# Migrating Tasks API from Core PHP to Slim Framework

Presented by: Javeed Mohammad (Yash – Dev)

## Purpose of the Migration

The **Core PHP version**:

* Mixed routing, logic, and database code in one file.
* Required manual handling for every HTTP method.
* Hard to maintain or extend.

The **Slim Framework version**:

* Separates routes, configuration, and logic into modular files (src/, public/, etc.).
* Uses **PSR-7** compliant request/response handling.
* Easier to test with Postman or automated tests.
* Cleaner, secure, and scalable for production.
* In the Slim framework, PSR-7 refers to the standard interfaces for representing HTTP messages, which slim uses for its request and response objects. This allows Slim to be flexible, as it can work with any PHP library that implements the PSR-7 standard, making it interoperable with other frameworks and components. Every Slim route receives a PSR-7 request and is responsible for returning a PSR-7 response object to the client.

## 1. Introduction

The Tasks API project was developed as a backend system designed to manage tasks using CRUD operations (Create, Read, Update, Delete). Initially, it was built using traditional PHP and MySQL. The system was later migrated to the Slim Framework (PHP micro-framework) to achieve better modularity, scalability, and cleaner architecture.

## 2. Core PHP + MySQL Setup

In the earlier version, the application used plain PHP files to handle routes and MySQL queries directly. Routing was handled using conditional statements like $\_SERVER['REQUEST\_METHOD'], and database operations were executed procedurally.

**Example routing in core PHP:**

* <?php  
  if ($\_SERVER['REQUEST\_METHOD'] === 'GET') {  
   $result = mysqli\_query($conn, 'SELECT \* FROM tasks');  
   echo json\_encode(mysqli\_fetch\_all($result, MYSQLI\_ASSOC));  
  } elseif ($\_SERVER['REQUEST\_METHOD'] === 'POST') {  
   $title = $\_POST['title'];  
   mysqli\_query($conn, "INSERT INTO tasks (title) VALUES ('$title')");  
   echo json\_encode(['message' => 'Task created']);  
  }  
  ?>

`

* **Purpose:**

Handles HTTP requests manually using conditional logic to simulate REST API routes.

* **Key Functions / Lines:**
  + $\_SERVER['REQUEST\_METHOD']: Detects the request type (GET, POST, etc.).
  + mysqli\_query (): Executes a SQL command using the MySQL extension.
  + mysqli\_fetch\_all (): Fetches all rows from a result set.
  + json\_encode (): Converts PHP arrays to JSON before sending as an API response.
  + $\_POST['title']: Retrieves input data from an HTML form or Postman request.
* **Workflow:**
  + When the user sends a **GET** request → fetches all tasks from the database.
  + When a **POST** request is received → inserts a new task record into the database.
* **Limitation:**

Hardcoded SQL, no input validation, no error handling, and code duplication across routes.

**Database connection example in Core PHP:**

* <?php  
  $conn = mysqli\_connect ('localhost', 'root', '', 'tasks\_db');  
  if (! $conn) {  
   die ('Connection failed: ‘. mysqli\_connect\_error ());  
  }  
  ?>
* **Purpose:**

Establishes a database connection using the procedural MySQL extension.

* **Key Functions / Lines:**
  + mysqli\_connect (): Opens a connection to the MySQL server.
  + mysqli\_connect\_error (): Returns the reason for connection failure.
  + die (): Immediately stops script execution and outputs an error message.
* **Limitation:**

This is a **procedural approach** with no exception handling — if the connection fails, the script stops entirely.

While functional, this approach led to repetitive code, tight coupling, and difficulty maintaining large applications.

## 3. Slim Framework Setup

Slim Framework provides a modern PHP architecture that follows REST principles. It separates routing, database configuration, and core logic into independent files, improving maintainability and clarity.

**Example Slim routing:**

* <?php  
  $app->get('/tasks', function ($request, $response) use ($pdo) {  
   $stmt = $pdo->query('SELECT \* FROM tasks');  
   $tasks = $stmt->fetchAll();  
   $response->getBody()->write(json\_encode($tasks));  
   return $response->with Header('Content-Type', 'application/json');  
  });  
  ?>
* **Purpose:**

Defines a RESTful route /tasks that returns all tasks using the Slim Framework.

* **Key Components / Lines:**
  + $app->get ('/tasks', ...): Registers a GET route using Slim’s routing system.
  + use ($pdo): Makes the PDO database connection available inside the function.
  + $pdo->query (): Executes an SQL query using PDO (object-oriented DB access).
  + $stmt->fetch All (): Retrieves all records as associative arrays.
  + $response->getBody ()->write (): Writes JSON data into the HTTP response body.
  + ->with Header ('Content-Type', 'application/json'): Sets a response header to tell clients it’s JSON.
* **Improvement:**

Cleaner route handling, automatic request/response objects, and standardized output.

**Database connection example in Slim:**

* <?php  
  $dsn = 'MySQL: host=localhost; dbname=tasks\_db; charset=utf8mb4';  
  $pdo = new PDO($dsn, 'root', '', [  
   PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,  
   PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC  
  ]);  
  ?>
* **Purpose:**

Connects to the MySQL database using PDO (PHP Data Objects) for better security and flexibility.

* **Key Components / Lines:**
  + $dsn: Data Source Name — defines database type, host, name, and charset.
  + new PDO(...): Creates a database connection instance.
  + PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION: Throws exceptions on database errors.
  + PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC: Fetches results as associative arrays instead of numeric ones.
* **Benefits over MySQL:**
  + Object-oriented
  + Built-in error handling
  + Supports multiple database types (MySQL, PostgreSQL, SQLite, etc.)

## 4. Comparison: Core PHP vs Slim Framework

|  |  |  |
| --- | --- | --- |
| Feature | Core PHP + MySQL | Slim Framework |
| Structure | Single index.php file handling all logic | Separated into public/, src/, vendor/ |
| Routing | Conditional if/else with $\_SERVER['REQUEST\_METHOD'] | $app->get(), $app->post(), etc. |
| Database Handling | MySQLi procedural connection | PDO with exception handling |
| Error Handling | Manual die() or echo messages | Middleware-driven error handling |
| Scalability | Hard to extend or modify | Easily scalable, modular architecture |
| Testing | Manual testing via URL parameters | Postman API testing with JSON |
| Extra Features | Limited functionality | Supports PATCH, Middleware, PSR-7 requests |

## 5. Implementation Process

1. Installed XAMPP (Apache + MySQL)  
2. Installed Composer for dependency management  
3. Created tasks-slim project inside htdocs  
4. Installed Slim Framework via composer require slim/slim:^4.0 slim/psr7  
5. Added db.php, routes.php, and index.php under src/ and public/ folders  
6. Tested CRUD and PATCH routes successfully via Postman

## 6. Postman Testing and Results

Each route was tested using Postman for functionality:  
- GET: Fetches all tasks in JSON format  
- POST: Adds new tasks  
- PUT: Updates an existing task  
- DELETE: Removes a task  
- PATCH: Toggles task status between 'pending' and 'completed'

## 7. Conclusion

Migrating from Core PHP to Slim Framework significantly improved the Tasks API’s structure, readability, and performance. Slim’s middleware and routing system allowed clean separation of logic, while PDO improved security and database interaction. This transition demonstrates modern API development principles using PHP.

## 🧩 8. Summary of Key Functions

| **Function / Feature** | **Description** |
| --- | --- |
| $\_SERVER['REQUEST\_METHOD'] | Detects incoming HTTP request type |
| mysqli\_query () | Executes SQL query in Core PHP |
| json\_encode () | Converts PHP array/object to JSON |
| $app->get () | Defines route in Slim |
| $response->getBody ()->write () | Writes response data in Slim |
| PDO: ATTR\_ERRMODE | Enables PDO exception mode |
| PDO: FETCH\_ASSOC | Fetches rows as associative arrays |
| die () | Terminates script on fatal error |