**Build A Full PHP MVC Blog From Scratch**

**Step 1: Environment Setup**

**- Install XAMPP:** Begin by downloading and installing XAMPP from the official website. This package contains the necessary tools such as Apache server, MySQL, and PHP to run the blog locally.

Here is the [link](https://www.apachefriends.org/download.html) to the xampp software or copy the link ***https://www.apachefriends.org/download.html*** and paste it on a browser URL bar.

**- Visual Studio Code:** Make sure to have Visual Studio Code installed. Enhance your coding experience with PHP extensions like PHP Intelephense, which provides improved code navigation and IntelliSense.

Get it here <https://code.visualstudio.com/>

**Step 2: Project Initialization**

**- Create a New Project Directory:** Establish the workspace by creating a new directory dedicated to this project.

cd xampp/htdocs

mkdir xampp/htdocs/PHPMVCBlog

and create your project there.

Mine I will name it **PHPMVCBlog.**

**- Initialize Git:** Implement version control by initializing a new Git repository within the project directory. This practice is essential for tracking changes and managing code versions throughout the development process.

git init

git add .

git commit -m "Initial commit"

git branch -M main

git remote add origin https://github.com/yourusername/PHPMVCBlog.git

git push -u origin master  # or 'main'

For incremental changes, the workflow is essentially:

1. Stage changes: **git add .** or **git add <specific-file-or-folder>**
2. Commit changes: **git commit -m "Your commit message"**
3. Push changes: **git push** (if the branch is already set up with upstream) or **git push -u origin main** (the first time you push your main branch, which you've already done).

**Step 3: Composer and Autoloading**

1. **Install Composer**: If not already installed, get Composer for managing dependencies.

How to check if installed.

Composer -v

If not installed go here [*https://getcomposer.org/*](https://getcomposer.org/)follow the instructions and you’re good to go.

* **Autoloading**: Set up autoloading using Composer by creating a **composer.json** file with the appropriate autoloading configuration.

**ii) Set Up Project Autoloading with Composer**

Once Composer is installed, you'll set up autoloading for your PHP MVC project. Here's how to do it:

**1. Create the composer.json file:**

* In your project root directory, create a file named **composer.json**.
* Add the following content to it:

{

    "name": "brian/my-mvc-blog",

    "description": "A Light Weight PHP Application",

    "type": "library",

    "license": "MIT",

    "autoload": {

        "psr-4": {

            "app\\": "app/"

        }

    },

    "authors": [

        {

            "name": "Brian"

        }

    ],

    "minimum-stability": "beta",

    "require": {}

}

The **composer.json** file is a configuration file for Composer, which manages dependencies for PHP projects. Here's a simple explanation of what this file does:

* **name**: This specifies a unique identifier for your project, typically using the format **vendor/package**.
* **description**: A brief description of what your project is.
* **type**: It indicates the type of project. Common types include **project** for applications and **library** for reusable packages.
* **license**: This defines the software license under which your project is released. MIT is one of the most open and permissive licenses.
* **autoload**: This section is important for defining how PHP class files are automatically loaded without the need for manual **require** statements. The **psr-4** entry tells Composer to use the PSR-4 standard for autoloading. It maps a namespace prefix to a directory.
	+ In this case, any PHP class that starts with the namespace **app\** will be autoloaded from the **app/** directory.
* **authors**: Lists the authors of the project.
* **minimum-stability**: This sets the minimum stability level for your dependencies. By setting it to **beta**, Composer will allow you to install dependencies that are still in beta.
* **require**: Lists the packages that your project depends on. Since it's empty, your project currently does not require any external packages.

The **autoload** section is crucial because it allows you to organize your project's PHP classes into namespaces, which helps prevent class name conflicts and makes your code easier to maintain. With autoloading, you don't need to manually include each class with a **require** or **include** statement. Instead, when you instantiate a new object, Composer's autoloader will automatically include the necessary file based on the namespace and class name.

**iii. Generate the autoloader:**

* Open the command prompt or terminal.
* Navigate to your project directory.
* Run **composer dump-autoload**. Composer will create the **vendor** directory and the autoloading files within it.

**iv.Update .gitignore:**

* Create a **.gitignore** file if it doesn't exist and add the following lines to avoid tracking vendor and other non-essential files:

/vendor/

/.vscode/

/node\_modules/