**WEEK-1**

**Composer**

Composer is a dependency manager for PHP that simplifies the process of managing and installing external libraries or packages that a PHP project relies on. It also keeps the track of the dependencies of those required components or packages as well as creates the class map for the downloaded libraries. It is also used to install Laravel framework in the system. Like npm for Node.js and bundler for Ruby.

composer require spatie/yaml-front-matter

composer create-project laravel/laravel project\_name

**Dependency**

A dependency is a relationship between two pieces of code where one piece of code (the dependent code) relies on another piece of code (the dependency) to function correctly.

Eg: Eloquent ORM library in Laravel framework that allows developers to work with databases using PHP objects, which can make it easier to write database-related code. Your application code depends on the Eloquent library to function correctly. If there is a change in Eloquent (e.g. a change in the way it handles database queries), your application code may be affected and may require changes to reflect those modifications.

Eg:

php artisan make:Model UserRegister // to create a model

protected $table = 'register'; // table name

protected $fillable = ['name', 'description', 'price']; //the attributes that can be filled

Using the create function on UserRegister class in the run method of the Seeder class. The create function auto timestamps the date and time of createdAt and UpdatedAt but the insert function will just put to null. Both the function does autoincrement the primary key.

UserRegister::create([

'name' => 'shyam',

'password'=>'qweawsdsf12345',

'email' => 'raadsfdsm12345@gmail.com',

'addresss'=>'kalimati',

]);

Then run the: php artisan db:seed command.

**Schema**

In a database, a schema is a collection of logical structures that define how the database is organized. It includes the description of tables, columns, relationships, and other database objects. In other words, a schema represents the blueprint or plan for a database. A schema defines the structure of a database, while a migration is a way to modify that structure over time.

Creating register entity

Schema::create('register', function(Blueprint $table){

$table->id();

$table->string('name');

$table->string('password') ->unique();

$table->string('email')->unique();

$table->string('addresss');

$table->timestamps();

});

Creating login table with foreign key

Schema::create('login', function(Blueprint $table){

$table->id();

$table->string('email');

$table->string('password');

$table->timestamps();

$table->foreign('password')->references('password')->on('register')->onDelete(null);

$table->foreign('email')->references('email')->on('register')->onDelete(null);

});

This will create register table in the database with id, name, password, email being unique and timestamps of createdAt and updatedAt.

**Migration**

Migrations are like version control for your database, allowing your team to define and share the application's database schema definition. The CRUD operation can be done and check and undo the process with the help of migration. It is like GIT for the database.

php artisan make:migration Register // to create migration

put the schema CRUD operation in the up/down method:

Schema::create('register', function(Blueprint $table){

$table->id();

$table->string('name');

$table->string('password') ->unique();

$table->string('email')->unique();

$table->string('addresss');

$table->timestamps();

});

php artisan migrate // to run all migrations or single migration

php artisan migrate --path=/database/migrations/full\_migration\_file\_name\_migration.php

**Seeders**

Laravel includes the ability to seed your database with data using seed classes. All seed classes are stored in the database/seeders directory.

php artisan make:seeder UserSeeder // to create seed

Insert the below code in the UserSeeder class of run() method

DB::table('register')->insert([

'name' => 'ram',

'password'=>'qwe12345',

'email' => 'ram12345@gmail.com',

'addresss'=>'kalimati',

]);

You need to write this line of code in the DatabaseSeeder.php where UserSeeder is your seeder name.

$this->call(UserSeeder::class);

Then run the following command.

Php artisan db:seed

**Entity**

An entity is a concept that represents a real-world object or concept within an application's domain. An entity can be a person, place, thing, or event that the application needs to keep track of and manipulate in some way.

Entities are often represented as objects in object-oriented programming languages or as rows in a database table. In either case, an entity typically has a unique identifier and a set of attributes that describe its properties.

For example, if the register table represents individual users, then each row of the table would represent a single user entity, with each column containing data about that user.

Id => 1

'name' => 'shyam',

'password'=>'qweawsdsf12345',

'email' => 'raadsfdsm12345@gmail.com',

'addresss'=>'kalimati',

‘createdAt => ‘2023-03-19 11:05:34’,

‘updatedAt => ‘2023-03-19 11:05:34 ‘

**Package Control**

In app directory, there is model and http package. The http package contains the controller and middleware. Laravel follows the MVC pattern where the Model and Controller is in the app directory. There is database directory where there is migration and seeder package. Migration helps to manage the database table structures. Seeder help to populate the data into the tables of database.The most important package is the routes. To go from model to view and controller to view, the web.php helps to display the information to the user. There is also api.php to handle the apis in the routes package. Inside the resource directory, there is the view package which contains the php or blade template. The .env file sets the environment file for the website and make connection details and more. The composer.json stores the information of the packages installed in the project.

WEEK2

**Login and Register page**

Register page has a form where user is requested to put name, email and password for registration.

$admin = new User();

$admin->name = $request->name;

$admin->email = $request->email;

$admin->password = Hash::make($request->password);

$admin->save();

This is to create a user in the User table in database .

Login page is where user puts the credentials so they can access the system if they are valid.

The system checks the credentials to the User table. If valid, they are sent to dashboard else sent to the login page again with error.

**Dashboard page to show the current customers information**

It has a navbar which contains a logo, home , add, move to trash and logout section in it. The information about the customers are shown in the tabular form where we are able to edit or delete the existing customer.

**CRUD operations of customers where delete should be softdelete**

**Create**

$customer = new AdminCustomer;

$customer->name = $request['name'];

$customer->email = $request['email'];

$customer->save();

It is to create a customer and store in database after creating an object of Eloquent Model.

**Read**

$customer = AdminCustomer::all();

We can use $customer object to access the customer information by looping the $customer to access all customers.

**Update**

$customer = AdminCustomer::find($id);

$customer->name = $req['name'];

$customer->gender = $req['gender'];

$customer->save();

It is to access the existing user info by using find method which searches the id and changes the information from the request.

**Delete**

To only soft delete the information so that it is not shown when accesss all rows from customer table but is stored in database. We do following steps:

1. Paste the code inside the Model of the table which needs soft delete i.e AdminCustomer.php for now.

use Illuminate\Database\Eloquent\SoftDeletes;

2. Paste “use SoftDeletes;” inside the AdminCustomer class

3. php artisan make:migration add\_deleted\_at\_to\_admin\_customers\_table – to create an column on existing table

4. $table->softDeletes(); inside the Schema function and run “php migrate”

5. So when we use this code: AdminCustomer::find($id)->delete(); It only soft deletes the information and deleted\_at columns has the soft delete time of that information

**Able to restore or hard delete the soft-deleted information of customer**

To restore the soft deleted information we use the following command.

$customer = AdminCustomer::withTrashed()->find($id);

$customer->restore();

To actually permanently deleting the information, we use this command.

$customer = AdminCustomer::withTrashed()->find($id);

$customer->forceDelete();

P.S : AdminCustomer::onlyTrashed()->get(); it returns the information of customers who are soft deleted only.

**Download the customer informations into the excel file**

1st step is to install the package which is : composer require maatwebsite/excel

After installing it. Use this command : php artisan make:export ExportUsers --model=AdminCustomer to create exports directory where it will export the Model you typed

In the ExportUsers, use this function:

public function collection()

{

return User::all();

}

Where ExportUsers must implements FromCollection to use the collection() function.

Finally call the class by typing: return Excel::download(new UsersExport, 'users.xlsx');

Which makes us to be able to export the customer information in the excel sheets.

**Basic validation**

**2 days gone on making auth middleware with session= failed**

**WEEK-3**

**Code optimization i.e. add and edit methods to do its tasks in single method**

**Pass to view function by using compact() instead of the with() function**

Old-> $data = compact('url', 'title', 'customer');

return view('admin\_add')->with($data);

New->return view('admin\_add', compact('url', 'title', 'customer'));

**Use all the input types found in the form of html**

**Retaining old values when editing the information of existing customers**

**Import the excel file and store it in database( with and without headings of many entries)**

If you have export the excel before , you dont need to do lots of tasks. You can just type:

php artisan make:import ImportUsers --model=User where User=Model

To enable multiple datas to be imported we need to put the protected fillables of our Model class i.e.

protected $fillable = ['name','gender','email','address','hobbies','blood\_group','file','description'];

Else there occurs errors during import process.

Then use this command to be able to import the excel :

Excel::import(new UsersImport, $request->file('file'));

It is to import the file that came from the browser where ‘file’ is the filename of the input type of file.

Finally, in the UsersImport.php , the class UsersImport must implements ToModel and we should put the following code inside the class:

public function model(array $row)

{

return new AdminCustomer([

'name' => $row[0],

'gender' => $row[1],

'email'=> $row[2],

'file'=>$row[3],

]);

}

Or we can put this : 'name' => $row['name'] where it will take the column from the ‘name’ of the Excel heading and put into ‘name’ key.

**Exporting in excel specific columns after selecting Model::all() in code.**

We can use this to export only specific column after selecting all the columns from Model.

public function collection()

{

$customers =AdminCustomer::all();

foreach($customers as $customer){

$data\_array[] = array(

'Name' =>$customer->name,

'Gender' => $customer->gender

);

}

return collect($data\_array);

}

We use all() to get all rows from AdminCustomer table and store in $customers. We then use foreach loop to get all the customers. Now, we create an array which consists of another array where the name and gender is the key and their respectives are the values. It is the nested array. Lastly, the array is changed into collection by collect() because we are using collection() so we need to return collection. If we try to return array, it will generate error.

**Popup after doing an event.**

return redirect('/dashboard')->with('success','Product successfully added.');

**Form validation**

**Changing default folder structure to HMVC**

**Software design patterns**

**Maintaining logs**

To create log file that is custom made which is stored in custom directory, we can do the following:

Go to logging.php in config directory and write following code inside it.

'custom' => [

'driver' => 'single',

'path' => storage\_path('logs/custom.log'),

'level' => 'debug',

'replace\_placeholders' => true,

],

And to access the logs and create logs we can do this:

Log::channel('custom')->info('Customer has been soft deleted',['id'=>$customer->id,'email'=>$customer->email]);