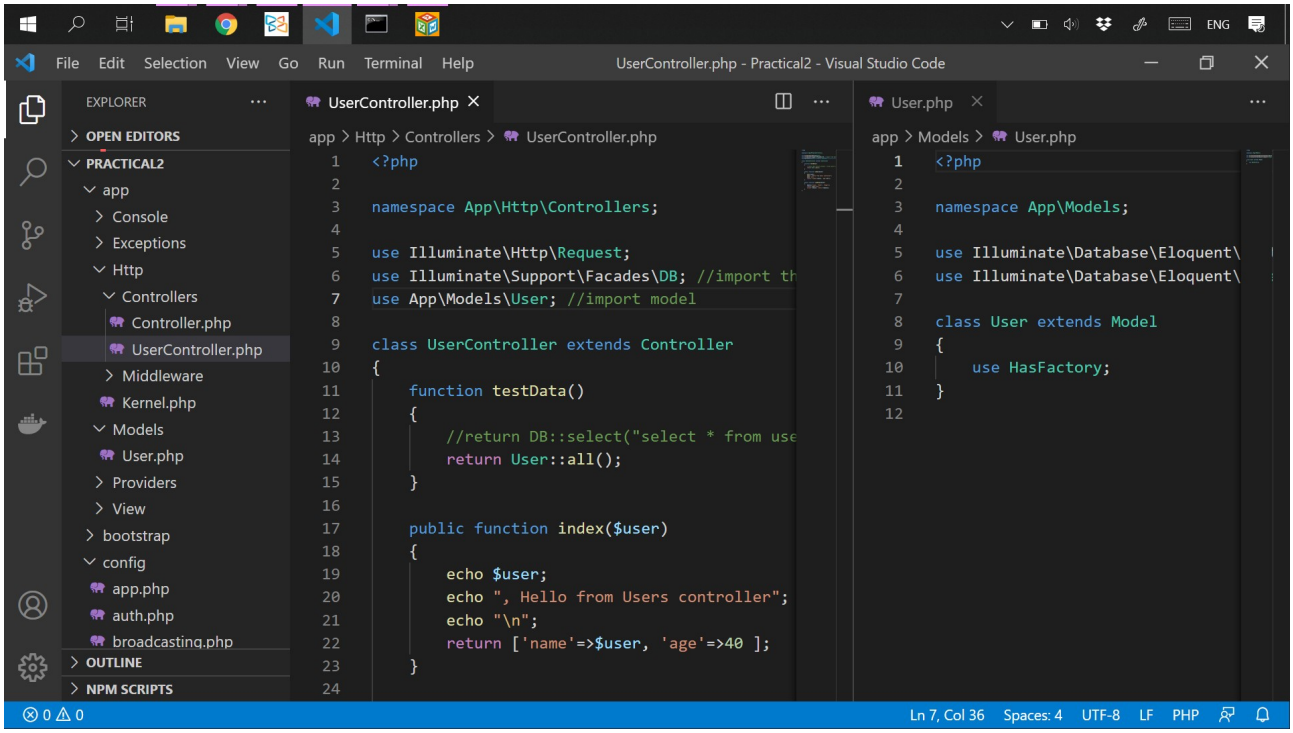


Figure 7: An echoing function in Users Controller.

Route to the controller and see that the data is automatically fetched from Users database table from User Model.

If in case a web developer insist on ignoring the eloquent ORM rule of Laravel framework (having the same name for database table and model class), connection to the database table from model class can still be done through a further configuration in the model class as shown in Figure 8.

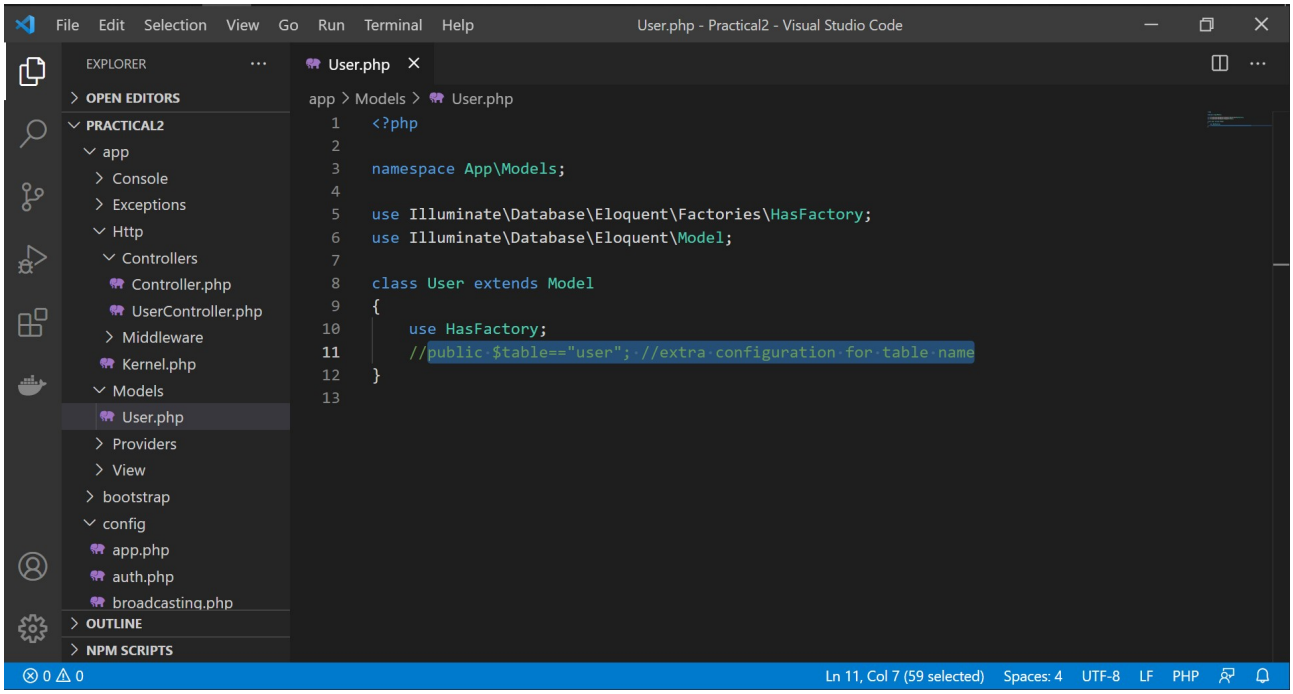


Figure 8: Extra configuration in model class for manual table name mapping.

Thus far, we've explored on fetching data from database using model as interface between controller and database. The output of the data fetch was shown using a return function in controller. In the following session of the practical, we will look into outputting the fetched data to view.

Showing a list of users to view.

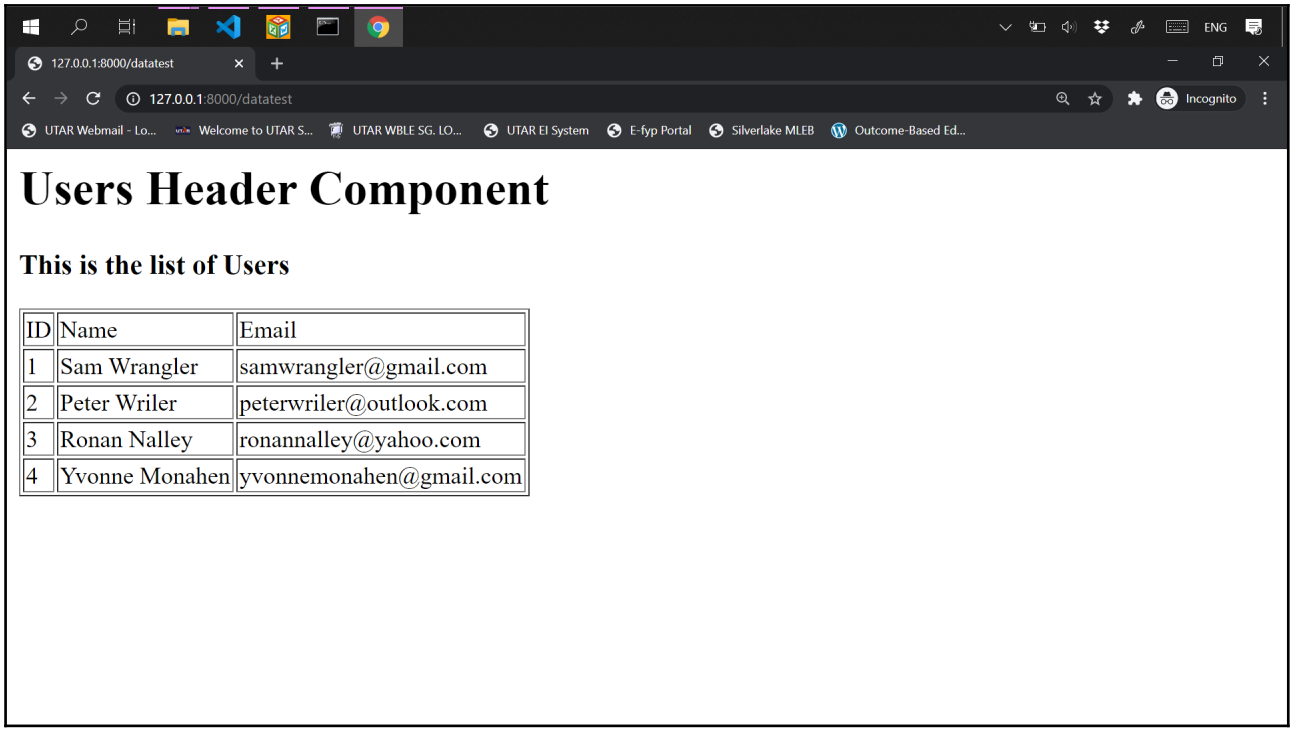
Within the controller, we know that "User::all()" contains all data from Users table. Previously, we used controller to return a view and passed some data to the view for output.

Exercise.

Based on the understanding and knowledge thus far;

1. Modify 'userInner' view to show a table; to list out the 'id', 'name' and 'email address' data.
2. Modify 'user' view in order to temporarily skip executing other php commands except for including 'userInner' view.
3. In controller, modify the testData() function to pass data into the returning view.

Your output can be as similar / as shown in figure below.



Use of Pagination.

In scenarios where the list of data is a lot, Laravel Framework offers a feature called 'pagination' to preset the number of data displayed in one page.

***In order to see the impact of having pagination, let's add data to the Users table so that it contains at least 15 data.*

Use paginate() in User table initialization within UserController instead of all() to enable pagination as shown in Figure 9. Then, referring to the same Figure 9, modify the view to enable navigation to display the rest of the users that are in second and third page.

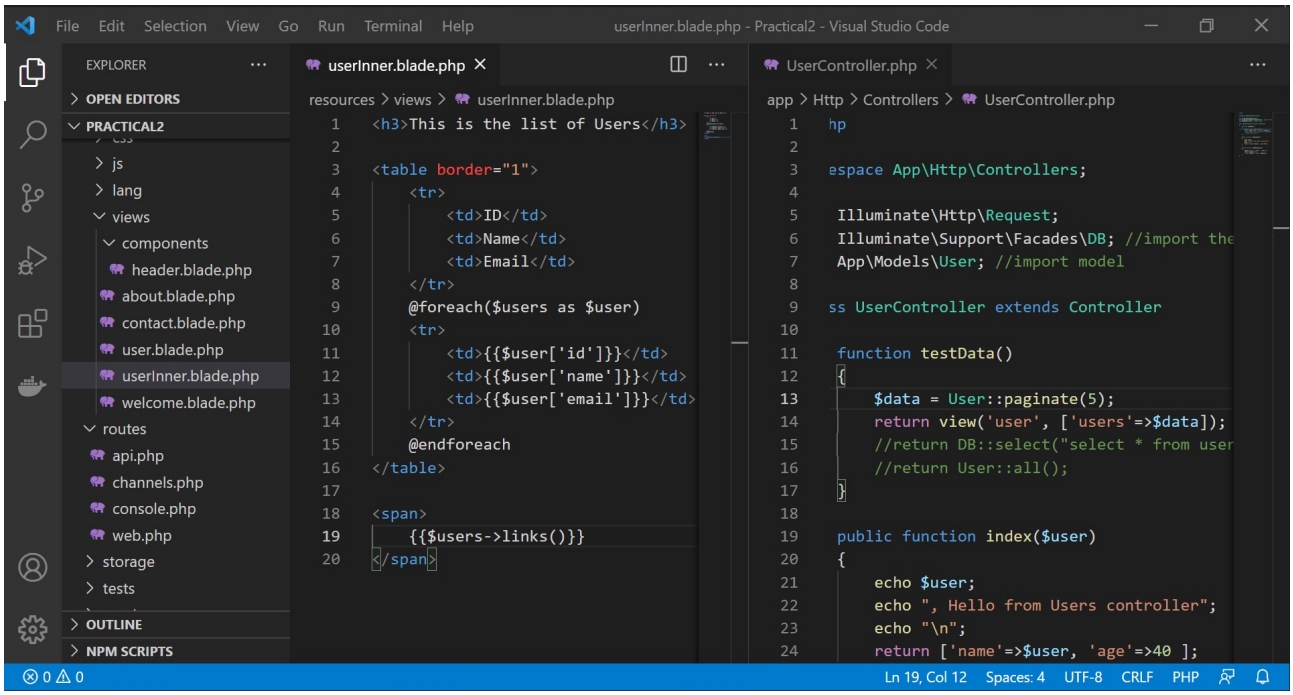


Figure 9: Applying pagination and displaying outputs in different pages.

As the result of applying pagination, the view should be as shown in Figure 10.

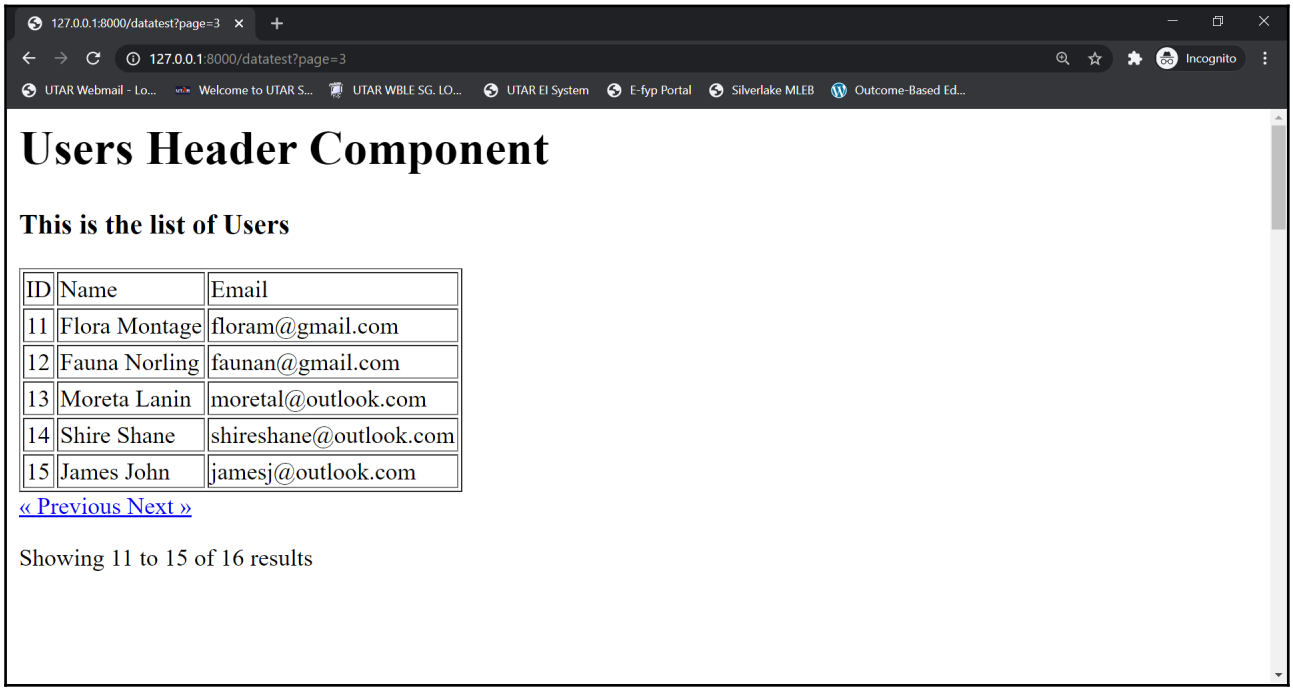


Figure 10: List of users with pagination.

**There is a mishandling of CSS within Laravel framework. In order to fix the mishandling, include following scripts to the view file.

```
<style>
      .w-5{
            display: none
      }
</style>
```

Thus far, we've explored how data can be modelled using model then displayed to a view that is invoked by a controller. Additional use of pagination to format the output data into different pages was explored as well. In the following session, let's explore on adding data into database.

Creating/Inserting Data into Database.

Firstly, let's create an interface for inputs; create a new addUser view with a simple form for name and email inputs which look like Figure 11. Then, create a view route for addUser.

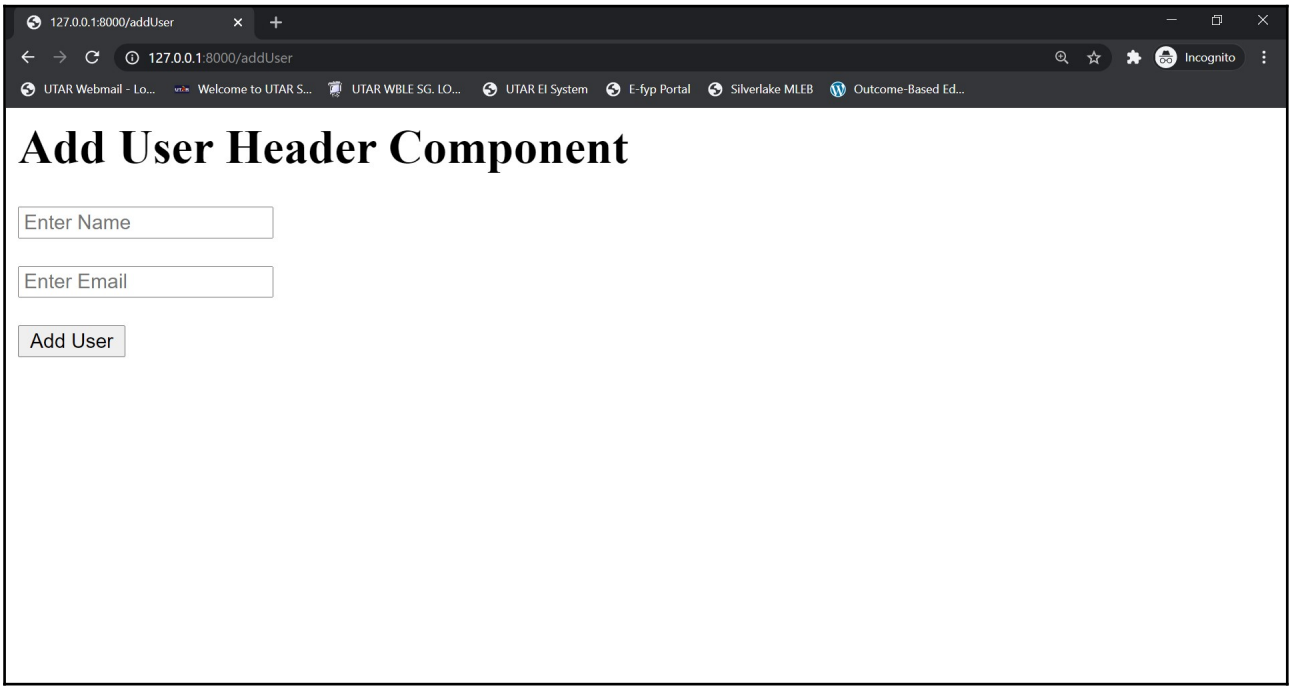


Figure 11: A simple form View for inputs.

Secondly, create an addUser function in UserController to fetch data input from the form as shown in Figure 12. Then, create a controller route to the controller's function.

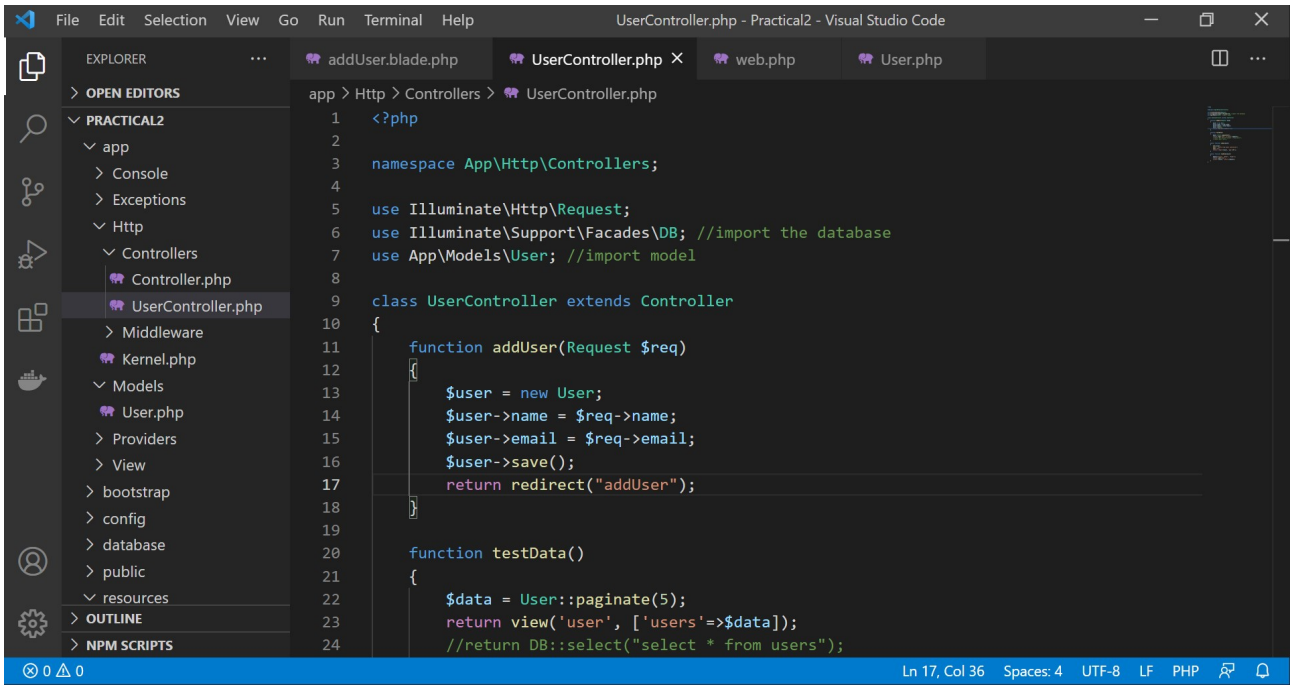


Figure 12: addUser function in UserController to process inputs.

Thirdly, ensure that User class has a public declaration of false for timestamps "`public $timestamps = false`" as Laravel expects every input to be accompanied by data of "updated_at" and "created_at", in which we can declare as false for Users table does not have this two fields.

Apart from outputting data to view, creation / insertion of data into database, deletion of data is a crucial function to be made available in a web application. Data deletion will be explored in the following session of practical.

Deleting Data from Database.

Looking back at the previous view for paginating all data from Users database table, add another column to the table to contain a “Delete” anchor as shown in Figure 13. Clicking the anchor will enable user to delete data of the same row.

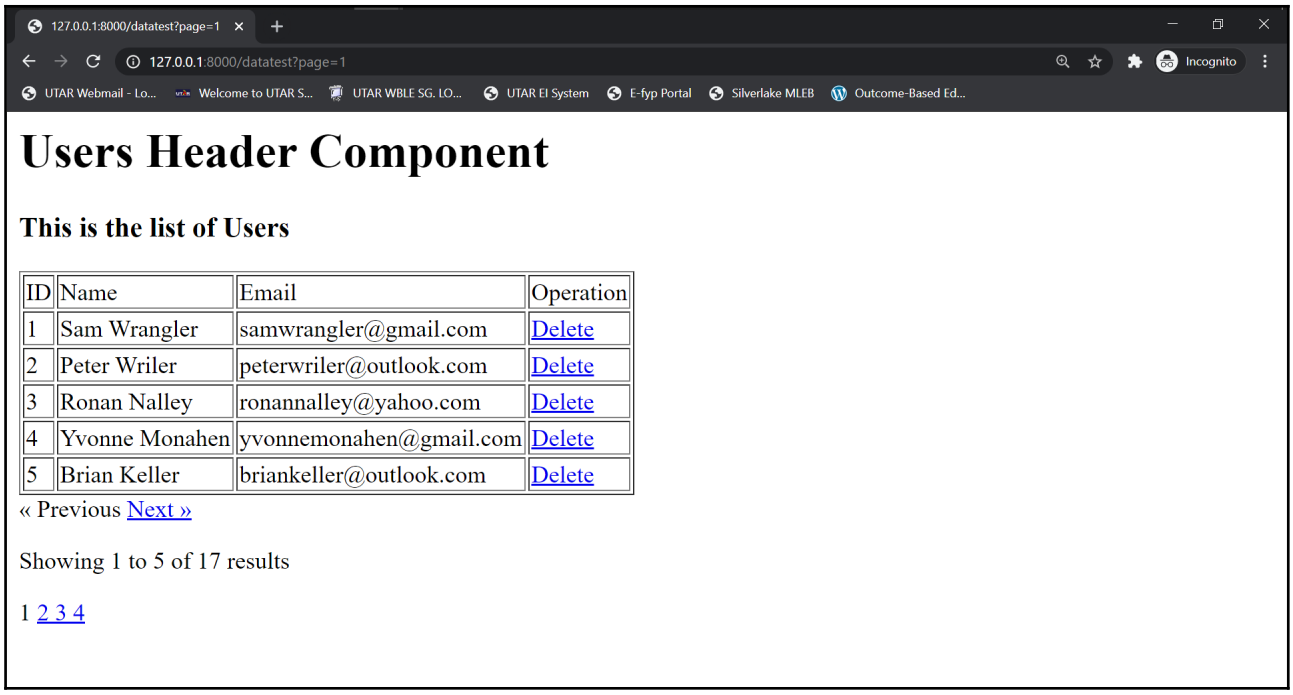


Figure 13: Delete function in userInner view for data deletion.

Then, create a deleteUser function in controller as well as the route to the controller as shown in Figure 14.

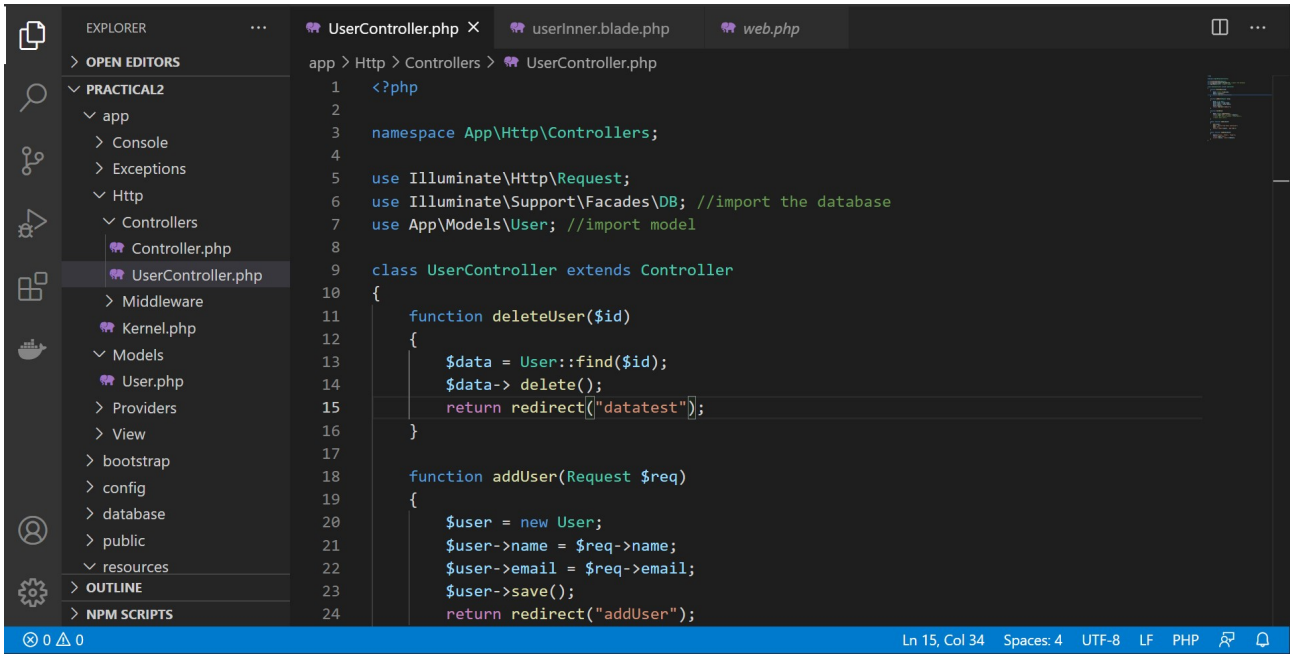


Figure 14: deleteUser function in UserController to process data deletion.

The final function that need to be included in a web application that deals with database will be updating existing data in a database. The following session will explore update of a data in Users database table.

Updating / Editing Data in Database.

Looking back at the previous view for deleting data from Users database table, add another column to the table to contain an “Update” anchor as shown in Figure 15. Clicking the anchor will enable invoke a form for user to update / edit the existing data.

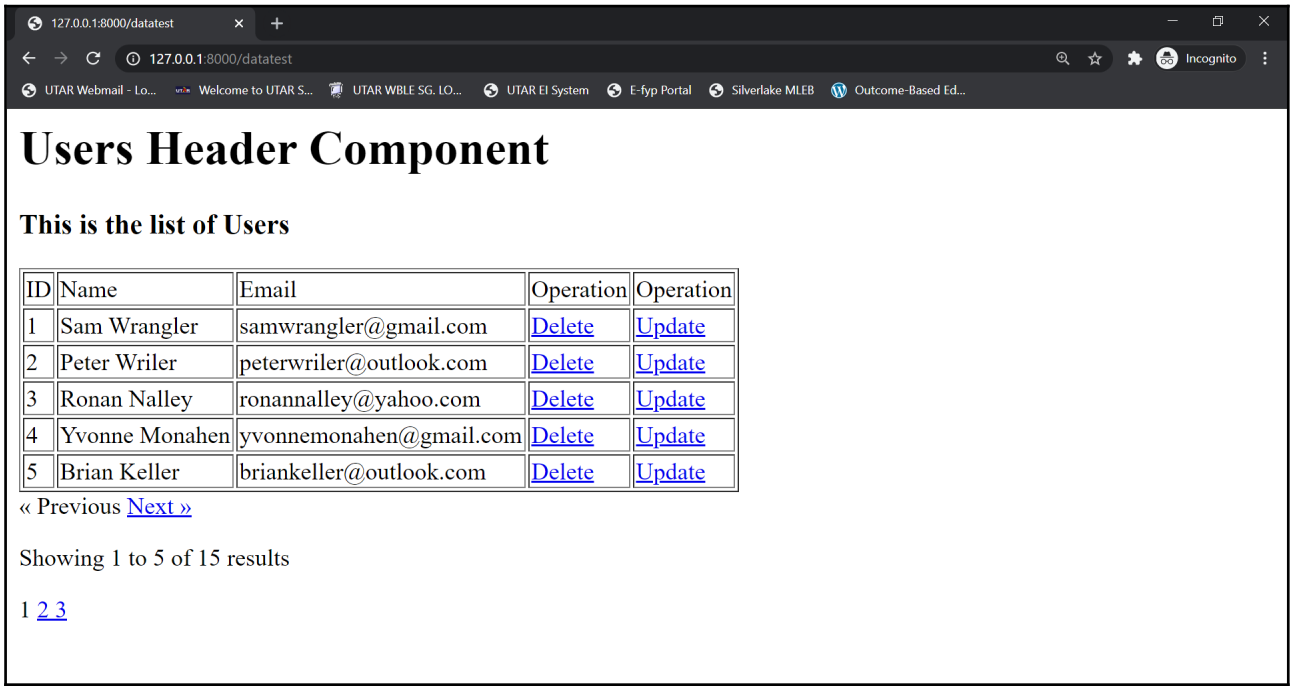


Figure 14: updateUser function in userInner view for data update.

Then create an “updateUser” form view, which will be invoked from the event of “Update” anchor being clicked to fetch data to be used to update a specific existing data as shown in Figure 15.

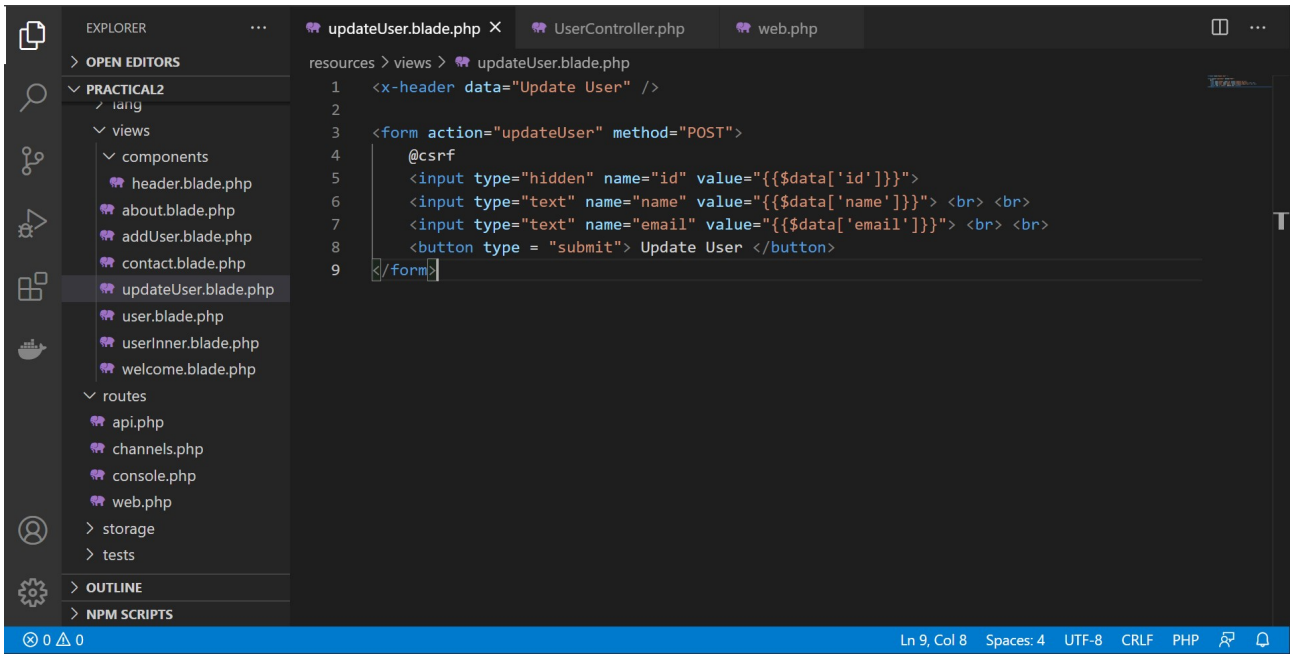


Figure 15: updateUser form view for data update.

Then, create two controller functions; one for showing the data selected for update into updateUser form, another one for updating logics as shown in Figure 16.

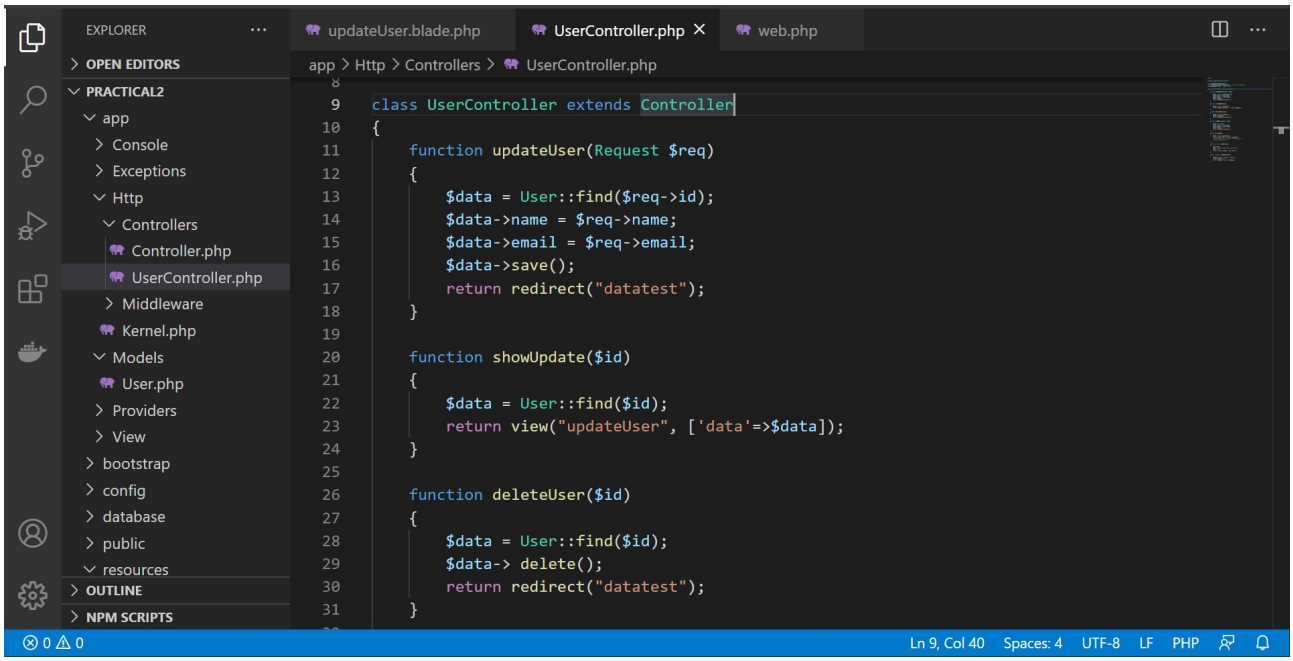


Figure 16: updateUser logics in UserController.

Finally, create the routes to the UserController show form and update logic functions as shown in Figure 17.

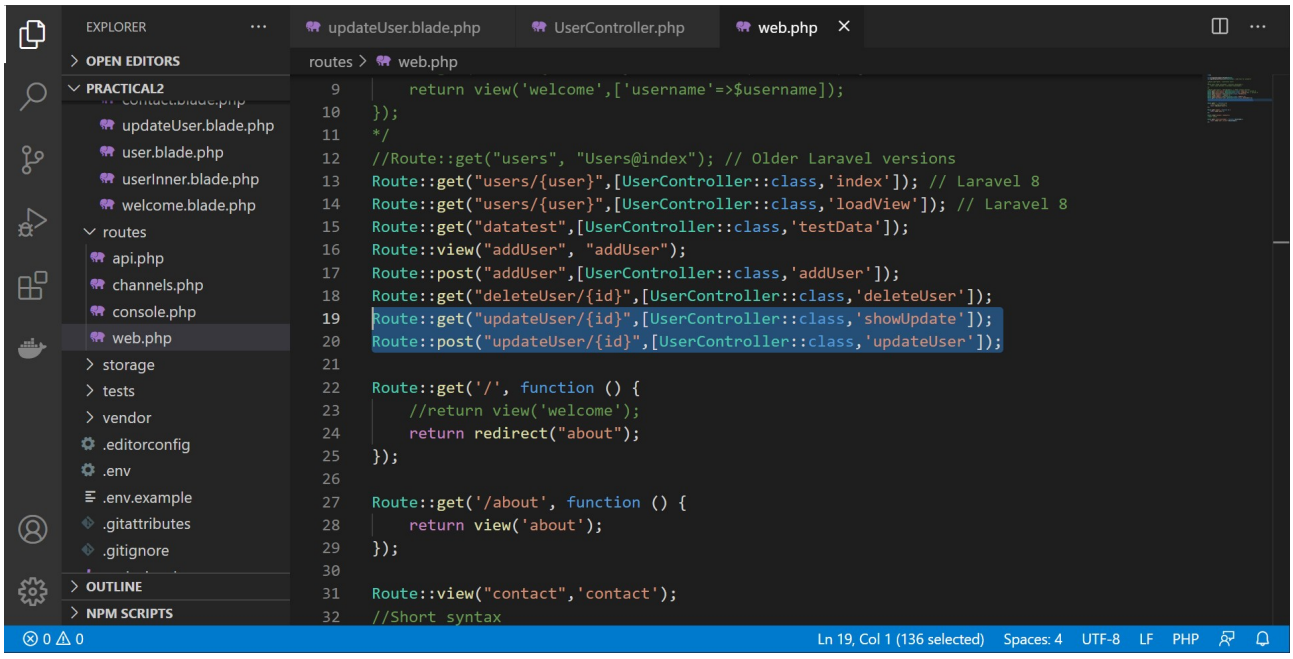


Figure 17: updateUser routes to UserController.

Thus far, main CRUD was explored on an existing database table “Users”. The following practical session expound on the concept of data migration.

Data Migration in Laravel.

Migration is a feature provided by Laravel framework for automating creation of database table. In order to explore the concept, create a new database table through Artisan CLI as shown in Figure 18.

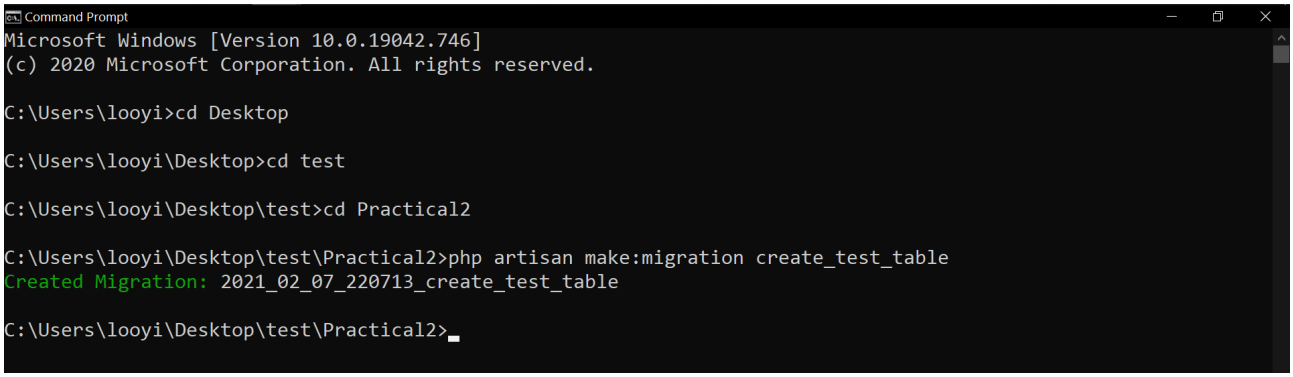


Figure 18: Data migration initiation using Artisan CLI.

The migration file created is located in \Database\Migrations. Within the migration file, further define the database table structure as shown in Figure 19.

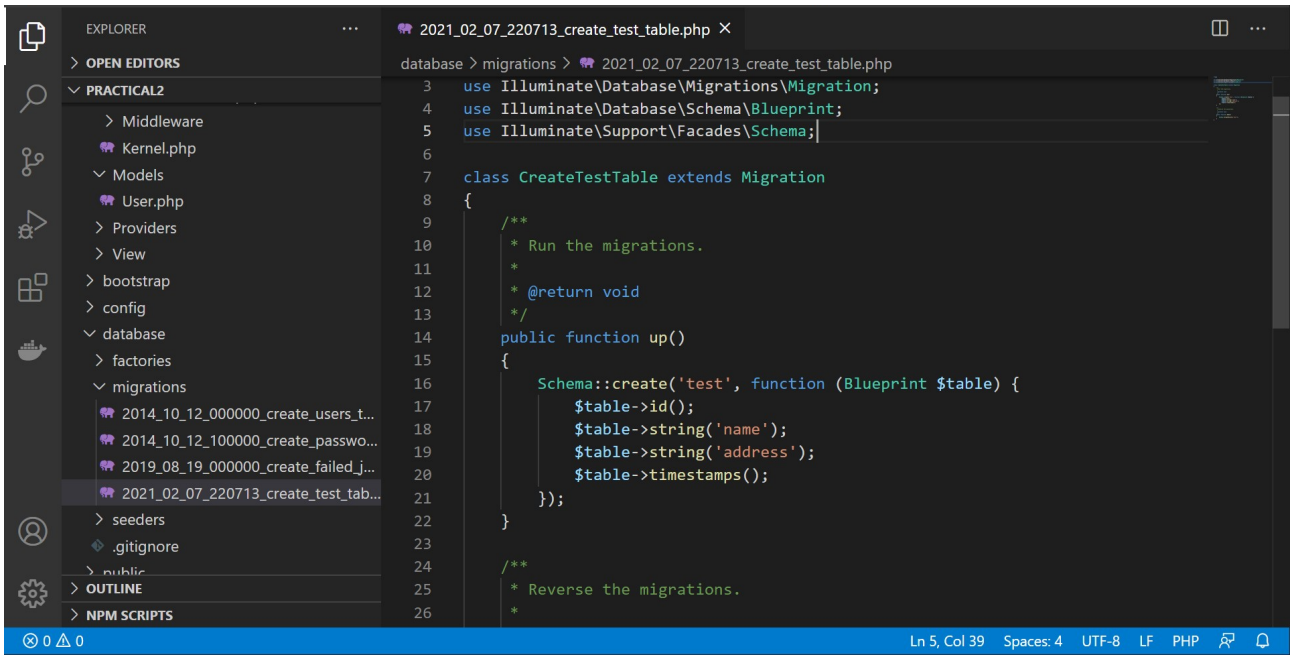


Figure 19: test database table structure.

In order to automate the creation of the table, execute Artisan CLI migrate command as shown in Figure 20.

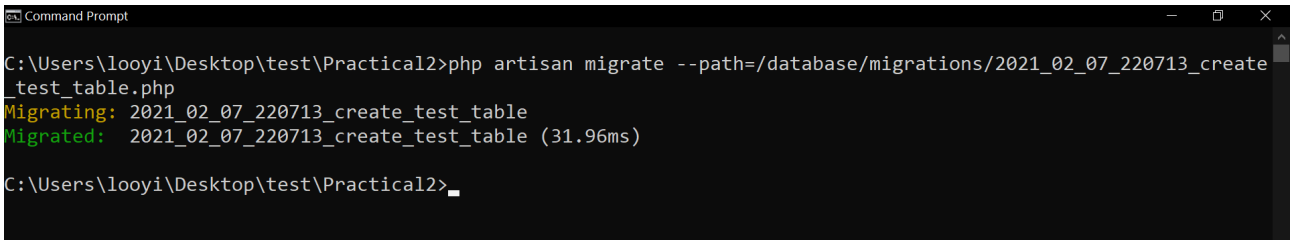


Figure 20: Migrate specific table with Artisan CLI.

Migration only create database table structure but not data. Following practical session expound on the automation of data creation with Laravel data seeding feature.

Data Seeding in Laravel.

The concept of data seeding is adding dummy data into a database table, in which, is a good practice for testing purpose. Data seeding file need to be first created using Artisan CLI as shown in Figure 21.

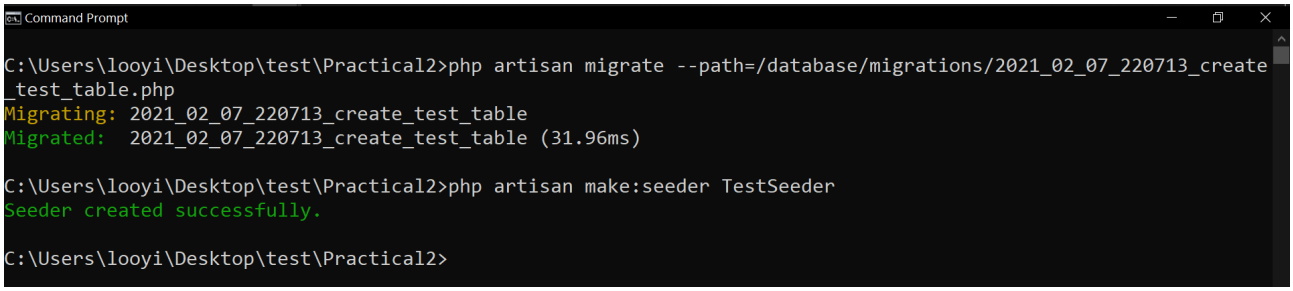


Figure 21: Creating data seeding file with Artisan CLI.

The seeder file created is located in Database\Seeders. Within the seeder file, further define the import of database that the seeder file will interact with and dummy data that should be generated. After specification of data seeding, execute Artisan CLI to execute data seeding as shown in Figure 22.

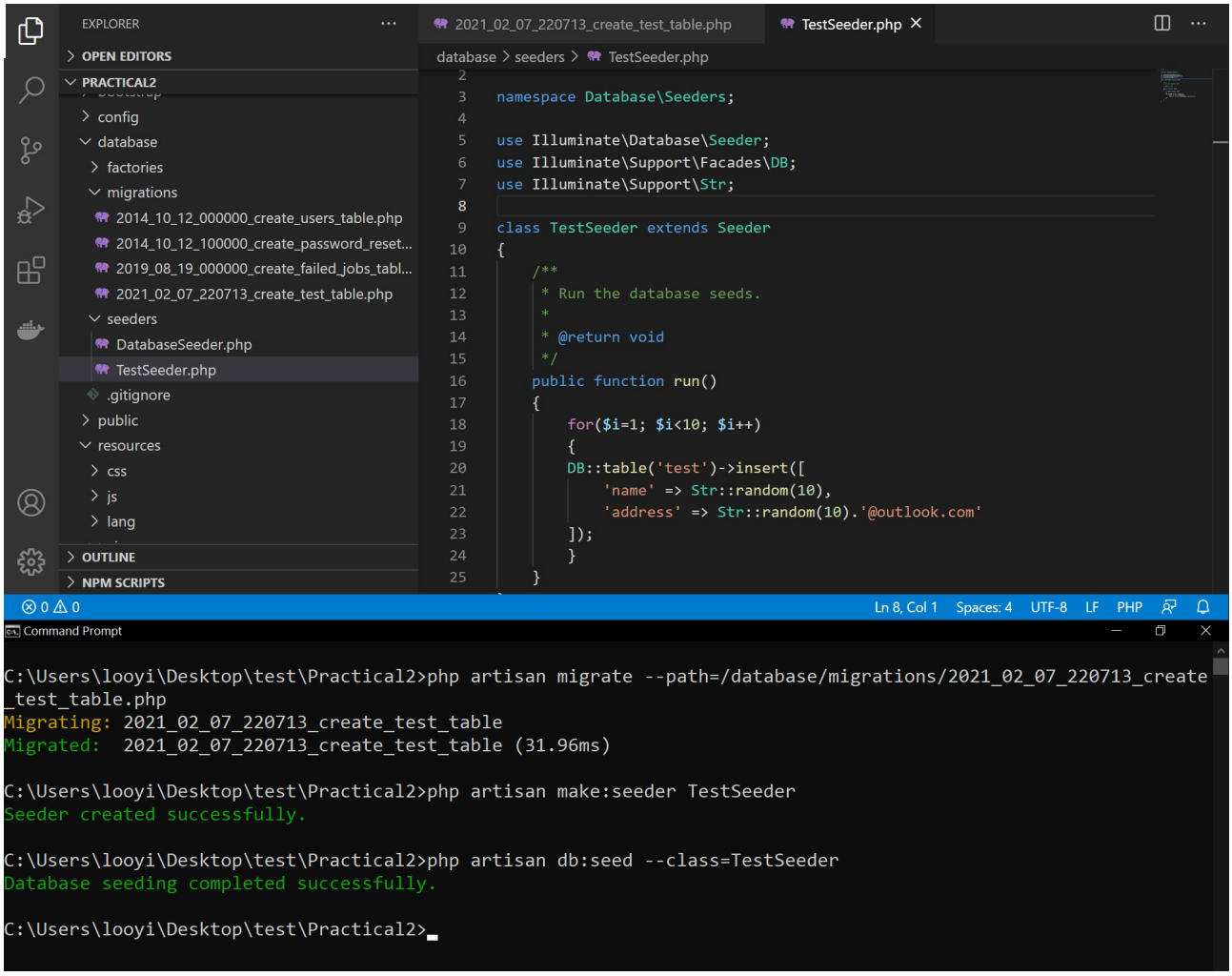


Figure 22: Data seeding specification and execution.