Complete PHP MVC Project Guide

Structure & Flow Explanation

Team Documentation

🏗️Project Architecture Overview

```
team-nieshaan-zoe-php/
  🗀 app/ # Private application code (NOT web-accessible)
     🗀 controllers/ # Business logic handlers
     🗀 models/ # Database interaction layer
     🗀 includes/ # Shared PHP files (config, DB, header, footer)
     🗀 views/ # HTML pages with PHP
  🗀 public/ # Web-accessible files ONLY
     🗀 api/ # API entry point
     🗀 assets/ # Static files (CSS, JS, images)
     🖹 index.php # Optional entry point
```

Key Concept: Only the `public` folder is directly accessible from web browsers. The `app` folder contains private application logic and is protected from direct access.

🔒Security First Design

This architecture follows the principle of least privilege. By separating public assets from application logic, we ensure that sensitive code, database credentials, and business logic cannot be accessed directly through the web browser.

🗁 The App Folder - Application Core

This folder contains all your private application logic. In production, this directory would never be accessible from web browsers.

Controllers - Business Logic Layer

Controllers are similar to those in Express.js or any MVC framework. They handle incoming requests and coordinate between models and responses.

```
// Example: EmployeeController.php
private static function getEmployees() {
    $employees = EmployeeModel::getAllEmployees(); // Call model
    echo json_encode(['employees' => $employees]); // Return JSON
}
```

Models - Data Access Layer

Models handle ALL database queries. No SQL should ever appear in controllers!

```
// Example: EmployeeModel.php
public static function getAllEmployees() {
    $query = "SELECT * FROM employees";
    return db()->query($query); // Returns array of data
}
```

💡 **Best Practice:** Keep your controllers thin and models fat. Business logic and data manipulation should happen in models, while controllers just orchestrate the flow.

Includes - Shared Components

Think of this like a `utils/` or `config/` folder in Node.js projects.

config.php - Environment Variables

```
define('DB_HOST', 'localhost');
define('DB_NAME', 'tracker_db');
define('BASE_URL', '/team-nieshaan-zoe-php/public');
```

db.php - Database Connection

Creates ONE database connection for the entire application (singleton pattern).

header.php & footer.php

Reusable HTML components, similar to React components but for PHP. Include these in every page to avoid repeating navbar and footer code.

Views - Frontend Pages

These are your actual HTML pages that users see. PHP files get processed on the server BEFORE being sent to the browser.

```
Dashboard content here
```

✅ **Important:** Views access CSS/JS using the BASE_URL constant to ensure proper path resolution regardless of where the view file is located.

🌐 The Public Folder

This is the ONLY folder that should be directly accessible from the web. In production, your web server (Apache) points to THIS folder as the document root.

Assets - Static Files

- **css/** - Stylesheets for visual design
- **js/** - Client-side JavaScript that runs in the browser
- **images/** - Images, logos, and graphics

Access Example:

```
http://localhost/team-nieshaan-zoe-php/public/assets/css/main.css
```

🎯 API Router - Backend Entry Point

The `api/index.php` file is THE MOST IMPORTANT FILE for understanding backend flow!

```
// This file receives ALL API requests and routes them
$uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
// URL: /team-nieshaan-zoe-php/public/api/employees/getEmployees
// Extracts: ['employees', 'getEmployees']

switch ($controller) {
    case 'employees':
        require_once '../../app/controllers/EmployeeController.php';
        EmployeeController::handleRequest($action, $method);
        break;
}
```

Think of it like Express.js routing:
`app.get('/api/employees/getEmployees', employeeController.getEmployees)`

🔄 Complete Request Flow

Example: Loading Dashboard with Employees

Step 1: User Visits the Page

```
Browser → http://localhost/team-nieshaan-zoe-php/app/views/home.php
```

↓

Step 2: Server Processes PHP (Server-Side)

```
// home.php runs on the server
```

Server sends final HTML (no PHP code visible in browser!)

↓

Step 3: Browser Loads Assets

Browser downloads:

- /team-nieshaan-zoe-php/public/assets/css/main.css
- /team-nieshaan-zoe-php/public/assets/js/dashboard.js

Step 4: JavaScript Makes API Call

```
// dashboard.js (runs in browser)
const data = await EmployeeAPI.getEmployees();
// Calls: http://localhost/team-nieshaan-zoe-php/public/api/employees/getEmploye
```

↓

Step 5: API Router Receives Request

```
// public/api/index.php receives the request
$uri = '/employees/getEmployees'
$controller = 'employees'
$action = 'getEmployees'
$method = 'GET'

// Routes to controller
require_once '../../app/controllers/EmployeeController.php';
EmployeeController::handleRequest('getEmployees', 'GET');
```

↓

Step 6: Controller Processes Request

```
// EmployeeController.php
public static function handleRequest($action, $method) {
    if ($action === 'getEmployees' && $method === 'GET') {
        self::getEmployees();
    }
}

private static function getEmployees() {
    $employees = EmployeeModel::getAllEmployees();
    echo json_encode(['employees' => $employees]);
}
```

Step 7: Model Queries Database

```
// EmployeeModel.php
public static function getAllEmployees() {
    $query = "SELECT * FROM employees";
    return db()->query($query); // Returns array from DB
}
```

↓

Step 8: JSON Response to Frontend

```
{
  "employees": [
    {"id": 1, "name": "John Doe", "email": "john@example.com"},
    {"id": 2, "name": "Jane Smith", "email": "jane@example.com"}
  ]
}
```

↓

Step 9: JavaScript Updates DOM

```
// dashboard.js receives the data
const data = await EmployeeAPI.getEmployees();
allEmployees = data.employees; // Extract array
renderEmployees(allEmployees); // Update HTML
```

🎉 **Complete!** The page now displays employee data fetched from the database through a clean MVC architecture.

What is .htaccess?

.htaccess is Apache web server configuration. Think of it like middleware in Express.js.

Your .htaccess File:

```
RewriteEngine On
RewriteBase /team-nieshaan-zoe-php/public/

# API Routes - URL rewriting
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^api/(.*)$ api/index.php [L,QSA]

# Security
Options -Indexes  # Prevent directory listing
```

What it does:

1. URL Rewriting - Makes URLs Clean

```
Without htaccess:
/api/index.php?controller=employees&action=getEmployees

With htaccess:
/api/employees/getEmployees  ← Clean URL
```

2. Security - Prevents Directory Browsing

```
Options -Indexes means:
/public/assets/      → Shows files (allowed)
/public/assets/css/  → 403 Forbidden (no listing)
```

3. API Routing - Centralized Entry Point

```
/api/employees/getEmployees  →  api/index.php
/api/admin/allKpiData        →  api/index.php
```

Public vs App Folder

Aspect	public/	app/
Web Access	✅ Direct browser access	❌ No direct access
Purpose	Static files & API entry	Application logic
Contains	CSS, JS, images, index.php	Controllers, Models, Views
Security	Public-facing	Private/Protected
Example Files	main.css, logo.png	EmployeeModel.php

Why This Separation?

Security! You don't want users accessing:

- Database credentials in config.php
- Raw model files with SQL queries
- Controller logic and business rules

Only CSS, JS, images, and the API router should be web-accessible.

How Views Access CSS & JS

Views use the `BASE_URL` constant to build absolute paths:

```
// In app/views/home.php
```

Path Resolution:

```
View location:    /team-nieshaan-zoe-php/app/views/home.php
CSS file:         /team-nieshaan-zoe-php/public/assets/css/main.css

Browser request:  http://localhost/team-nieshaan-zoe-php/public/assets/css/mai
```

Why Use BASE_URL?

- **Consistency:** All paths work the same way across different views
- **Portability:** Easy to move the project to production by changing one constant
- **Reliability:** No relative path confusion (../../)
- **Maintainability:** Update path structure in one place

📊MVC Pattern in This Project

Comparison with Node.js frameworks:

Component	This Project	Express.js Equivalent
Routes	api/index.php	app.get('/api/...')
Controllers	EmployeeController.php	employeeController.js
Models	EmployeeModel.php	Sequelize/Mongoose models
Views	home.php	React/Vue or EJS templates
Config	config.php	.env + config.js
Database	db.php	Sequelize connection
Static Files	public/assets	public/ in Express

🎯 Key Differences from Node.js MVC

1. Server-Side Rendering (SSR)

```
// PHP processes BEFORE sending to browser
  // Runs on server
↓

John Doe

  // Browser receives this
```

2. No Build Step

- No webpack, no npm build
- PHP files run directly on server
- Changes take effect immediately (just refresh browser)

3. Synchronous by Default

```
// PHP is synchronous (no async/await needed for DB)
$data = $model->getData();  // Waits for DB
echo json_encode($data);    // Then responds
```

4. Include Instead of Import

```
require_once 'file.php';  // Like import in JS
```

🚀 **Pro Tip:** PHP's synchronous nature makes it simpler to reason about code flow, but remember that includes happen at runtime, not compile-time like JS imports.

Common Gotchas

1. Path Resolution

```
// Relative paths are from the CURRENT file location
__DIR__                // Current directory
__DIR__ . '/../'       // Go up one level
```

2. Browser Caching

- **CSS/JS changes not showing?** Hard refresh: `Ctrl+Shift+R` (or `Cmd+Shift+R` on Mac)
- **PHP changes** show immediately (no cache)

3. Error Visibility

```
// In config.php
error_reporting(E_ALL);      // Show all errors
ini_set('display_errors', 1); // Display in browser
```

4. Database Connection Issues

- Always check your credentials in config.php
- Ensure MySQL service is running
- Verify database exists and user has permissions

5. Case Sensitivity

- PHP function names are case-insensitive
- Class names ARE case-sensitive
- File names on Linux servers ARE case-sensitive

📝 Summary for Your Team

"This is a traditional PHP MVC application with a modern API architecture"

Key Components:

1. Frontend: HTML pages (views) with client-side JavaScript

2. Backend API: RESTful JSON API (index.php routes to controllers)

3. Database: MySQL accessed through model layer

4. MVC Pattern:
• Models talk to database
• Controllers handle business logic
• Views render HTML

5. Security: Only public folder is web-accessible

6. No Build Tools: Pure PHP, no compilation needed

🔄 Complete Flow Summary

Browser