Root Level Files

composer.json

Purpose: Defines project dependencies and metadata

Content: Contains project name, description, required packages, autoloading settings, scripts, and PHP version requirements

Example:

json

{

"name": "your-vendor/your-project",

"description": "Project description",

"type": "project",

"require": {

"php": "^8.1",

"monolog/monolog": "^2.8"

},

"autoload": {

"psr-4": {

"App\\": "src/"

}

}

}

composer.lock

Purpose: Locks dependencies to specific versions

Content: Auto-generated file that records exact versions of all installed packages

Importance: Should be committed to version control to ensure all developers use identical dependencies

.env

Purpose: Stores environment-specific configuration values

Content: API keys, database credentials, application settings

Security: Should never be committed to version control

Example:

DB\_HOST=localhost

DB\_NAME=my\_database

DB\_USER=user

DB\_PASS=password

APP\_ENV=development

.env.example

Purpose: Template for the .env file

Content: Same structure as .env but with placeholder values

Usage: Developers copy this file to create their own .env

.gitignore

Purpose: Specifies files/directories to exclude from version control

Content: Lists patterns of files to ignore (vendor/, .env, logs/, etc.)

Example:

/vendor/

/node\_modules/

.env

/storage/logs/\*

/public/uploads/\*

README.md

Purpose: Project documentation

Content: Installation instructions, configuration, usage examples, contributing guidelines

Directories

public/

Purpose: The only web-accessible directory

Security: All other directories are inaccessible via HTTP requests

Contents:

index.php: The entry point for all HTTP requests

Bootstraps the application

Routes requests to appropriate controllers

assets/: Contains static files

css/: Stylesheets (compiled and source)

js/: JavaScript files (compiled and source)

images/: Image assets

.htaccess: Apache configuration for URL rewriting and security

Enables clean URLs (without index.php)

Prevents direct access to sensitive files

src/

Purpose: Application source code (following PSR-4 autoloading)

Contents:

Controllers/: Handle HTTP requests and responses

Receive input from routes

Delegate to services/models

Return appropriate responses

Models/: Represent data and business logic

Define database relationships

Implement data validation rules

Views/: Template files

Usually in PHP, Twig, Blade, or other templating systems

Responsible for HTML output

Services/: Business logic separated from controllers

Reusable functionality

Third-party integrations

Helpers/: Utility functions and classes

Date/time manipulation

String formatting

Common operations

config/

Purpose: Configuration settings for the application

Contents:

app.php: General application settings

Timezone, locale, debugging mode

database.php: Database connection settings

Multiple connection configurations

Query logging settings

routes.php: URL routing definitions

Maps URLs to controller actions

database/

Purpose: Database-related files

Contents:

migrations/: Version-controlled database schema changes

Create/modify tables

Add/modify columns

Create indexes

seeds/: Populate database with test/initial data

Default users

Reference data

tests/

Purpose: Automated tests

Contents:

Unit/: Tests for individual components in isolation

Model tests

Service tests

Helper tests

Integration/: Tests for component interactions

API endpoint tests

Database integration tests

vendor/

Purpose: Third-party dependencies

Content: Packages installed via Composer

Management: Completely managed by Composer, never edit manually

Version Control: Should be excluded from git (.gitignore)

storage/

Purpose: Application-generated files

Contents:

logs/: Application log files

Error logs

Activity logs

System events

cache/: Generated cache files

View cache

Data cache

Route cache

uploads/: User-uploaded files

Images

Documents

Media files

bootstrap/

Purpose: Application initialization

Contents:

app.php: Bootstraps the application

Loads environment variables

Registers service providers

Sets up error handling

Initializes key components

Additional Components in Advanced Projects

cli/

Command-line interface scripts for maintenance tasks, cron jobs, and administrative functions

resources/

Raw assets (SCSS, JavaScript modules) that need compilation

Language files for internationalization

docs/

Additional documentation beyond the README

API documentation

Developer guides

bin/

Executable scripts and tools

Deployment scripts

Development utilities

This structured approach helps maintain a clean codebase, provides clear separation of concerns, and follows modern PHP development practices. It's adaptable to the specific needs of your project and can be modified to work with various PHP frameworks.