**MODULE – 9, 10(Forms, Controls)**

1. Explain ORM

=>Laravel includes Eloquent, an object-relational mapper (ORM) that makes it enjoyable to interact with your database. When using Eloquent, each database table has a corresponding "Model" that is used to interact with that table.

=> Eloquent ORM is the very powerful yet very expressive ORM, which allow us to work with the database objects and relationships using much eloquent and expressive syntax.

=> In Laravel, each database table is mapped into corresponding eloquent model and each of the eloquent model object include various methods for retrieving and updating the database.

1. Explain - Eloquent Relationships

=> Database tables are often related to one another. For example, a blog post may have many comments or an order could be related to the user who placed it. Eloquent makes managing and working with these relationships easy, and supports a variety of common relationships: -

* One To One

=> A one-to-one relationship is a very basic type of database relationship. For example, a User model might be associated with one Phone model.

* One To Many

=> A one-to-many relationship is used to define relationships where a single model is the parent to one or more child models.

* Many To Many

=> Many-to-many relations are slightly more complicated than hasOne and hasMany relationships.

* Hash One Through

=> The "has-one-through" relationship defines a one-to-one relationship with another model. However, this relationship indicates that the declaring model can be matched with one instance of another model by proceeding through a third model.

* Hash Many Through

=> The "has-many-through" relationship provides a convenient way to access distant relations via an intermediate relation. For example, let's assume we are building a deployment platform like [Laravel Vapor](https://vapor.laravel.com/).

* One To One(Polymorphic)

=> A one-to-one polymorphic relation is similar to a typical one-to-one relation; however, the child model can belong to more than one type of model using a single association.

* One To Many(Polymorphic)

=> A one-to-many polymorphic relation is similar to a typical one-to-many relation; however, the child model can belong to more than one type of model using a single association.

* Many To Many(Polymorphic)

=> Many-to-many polymorphic relations are slightly more complicated than "morph one" and "morph many" relationships.

3.What is Eager Loading and lazy loading?

=> While lazy loading delays the initialization of a resource, eager loading initializes or loads a resource as soon as the code is executed. Eager loading also involves pre-loading related entities referenced by a resource. For example, a PHP script with an include statement performs eager loading—as soon as it executes, eager loading pulls in and loads the included resources.

4. Do Session for Employee Management System.

1. **Set Up a New Laravel Project:** If you haven't already, create a new Laravel project using Composer. Open your terminal and run:

bashCopy code:-

composer create-project --prefer-dist laravel/laravel employee-management-system

composer create-project --prefer-dist laravel/laravel employee-management-system

1. **Create Employee Model and Migration:** Generate a migration and model for the **Employee** entity:

bashCopy code

php artisan make:model Employee -m

php artisan make:model Employee -m

In the generated migration file (e.g., **database/migrations/yyyy\_mm\_dd\_create\_employees\_table.php**), define the schema for the **employees** table. For simplicity, let's assume we have **name**, **email**, and **designation** columns:

phpCopy code

public function up()

{

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

$table->id();

$table->string('name');

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

$table->string('designation');

$table->timestamps();

});

}

Then, run the migration to create the table:

bashCopy code

php artisan migrate

php artisan migrate

1. **Create EmployeeController:** Create a controller for managing employees:

bashCopy code

php artisan make:controller EmployeeController

php artisan make:controller EmployeeController

In the **EmployeeController**, you can define actions to manage employees such as creating, listing, editing, and deleting.

1. **Define Routes:** In the **routes/web.php** file, define routes for your employee management actions. Here's a basic example:

phpCopy code

Route::get('/employees', 'EmployeeController@index');

Route::get('/employees/create', 'EmployeeController@create');

Route::post('/employees', 'EmployeeController@store');

Route::get('/employees/{id}/edit', 'EmployeeController@edit');

Route::put('/employees/{id}', 'EmployeeController@update');

Route::delete('/employees/{id}', 'EmployeeController@destroy');

1. **Create Views:** Create Blade views for your employee management actions. Place these views in the **resources/views** directory. For example, create **create.blade.php**, **edit.blade.php**, **index.blade.php**, etc.
2. **Implement EmployeeController Actions:** In the **EmployeeController**, implement the actions for creating, listing, editing, and deleting employees. You'll use Laravel's session storage to manage employee data. Here's a simplified example for creating and listing employees:

phpCopy code

public function create()

{

return view('employees.create');

}

public function store(Request $request)

{

$employee = $request->only(['name', 'email', 'designation']);

session()->push('employees', $employee);

return redirect('/employees');

}

public function index()

{

$employees = session('employees', []);

return view('employees.index', compact('employees'));

}

}

1. **Create Views for Employee Actions:** Create Blade views for creating, listing, editing, and deleting employees. These views will display and collect data from users.
2. **Display Employee Data in Views:** In your views, you can loop through the **employees** session data to display the employee information.
3. **Implement Editing and Deleting:** Implement the **edit** and **destroy** actions in your **EmployeeController** to allow for editing and deleting employees from the session.
4. **Run Your Application:** Start your Laravel development server:

bashCopy code

php artisan serve

php artisan serve

Access your application in a web browser and navigate to the appropriate routes you've defined for employee management.

This is a basic example of implementing an employee management system using Laravel's session storage. In a real-world application, you would likely use a database to store and manage employee data for better persistence and scalability.Top of Form

1. Do Curd using Eloquent Query?

**1.Create (Insert) a New Employee:**

To create a new employee record, you can use the **create** method on your model. Make sure to define the **fillable** property in your **Employee** model to specify which fields can be mass-assigned:

use App\Models\Employee;

// Create a new employee

$employee = Employee::create([

'name' => 'John Doe',

'email' => 'john@example.com',

'designation' => 'Developer',

]);

**2.Read (Retrieve) Employees:**

To retrieve employees, you can use various Eloquent methods, such as **get**, **first**, **find**, and **where**. Here are some examples:

* 1. Get all employees:

phpCopy code

$employees = Employee::all();

* 1. Get the first employee:

phpCopy code

$employee = Employee::first();

* 1. Find an employee by ID:

phpCopy code

$employees = Employee::where('email', 'john@example.com')->get();

* 1. Find employees that match certain criteria (e.g., by email):

phpCopy code

**3.Update an Employee:**

To update an existing employee, retrieve the record and then modify its attributes. After modifying the attributes, you can call the **save** method:

phpCopy code  
$employee = Employee::find(1);

$employee->name = 'Jane Doe';

$employee->email = 'jane@example.com';

$employee->designation = 'Manager';

$employee->save();

Alternatively, you can use the **update** method to update multiple records at once based on a condition:

phpCopy code

Employee::where('designation', 'Developer')

->update(['designation' => 'Senior Developer']);

**4.Delete an Employee:**

To delete an employee, retrieve the record and call the **delete** method:

phpCopy code

$employee = Employee::find(1);

$employee->delete();

You can also delete multiple records based on a condition:

phpCopy code

Employee::where('designation', 'Manager')->delete();

These are the basic CRUD operations you can perform using Eloquent queries in Laravel. You can customize and expand upon these operations to suit the specific requirements of your application. Additionally, you can use relationships, validation, and other Laravel features to enhance your CRUD operations.