Lab Tutorial: Backend Web Development

Table of Contents

1. Introduction

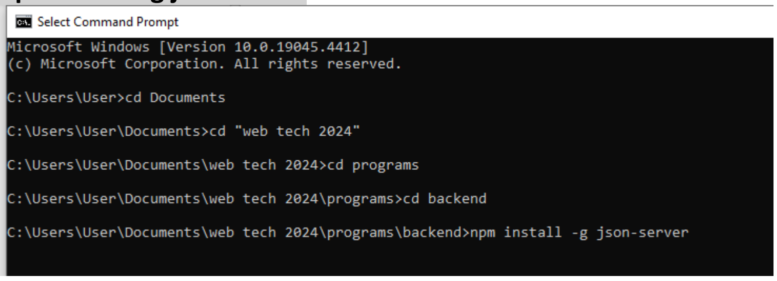
In this tutorial, we will explore backend web development using various technologies. We'll start with **JSON Server** for quick API prototyping, then move on to building a backend using **plain PHP**, first develop a non-restful then transitioning to a RESTful approach, and finally utilizing the **Slim framework** for a more structured and scalable solution.
The resources we will be working with are **/users** and **/products**.

2. Setting Up JSON Server

Installing JSON Server

✓ Prerequisites: Ensure you have Node.js installed.
✓ Installation:
Create a new project folder, ie /backend
Run command to install json-server

```
npm install -g json-server
```



Select Command Prompt

```
Microsoft Windows [Version 10.0.19045.4412]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>cd Documents

C:\Users\User\Documents>cd "web tech 2024"

C:\Users\User\Documents\web tech 2024>cd programs

C:\Users\User\Documents\web tech 2024\programs>cd backend

C:\Users\User\Documents\web tech 2024\programs\backend>npm install -g json-server
```

Creating a Mock Database

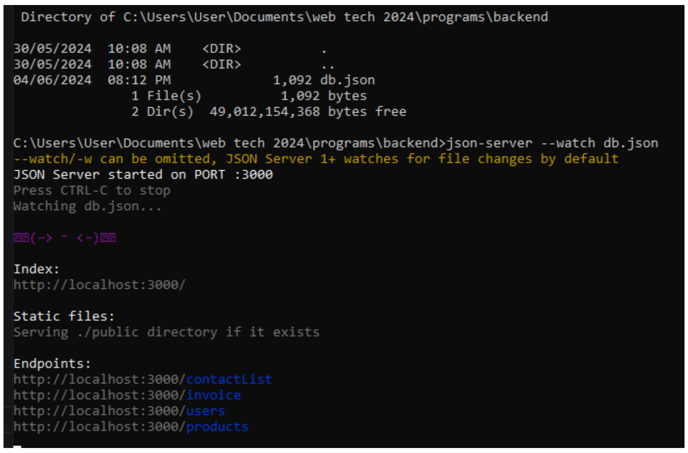
✓ Open the project /backend folder in VSCode.
✓ Create a db.json file with some initial data:

```json
{
  "users": [
    { "id": 1, "name": "Ali Johan", "email": "alijo@utm.my", "role":"admin" },
    { "id": 2, "name": "Seman Rihan", "email": "semrihan@utm.my", "role":"sales" }
  ],
  "products": [
    { "id": 1, "name": "Laptop", "price": 2999.99 },
    { "id": 3, "name": "Phone", "price": 1999.50 },
    { "id": 5, "name": "Desk", "price": 500.00 },
    { "id": 9, "name": "Book", "price": 45.50 }
  ]
}
```

Running JSON Server

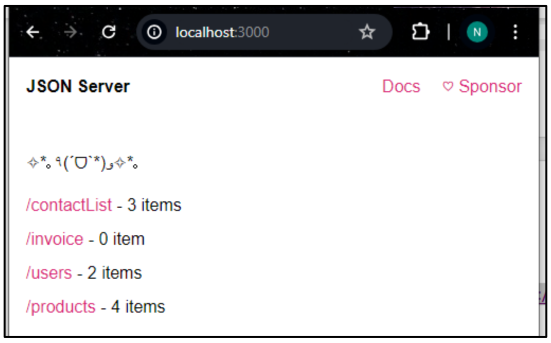
✓ Run command to start the server.

```
json-server --watch db.json
```



```
 Directory of C:\Users\User\Documents\web tech 2024\programs\backend

30/05/2024  10:08 AM    <DIR>          .
30/05/2024  10:08 AM    <DIR>          ..
04/06/2024  08:12 PM             1,092 db.json
               1 File(s)          1,092 bytes
               2 Dir(s)  49,012,154,368 bytes free

C:\Users\User\Documents\web tech 2024\programs\backend>json-server --watch db.json
--watch/-w can be omitted, JSON Server 1+ watches for file changes by default
JSON Server started on PORT :3000
Press CTRL-C to stop
Watching db.json...

⍰⍰(~> ˘ <~)⍰⍰

Index:
http://localhost:3000/

Static files:
Serving ./public directory if it exists

Endpoints:
http://localhost:3000/contactList
http://localhost:3000/invoice
http://localhost:3000/users
http://localhost:3000/products
```

✓ Access the server at http://localhost:3000. using your web browser (ie get users)



JSON Server

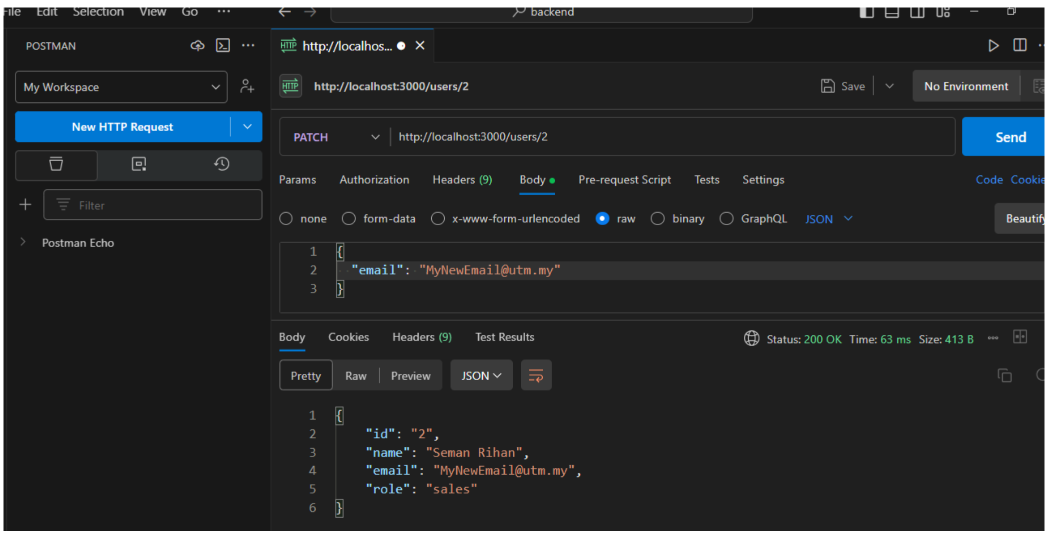
✧*｡٩(ˊᗜˋ*)و✧*｡

/contactList - 3 items
/invoice - 0 item
/users - 2 items
/products - 4 items

✓ Making the following API Requests

Test using Postman

a.	GET all users:	GET	http://localhost:3000/users
b.	GET a specific user:	GET	http://localhost:3000/users/1
c.	POST a new user:	POST	http://localhost:3000/users
d.	PUT update a user:	PUT	http://localhost:3000/users/1
e.	PATCH update a user:	PATCH	http://localhost:3000/users/1
f.	DELETE a user:	DELETE	http://localhost:3000/users/1



http://localhost:3000/users/2

PATCH http://localhost:3000/users/2

```
{
    "email": "MyNewEmail@utm.my"
}
```

Status: 200 OK Time: 63 ms Size: 413 B

```
{
    "id": "2",
    "name": "Seman Rihan",
    "email": "MyNewEmail@utm.my",
    "role": "sales"
}
```

Your Tasks:

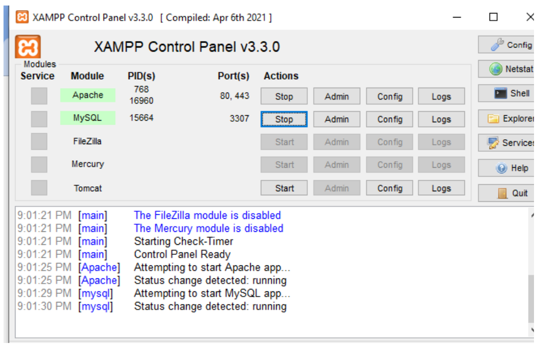
✓ **Follow the similar steps above for the resources : /products & /orders.**

✓ **complete the implementation of the frontend (Vue project) by integrating it with the backend, specifically for the modules: user login & registration, and product & orders (covering all CRUD operations).**

3. Using PHP for Backend Development (Non-RESTful)

Setting Up a PHP Environment

- ✓ Install PHP: Follow the installation instructions for your operating system.
- ✓ Set Up a Local Server: Use XAMPP, WAMP, or MAMP.


XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]

XAMPP Control Panel v3.3.0

Modules

Service	Module	PID(s)	Port(s)	Actions			
	Apache	768 16960	80, 443	Stop	Admin	Config	Logs
	MySQL	15664	3307	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

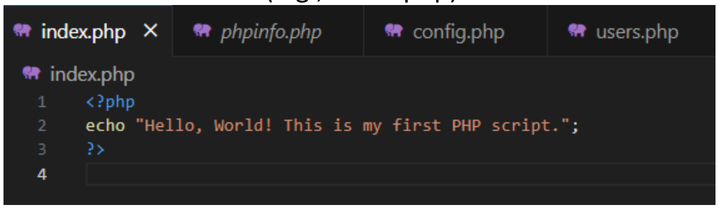
```
9:01:21 PM  [main]      The FileZilla module is disabled
9:01:21 PM  [main]      The Mercury module is disabled
9:01:21 PM  [main]      Starting Check-Timer
9:01:21 PM  [main]      Control Panel Ready
9:01:25 PM  [Apache]    Attempting to start Apache app...
9:01:25 PM  [Apache]    Status change detected: running
9:01:29 PM  [mysql]     Attempting to start MySQL app...
9:01:30 PM  [mysql]     Status change detected: running
```

Creating a Simple PHP Backend

- ✓ Create the config.php for database config

```php
<?php
 class db{
    // Properties
    private $host = 'localhost';
    private $user = 'root';
    private $password = '';
    private $dbname = 'my_database';

    // Connect
    function connect(){
        $mysql_connect_str = "mysql:host=$this->host;dbname=$this->dbname";
        $dbConnection = new PDO($mysql_connect_str, $this->user, $this->password);
        $dbConnection->setAttribute( PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
        return $dbConnection;
    }
  }
?>
```

- ✓ Create a PHP file (e.g., index.php):



```php
<?php
echo "Hello, World! This is my first PHP script.";
?>
```

- ✓ Complete the PHP code (index.php)

```php
<?php
require_once './config.php';

$action = isset($_GET['action']) ? $_GET['action'] : '';
switch ($action) {
    case 'getUsers':
        getUsers();
        break;
    case 'getUserById':
        getUserById();
        break;
    case 'createUser':
        createUser();
        break;
    case 'updateUser':
        updateUser();
        break;
    case 'deleteUser':
        deleteUser();
        break;
    default:
        echo json_encode(["error" => "Invalid action"]);
        break;
}

function getUsers(){
    try {
        $db = new db();
        $conn = $db->connect();
        $sql = "SELECT * FROM users";
        $stmt = $conn->prepare($sql);
        $stmt->execute();
        $result = $stmt->fetchAll(PDO::FETCH_ASSOC);
        $conn = null; // Close the connection
        echo json_encode($result);
    } catch (PDOException $e) {
        echo "Error: " . $e->getMessage();
    }
}

function getUserById(){
    try {
        $db = new db();
        $conn = $db->connect();
        $id = $_GET['id'];
        $sql = "SELECT * FROM users WHERE id = :id";
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(':id', $id);
        $stmt->execute();
        $user = $stmt->fetch(PDO::FETCH_ASSOC);
        if ($user) {
            echo json_encode($user);
        } else {
            echo json_encode(["error" => "User not found"]);
        }
        $conn = null; // Close the connection
    } catch (PDOException $e) {
        echo json_encode(["error" => "Database error: " . $e->getMessage()]);
    }
}

function createUser()
{
    try {
        $db = new db();
        $conn = $db->connect();
        // Get JSON data from the request body
        $json_data = file_get_contents('php://input');
        // Decode JSON data into associative array
        $data = json_decode($json_data, true);
        // Check if required fields are present
        if (!isset($data['name']) || !isset($data['email'])) {
            throw new Exception("Name and email are required.");
        }
        $name = $data['name'];
        $email = $data['email'];
        $sql = "INSERT INTO users (name, email) VALUES (:name, :email)";
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(':name', $name);
        $stmt->bindValue(':email', $email);
        $stmt->execute();
        echo json_encode(["message" => "User created successfully"]);
        $conn = null; // Close the connection
    } catch (Exception $e) {
        echo json_encode(["error" => "Error creating user: " . $e->getMessage()]);
    }
}

function updateUser(){
//pls complete
}
```

```
function patchUser(){
//pls complete
}
function deleteUser(){
//pls complete
}
```

Test using Postman

a.	GET all users:	GET	http://localhost/index.php?action=getUsers.
b.	GET specific user:	GET	http://localhost/index.php?action=getUserById&id=1.
c.	POST a new user:	GET	http://localhost/index.php?action=createUser.
d.	PUT update a user:	GET	http://localhost/index.php?action=updateUser&id=1.
e.	PATCH a user:	GET	http://localhost/index.php?action=patchUser&id=1.
f.	DELETE a user:	GET	http://localhost/index.php?action=deleteUser&id=1.GET

Your Tasks:

- ✓ **Follow the similar steps above for the resources : /products & /orders.**
- ✓ **complete the implementation of the frontend (Vue project) by integrating it with the backend (this non-restful), specifically for the modules: user login & registration, and product & orders (covering all CRUD operations).**

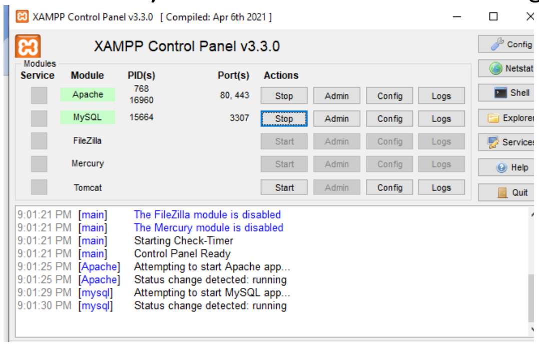
4. Building a RESTful API with PHP

RESTful API Principles

- ✓ Statelessness
- ✓ Client-Server Architecture
- ✓ Uniform Interface
- ✓ Resource-Based URIs

Setting Up the Environment

- ✓ Ensure your PHP environment is running.


XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]
XAMPP Control Panel v3.3.0
Modules
Config
Netstat
Shell
Explorer
Services
Help
Quit

Service	Module	PID(s)	Port(s)	Actions			
	Apache	768 16960	80, 443	Stop	Admin	Config	Logs
	MySQL	15664	3307	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
9:01:21 PM  [main]     The FileZilla module is disabled
9:01:21 PM  [main]     The Mercury module is disabled
9:01:21 PM  [main]     Starting Check-Timer
9:01:21 PM  [main]     Control Panel Ready
9:01:25 PM  [Apache]   Attempting to start Apache app...
9:01:25 PM  [Apache]   Status change detected: running
9:01:29 PM  [mysql]    Attempting to start MySQL app...
9:01:30 PM  [mysql]    Status change detected: running
```

Creating RESTful Endpoints

Preparation:

Directory Structure:

```
project_root/
├── config.php
└── api/
    ├── index.php
    └── .htaccess
```

Step 1: Code the config.php

Create a config.php file with the database configuration details.

```php
<?php
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'root');
define('DB_PASSWORD', '');
define('DB_NAME', 'my_database');

function getDbConnection() {
    $conn = new mysqli(DB_SERVER, DB_USERNAME, DB_PASSWORD, DB_NAME);
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    return $conn;
}
?>
```

Step 2: Code the api/index.php

Create an index.php file inside the api directory that includes the config.php for the database connection.

```php
<?php
header('Content-Type: application/json');
require_once '../config.php';
```

```php
// Helper function to get input data
function get_input_data() {
    return json_decode(file_get_contents('php://input'), true);
}

// Connect to the database
$conn = getDbConnection();

// Parse the request URL to determine the resource and ID
$request_uri = explode('?', $_SERVER['REQUEST_URI'], 2);
$path = trim($request_uri[0], '/');
$segments = explode('/', $path);
$resource = $segments[1] ?? null;
$id = $segments[2] ?? null;

// Fetch all users
if ($_SERVER['REQUEST_METHOD'] == 'GET' && $resource == 'users' && !$id) {
    $sql = "SELECT * FROM users";
    $result = $conn->query($sql);
    $users = [];
    if ($result->num_rows > 0) {
        while($row = $result->fetch_assoc()) {
            $users[] = $row;
        }
    }
    echo json_encode($users);
}

// Fetch a specific user
if ($_SERVER['REQUEST_METHOD'] == 'GET' && $resource == 'users' && $id) {
    $id = intval($id);
    $sql = "SELECT * FROM users WHERE id = $id";
    $result = $conn->query($sql);
    if ($result->num_rows > 0) {
        echo json_encode($result->fetch_assoc());
    } else {
        echo json_encode(["message" => "User not found"]);
    }
}

// Insert a new user
if ($_SERVER['REQUEST_METHOD'] == 'POST' && $resource == 'users') {
    $input = get_input_data();
    $name = $conn->real_escape_string($input['name']);
    $email = $conn->real_escape_string($input['email']);
    $sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')";
    if ($conn->query($sql) === TRUE) {
        echo json_encode(["message" => "User created successfully", "id" =>
$conn->insert_id]);
    } else {
        echo json_encode(["message" => "Error: " . $conn->error]);
    }
}

// Update a user
//complete the code here

// patch a user
//complete the code here

// Delete a user
//complete the code here

$conn->close();
?>
```

Test using Postman

a. GET all users: GET http://localhost/index.php/users

b.	GET a specific user:	GET	http://localhost/index.php/users/1
c.	Post a new user:	POST	http://localhost/index.php/users
d.	PUT update a user:	PUT	http://localhost/index.php/users/1
e.	PATCH update a user:	PATCH	http://localhost/index.php/users/1
f.	DELETE a user:	DELETE	http://localhost/index.php/users/1

~~**Your Task: Follow similar steps for the /products resource.**~~

Step 3: Create .htaccess File

Create an .htaccess file in the api directory to handle URL rewriting.

```
RewriteEngine On
# Redirect all requests to index.php
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php [QSA,L]
```

Test using Postman

g.	GET all users:	GET	http://localhost/api/users
h.	GET a specific user:	GET	http://localhost/api/users/1
i.	POST a new user:	POST	http://localhost/api/users
j.	PUT update a user:	PUT	http://localhost/api/users/1
k.	PATCH update a user:	PATCH	http://localhost/api/users/1
l.	DELETE a user:	DELETE	http://localhost/api/users/1

Your Tasks:

- ✓ **Follow the similar steps above for the resources : /products & /orders.**
- ✓ **complete the implementation of the frontend (Vue project) by integrating it with the backend, specifically for the modules: user login & registration, and product & orders (covering all CRUD operations).**

5. Using the PHP Slim Framework

Introduction to Slim Framework

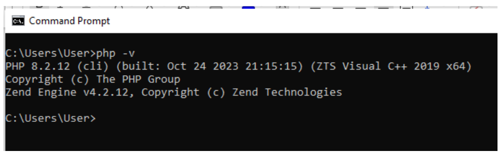
Slim is a PHP micro-framework that helps us to quickly write simple yet powerful web applications and APIs. It is great for building simple and fast web backends.

Steps:

Install PHP:

✓ Make sure you have PHP installed on your system. You can check by running:

php -v



```
C:\Users\User>php -v
PHP 8.2.12 (cli) (built: Oct 24 2023 21:15:15) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.2.12, Copyright (c) Zend Technologies

C:\Users\User>
```

Install Slim

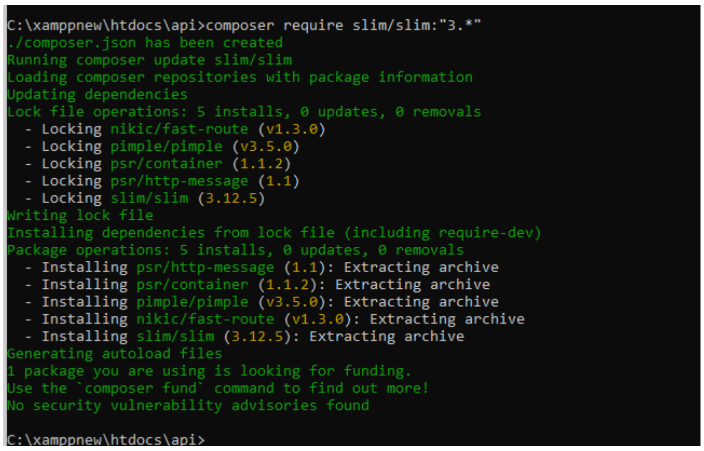
✓ Install Composer (PHP dependency manager):

✓ Composer is a dependency manager for PHP. You can download it from getcomposer.org and install it following the instructions there

Create a new Slim project:

```
mkdir api
cd api
composer require slim/slim:"3.*"
code .
```

Open in VSCode



```
C:\xamppnew\htdocs\api>composer require slim/slim:"3.*"
./composer.json has been created
Running composer update slim/slim
Loading composer repositories with package information
Updating dependencies
Lock file operations: 5 installs, 0 updates, 0 removals
  - Locking nikic/fast-route (v1.3.0)
  - Locking pimple/pimple (v3.5.0)
  - Locking psr/container (1.1.2)
  - Locking psr/http-message (1.1)
  - Locking slim/slim (3.12.5)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 5 installs, 0 updates, 0 removals
  - Installing psr/http-message (1.1): Extracting archive
  - Installing psr/container (1.1.2): Extracting archive
  - Installing pimple/pimple (v3.5.0): Extracting archive
  - Installing nikic/fast-route (v1.3.0): Extracting archive
  - Installing slim/slim (3.12.5): Extracting archive
Generating autoload files
1 package you are using is looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found

C:\xamppnew\htdocs\api>
```

Setting Up a Slim Project

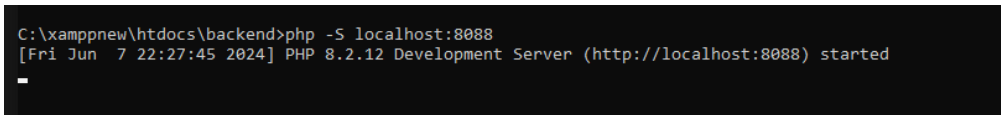
✓ Create an index.php file:

```php
<?php
require 'vendor/autoload.php';

$app = new \Slim\App;
$app->get('/', function ($request, $response, $args) {
    return $response->write("Hello, World!");
});

$app->get('/hello/{name}', function ($request, $response, $args) {
    $name = $args['name'];
    return $response->write("Selamat datang ..., $name");
});

$app->get('/users', function ($request, $response, $args) {
    $userObject = [
        'id' => 1,
        'name' => 'John Doe',
        'email' => 'john@example.com',
        // Add other user attributes here
    ];
    return $response->withJson($userObject);
});

$app->run();
```

Run Your Application

✓ Run the PHP Built-in Server: You can use PHP's built-in server to run your application:

php -S localhost:8088



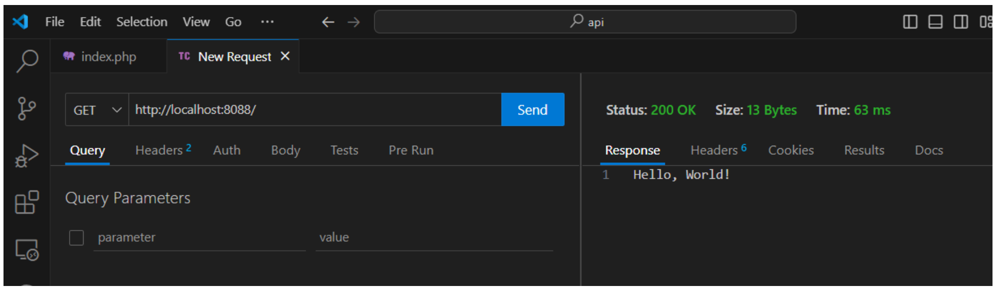
```
C:\xamppnew\htdocs\backend>php -S localhost:8088
[Fri Jun  7 22:27:45 2024] PHP 8.2.12 Development Server (http://localhost:8088) started
```

Test using Postman or ThunderClient

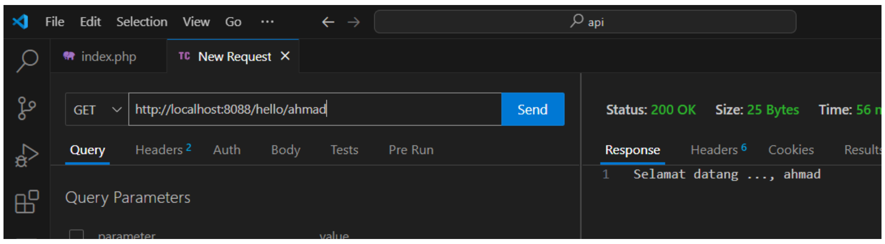
✓ Make a GET request: http://localhost:8088/

✓ Make a GET request: http://localhost:8088/hello/ahmad

✓ Make a GET request: http://localhost:8088/users


File Edit Selection View Go ···
api
index.php
TC New Request ✕
Query Headers 2 Auth Body Tests Pre Run
Query Parameters
parameter
value
Send
Response Headers 6 Cookies Results Docs

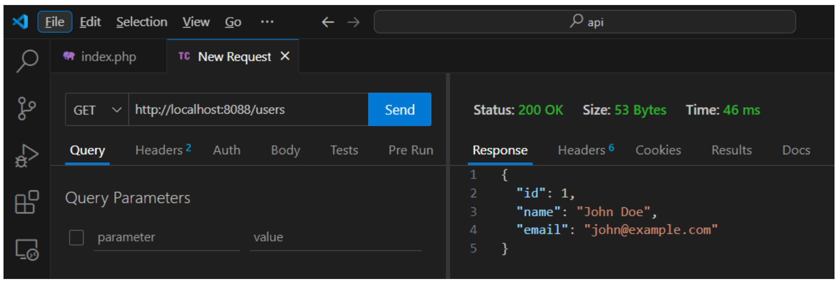
```
GET http://localhost:8088/
Status: 200 OK   Size: 13 Bytes   Time: 63 ms
1  Hello, World!
```



GET http://localhost:8088/hello/ahmad

Status: 200 OK Size: 25 Bytes Time: 56

```
Selamat datang ..., ahmad
```



GET http://localhost:8088/users

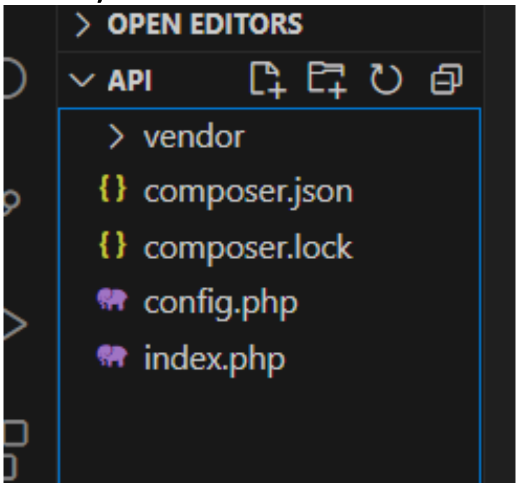
Status: 200 OK Size: 53 Bytes Time: 46 ms

```
{
  "id": 1,
  "name": "John Doe",
  "email": "john@example.com"
}
```

Complete the code untuk endpoints : /users, /products, /orders. Store persistent dlm mysql database tables

Steps:

Directory structure



```
vendor
composer.json
composer.lock
config.php
index.php
```

config.php

```php
<?php
class db
{
    // Properties
    private $host = 'localhost';
    private $user = 'root';
    private $password = '';
    private $dbname = 'my_database';

    // Connect
    function connect()
    {
        $mysql_connect_str = "mysql:host=$this->host;dbname=$this->dbname";
        $dbConnection = new PDO($mysql_connect_str, $this->user, $this->password);
        $dbConnection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        return $dbConnection;
    }
}
```

index.php

```php
<?php
header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Headers: X-Requested-With, Content-Type, Accept,
Origin, Authorization");
header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS");


require 'vendor/autoload.php';
require_once './config.php';
// Instantiate db class
$db = new db();
// Create Slim app
$app = new \Slim\App;
// Routes for users
$app->get('/users', function ($request, $response, $args) use ($db) {
    try {
        $conn = $db->connect();
        $sql = "SELECT * FROM users";
        $stmt = $conn->prepare($sql);
        $stmt->execute();
        $result = $stmt->fetchAll(PDO::FETCH_ASSOC);
        return $response->withJson($result);
    } catch (PDOException $e) {
        return $response->withJson(["error" => "Error: " . $e->getMessage()]);
    }
});
$app->get('/users/{id}', function ($request, $response, $args) use ($db) {
    try {
        $id = $args['id'];
        $conn = $db->connect();
        $sql = "SELECT * FROM users WHERE id = :id";
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(':id', $id);
        $stmt->execute();
        $user = $stmt->fetch(PDO::FETCH_ASSOC);
        if ($user) {
            return $response->withJson($user);
        } else {
            return $response->withJson(["error" => "User not found"]);
        }
    } catch (PDOException $e) {
        return $response->withJson(["error" => "Database error: " .
$e->getMessage()]);
    }
});
$app->post('/users', function ($request, $response, $args) use ($db) {
    try {
        $conn = $db->connect();
        $data = $request->getParsedBody();
        if (!isset($data['name']) || !isset($data['email'])) {
            throw new Exception("Name and email are required.");
        }
        $name = $data['name'];
        $email = $data['email'];
        $sql = "INSERT INTO users (name, email) VALUES (:name, :email)";
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(':name', $name);
        $stmt->bindValue(':email', $email);
        $stmt->execute();
        return $response->withJson(["message" => "User created successfully"]);
    } catch (Exception $e) {
        return $response->withJson(["error" => "Error creating user: " .
$e->getMessage()]);
    }
});
```

```
$app->put('/users/{id}', function ($request, $response, $args) use ($db) {
    try {
        $id = $args['id'];
        $conn = $db->connect();
        $data = $request->getParsedBody();
        if (!isset($data['name']) || !isset($data['email'])) {
            throw new Exception("Name and email are required.");
        }
        $name = $data['name'];
        $email = $data['email'];
        $sql = "UPDATE users SET name = :name, email = :email WHERE id = :id";
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(':name', $name);
        $stmt->bindValue(':email', $email);
        $stmt->bindValue(':id', $id);
        $stmt->execute();
        return $response->withJson(["message" => "User updated successfully"]);
    } catch (Exception $e) {
        return $response->withJson(["error" => "Error updating user: " .
$e->getMessage()]);
    }
});
$app->delete('/users/{id}', function ($request, $response, $args) use ($db) {
    try {
        $id = $args['id'];
        $conn = $db->connect();
        $sql = "DELETE FROM users WHERE id = :id";
        $stmt = $conn->prepare($sql);
        $stmt->bindValue(':id', $id);
        $stmt->execute();
        return $response->withJson(["message" => "User deleted successfully"]);
    } catch (PDOException $e) {
        return $response->withJson(["error" => "Error deleting user: " .
$e->getMessage()]);
    }
});
// Routes for products
// lengkapkan routes for products here
$app->get('/products', function ($request, $response, $args){
    return $response->write("get all products executes!");
});
$app->get('/products/{id}', function($request, $response, $args){
    $id = $args['id'];
    return $response->write("Get product with ID: $id");
});
// Routes for orders
// lengkapkan routes untuk orders here
$app->get('/orders', function ($request, $response, $args){
    return $response->write("get all orders executes!");
});
//dummy routes...cuba disini
$app->get('/xyz', function ($request, $response, $args) {
    $userObject = [
        'id' => 101,
        'name' => 'Johanna Alis',
        'email' => 'johalis@utm.my'
    ];
    return $response->withJson($userObject);
});
$app->run();
```

Test using Postman

✓ GET all users: GET http://localhost/users

✓ GET a specific user: GET http://localhost/users/1

- ✓ POST a new user: POST http://localhost/users
- ✓ PUT update a user: PUT http://localhost/users/1
- ✓ PATCH update a user: PATCH http://localhost/users/1
- ✓ DELETE a user: DELETE http://localhost/users/1

Your Final Tasks:

- ✓ **Follow the similar steps above for the resources : /products & /orders.**
- ✓ **complete the implementation of the frontend (Vue project) by integrating it with the backend, specifically for the modules: user login & registration, and product & orders (covering all CRUD operations).**

6. Understanding Web Backend Development

Concept of Web Backend

The backend of a web application is the part that handles the business logic, database interactions, authentication, authorization, and server-side processing. It serves as the intermediary between the frontend (user interface) and the database, ensuring that data is retrieved, processed, and sent back to the frontend securely and efficiently.

Advantages and Disadvantages

Advantages

Scalability: Backend systems can be scaled horizontally by adding more servers or vertically by enhancing server capabilities.
Security: Sensitive operations and data can be handled securely on the server side.
Centralized Data Management: A single point of truth for data makes it easier to manage and maintain consistency.
Performance Optimization: Server-side processing can handle heavy computations and optimizations before sending the data to the client.

Disadvantages

Complexity: Building and maintaining a robust backend can be complex and require specialized knowledge.
Maintenance: Continuous maintenance and updates are necessary to keep the backend secure and efficient.
Latency: Network delays can affect the performance, especially if the backend is not optimized properly.
Cost: Backend infrastructure and development can be costly, especially for large-scale applications.

Technologies for Backend Implementation

Programming Languages: PHP, Python, Java, Ruby, Node.js
Frameworks:

- ✓ PHP: Laravel, Slim, Symfony
- ✓ Python: Django, Flask
- ✓ JavaScript (Node.js): Express.js, Koa.js
- ✓ Java: Spring Boot
- ✓ Ruby: Ruby on Rails

Databases: MySQL, PostgreSQL, MongoDB, SQLite
Web Servers: Apache, Nginx
API Protocols: REST, GraphQL
Authentication: JWT (JSON Web Tokens), OAuth
Tools and Platforms: Docker, Kubernetes, AWS, Azure, Google Cloud

7. Conclusion

In this tutorial, we covered the basics of backend web development using JSON Server, plain PHP, RESTful APIs, and the Slim framework. We also explored the concept of web backend, its advantages and disadvantages, and a variety of technologies available for its implementation. This foundational knowledge equips you with the skills needed to build and manage robust backend systems for web applications.