***MODULE – (Web Services)***

**1) Create API for Registration & Login Page**

**Ans.** Creating an API for registration and login in PHP involves setting up endpoints, handling requests, and managing a database. Below is a simple example using PHP with PDO for database interaction.

**Prerequisites**

1. A web server with PHP support (like Apache or Nginx).
2. A MySQL database.
3. Basic knowledge of PHP and REST APIs.

**Step-by-Step Guide**

**1. Database Setup**

First, create a database and a users table. You can use the following SQL:

CREATE DATABASE auth\_db;

USE auth\_db;

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL UNIQUE,

password VARCHAR(255) NOT NULL

);

**2. Project Structure**

Create a directory for your project, e.g., auth-api, and create the following files:

auth-api/

├── config.php

├── register.php

└── login.php

**3. Configuration File**

**config.php**

This file will handle database connections.

<?php

$host = 'localhost';

$db = 'auth\_db';

$user = 'your\_db\_user';

$pass = 'your\_db\_password';

try {

$pdo = new PDO("mysql:host=$host;dbname=$db", $user, $pass);

$pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

} catch (PDOException $e) {

die("Could not connect to the database: " . $e->getMessage());

}

?>

**4. Registration Script**

**register.php**

This script will handle user registration.

<?php

header('Content-Type: application/json');

require 'config.php';

$data = json\_decode(file\_get\_contents("php://input"));

if (isset($data->username) && isset($data->password)) {

$username = $data->username;

$password = password\_hash($data->password, PASSWORD\_BCRYPT);

try {

$stmt = $pdo->prepare("INSERT INTO users (username, password) VALUES (?, ?)");

$stmt->execute([$username, $password]);

echo json\_encode(["message" => "User registered successfully"]);

} catch (PDOException $e) {

if ($e->getCode() === 23000) { // Unique constraint violation

echo json\_encode(["error" => "Username already exists"]);

} else {

echo json\_encode(["error" => "Registration failed"]);

}

}

} else {

echo json\_encode(["error" => "Invalid input"]);

}

?>

**5. Login Script**

**login.php**

This script will handle user login.

<?php

header('Content-Type: application/json');

require 'config.php';

$data = json\_decode(file\_get\_contents("php://input"));

if (isset($data->username) && isset($data->password)) {

$username = $data->username;

$password = $data->password;

$stmt = $pdo->prepare("SELECT \* FROM users WHERE username = ?");

$stmt->execute([$username]);

$user = $stmt->fetch(PDO::FETCH\_ASSOC);

if ($user && password\_verify($password, $user['password'])) {

// Generate a simple token (for demonstration purposes)

$token = base64\_encode($username . ':' . $password);

echo json\_encode(["token" => $token]);

} else {

echo json\_encode(["error" => "Invalid credentials"]);

}

} else {

echo json\_encode(["error" => "Invalid input"]);

}

?>

**Testing the API**

You can test the API using tools like Postman or curl.

**1. Register a new user**:

* **Endpoint**: POST /register.php
* **Body**:

{

"username": "yourUsername",

"password": "yourPassword"

}

**2. Log in**:

* **Endpoint**: POST /login.php
* **Body**:

{

"username": "yourUsername",

"password": "yourPassword"

}

**Conclusion**

This basic setup provides a simple API for user registration and login using PHP and MySQL. Depending on your needs, you might want to enhance it with features like token expiration, email verification, or a more robust token generation method (like JWT).

**2) Create Web Service for your MVC Project.**

**Ans.** Creating a web service for an MVC project involves setting up a controller to handle API requests and defining routes for your endpoints. Below, I'll provide a basic example using PHP with an MVC structure. This example includes user registration and login functionality.

**Project Structure**

Assuming you're using a standard MVC architecture, your project structure might look something like this:

my-mvc-app/

├── app/

│ ├── controllers/

│ │ └── UserController.php

│ ├── models/

│ │ └── User.php

│ ├── views/

│ └── config.php

├── public/

│ └── index.php

└── .htaccess

**Step 1: Database Setup**

Make sure you have a database set up. Use the following SQL to create the users table:

CREATE DATABASE my\_mvc\_db;

USE my\_mvc\_db;

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL UNIQUE,

password VARCHAR(255) NOT NULL

);

**Step 2: Configuration File**

**app/config.php**

This file handles the database connection.

<?php

$host = 'localhost';

$db = 'my\_mvc\_db';

$user = 'your\_db\_user';

$pass = 'your\_db\_password';

try {

$pdo = new PDO("mysql:host=$host;dbname=$db", $user, $pass);

$pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

} catch (PDOException $e) {

die("Could not connect to the database: " . $e->getMessage());

}

?>

**Step 3: User Model**

**app/models/User.php**

This model will handle user data and interactions with the database.

<?php

require\_once 'config.php';

class User {

public static function register($username, $password) {

global $pdo;

$hashedPassword = password\_hash($password, PASSWORD\_BCRYPT);

try {

$stmt = $pdo->prepare("INSERT INTO users (username, password) VALUES (?, ?)");

$stmt->execute([$username, $hashedPassword]);

return true;

} catch (PDOException $e) {

return false; // Registration failed (e.g., username already exists)

}

}

public static function login($username, $password) {

global $pdo;

$stmt = $pdo->prepare("SELECT \* FROM users WHERE username = ?");

$stmt->execute([$username]);

$user = $stmt->fetch(PDO::FETCH\_ASSOC);

if ($user && password\_verify($password, $user['password'])) {

return true; // Successful login

}

return false; // Invalid credentials

}

}

?>

**Step 4: User Controller**

**app/controllers/UserController.php**

This controller will handle API requests related to users.

<?php

require\_once '../models/User.php';

class UserController {

public function register() {

header('Content-Type: application/json');

$data = json\_decode(file\_get\_contents("php://input"));

if (isset($data->username) && isset($data->password)) {

$username = $data->username;

$password = $data->password;

if (User::register($username, $password)) {

echo json\_encode(["message" => "User registered successfully"]);

} else {

echo json\_encode(["error" => "Registration failed, username may already exist"]);

}

} else {

echo json\_encode(["error" => "Invalid input"]);

}

}

public function login() {

header('Content-Type: application/json');

$data = json\_decode(file\_get\_contents("php://input"));

if (isset($data->username) && isset($data->password)) {

$username = $data->username;

$password = $data->password;

if (User::login($username, $password)) {

echo json\_encode(["message" => "Login successful"]);

} else {

echo json\_encode(["error" => "Invalid credentials"]);

}

} else {

echo json\_encode(["error" => "Invalid input"]);

}

}

}

?>

**Step 5: Entry Point**

**public/index.php**

This is the main entry point of your application.

<?php

require\_once '../app/controllers/UserController.php';

$requestMethod = $\_SERVER['REQUEST\_METHOD'];

$requestUri = explode('/', trim($\_SERVER['REQUEST\_URI'], '/'));

if ($requestUri[0] === 'api' && $requestUri[1] === 'register' && $requestMethod === 'POST') {

$controller = new UserController();

$controller->register();

} elseif ($requestUri[0] === 'api' && $requestUri[1] === 'login' && $requestMethod === 'POST') {

$controller = new UserController();

$controller->login();

} else {

header("HTTP/1.0 404 Not Found");

echo json\_encode(["error" => "Endpoint not found"]);

}

?>

**Step 6: .htaccess for URL Rewriting**

**.htaccess**

Ensure that the server rewrites requests to the index.php file.

<IfModule mod\_rewrite.c>

RewriteEngine On

RewriteRule ^api/(.\*)$ public/index.php [QSA,L]

</IfModule>

**Testing the API**

You can test your API using tools like Postman or curl.

**1. Register a new user**:

* **Endpoint**: POST /api/register
* **Body**:

{

"username": "yourUsername",

"password": "yourPassword"

}

**2. Log in**:

* **Endpoint**: POST /api/login
* **Body**:

{

"username": "yourUsername",

"password": "yourPassword"

}

**Conclusion**

This setup gives you a basic MVC structure for a web service with user registration and login functionality. You can expand upon this by adding more features, such as password reset, email verification, or user role management, as needed.

**3) Create API for Image Uploading.**

**Ans.** Creating an API for image uploading involves setting up an endpoint that accepts image files, processes them, and stores them on the server or in a cloud service. Below is a simple example using PHP and a basic MVC structure.

**Project Structure**

Here’s how your project might look:

image-upload-api/

├── app/

│ ├── controllers/

│ │ └── ImageController.php

│ ├── models/

│ ├── config.php

├── public/

│ └── index.php

└── .htaccess

**Step 1: Configuration File**

**app/config.php**

This file will handle basic configuration.

<?php

define('UPLOAD\_DIR', \_\_DIR\_\_ . '/../uploads/');

if (!file\_exists(UPLOAD\_DIR)) {

mkdir(UPLOAD\_DIR, 0777, true);

}

?>

**Step 2: Image Controller**

**app/controllers/ImageController.php**

This controller handles the image upload logic.

<?php

require\_once '../config.php';

class ImageController {

public function upload() {

header('Content-Type: application/json');

if (isset($\_FILES['image']) && $\_FILES['image']['error'] === UPLOAD\_ERR\_OK) {

$fileTmpPath = $\_FILES['image']['tmp\_name'];

$fileName = $\_FILES['image']['name'];

$fileSize = $\_FILES['image']['size'];

$fileType = $\_FILES['image']['type'];

$fileNameCmps = explode(".", $fileName);

$fileExtension = strtolower(end($fileNameCmps));

// Allowed file extensions

$allowedExts = ['jpg', 'jpeg', 'png', 'gif'];

if (in\_array($fileExtension, $allowedExts)) {

$newFileName = uniqid() . '.' . $fileExtension;

$destination = UPLOAD\_DIR . $newFileName;

if (move\_uploaded\_file($fileTmpPath, $destination)) {

echo json\_encode(["message" => "File uploaded successfully", "file" => $newFileName]);

} else {

echo json\_encode(["error" => "There was an error moving the uploaded file"]);

}

} else {

echo json\_encode(["error" => "Invalid file type"]);

}

} else {

echo json\_encode(["error" => "No file uploaded or there was an upload error"]);

}

}

}

?>

**Step 3: Entry Point**

**public/index.php**

This file routes requests to the appropriate controller.

<?php

require\_once '../app/controllers/ImageController.php';

$requestMethod = $\_SERVER['REQUEST\_METHOD'];

if ($requestMethod === 'POST' && isset($\_GET['action']) && $\_GET['action'] === 'upload') {

$controller = new ImageController();

$controller->upload();

} else {

header("HTTP/1.0 404 Not Found");

echo json\_encode(["error" => "Endpoint not found"]);

}

?>

**Step 4: .htaccess for URL Rewriting**

**.htaccess**

Make sure to rewrite requests to the index.php file.

<IfModule mod\_rewrite.c>

RewriteEngine On

RewriteRule ^api/(.\*)$ public/index.php [QSA,L]

</IfModule>

**Step 5: Testing the API**

You can test the API using tools like Postman or curl.

**1. Upload an image**:

* **Endpoint**: POST /api/upload
* **Body**: Use form-data to send the image file.
  + Key: image (type: File)

**Example curl command**

bash

curl -X POST -F "image=@/path/to/your/image.jpg" http://yourdomain.com/api/upload

**Conclusion**

This basic setup provides an API for uploading images using PHP. You can expand upon this by adding more features, such as image validation, resizing, or saving image metadata in a database. Remember to implement security measures to protect against file upload vulnerabilities.