INSTITUTE OF SCIENCE & TECHNOLOGY FOR ADVANCED STUDIES & RESEARCH

THE CHARUTAR VIDYA MANDAL (CVM) UNIVERSITY

MASTER OF COMPUTER APPLICATION

ACADEMIC YEAR:2025-2026(EVEN) SEMESTER: 2

PAPER CODE :101550234

PAPER TITLE : PRACTICAL BASED ON PHP FRAMEWORK

PREPARED BY : DR. NIKY JAIN

PRACTICAL LIST

1.) Develop a Laravel Web Application Based on the Following Instructions:

A. Project Setup

- Create a folder for your Laravel project.
- Open the terminal in that folder.

B. Create a Laravel Project

- Create a Laravel project using the terminal with a project name (e.g., blog).
- Choose "None" for the starter kit and press enter for default options.
- Select MySQL as the database.

C. Build and Serve the Application

- Move into the project directory.
- Run the necessary NPM and Artisan commands to serve the application.

D. Modify the Welcome Page

- Open welcome.blade.php and display your full name.

E. Create a New Blade View

- Create a new Blade file named home.blade.php.
- Display any custom heading in it.

F. Update Routes

- Define a route in web.php to load the new view when /home is accessed.
- Set this route to replace the default / route if needed.

2.) Perform the Following Tasks Related to Laravel Routing:

A. Routing Basics

- Create a Blade view file and display content via a defined route.

B. Define Routes

- Define a basic route using Route::get() method.
- Change the route path and access it via a browser.

C. Pass Data via Routes

- Create a route that accepts a parameter (e.g., /about/{name}).
- Pass and display this parameter in a Blade view using Blade syntax.

D. Anchor Tag with Routing

- Add anchor (<a>) tags inside a Blade view to link to other defined routes.

E. Redirection

- ➢ Create a route that redirects from one path to another using redirect().

F. Routing Methods : Explore the methods

3.) **Perform the Following Tasks and Implement Each One Practically:**

A. Create a Controller

- ➢ Create a controller using the command line (php artisan make:controller).

B. Call Controller Methods Using Routes

- ➢ Define a method inside the controller (e.g., getUser()).
- ➢ Connect the method to a route in web.php.

C. Pass Data from Route to Controller

- ➢ Create a route that accepts a parameter (e.g., /user/{name}).
- ➢ Access and use the parameter inside the controller method.

D. Call a View from a Controller

- ➢ Create a Blade view (e.g., user.blade.php).
- ➢ Return the view from a controller method using return view('user').

E. Pass Data from Controller to View

- ➢ Return the view with data using an associative array.
- ➢ Access the data in the Blade view using Blade syntax ({{ $variable }}).

4.) **Perform the Following Tasks and Implement Each One Practically:**

A. Create and Implement a View

- ➢ Create a Blade view file manually inside the resources/views folder.
- ➢ Alternatively, use the command: php artisan make:view about
- ➢ Implement: Ensure the view renders text correctly when loaded.

B. Call and Implement a View from Route and Controller

- ➢ Create a route in web.php that directly returns a view.
- ➢ Create a controller method that returns a view.
- ➢ Implement: Test both route-based and controller-based view rendering.

C. Create and Implement a Nested View

- ➢ Create a view file in a subfolder (e.g., admin/login.blade.php).
- ➢ Call the nested view using view('admin.login').
- ➢ Implement: Access it via browser to confirm nested view loading.

D. Pass and Implement Data in a View

- ➢ Pass a parameter from the route to the controller and then to the view.
- ➢ Use Blade syntax {{ $variable }} to display the data.
- ➢ Implement: Confirm the data is received and shown on the view.

E. Verify and Implement View Existence Check

- ➢ Use View::exists('viewname') inside the controller.
- ➢ Show fallback message if view is not found.

- Implement: Test with both existing and non-existing views.

5.) Perform the Following Tasks and Implement Each One Practically:

A. Understand and Implement Blade Template
- Identify the file extension used for Blade (.blade.php).
- Implement: Create a simple Blade file and verify output in the browser.

B. Use and Implement Blade Expressions
- Use {{ }} syntax to display dynamic values.
- Implement: Pass a variable from controller to view and display it using Blade syntax.

C. Execute and Implement PHP Functions
- Use Blade to call PHP functions like rand(), date(), etc.
- Implement: Display the output of a function in your Blade view.

D. Implement Conditional Logic (if-else) in Blade
- Use @if, @elseif, and @else directives.
- Implement: Display a custom message based on a passed variable value.

E. Implement For Loop in Blade
- Use @for directive to loop through numeric values.
- Implement: Display a list of numbers using a Blade for-loop.

F. Implement Foreach Loop in Blade
- Use @foreach to iterate over an array passed from the controller.
- Implement: Display each value of the array in your view.

6.) Execute the Following Tasks and Demonstrate Each One Practically:

A. Understand and Implement Subviews
- Create a subview using a folder structure inside resources/views.
- Implement: Create a subview like common.header and verify its structure.

B. Include and Implement Subviews in Main View
- Use @include() directive to include a subview inside a main view (e.g., home1.blade.php).
- Implement: Verify both views render together correctly in the browser.

C. Create and Implement Multiple Subviews
- Create additional subviews like common.inner.blade.php.
- Include multiple subviews inside a main Blade file.
- Implement: Load them together and ensure they render properly.

D. Pass and Implement Data from Main View to Subview
- Use @include('viewname', ['key' => 'value']) to pass data.
- Access and display the variable in the subview using {{ $variable }}.
- Implement: Confirm the passed data renders correctly.

7.) Execute the Following Tasks and Demonstrate Each One Practically:

A. Understand and Define a Named Route

- Define a route using ->name('routeName').
- Implement: Create a named route for a view and verify its output.

B. Use Named Route in View

- Use route('routeName') within a Blade file for navigation.
- Implement: Add an anchor tag that navigates using the named route.

C. Use Named Route in Controller

- Redirect to a named route from a controller using redirect()->route('name') or to_route('name').
- Implement: Call a controller method that redirects to a named route.

D. Pass Parameters in a Named Route

- Define a named route that accepts a parameter (e.g., {name}).
- Use route('routeName', ['name' => 'value']) in the controller or view.
- Implement: Access the dynamic parameter in the Blade view.

E. Pass Dynamic Data from Controller Using Named Routes

- Use to_route('routeName', ['key' => 'value']) from a controller.
- Implement: Redirect from a controller to a named route with parameter and display it in view.

8.) Execute the Following Tasks and Demonstrate Each One Practically:

A. Understand and Define Route Grouping with Prefix

- Implement: Use Route::prefix('student') to define grouped routes.

B. Create Routes for View and Controller

- Create a Blade view file (e.g., home1.blade.php) and a controller (e.g., HomeController).
- Define individual routes like /student/home, /student/show, and /student/add.
- Implement: Return views and simple text responses from controller methods.

C. Group the Routes Under a Common Prefix

- Use Route::prefix('student')->group(function(){ ... });
- Include multiple routes inside the group (both view and controller-based).
- Implement: Access and verify all grouped URLs in the browser.

9.) Execute the Following Tasks and Demonstrate Each One Practically:

A. Understand and Implement Route Grouping with Controller

- ➢ Implement: Use Route::controller(ControllerName::class)->group(...) syntax.

B. Create Routes for Controller Methods

- ➢ Create a controller (e.g., StudentController).
- ➢ Add methods like show(), add(), and delete() inside the controller.
- ➢ Implement: Define corresponding routes in web.php and check the output.

C. Group Controller Routes Using Route::controller

- ➢ Define a grouped route structure using Route::controller(...)->group(...).
- ➢ Inside the group, define paths that call methods directly without repeating the controller.
- ➢ Implement: Access routes like /show, /add, /delete and confirm functionality.

D. Pass Parameters with Controller-Based Routes

- ➢ Add a route with a dynamic parameter (e.g., /about/{name}) in the controller group.
- ➢ Create a method (e.g., about($name)) in the controller to receive it.
- ➢ Implement: Pass and display the parameter in the browser.

10.) Execute the Following Tasks and Demonstrate Each One Practically:

A. Understand and Describe Middleware

B. Create Middleware

- ➢ Use Artisan to create middleware using the command:
- ➢ Implement: Verify that AgeCheck.php is created inside the Http/Middleware folder.

C. Apply Middleware Globally

- ➢ Open bootstrap/app.php.
- ➢ Add the middleware class inside the withMiddleware block.
- ➢ Implement: Confirm it executes for every incoming request.

D. Filter Requests Using Middleware Logic

- ➢ Modify the handle() method to include a custom condition using request parameters.

E. Create Additional View and Test Middleware

- ➢ Create a view file named about.blade.php.
- ➢ Define a route for the view (e.g., /about).
- ➢ Implement: Apply middleware and test how it filters access based on the

given request.

11.) **Execute the Following Tasks and Demonstrate Each One Practically:**
A. Understand and Describe Middleware Group
B. Create Middleware Files
- ➢ Use Artisan commands to create multiple middleware:
- ➢ Implement: Add simple logic inside each middleware's handle() method.

C. Register Middleware Group
D. Apply Middleware Group to Single Route
E. Apply Middleware Group to Route Group

12.) **Execute the Following Tasks and Demonstrate Each One Practically:**
A. Create and Implement Middleware
- ➢ Use the following command to create middleware:
  - ✓ **php artisan make:middleware AgeCheck**
- ➢ Implement: Add basic logic to print a message or block underaged users.

B. Create View Files
- ➢ Create two Blade view files:
  - ✓ **php artisan make:view home**
  - ✓ **php artisan make:view about**
- ➢ Implement: Load these views using Route::view() in web.php.

C. Assign Middleware to a Specific Route
- ➢ Import the middleware class directly in web.php.
- ➢ Apply middleware to a single route like so:
  - ✓ **Route::view('home1', 'home')->middleware(AgeCheck::class);**
- ➢ Implement: Add validation in the middleware (e.g., restrict if age < 18) and test using ?age= parameter.

D. Create and Apply Multiple Middleware
- ➢ Create another middleware named CountryCheck:
  - ✓ **php artisan make:middleware CountryCheck**
  - ✓ **Add logic to block access if country != 'india'.**
  - ✓ **Apply both middleware to a single route:**
  - ✓ **Route::view('home1', 'home')->middleware([AgeCheck::class, CountryCheck::class]);**

13.) **Execute the Following Tasks and Demonstrate Each One Practically:**
A. Create View and Define Route for the Form
- ✓ Create a Blade view named user-form.blade.php using Artisan.
- ✓ Define a route using Route::view('user-form', 'user-form');.
- ✓ Implement: Load the form in the browser at /user-form.

B. Design the Form with Input Fields

- ➢ Create an HTML form in the Blade file with fields:
  - ✓ Name (name="username")
  - ✓ Email (name="email")
  - ✓ City (name="city")
- ➢ Implement: Use the POST method and add @csrf inside the form.

C. Create the Controller

✓**php artisan make:controller UserController**

- ➢ Implement: Create a method named adduser to handle the form data.

D. Create the Form Submission Route

- ➢ Route::post('adduser', [UserController::class, 'adduser'])
- ➢ Implement: Ensure this matches the action="adduser" in the form.

E. Retrieve and Display Form Data in Controller

- ➢ In the adduser() method of the controller, use the $request object.
- ➢ Implement: Submit the form and verify that the values appear correctly in the browser.

14.) Execute the Following Tasks and Demonstrate Each One Practically:

A. Create a View and Define Its Route

✓**php artisan make:view user-form**

- ➢ Define the route using:

  ✓**Route::view('user-form', 'user-form');**
- ➢ Implement: Access /user-form and verify the page loads.

B. Add and Implement Form Elements

- ➢ Inside user-form.blade.php, create a form with the following elements:
  - ✓ **Checkboxes** for skills (e.g., PHP, Node, Java)
  - ✓ **Radio buttons** for gender
  - ✓ **Dropdown (select)** for city options (e.g., Delhi, Anand, Mumbai)
  - ✓ **Range input** for age (min=18, max=100)
  - ✓ A **submit button** and @csrf token for security

C. Create the Controller

- ➢ Generate a controller using:

  ✓**php artisan make:controller UserController**

  ✓**Add a method addUser() to handle form submissions.**

D. Set Up the Route for Form Submission

✓Route::post('adduser', [UserController::class, 'addUser']);

E. Retrieve and Display Input Values

- Implement: Submit the form with different input combinations and verify the output.

15.) Execute the Following Tasks and Demonstrate Each One Practically:

A. Connect to MySQL Database Using .env and Migration

- **Create a database** in phpMyAdmin with the name laravel.
- **Update your .env file** with the following credentials:
  - ✓ DB_CONNECTION=mysql
  - ✓ DB_HOST=127.0.0.1
  - ✓ DB_PORT=3306
  - ✓ DB_DATABASE=laravel
  - ✓ DB_USERNAME=root
  - ✓ DB_PASSWORD=
- **Run the migration using the command**

B. Retrieve and Display Data from the Database

C. Connect to a Database Without Migration

D. Display DB Data on UI

16.) Execute the Following Tasks and Demonstrate Each One Practically:

A. Create and Connect the Students Table in Database

- Create a table named students in your MySQL laravel database with fields:
  - ✓ id, name, email, batch
- Insert sample records into this table via phpMyAdmin.
- Confirm .env database credentials are correct:

B. Create Controller and Model

C. Fetch and Display Data

D. Create a Blade View and Display Data

E. Define and Use a Method in the Model

17.) Execute the Following Tasks and Demonstrate Each One Practically Using Query Builder:

A. Controller and View Setup

- Create a controller and view using:
  - ✓ **php artisan make:controller UsersController**
  - ✓ **php artisan make:view users**

B. Retrieve and Display Data

- Use Query Builder to fetch all user records from the users table:

C. Use where() and first()

D. Insert Data into Table

F. Update Records
G. Delete Records

18.) Execute the Following Tasks and Demonstrate Each One Practically Using Form Validation:

A. Create and Display the Registration Form

- Create a Blade view file: user-form1.blade.php.
- Add input fields:
  - ✓ First Name
  - ✓ Last Name
  - ✓ Email
  - ✓ Password
  - ✓ Gender (radio button)
- Define the GET route to load the form and POST route to submit it:

B. Add Basic Form Validation in Controller

- In UserController, define the adduser(Request $request) method.
- Validate input using $request->validate([...]).
- Add validation rules:
  - ✓ Required, min/max for names
  - ✓ Required and email format for email
  - ✓ Min length for password
  - ✓ Required gender
- Implement custom error messages using the second argument to validate().

C. Display Error Messages and Retain Old Values
D. Highlight Invalid Input Fields
E. Customize Laravel's Validation Messages (Optional)

19.) To demonstrate the usage of various string helper functions provided by Laravel for effective and fluent string manipulation.

A. Use the following string helpers in your implementation:

- ✓ Str::of() and str()
- ✓ Str::slug()
- ✓ Str::camel()
- ✓ Str::snake()
- ✓ Str::title()
- ✓ Str::limit()
- ✓ Str::contains()
- ✓ Str::startsWith() and Str::endsWith()

20.) **Develop a simple Laravel-based web application to** Add, List, Edit, and Delete **student records in a MySQL database.**

A. Insert Data in MySQL DB Table

- ➢ Configure .env for MySQL connection.
- ➢ Create necessary components:
  - ✓ Controller: StudentController
  - ✓ Model: Student
  - ✓ View: add-student.blade.php
- ➢ Create an HTML form to input student name, email, and phone.
- ➢ Create MySQL table students.
- ➢ Submit form and store data in DB.

B. Display Data from DB

- ➢ Create view list-student.blade.php and route.
- ➢ Retrieve all records using Student::all().
- ➢ Display data in a table format.

C. Delete Data from MySQL Table

- ➢ Add a **Delete** button in the student list.
- ➢ Define delete route and controller method to remove record.
- ➢ After deletion, redirect to the list view.

D. Populate Data in Input Fields for Editing

- ➢ Add an **Edit** button next to each record in the list.
- ➢ Fetch data using student ID and display it in a form.
- ➢ Populate the form fields with existing data.

E. Update Data in MySQL Table

- Submit the edit form using a PUT request.
- Update the record in the database using the given ID.
- After successful update, redirect back to the list.