PHP Laravel Industry Assignments

Module 1 – Core PHP

PHP Syntax

THEORY EXERCISE:

- Discuss the structure of a PHP script and how to embed PHP in HTML.
- What are the rules for naming variables in PHP?

LAB EXERCISE:

- Write a PHP script to print "Hello, World!" on a web page.

3. PHP Variables

THEORY EXERCISE:

- Explain the concept of variables in PHP and their scope.

LAB EXERCISE:

- Create a PHP script to declare and initialize different types of variables (integer, float, string, boolean). Display them using echo.

4. Super Global Variables

THEORY EXERCISE:

- What are super global variables in PHP? List at least five super global arrays and their use.

LAB EXERCISE:

- Create a form that takes a user's name and email. Use the $_POST super global to display the entered data.

5. Practical Example: Multiple Tables and SQL Queries

LAB EXERCISE:

- Create multiple tables and perform queries using:
  - SELECT, UPDATE, DELETE, INSERT
  - WHERE, LIKE, GROUP BY, HAVING
  - LIMIT, OFFSET, Subqueries, AND, OR, NOT, IN

6. Conditions, Events, and Flows

THEORY EXERCISE:

- Explain how conditional statements work in PHP.

7. If Condition and If-Else If

LAB EXERCISE:

- Write a PHP program to determine if a number is even or odd using if conditions.

8. Practical Example: Calculator and Day Finder

LAB EXERCISE:

1. **Simple Calculator**: Create a calculator using if-else conditions that takes two inputs and an operator (+, -, *, /).
2. **Day Finder**: Write a script that finds the current day. If it is Sunday, print "Happy Sunday."

9. Switch Case and Ternary Operator

LAB EXERCISE:

1. **Restaurant Food Category Program**: Use a switch case to display the category (Starter/Main Course/Dessert) and dish based on user selection.
2. **Ternary Operator Example**: Write a script using the ternary operator to display a message if the age is greater than 18.
3. **Color Selector**: Write a program to display the name of a color based on user input (red, green, blue).

10. Loops: Do-While, For Each, For Loop

THEORY EXERCISE:

- Discuss the difference between for loop, foreach loop, and do-while loop in PHP.

LAB EXERCISE:

1. **For Loop**: Write a script that displays numbers from 1 to 10 on a single line.
2. **For Loop (Addition)**: Add all integers from 0 to 30 and display the total.
3. **Chessboard Pattern**: Use a nested loop to create a chessboard pattern (8x8 grid).
4. **Various Patterns**: Generate different patterns using loops.

11. PHP Array and Array Functions

THEORY EXERCISE:

- Define arrays in PHP. What are the different types of arrays?

LAB EXERCISE:

1. Display the value of an array.
2. Find and display the number of odd and even elements in an array.
3. Create an associative array for user details (name, email, age) and display them.
4. Write a script to shift all zero values to the bottom of an array.

12. PHP Date-Time Function

LAB EXERCISE:

- Write a script to display the current date and time in different formats.

13. Header Function

THEORY EXERCISE:

- What is the header function in PHP and how is it used?

LAB EXERCISE:

- Redirect users to another page using the `header()` function.

14. Include and Require

THEORY EXERCISE:

- Explain the difference between include and require in PHP.

LAB EXERCISE:

- Use include and require to insert common header and footer files into multiple PHP pages.

15. Practical Example: Calculator, Factorial, String Reverse

LAB EXERCISE:

1. **Calculator**: Create a calculator using user-defined functions.
2. **Factorial**: Write a function that finds the factorial of a number using recursion.
3. **String Reverse**: Reverse a string without using built-in functions.
4. **Download File**: Create a button that allows users to download a file.

16. PHP Expressions, Operations, and String Functions

THEORY EXERCISE:

- Explain what PHP expressions are and give examples of arithmetic and logical operations.

LAB EXERCISE:

- Write a script to perform various string operations like concatenation, substring extraction, and string length determination.

Extra LAB EXERCISES for Core PHP

1. PHP Syntax

Extra LAB EXERCISES:

- **PHP Comments**: Write a PHP script that demonstrates the use of single-line (`//`), multi-line (`/* */`), and inline (`#`) comments.
- **Embedding HTML and PHP**: Create a web page that uses PHP to dynamically generate HTML content (e.g., a table with user information using PHP).
- **Output Statements**: Experiment with `echo`, `print`, and `var_dump`. Write a script that outputs different types of data using these functions.

---

2. PHP Variables

Extra LAB EXERCISES:

- **Type Casting**: Write a script that declares variables of different types and converts them into other types (e.g., integer to float, string to integer). Display the type and value before and after the conversion.
- **Variable Variables**: Demonstrate the use of variable variables in PHP. Write a script where a variable name is stored in another variable, and then use it to print the value.
- **Global and Local Scope**: Write a script that shows how global and local variables work. Use the `global` keyword inside a function to access a global variable.

---

3. Super Global Variables

Extra LAB EXERCISES:

- **$_GET and $_POST**: Create two separate forms: one that uses the `$_GET` method and one that uses `$_POST`. Display the difference in the URL and how data is passed.
- **$_SERVER**: Write a script to display various details of the server environment using `$_SERVER` (like `PHP_SELF`, `SERVER_NAME`, `HTTP_USER_AGENT`, etc.).
- **$_FILES**: Create a form that allows users to upload a file. Handle the uploaded file using the `$_FILES` super global and display information about the file.

---

4. Practical Example: Multiple Tables and SQL Queries

Extra LAB EXERCISES:

- **Complex Joins**: Create a PHP script that connects two or more tables using INNER JOIN, LEFT JOIN, and RIGHT JOIN. Display data from these tables based on specific conditions.
- **Prepared Statements**: Implement SQL queries using prepared statements with placeholders to prevent SQL injection in `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.
- **Transaction Management**: Write a PHP script that uses SQL transactions to insert data into multiple tables, ensuring data integrity in case of an error.

---

5. Conditions, Events, and Flows

Extra LAB EXERCISES:

- **Nested Conditions**: Write a script that uses nested if-else conditions to categorize a number as positive, negative, or zero, and also check if it's an even or odd number.
- **Switch Case with Multiple Cases**: Write a script that accepts a grade (A, B, C, D, F) and displays a message using a switch statement. Handle multiple cases that fall under the same logic (e.g., A and B show "Excellent").

---

6. If Condition and If-Else If

Extra LAB EXERCISES:

- **Grading System**: Write a PHP program that accepts a student's marks and outputs their grade using if-else conditions (A, B, C, D, Fail based on score).
- **Temperature Converter**: Write a script that takes temperature in Celsius or Fahrenheit as input and converts it to the other format using if conditions.

---

7. Practical Example: Calculator and Day Finder

Extra LAB EXERCISES:

- **Enhanced Calculator**: Modify the calculator to handle more complex operations such as exponentiation (^), modulus (%), and square root (√).
- **Date Finder with Time Zone**: Write a script that finds the current day and prints "Happy Sunday" if it's Sunday, but also adjusts for different time zones.

---

8. Switch Case and Ternary Operator

Extra LAB EXERCISES:

- **Month Display**: Create a program using switch case that takes a number (1-12) and displays the corresponding month.
- **Discount Calculation (Ternary Operator)**: Write a script that calculates and displays the discount on a product based on a user-defined price. If the price is above 500, give a 10% discount; otherwise, no discount (use the ternary operator).

---

9. Loops: Do-While, For Each, For Loop

Extra LAB EXERCISES:

- **FizzBuzz Program**: Write a program using a for loop that prints numbers from 1 to 100. But for multiples of 3, print "Fizz" instead of the number, for multiples of 5 print "Buzz", and for multiples of both 3 and 5 print "FizzBuzz".
- **Multiplication Table**: Write a PHP script using a nested for loop to generate a multiplication table from 1 to 10.
- **Reverse Number Sequence**: Write a script using a do-while loop that displays numbers from 10 to 1.

---

10. PHP Array and Array Functions

Extra LAB EXERCISES:

- **Sorting Arrays**: Write a script that demonstrates the use of `sort()`, `rsort()`, `asort()`, and `ksort()` functions to sort arrays.
- **Multi-dimensional Array**: Create a multi-dimensional array to store information about products (name, price, and stock). Write a script to display the information in a tabular format.
- **Array Merge and Diff**: Write a PHP script that merges two arrays and finds the difference between them using `array_merge()` and `array_diff()`.

---

11. PHP Date-Time Function

Extra LAB EXERCISES:

- **Time Difference**: Write a script that calculates the time difference between two dates (e.g., "today" and "next birthday").
- **Custom Date Formats**: Create a script that displays the current date in different formats (e.g., `Y-m-d`, `d/m/Y`, `l, F jS Y`).

---

12. Header Function

Extra LAB EXERCISES:

- **Page Redirect Based on Condition**: Write a script that checks if a user is logged in (use a boolean variable). If not, use the `header()` function to redirect them to a login page.
- **Content-Type Header**: Write a script that sets the `Content-Type` header to return a plain text file or a JSON response.

---

13. Include and Require

Extra LAB EXERCISES:

- **Template System**: Write a PHP script that includes header, navigation, and footer files in multiple web pages to create a basic template system.
- **File Not Found Handling**: Use `require` to include a critical file. If the file doesn't exist, display a custom error message instead of the default PHP error.

---

14. Practical Example: Calculator, Factorial, String Reverse

Extra LAB EXERCISES:

- **Enhanced Factorial**: Write a recursive and non-recursive function to calculate the factorial of a number. Compare their performance for large numbers.
- **Palindrome Checker**: Create a function that checks if a given string is a palindrome.
- **File Upload**: Create a form that allows users to upload a file. Upon submission, download the file using a button click and display the file's details (name, type, size).

Module 6 – HTML, CSS and JS in PHP

HTML Basics

- **What is HTML? Explain its structure.**
- **Describe the purpose of HTML tags and provide examples of commonly used tags.**
- **What are the differences between block-level and inline elements? Give examples of each.**
- **Explain the concept of semantic HTML and why it is important.**

2. CSS Fundamentals

- **What is CSS? How does it differ from HTML?**
- **Explain the three ways to apply CSS to a web page.**
- **What are CSS selectors? List and describe the different types of selectors.**
- **What is the box model in CSS? Explain its components.**

3. Responsive Web Design

- **What is responsive web design? Why is it important?**
- **Explain the use of media queries in CSS. Provide an example.**
- **What are the benefits of using a mobile-first approach in web design?**

4. PHP Integration

- **How can PHP be used to dynamically generate HTML content? Provide examples.**
- **Explain how to include CSS files in a PHP-generated HTML page.**
- **What are the advantages of using PHP to manage HTML forms?**

LAB EXERCISES

1. Creating a Simple Web Page

- **Objective:** Create a basic web page using HTML and style it with CSS.
  - **Instructions:**
    - Create an HTML file (e.g., `index.html`) that includes a header, a navigation bar, a main content section, and a footer.
    - Style the page using an external CSS file (e.g., `styles.css`).
    - Use CSS properties such as `color`, `background-color`, `font-size`, and `padding` to enhance the design.

2. Form Handling with PHP

- **Objective:** Create a simple HTML form and process it using PHP.
  - **Instructions:**
    - Create an HTML form that collects user information (e.g., name, email, and message).
    - Use PHP to process the form data and display a confirmation message with the submitted information.
    - Validate user inputs and provide appropriate feedback.

3. Dynamic Content Generation

- **Objective:** Use PHP to generate dynamic HTML content.
  - **Instructions:**
    - Create a PHP script (e.g., `dynamic-content.php`) that generates a list of items (e.g., products or blog posts) from an array.
    - Use a loop to display the items in a styled HTML list.
    - Style the list using CSS.

4. CSS Grid and Flexbox

- **Objective:** Create a responsive layout using CSS Grid or Flexbox.
  - **Instructions:**
    - Build a grid layout for a gallery of images or a product showcase using either CSS Grid or Flexbox.

- Ensure that the layout is responsive and adjusts based on the screen size.
- Use media queries to change the layout for mobile devices.

5. Styling a PHP Application

- **Objective:** Apply CSS styles to a PHP web application.
  - **Instructions:**
    - Create a simple PHP application (e.g., a user registration page).
    - Use an external CSS file to style the form elements (e.g., inputs, buttons, labels).
    - Ensure that the application is visually appealing and user-friendly.

6. Implementing a Responsive Navigation Bar

- **Objective:** Create a responsive navigation bar using HTML and CSS.
  - **Instructions:**
    - Build a navigation bar using HTML `<ul>` and `<li>` elements.
    - Use CSS to style the navigation bar and make it responsive (e.g., using media queries).
    - Implement a dropdown menu for sub-navigation items.

7. Image Gallery with Lightbox Effect

- **Objective:** Create an image gallery that opens images in a lightbox effect.
  - **Instructions:**
    - Use HTML to create a gallery of images.
    - Implement CSS for styling and layout.
    - Use JavaScript or a CSS library to create a lightbox effect when images are clicked.

Module 2 – Advanced PHP Excercises

OOPs Concepts

THEORY EXERCISE:

- Define Object-Oriented Programming (OOP) and its four main principles: Encapsulation, Inheritance, Polymorphism, and Abstraction.

Practical Exercise:

- Create a simple class in PHP that demonstrates encapsulation by using private and public properties and methods.

Class

THEORY EXERCISE:

- Explain the structure of a class in PHP, including properties and methods.

Practical Exercise:

- Write a PHP script to create a class representing a "Car" with properties like `make`, `model`, and `year`, and a method to display the car details.

Object

THEORY EXERCISE:

- What is an object in OOP? Discuss how objects are instantiated from classes in PHP.

Practical Exercise:

- Instantiate multiple objects of the "Car" class and demonstrate how to access their properties and methods.

Extends

THEORY EXERCISE:

- Explain the concept of inheritance in OOP and how it is implemented in PHP.

Practical Exercise:

- Create a "Vehicle" class and extend it with a "Car" class. Include properties and methods in both classes, demonstrating inherited behavior.

Overloading

THEORY EXERCISE:

- Discuss method overloading and how it is implemented in PHP.

Practical Exercise:

- Create a class that demonstrates method overloading by defining multiple methods with the same name but different parameters.

Abstraction Interface

THEORY EXERCISE:

- Explain the concept of abstraction and the use of interfaces in PHP.

Practical Exercise:

- Define an interface named `VehicleInterface` with methods like `start()`, `stop()`, and implement this interface in multiple classes.

Constructor

THEORY EXERCISE:

- What is a constructor in PHP? Discuss its purpose and how it is used.

Practical Exercise:

- Create a class with a constructor that initializes properties when an object is created.

Destructor

THEORY EXERCISE:

- Explain the role of a destructor in PHP and when it is called.

Practical Exercise:

- Write a class that implements a destructor to perform cleanup tasks when an object is destroyed.

Magic Methods

THEORY EXERCISE:

- Define magic methods in PHP. Discuss commonly used magic methods like `__get()`, `__set()`, and `__construct()`.

Practical Exercise:

- Create a class that uses magic methods to handle property access and modification dynamically.

Scope Resolution

THEORY EXERCISE:

- Explain the scope resolution operator (`::`) and its use in PHP.

Practical Exercise:

- Create a class with static properties and methods, and demonstrate their access using the scope resolution operator.

Traits

THEORY EXERCISE:

- Define traits in PHP and their purpose in code reuse.

Practical Exercise:

- Create two traits and use them in a class to demonstrate how to include multiple behaviors.

Visibility

THEORY EXERCISE:

- Discuss the visibility of properties and methods in PHP (public, private, protected).

Practical Exercise:

- Write a class that shows examples of each visibility type and how they restrict access to properties and methods.

Type Hinting

THEORY EXERCISE:

- Explain type hinting in PHP and its benefits.

Practical Exercise:

- Write a method in a class that accepts type-hinted parameters and demonstrate how it works with different data types.

Final Keyword

THEORY EXERCISE:

- Discuss the purpose of the `final` keyword in PHP and how it affects classes and methods.

Practical Exercise:

- Create a class marked as `final` and attempt to extend it to show the restriction.

Email Security Function

THEORY EXERCISE:

- Explain the importance of email security and common practices to ensure secure email transmission.

Practical Exercise:

- Write a function that sanitizes email input and validates it before sending.

File Handling

THEORY EXERCISE:

- Discuss file handling in PHP, including opening, reading, writing, and closing files.

Practical Exercise:

- Create a script that reads from a text file and displays its content on a web page.

Handling Emails

THEORY EXERCISE:

- Explain how to send emails in PHP using the `mail()` function and the importance of validating email addresses.

Practical Exercise:

- Write a PHP script to send a test email to a user using the `mail()` function.

MVC Architecture

THEORY EXERCISE:

- Discuss the Model-View-Controller (MVC) architecture and its advantages in web development.

Practical Exercise:

- Create a simple MVC application that demonstrates the separation of concerns by implementing a basic "User" module with a model, view, and controller.

Practical Example: Implementation of all the OOPs Concepts

Practical Exercise:

- Develop a mini project (e.g., a Library Management System) that utilizes all OOP concepts like classes, inheritance, interfaces, magic methods, etc.

Connection with MySQL Database

THEORY EXERCISE:

- Explain how to connect PHP to a MySQL database using `mysqli` or `PDO`.

Practical Exercise:

- Write a script to establish a database connection and handle any errors during the connection process.

SQL Injection

THEORY EXERCISE:

- Define SQL injection and its implications on security.

Practical Exercise:

- Demonstrate a vulnerable SQL query and then show how to prevent SQL injection using prepared statements.

Practical: Exception Handling with Try-Catch for Database Connection and Queries

Practical Exercise:

- Implement try-catch blocks in a PHP script to handle exceptions for database connection and query execution.

Server-Side Validation while Registration using Regular Expressions

Practical Exercise:

- Write a registration form that validates user input (e.g., email, password) using regular expressions before submission.

Send Mail While Registration

Practical Exercise:

- Extend the registration form to send a confirmation email upon successful registration.

Session and Cookies

THEORY EXERCISE:

- Explain the differences between sessions and cookies in PHP.

Practical Exercise:

- Write a script to create a session and store user data, and then retrieve it on a different page. Also, demonstrate how to set and retrieve a cookie.

File Upload

THEORY EXERCISE:

- Discuss file upload functionality in PHP and its security implications.

Practical Exercise:

- Create a file upload form that allows users to upload files and handle the uploaded files safely on the server.

PHP with MVC Architecture

Practical Exercise:

- Implement a CRUD application (Create, Read, Update, Delete) using the MVC architecture to manage user data.

Insert, Update, Delete MVC

Practical Exercise:

- Extend the CRUD application to include functionalities for inserting, updating, and deleting user records, ensuring proper separation of concerns in the MVC structure.

Extra Practise for Grade A

1. **Practical Exercise:**
    - Develop a class hierarchy for a simple e-commerce system with classes like `Product`, `Category`, and `Order`. Implement encapsulation by using private properties and public methods to access them.

Class

2. **Practical Exercise:**

- Create a class called `Book` with properties like `title`, `author`, and `price`. Implement a method to apply a discount to the book's price and return the new price.

Object

3. **Practical Exercise:**
   - Instantiate an object of the `Book` class and demonstrate the usage of its methods. Create multiple instances of `Book` and display their details in a formatted manner.

Extends

4. **Practical Exercise:**
   - Create a base class called `Employee` with properties like `name` and `salary`. Extend it with subclasses `FullTimeEmployee` and `PartTimeEmployee`, each having specific methods to calculate bonuses.

Overloading

5. **Practical Exercise:**
   - Create a `Calculator` class with a method `calculate` that can add, subtract, or multiply based on the number and type of arguments passed.

Abstraction Interface

6. **Practical Exercise:**
   - Define an interface `PaymentInterface` with methods like `processPayment()`, `refund()`, and implement it in classes like `CreditCardPayment` and `PaypalPayment`.

Constructor

7. **Practical Exercise:**
   - Create a class `Student` with properties like `name`, `age`, and `grade`. Use a constructor to initialize these properties and a method to display student details.

Destructor

8. **Practical Exercise:**
   - Write a class that connects to a database, with a destructor that closes the connection when the object is destroyed.

Magic Methods

9. **Practical Exercise:**
   - Create a class that uses the `__set()` and `__get()` magic methods to dynamically create and access properties based on user input.

Scope Resolution

10. **Practical Exercise:**
    - Define a class with static properties and methods to keep track of the number of instances created. Use the scope resolution operator to access these static members.

Traits

11. **Practical Exercise:**
    - Create two traits: `Logger` and `Notifier`. Use these traits in a class `User` to log user activities and send notifications.

Visibility

12. **Practical Exercise:**
    - Develop a class `Account` with properties for `username` (public), `password` (private), and `accountBalance` (protected). Demonstrate how to access these properties in a derived class.

Type Hinting

13. **Practical Exercise:**
    - Write a method in a class `Order` that accepts an array of products (type-hinted) and calculates the total order amount.

Final Keyword

14. **Practical Exercise:**
    - Create a base class `Animal` and a final class `Dog`. Attempt to extend `Dog` and demonstrate the restriction imposed by the `final` keyword.

Email Security Function

15. **Practical Exercise:**
    - Write a function that sanitizes user input for an email address, validates it, and throws an exception if it fails validation.

File Handling

16. **Practical Exercise:**
    - Create a script that uploads a file and reads its content. Implement error handling to manage any file-related exceptions.

Handling Emails

17. **Practical Exercise:**
    - Develop a function to send a welcome email to a user upon registration, ensuring the email format is validated first.

MVC Architecture

18. **Practical Exercise:**
    - Extend the simple MVC application to include a model for managing user profiles, a view for displaying user details, and a controller for handling user actions.

Practical Example: Implementation of all the OOPs Concepts

19. **Practical Exercise:**
    - Develop a project that simulates a library system with classes for `User`, `Book`, and `Transaction`, applying all OOP principles.

Connection with MySQL Database

20. **Practical Exercise:**
    - Write a class `Database` that handles database connections and queries. Use this class in another script to fetch user data from a `users` table.

SQL Injection

21. **Practical Exercise:**
    - Create a vulnerable PHP script that demonstrates SQL injection. Then, rewrite it using prepared statements to prevent SQL injection attacks.

Practical: Exception Handling with Try-Catch for Database Connection and Queries

22. **Practical Exercise:**
    - Implement a complete registration process with a database connection that uses try-catch blocks to handle exceptions for all operations.

Server-Side Validation while Registration using Regular Expressions

23. **Practical Exercise:**
    - Write a PHP script that validates user inputs (username, password, email) using regular expressions, providing feedback on any validation errors.

Send Mail While Registration

24. **Practical Exercise:**
    - Extend the registration process to send a confirmation email to the user after successful registration and validate the email format.

Session and Cookies

25. **Practical Exercise:**

- Implement a login system that uses sessions to keep track of user authentication and demonstrates cookie usage for "Remember Me" functionality.

File Upload

26. **Practical Exercise:**
    - Create a file upload feature that allows users to upload images. Ensure that the uploaded images are checked for file type and size for security.

PHP with MVC Architecture

27. **Practical Exercise:**
    - Build a small blog application using the MVC architecture, where users can create, read, update, and delete posts.

Insert, Update, Delete MVC

28. **Practical Exercise:**
    - Expand the blog application to include a feature for user comments, allowing users to insert, update, and delete their comments.

Module 3 – WebServices, API, Extensions

THEORY EXERCISEs

1. **Payment Gateway Integration**
    - **Objective:** Understand the concept and importance of payment gateways in e-commerce.
    - **Questions:**
        - Explain the role of payment gateways in online transactions.
        - Compare and contrast different payment gateway options (e.g., PayPal, Stripe, Razorpay).
        - Discuss the security measures involved in payment gateway integration.
2. **API with Header**
    - **Objective:** Learn about the significance of headers in API requests and responses.
    - **Questions:**
        - What are HTTP headers, and how do they facilitate communication between client and server?
        - Describe how to set custom headers in an API request.
3. **API with Image Uploading**
    - **Objective:** Understand the process of uploading images through an API.
    - **Questions:**
        - What are the common file formats for images that can be uploaded via API?
        - Explain the process of handling file uploads securely in a web application.
4. **SOAP and REST APIs**

- **Objective:** Differentiate between SOAP and REST API architectures.
- **Questions:**
  - What are the key characteristics of SOAP APIs?
  - Describe the principles of RESTful API design.

5. **Product Catalog**
   - **Objective:** Explore the structure and implementation of a product catalog in an e-commerce system.
   - **Questions:**
     - What are the key components of a product catalog?
     - How can you ensure that a product catalog is scalable?
6. **Shopping Cart**
   - **Objective:** Understand the functionality and design of a shopping cart system.
   - **Questions:**
     - What are the essential features of an e-commerce shopping cart?
     - Discuss the importance of session management in maintaining a shopping cart.
7. **Web Services**
   - **Objective:** Understand the concept of web services and their applications.
   - **Questions:**
     - Define web services and explain how they are used in web applications.
     - Discuss the difference between RESTful and SOAP web services.
8. **RESTful Principles**
   - **Objective:** Familiarize with RESTful principles and best practices for API design.
   - **Questions:**
     - Explain the importance of statelessness in RESTful APIs.
     - What is resource identification in REST, and why is it important?
9. **OpenWeatherMap API**
   - **Objective:** Explore the functionality and usage of the OpenWeatherMap API.
   - **Questions:**
     - Describe the types of data that can be retrieved using the OpenWeatherMap API.
     - Explain how to authenticate and make requests to the OpenWeatherMap API.
10. **Google Maps Geocoding API**
    - **Objective:** Understand the use of Google Maps Geocoding API for location services.
    - **Questions:**
      - What is geocoding, and how does it work with the Google Maps API?
      - Discuss the potential applications of the Google Maps Geocoding API in web applications.

LAB EXERCISES

1. **Payment Gateway Integration**
   - **Exercise:** Implement a payment gateway (e.g., Stripe or PayPal) in a sample e-commerce application.
   - **Tasks:**
     - Set up the payment gateway account.
     - Create an API endpoint for processing payments.

        - Handle payment success and failure responses.
2. **Create API with Header**
    - **Exercise:** Develop a simple REST API that accepts custom headers.
    - **Tasks:**
        - Create an API endpoint that accepts a custom header and responds with the header value.
3. **API with Image Uploading**
    - **Exercise:** Create an API that allows users to upload images.
    - **Tasks:**
        - Implement file upload functionality with validation.
        - Store the uploaded images on the server.
4. **SOAP and REST APIs**
    - **Exercise:** Create a simple REST API for a product catalog.
    - **Tasks:**
        - Implement endpoints for CRUD operations (Create, Read, Update, Delete) on products.
5. **Product Catalog**
    - **Exercise:** Design a product catalog with product details.
    - **Tasks:**
        - Create a database schema for products.
        - Develop an interface to display products.
6. **Shopping Cart**
    - **Exercise:** Implement a shopping cart feature in an e-commerce application.
    - **Tasks:**
        - Allow users to add, update, and remove products from the cart.
        - Persist cart data using sessions or cookies.
7. **Web Services**
    - **Exercise:** Create a web service that returns product data.
    - **Tasks:**
        - Implement a RESTful service to fetch product details.
        - Handle errors gracefully.
8. **Create Web Services for MVC Project**
    - **Exercise:** Extend an existing MVC project with web services.
    - **Tasks:**
        - Add web services for user authentication and product management.
9. **Integration of API in Project**
    - **Exercise:** Integrate an external API (e.g., OpenWeatherMap) into a project.
    - **Tasks:**
        - Make API calls and display data on the frontend.
10. **Implement RESTful principles**
    - **Exercise:** Design an API following RESTful principles.
    - **Tasks:**
        - Implement resource identification and statelessness in your API design.
11. **OpenWeatherMap API**
    - **Exercise:** Build a weather dashboard using the OpenWeatherMap API.
    - **Tasks:**
        - Retrieve and display current weather data for a user-specified location.
12. **Google Maps Geocoding API**
    - **Exercise:** Create a location-based application using the Google Maps Geocoding API.

- **Tasks:**
  - Allow users to enter an address and display its coordinates on a map.

13. **GitHub API**
    - **Exercise:** Build a simple application that retrieves user data from the GitHub API.
    - **Tasks:**
      - Allow users to search for GitHub users and display their repositories.
14. **Twitter API**
    - **Exercise:** Integrate Twitter functionality into your application using the Twitter API.
    - **Tasks:**
      - Fetch and display tweets based on a specific hashtag.
15. **Email Sending APIs**
    - **Exercise:** Implement email functionality using a service like SendGrid or Mailgun.
    - **Tasks:**
      - Set up email sending for user registration confirmations.
16. **Social Authentication**
    - **Exercise:** Implement social authentication in your application.
    - **Tasks:**
      - Allow users to log in using Google or Facebook accounts.
17. **Normal Payments**
    - **Exercise:** Create a payment processing feature using PayPal or Stripe.
    - **Tasks:**
      - Develop a checkout page that integrates with the payment gateway.
18. **SMS Sending APIs**
    - **Exercise:** Integrate SMS notifications into your application using Twilio.
    - **Tasks:**
      - Set up SMS notifications for important events (e.g., order confirmations).
19. **File Upload**
    - **Exercise:** Implement a file upload feature for users to upload documents.
    - **Tasks:**
      - Validate and store uploaded files securely.
20. **MVC with Insert, Update, Delete**
    - **Exercise:** Extend an existing MVC project to manage user comments.
    - **Tasks:**
      - Implement functionality to insert, update, and delete comments.

Extra Practise

Challenging Practical Exercises

1. **Payment Gateway Integration**
    - **Exercise:** Build a fully functional e-commerce site with multiple payment gateways.
      - **Tasks:**
        - Integrate at least two different payment gateways (e.g., PayPal and Stripe).

        - Implement a user-friendly checkout process, including error handling for payment failures.
        - Use webhooks to update the order status based on payment results.
2. **Create API with Header & API with Image Uploading**
    - **Exercise:** Develop a RESTful API that handles user registration with image uploads and custom headers.
        - **Tasks:**
            - Create an endpoint that accepts user data and an avatar image, validating the image type and size.
            - Implement authentication using custom headers.
            - Return appropriate status codes and messages based on the request outcome.
3. **Payment Gateway Implementation on MVC Project**
    - **Exercise:** Create a multi-step checkout process in an MVC application.
        - **Tasks:**
            - Implement user authentication and store cart items in sessions.
            - Allow users to enter shipping details and select payment methods.
            - Handle payment processing and order confirmation using MVC architecture.
4. **SOAP and REST API Creation for CRUD Operations**
    - **Exercise:** Create both a SOAP and REST API for a library system managing books.
        - **Tasks:**
            - Implement CRUD operations for books in both APIs.
            - Ensure the APIs handle input validation and error reporting effectively.
            - Compare the implementations and discuss the differences in handling requests and responses.
5. **Product Catalog**
    - **Exercise:** Design and implement a dynamic product catalog with search and filtering features.
        - **Tasks:**
            - Use a database to store product information and images.
            - Implement search functionality based on keywords and filtering by category and price range.
            - Use AJAX for live search results without page reloads.
6. **Shopping Cart**
    - **Exercise:** Develop a persistent shopping cart that remembers items even after the user logs out.
        - **Tasks:**
            - Store cart items in the database, linked to user accounts.
            - Implement functionality to modify cart items (add, remove, update quantities).
            - Create a summary page displaying the cart's contents before checkout.
7. **Web Services**
    - **Exercise:** Create a comprehensive web service that provides data from multiple APIs.

        - **Tasks:**
            - Integrate data from at least three different APIs (e.g., weather, country info, and GitHub).
            - Build a single endpoint that consolidates this data for the client.
            - Ensure proper error handling if one or more APIs fail.
8. **Create Web Services for MVC Project**
    - **Exercise:** Extend an existing MVC project to provide a web service for mobile app integration.
        - **Tasks:**
            - Create a secure API for retrieving user data and submitting feedback.
            - Implement token-based authentication for the API.
            - Ensure the API adheres to RESTful principles.
9. **Integration of API in Project**
    - **Exercise:** Build a weather dashboard that combines the OpenWeatherMap API with user input.
        - **Tasks:**
            - Allow users to enter a city name and display current weather conditions, forecasts, and historical data.
            - Implement caching to improve performance and reduce API calls.
            - Create an admin panel to manage user API keys for access control.
10. **RESTful Principles Implementation**
    - **Exercise:** Design an API following RESTful principles that handles inventory management.
        - **Tasks:**
            - Implement endpoints for adding, retrieving, updating, and deleting inventory items.
            - Ensure that the API is stateless and follows uniform resource identifiers (URIs).
            - Document the API using Swagger or a similar tool.
11. **OpenWeatherMap API**
    - **Exercise:** Create a weather application that allows users to compare weather conditions across multiple cities.
        - **Tasks:**
            - Use the OpenWeatherMap API to fetch data for at least three different locations.
            - Display a comparative view of temperatures, humidity, and wind speeds.
            - Implement error handling for invalid city inputs.
12. **Google Maps Geocoding API**
    - **Exercise:** Build a location-based service that provides directions and distance between two addresses.
        - **Tasks:**
            - Use the Google Maps Geocoding API to convert addresses to coordinates.
            - Integrate Google Maps JavaScript API to display the route on a map.
            - Allow users to save their frequently used addresses.

13. **GitHub API**
    - **Exercise:** Create a GitHub profile viewer that displays user repositories and contributions.
        - **Tasks:**
            - Fetch user data from the GitHub API, including repositories and their stars, forks, and issues.
            - Display a graphical representation of contributions over the last year.
            - Allow users to search for GitHub users and view their profile data.
14. **Twitter API**
    - **Exercise:** Build a Twitter sentiment analysis tool that fetches and analyzes tweets based on a keyword.
        - **Tasks:**
            - Use the Twitter API to retrieve tweets containing the keyword.
            - Implement a simple sentiment analysis algorithm to categorize tweets as positive, negative, or neutral.
            - Display the results in a dashboard format.
15. **REST Countries API**
    - **Exercise:** Develop a travel application that uses the REST Countries API to provide information about different countries.
        - **Tasks:**
            - Allow users to search for countries and view details like population, languages, and currencies.
            - Implement a comparison feature to compare multiple countries side-by-side.
            - Create a favorites list for users to save countries of interest.
16. **SendGrid Email API**
    - **Exercise:** Implement a notification system using SendGrid that sends emails for different events.
        - **Tasks:**
            - Set up email templates for different notifications (e.g., order confirmations, newsletters).
            - Track email delivery status and handle failures.
            - Create an admin interface to manage email templates and view delivery reports.
17. **Social Authentication**
    - **Exercise:** Build a user registration and login system that integrates with Google and Facebook for social authentication.
        - **Tasks:**
            - Implement OAuth for Google and Facebook authentication.
            - Allow users to register and log in using their social media accounts.
            - Store user data securely and handle the initial setup.
18. **Email Sending APIs**
    - **Exercise:** Create a marketing email system using Mailgun that allows users to subscribe and unsubscribe from newsletters.
        - **Tasks:**
            - Implement an interface for users to manage their subscription preferences.

        - Use Mailgun to send bulk marketing emails.
        - Track open and click rates for email campaigns.

19. **SMS Sending APIs**
    - **Exercise:** Develop an application that sends SMS notifications using Twilio.
        - **Tasks:**
            - Set up a user interface to input phone numbers and messages.
            - Implement functionality to send SMS reminders for events or appointments.
            - Track delivery status and handle any errors.

20. **Google Map API**
    - **Exercise:** Build a location-sharing application that allows users to share their locations in real-time.
        - **Tasks:**
            - Use Google Maps API to display a map and user locations.
            - Implement functionality for users to check in and share their location with friends.
            - Include a feature for users to view friends' locations on the map.

Module 4) Laravel Framework

Theory Assignments

1. **Introduction to Laravel**
    - **Assignment**: Write a detailed report on the history of Laravel. Include its versioning, key features, and how it differs from other PHP frameworks.
2. **Laravel MVC Architecture**
    - **Assignment**: Explain the MVC (Model-View-Controller) architecture. Provide examples of how Laravel implements this architecture in web applications.
3. **Routing in Laravel**
    - **Assignment**: Describe how routing works in Laravel. Explain the difference between named routes and route parameters with examples.
4. **Blade Templating Engine**
    - **Assignment**: Write an essay on the Blade templating engine in Laravel. Discuss its features, syntax, and how it enhances the development process.
5. **Database Migrations and Eloquent ORM**
    - **Assignment**: Explain the concept of database migrations in Laravel. Discuss how Eloquent ORM simplifies database interactions and provide examples of CRUD operations.
6. **Laravel Middleware**
    - **Assignment**: Define middleware in Laravel. Explain how middleware can be used for authentication, logging, and CORS handling.
7. **Laravel Authentication**
    - **Assignment**: Write a report on Laravel's built-in authentication system. Explain how to set up user authentication and discuss the use of guards and providers.
8. **Testing in Laravel**
    - **Assignment**: Discuss the importance of testing in web applications. Explain the testing tools available in Laravel and write a brief guide on how to write basic tests.

Practical Assignments

1. **Setting Up a Laravel Project**
   - **Task**: Install Laravel using Composer and create a new Laravel project. Set up your development environment (including a local server) and configure the `.env` file for database connections.
2. **Creating a Simple Blog**
   - **Task**: Develop a simple blog application where users can create, read, update, and delete (CRUD) blog posts.
   - **Requirements**:
     - Use Eloquent ORM for database operations.
     - Implement route controllers for handling requests.
     - Use Blade templates for the front end.
3. **User Registration and Authentication**
   - **Task**: Implement a user registration and login system using Laravel's built-in authentication.
   - **Requirements**:
     - Allow users to register with a username, email, and password.
     - Implement email verification.
     - Create a dashboard that displays user information after login.
4. **Form Validation**
   - **Task**: Create a contact form for users to submit their inquiries.
   - **Requirements**:
     - Implement form validation to ensure all fields are filled out correctly.
     - Display validation error messages using Blade.
5. **RESTful API Development**
   - **Task**: Create a RESTful API for managing products in an inventory.
   - **Requirements**:
     - Implement endpoints for creating, retrieving, updating, and deleting products.
     - Use Laravel's resource controllers and API routes.
     - Test the API using Postman.
6. **Using Laravel Middleware**
   - **Task**: Create a middleware that checks if a user is an admin.
   - **Requirements**:
     - Apply this middleware to certain routes to restrict access based on user roles.
7. **Laravel Notifications**
   - **Task**: Implement a notification system that sends email notifications when a new blog post is published.
   - **Requirements**:
     - Use Laravel's notification system to send notifications.
     - Set up a queue for processing notifications.
8. **Integrating a Payment Gateway**
   - **Task**: Integrate a payment gateway (e.g., Stripe or PayPal) into your blog application.
   - **Requirements**:
     - Create a feature that allows users to make donations or purchases.
     - Handle payment processing and success/error responses.
9. **Building a CRUD Application with Resource Controllers**

- **Task**: Create a resource controller for managing categories in the blog application.
- **Requirements**:
  - Implement all CRUD operations using resource routes.
  - Create views for adding and editing categories.

10. **Deployment of Laravel Application**
    - **Task**: Deploy your Laravel application to a web server (like DigitalOcean, Heroku, or any shared hosting).
    - **Requirements**:
      - Document the deployment process, including environment configuration and database setup.