Welcome Letter

To the new student beginning this journey

Dear Student,

Welcome to your new adventure in learning how to code.
I'm genuinely grateful and excited that you're here. This journey you're starting is not just about PHP, Laravel, Vue, or Docker — it's about discovering what you're capable of, even when something feels completely new or unfamiliar.

You may feel nervous.
You may feel unsure.
You may even feel like you don't belong in the world of programming.

I want you to know: **those feelings are normal** — and more importantly, they do **not** define what you can achieve.

I've spent many years as a software developer, and I can tell you with complete honesty:
the best developers are not the ones who never make mistakes.
They're the ones who keep trying, keep learning, and keep asking questions.

That's exactly what I hope for you.

This bootcamp is made for you — step-by-step, beginner-friendly, paced so you can grow steadily even if you only have 30 minutes to 1 hour a day. You don't need a technical background, and you don't need to understand everything right away. Coding is a skill, and like acupuncture, it takes patience, practice, and curiosity.

You will have days where everything clicks.
You will have days where nothing seems to work.
But both kinds of days move you forward.

Please remember that you're not alone on this journey.
You have tools to guide you — Visual Studio Code, Docker, your GitHub repo, documentation libraries, and yes, ChatGPT. Use them freely. Asking questions is not cheating; it's how real developers learn.

I will be guiding you through this program, and everything you need — every exercise, note, cheat sheet, and project — has been carefully prepared so you can learn with confidence.

Most of all, I want you to enjoy this.
To celebrate small wins.
To be proud of every step.
Because choosing to learn something new — especially something challenging — already makes you brave.

Thank you for trusting me with your learning journey.
I believe in you.
And I’m excited to see what you build.

Sincerely,

Keen Rosal
Your guide, mentor, and fellow lifelong learner

Introduction

💛 INTRODUCTION: A Gentle Start to Your Coding Journey

For adults learning programming for the first time

Learning to code can feel intimidating — especially if you've spent most of your life in a completely different profession. Maybe you're thinking:

- "What if I'm not smart enough?"
- "What if I don't understand it?"
- "What if it's too technical?"

These worries are **normal**.

Every developer — even senior engineers with 20 years of experience — started exactly where you are: **confused, slow, uncertain, and afraid of messing up.**

The truth is:

You do not need to be "technical" to learn programming.
You only need patience, consistency, and the willingness to try.

🌱 Coding Is a Skill, Not a Talent

Just like acupuncture, cooking, learning a language, or playing piano — coding is a skill you build through:

- **Repetition**
- **Practice**
- **Following steps**
- **Making mistakes**

- **Trying again**

Your brain learns one tiny piece at a time.

You don’t have to memorize everything.
You don’t have to understand everything instantly.
You don’t have to be perfect.

You just have to **show up**, even for 30 minutes.

🌤️ Be Patient With Yourself

Some days, things will click.
Some days, you’ll stare at the screen confused.
Both are part of the process.

Be kind to yourself.
Take breaks.
Celebrate small wins.

The goal is **progress**, not perfection.

🤝 You Are Not Alone — Use Your Resources

Great programmers don’t know everything.
They simply know how to **find answers**.

You will use tools like:

✔ ChatGPT

To ask questions, fix mistakes, explain errors, and learn concepts.

✔ PHP Documentation (php.net)

To look up functions, read examples, and understand how PHP works.

✔ Online Guides & Tutorials

If you're unsure, you can always search for simple explanations.

✔ Your GitHub Repository

All starter code, notes, and exercises are organized for you.

✔ Friends, mentors, and your support system

Asking for help is a sign of strength, not weakness.

Good developers ask questions all the time.
Great developers ask questions **even more**.

🧘 You’re Exactly Where You Should Be

If you’re nervous, that’s okay.
If you're curious, that’s enough.
If you’re willing to try, you can learn this.

Coding is not about being the smartest person in the room —
it’s about being the most **persistent**.

Every step you take is a win.
Every exercise you complete is progress.
Every error you fix makes you better.

And at the end of this journey, you’ll look back and realize:

“I can do this. I DID this.”

Monthly Lessons

🌟 Overall Goal (6-8 Months)

By the end of 8 months, she will be able to:

✔ Build functional PHP applications
✔ Build simple Laravel apps (CRUD, authentication, APIs)
✔ Understand the basics of backend development
✔ Solve beginner HackerRank challenges
✔ Push projects to GitHub
✔ Apply to junior roles or freelance PHP/Laravel jobs

🌿 Core Principles (Designed for Adult Beginners)

- **Short, digestible lessons** (15–45 minutes)
- **No jargon** — simple language
- **Hands-on, not theoretical**
- **Repeating patterns to reinforce learning**
- **End-of-week review days**
- **End-of-month mini-project**
- **Everything running inside Docker (stable & predictable)**

📅 8-Month Daily Learning Plan

⏱ 30–60 minutes per day

🧘 5 days/week (with built-in rest)

📘 MONTH 01 — PHP Basics (Days 1–20)

🧩 *Goal: Comfort with syntax, variables, arrays, loops, and functions.*

Week 1 — Foundations

- **Day 1:** Intro to PHP, running Docker
- **Day 2:** Variables
- **Day 3:** Strings + numbers
- **Day 4:** Arrays
- **Day 5:** Exercise Review + Reflection

Week 2 — Logic

- **Day 6:** If/else
- **Day 7:** Comparison + logical operators
- **Day 8:** Loops (for / while)
- **Day 9:** Loops practice
- **Day 10:** Review day

Week 3 — Functions

- **Day 11:** Basic functions
- **Day 12:** Function parameters
- **Day 13:** Return values
- **Day 14:** Functions practice
- **Day 15:** Review

Week 4 — Mini Project

- **Day 16:** Start — build “Daily Wellness Tracker” (simple CLI)
- **Day 17:** Continue
- **Day 18:** Add user input
- **Day 19:** Add simple storage (text JSON)
- **Day 20:** Final polish + reflection

📘 MONTH 02 — PHP OOP (Days 21–40)

🧩 *Goal: Understand classes, objects, methods, inheritance, traits.*

Week 1

- **Day 21:** What is a class?
- **Day 22:** Properties + methods
- **Day 23:** Constructors
- **Day 24:** Build simple class
- **Day 25:** Review

Week 2

- **Day 26:** Inheritance
- **Day 27:** Overriding methods
- **Day 28:** Interfaces
- **Day 29:** Traits

- **Day 30:** Review

Week 3

- **Day 31:** Practice OOP exercises
- **Day 32:** Create your own class
- **Day 33:** Extend it
- **Day 34:** Add interface/trait
- **Day 35:** Review

Week 4 — Mini Project

🧘‍♀️ *"Acupuncture Appointment Scheduler" (OOP only)*

- **Day 36:** Class design
- **Day 37:** Create appointment class
- **Day 38:** Add filters + sorting
- **Day 39:** Export data
- **Day 40:** Final review

📘 MONTH 03 — PHP + MySQL (Days 41–60)

🧩 *Goal: Learn CRUD, databases, PDO.*

Week 1 — MySQL Basics

- **Day 41:** Intro to DB + Docker

- **Day 42:** Tables + rows
- **Day 43:** SELECT
- **Day 44:** INSERT
- **Day 45:** Review

Week 2 — PDO

- **Day 46:** Database connection (PDO)
- **Day 47:** SELECT with PDO
- **Day 48:** INSERT with PDO
- **Day 49:** UPDATE + DELETE
- **Day 50:** Review

Week 3 — CRUD App

- **Day 51:** Build CREATE page
- **Day 52:** Build READ list
- **Day 53:** Build UPDATE form
- **Day 54:** Build DELETE
- **Day 55:** Review

Week 4 — Mini Project

🧘 *“Client Treatment Log System”*

- **Day 56:** Table structure
- **Day 57:** Add treatment

- **Day 58:** View list
- **Day 59:** Edit & delete
- **Day 60:** Final review

📘 MONTH 04 — Laravel Basics (Days 61–80)

🧩 *Goal: MVC, routing, controllers, views, migrations.*

Week breakdown:

- Routing
- Controllers
- Blade templates
- Database migrations
- Models
- Forms

📌 **Project:** Simple “Journal App”

📘 MONTH 05 — Laravel Advanced (Days 81–100)

🧩 *Goal: APIs, validation, resources, relationships, auth.*

📌 **Project:** “Habit Tracker API”

📘 MONTH 06 — Vue Basics (Days 101–120)

🧩 *Goal: Components, props, events, router.*

📌 **Project:** “Meditation Timer SPA”

📘 MONTH 07 — Vue Advanced (Days 121–140)

🧩 *Goal: Pinia, API calls, patterns, state management.*

📌 **Project:** “Wellness Dashboard”

📘 MONTH 08 — Fullstack Apps (Days 141–160)

🧩 *Goal: Connecting Laravel API ↔ Vue Frontend*

📌 **Capstone Project:**
“Wellness Journal — Fullstack Application”
(in your repo)

Daily tasks:

- Set up backend API
- Protect routes
- Build SPA pages

- Connect forms
- Style with Tailwind
- Deploy (optional)

🎉 Total Time Commitment

- **5 days/week**
- **30–60 minutes per day**
- Duration: **8 months**
- Beginner-friendly pace
- Zero technical background required

Month 0

📦 MONTH 0 — WELCOME PACKAGE

A Beginner-Friendly Start Before Your Coding Journey Begins

💛 1. Welcome Message

Before you write your first line of code, I want you to know this:

You *can* learn programming.
You *don't* need to be technical.
You *don't* need to understand everything at once.
You *just* need patience, curiosity, and 30–60 minutes a day.

Everything else, we will learn together — step by step.

This Month 0 is your soft landing.
It prepares your computer **and your mindset** for success.

🧭 2. What You Will Do in Month 0

Month 0 helps you:

✔ Install the tools you will use
✔ Learn what each tool is for
✔ Set up your GitHub repository
✔ Open your project in VS Code
✔ Understand Docker containers
✔ Build confidence before starting Month 01

No coding yet — just preparation.

🖥️ 3. Install the Tools

🔹 TOOL 1 — Visual Studio Code (Your Coding Workspace)

What is VS Code?

A smart, beginner-friendly editor where you write, organize, and view your code.
Like Microsoft Word, but for coding.

Why you need it

- You edit every lesson here
- You run commands through the built-in terminal
- It highlights errors for you
- Professionals use it daily

Install steps

👉 https://code.visualstudio.com/

- Download
- Install
- Open the app
- Keep the icon in your Dock or Taskbar

Extensions to install

Inside VS Code → Extensions (left sidebar)

Search and install:

- PHP Intelephense
- Laravel Extra Intellisense
- Blade Snippets

- Vue Language Features (Volar)
- Prettier
- Docker Extension
- GitLens

🔹 TOOL 2 — Docker Desktop (Your Portable Coding Environment)

What is Docker?

A tool that gives you a clean, isolated space to run:

- PHP
- MySQL
- Node.js
- Laravel
- Vue
- Databases
- Web servers

So your computer never breaks or gets messy.

Why you need it

✔ You don’t need to install PHP manually
✔ You don’t need to worry about versions
✔ Everything “just works”
✔ Matches real-world developer environments

Install steps

👉 https://www.docker.com/products/docker-desktop/

After installing:

Open Terminal → type:

docker --version

If you see a version number → it works.

🔹 TOOL 3 — GitHub (Your Online Folder for Projects)

What is GitHub?

A website where you store, back up, and share your coding projects.

Think of it like a Google Drive for developers.

Why you need it

- You download the bootcamp repo
- You'll store your own projects
- Employers will see your progress
- You'll track changes safely

Steps

1. Visit your repository link
2. Click **Code** → **Download ZIP**
3. Unzip
4. Move the folder somewhere you can find easily
5. Open in VS Code

📁 4. Open Your Bootcamp in VS Code

Once you've downloaded your repo:

Step-by-step

1. Open **VS Code**
2. Click **File → Open Folder**
3. Select your repo folder:

bootcamp-fullstack-php-laravel-vue-docker

4. You should now see all folders:

month01-php-basics
month02-php-oop
month03-php-mysql
...
month08-fullstack-projects

You're ready.

🐳 5. Understanding Docker (Super Simple Explanation)

Imagine your project needs:

- PHP
- MySQL

- Node
- Composer
- Laravel
- Vue
- A web server
- Tools and libraries

Instead of installing them yourself, Docker gives you “mini-computers” called **containers** with everything already inside.

Why this is amazing

- No breaking your computer
- No version conflicts
- Clean, safe, controlled environment
- Easy to reset if something goes wrong

One command starts everything

docker-compose up -d

Done.

🧘 6. Mindset: How to Learn Without Feeling Overwhelmed

Learning to code is like learning a new language, or like starting acupuncture for the first time:

- It looks hard at first

- It becomes easier with practice
- Your hands start to “remember” the motions
- Eventually everything flows

Remember these rules:

✔ You don’t need to memorize everything
✔ Mistakes are normal (and expected!)
✔ Ask lots of questions — professionals still do
✔ Google + ChatGPT are part of the job
✔ Celebrate small wins
✔ Progress > perfection
✔ Be patient with yourself

You are not behind.
You are learning at the perfect pace.

📚 7. How to Ask Good Questions (Beginner Guide)

When confused, ask:

- “What does this error mean?”
- “Can you explain this line to me?”
- “Why is this not working?”
- “How do I fix this?”
- “What is the simplest way to understand this?”

Tools like ChatGPT, PHP.net, and documentation exists **because confusion is normal**.

🧱 8. Month 0 Mini Task Checklist (No Code Yet)

✔ Install VS Code
✔ Install Docker Desktop
✔ Download GitHub Repository
✔ Verify Docker is working
✔ Open repo in VS Code
✔ Explore the folders
✔ Read the Month 01 overview
✔ Breathe — you're ready

🎉 9. Your Journey Starts Tomorrow (Month 01)

The hard part — getting ready — is done.
From here, each lesson is small, manageable, and beginner-friendly.

You're not expected to know everything.
You're only expected to **show up**.

This is the beginning of something great.

Month 1

📘 MONTH 01 — PHP BASICS (Beginner-Friendly Curriculum)

Duration: 20 days (4 weeks)
Time per day: 30–60 minutes
Goal: Learn fundamental PHP skills — variables, arrays, loops, functions — using your Docker environment.

🌿 Overview

Month 01 introduces absolute beginners to programming concepts gently and gradually.
This month builds confidence, familiarity with code, and comfort running PHP scripts in Docker.

Each day includes:

- A **simple lesson**
- A **hands-on task**
- A **1-line reflection** (optional)

🧰 Required Setup

Folder: `month01-php-basics`
Run Docker:

cd month01-php-basics/docker
docker compose up -d
docker compose exec php bash

Your working directory inside Docker:

/var/www/html

📅 WEEK 1 — PHP Foundations

Goal: Understand basic syntax and how code runs.

Day	Topic	What You Do
Day 1	Welcome to PHP + Docker	Run your first PHP script (“Hello World”). Learn how to enter the container.
Day 2	Variables	Create variables like `$name` and `$age`. Echo them.
Day 3	Strings & Numbers	Practice concatenation and simple math.
Day 4	Arrays	Create a list of items and loop through them.
Day 5	Review Day	Re-run Day 1–4 exercises and write a 1-sentence summary of each.

📅 WEEK 2 — Logic & Thinking Like a Programmer

Goal: Understand decision-making and repetition.

Day	Topic	What You Do
Day 6	Conditions (if/else)	Build simple branching logic.
Day 7	Comparisons + Logical Operators	Practice `==`, `===`, `>`, `<`, `&&`, `
Day 8	Loops: `for`	Loop 1–10 with simple messages.
Day 9	Loops: `while`	Convert Day 8 `for` loop to `while`.

Day 10	Review Day	Combine loops + conditions in a simple exercise.

📅 WEEK 3 — Functions (Your First "Tools")

Goal: Write reusable pieces of code.

Day	Topic	What You Do
Day 11	What is a function?	Create a function that prints a greeting.
Day 12	Function Parameters	Add a `$name` parameter to your greeting function.
Day 13	Return Values	Write functions that calculate values.
Day 14	Practice Day	Build your own utility function (ex: BMI calculator).
Day 15	Review	Run all exercises from Days 11–14.

📅 WEEK 4 — Mini Project (Wellness Tracker)

Goal: Build a small real program using everything you learned.

Mini Project: "Daily Wellness Tracker (CLI Version)"
You will build a script that:

- Asks the user for daily inputs:
    - Mood (1–10)
    - Hours slept
    - Stress level

- Stores each day's entry (into a text file or JSON)
- Displays a summary

Day	Task
Day 16	Create project file + ask user for input
Day 17	Store the data (text or JSON file)
Day 18	Display stored entries in the terminal
Day 19	Add averages (ex: average sleep)
Day 20	Final polish + reflection

🧪 Included Exercises (already in your repo)

Inside:
`month01-php-basics/exercises`

You have:

1. **01_variables.php**
2. **02_arrays.php**
3. **03_loops.php**
4. **04_functions.php**

Each of these can be run using:

php /var/www/html/../exercises/FILENAME.php

🎯 Learning Goals for Month 01

By the end of Month 01, you will be able to:

- Run PHP scripts confidently in Docker
- Write and use variables
- Create and loop through arrays
- Use if/else logic
- Write functions
- Build a small working CLI application

Everything here is foundational and sets the stage for Month 02 (OOP).

Cheatsheet1

📄 MONTH 01 — PHP BASICS CHEAT SHEET (PRINTABLE)

For absolute beginners — 30–60 minutes/day

🚀 How to Run PHP (Using Docker)

Start and enter container:

```
cd month01-php-basics/docker
docker compose up -d
docker compose exec php bash
```

Run a PHP file:

```
php filename.php
```

🔤 PHP Syntax Essentials

Echo/Print

```
echo "Hello World!";
```

Variables

```
$name = "Keen";
$age = 30;
```

Strings & Numbers

```
echo $name . " is " . $age;
$sum = 5 + 10;
```

📦 Arrays

```
$fruits = ["apple", "banana", "orange"];
echo $fruits[0];
```

Looping arrays:

```
foreach ($fruits as $fruit) {
    echo $fruit;
}
```

🔀 Conditions

```
if ($age > 18) {
    echo "Adult";
} else {
    echo "Minor";
}
```

Comparison Operators:

- `==` equal
- `===` equal + same type
- `!=` not equal
- `>` greater than
- `<` less than
- `&&` AND
- `||` OR

🔁 Loops

For Loop

```
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
```

While Loop

```
$i = 1;
while ($i <= 10) {
    echo $i;
    $i++;
}
```

🛠️ Functions

Basic Function

```
function greet() {
    echo "Hello!";
}
```

With Parameter

```
function greet($name) {
    echo "Hello, $name!";
}
```

With Return Value

```
function add($a, $b) {
    return $a + $b;
}
```

📘 Month 01 Mini Project

"Daily Wellness Tracker" (CLI App)
You can include:

- Ask user for:
    - Mood (1–10)
    - Hours slept
    - Stress level
- Save to a simple file
- Show history
- Show averages

🎯 What You Should Know by End of Month 01

✔ Run PHP scripts
✔ Work with variables
✔ Build arrays
✔ Use if/else logic
✔ Write loops
✔ Create functions
✔ Build small real programs

Month 2

📘 MONTH 02 — PHP OOP (Beginner-Friendly Curriculum for Non-Technical Adults)

Duration: 20 days (4 weeks)
Time per day: 30–60 minutes
Goal: Understand PHP classes, objects, methods, inheritance, and basic design.

🌿 Overview

In Month 02, the learner begins to “think in objects,” which is a gentle shift from Month 01. This month emphasizes:

- real-world metaphors
- hands-on exercises
- repetition to build confidence
- no complex theory

By the end of the month, the learner can create simple structured programs using classes and objects.

🧰 Required Setup

Folder: `month02-php-oop`
Start Docker:

```
cd month02-php-oop/docker
docker compose up -d
docker compose exec php bash
```

Your working directory inside Docker will be:

/var/www/html

📅 WEEK 1 — Understanding Objects & Classes

Goal: Learn what an object is and how to create one.

Day	Topic	What You Do
Day 21	What is OOP? Objects in real life	Learn analogies: “class = blueprint,” “object = instance.” Create your first class.
Day 22	Properties (variables inside objects)	Add `$name`, `$age`, `$title` to a class. Instantiate objects.
Day 23	Methods (functions inside objects)	Add methods like `introduce()` or `addNumber()`.
Day 24	Constructors	Learn `__construct()` and set default values.
Day 25	Review + small exercise	Build a simple class from scratch.

📅 WEEK 2 — Inheritance & Code Reuse

Goal: Learn how classes can extend other classes.

Day	Topic	What You Do
Day 26	Inheritance basics	Create a parent class (“Person”) and child class (“Acupuncturist”).
Day 27	Overriding methods	Override a method (ex: child class custom greeting).

Day 28	Interfaces	Learn: “A contract for methods.” Build simple interface.
Day 29	Traits	Understand reusable behavior. Add a “LoggerTrait.”
Day 30	Review	Combine inheritance + interface + trait into a small script.

📅 WEEK 3 — Applying OOP Concepts

Goal: Build confidence by creating small reusable components.

Day	Topic	What You Do
Day 31	Create your own class	Pick a real-world item (tea, needle kit, appointment).
Day 32	Add methods	Ex: calculate duration, format dates, compute totals.
Day 33	Add inheritance	Create a more specific version (e.g., “FollowUpAppointment”).
Day 34	Add interface or trait	Apply a reusable method like logging or formatting.
Day 35	Review	Organize classes into folders (`src/Entities`).

📅 WEEK 4 — Mini Project: “Acupuncture Appointment Scheduler”

Goal: Build a small real app using OOP principles.

Day	Task
Day 36	Design your classes (Appointment, Client, Scheduler)

Day 37	Implement `Appointment` class with date/time and notes
Day 38	Add list of appointments + filtering
Day 39	Add method to reschedule or cancel
Day 40	Final polishing + review

🧪 Included Exercises (already in your repo)

Inside:
`month02-php-oop/exercises`

Includes:

- Class creation exercises
- Constructor exercises
- Inheritance mini tasks
- Trait + interface practice

Run them via:

php /var/www/html/../exercises/FILENAME.php

🎯 What You Should Know by the End of Month 02

By the end of this month, you will confidently understand:

✔ What classes and objects are
✔ How methods and properties work

✔ How to create and extend classes
✔ How to use constructors
✔ The purpose of interfaces and traits
✔ How to assemble a simple OOP-based app

This foundation prepares you perfectly for **Month 03 (PHP + MySQL)**, where they connect OOP to databases.

Cheatsheet2

📄 MONTH 02 — PHP OOP CHEAT SHEET (PRINTABLE)

A beginner-friendly guide for adults with no technical background

---

🌿 What Is OOP?

- **Class** → A blueprint (like a template).
- **Object** → A real thing created from the class.
- **Property** → Data inside an object.
- **Method** → Actions an object can take.

Example:

```
class Person {
   public $name;
   public function sayHello() {
      echo "Hello!";
   }
}
```

🏠 Classes & Objects

Define a class:

```
class Animal {
   public $type = "Dog";
}
```

Create an object:

```
$pet = new Animal();
```

```
echo $pet->type;
```

🔧 Methods (Functions Inside Classes)

```
class Calculator {
    public function add($a, $b) {
        return $a + $b;
    }
}
```

Use it:

```
$calc = new Calculator();
echo $calc->add(2, 3);
```

🚪 Constructors

Used to set initial values automatically.

```
class Person {
    public $name;
    public function __construct($name) {
        $this->name = $name;
    }
}
```

🧬 Inheritance (Extending Classes)

A child class inherits from a parent class.

```
class Person {
    public function greet() {
        echo "Hello!";
    }
}

class Student extends Person {
```

```
}
```

Use it:

```
$s = new Student();
$s->greet();
```

🎨 Overriding Methods

Child class replaces parent behavior.

```
class Student extends Person {
    public function greet() {
        echo "Hi, I'm a student!";
    }
}
```

📜 Interfaces

A contract that ensures a class has certain methods.

```
interface Logger {
    public function log($msg);
}

class FileLogger implements Logger {
    public function log($msg) {
        echo $msg;
    }
}
```

🔩 Traits

Reusable pieces of code.

```
trait Timestamps {
```

```
    public function now() {
        return date('Y-m-d H:i:s');
    }
}

class Post {
    use Timestamps;
}
```

📦 Organizing OOP Code

Suggested folder structure:

```
src/
  Classes/
  Models/
  Interfaces/
  Traits/
```

🌱 Mini Project (End of Month 02)

Acupuncture Appointment Scheduler

Includes:

- Appointment class
- Client class
- Scheduler class
- Methods for booking, viewing, canceling
- Real-world OOP application

🎯 What You Should Know by the End of Month 02

✔ What classes, objects, methods, and properties are
✔ How constructors work
✔ How inheritance helps reuse code
✔ When to use interfaces and traits
✔ How to build a real program using OOP

Month 3

📘 MONTH 03 — PHP + MySQL (Beginner-Friendly Curriculum)

Duration: 20 days (4 weeks)
Time per day: 30–60 minutes
Goal: Learn how to store, retrieve, update, and delete data using MySQL and PDO.

🌿 Overview

Month 03 introduces databases — a core skill of all backend developers.

This month teaches:

- How data is stored
- How to interact with a database
- How to build CRUD (Create, Read, Update, Delete) applications
- How PHP connects to MySQL using PDO

Everything is taught with **simple language**, real-world metaphors, and repetition for confidence.

🧰 Required Setup

Folder:
`month03-php-mysql`

1. Start Docker

cd month03-php-mysql/docker
docker compose up -d

2. Enter the PHP container

docker compose exec php bash

3. Verify MySQL is running

Inside container:

mysql -h mysql -u root -p

Password: `password`

You're now ready for Month 03 lessons.

📅 WEEK 1 — Database Foundations

Goal: Understand how databases store structured information.

Day	Topic	What You Do
Day 41	What is a database? Why MySQL?	Learn tables, rows, columns. View your `students` table from init.sql.
Day 42	SELECT	Run simple queries like: `SELECT * FROM students;`
Day 43	INSERT	Add rows manually into the database.
Day 44	UPDATE	Modify existing data.
Day 45	Review Day	Run SELECT/INSERT/UPDATE/DELETE in MySQL directly.

📅 WEEK 2 — Connecting PHP to MySQL (PDO)

Goal: Learn how PHP talks to the database.

Day	Topic	What You Do
Day 46	PDO connection	Write `pdo.php` to connect to MySQL using DSN.
Day 47	SELECT with PDO	Fetch all students using `prepare()` and `execute()`.
Day 48	INSERT with PDO	Insert form-like data using placeholders (`:name`).
Day 49	UPDATE + DELETE	Update rows, delete rows, practice error handling.
Day 50	Review Day	Rewrite all queries using prepared statements.

📅 WEEK 3 — Building Your First CRUD App

Goal: Combine PHP + MySQL + HTML basics (still simple).

You will build a **Student Manager** with 4 pages:

- create.php
- list.php
- edit.php
- delete.php

Day	Topic	What You Do
Day 51	Create page	HTML form → INSERT into students table.
Day 52	Listing page	SELECT rows + display table.
Day 53	Edit page	Load a row → UPDATE logic.
Day 54	Delete page	Add delete link → DELETE logic.
Day 55	Review Day	Test entire flow end-to-end.

📅 WEEK 4 — Mini Project: “Client Treatment Log System”

Goal: Build a useful real-world database app.

Tables suggested:

clients: id, name, phone
treatments: id, client_id, notes, created_at

Day	Task
Day 56	Create database tables (PDO or raw SQL)
Day 57	Build “Add Treatment” page
Day 58	Build treatments list for each client
Day 59	Build Edit/Delete
Day 60	Final Review + Reflection

🧪 Included Exercises (Located in Your Repo)

Folder:
`month03-php-mysql/exercises`

These include:

1. **Connecting to MySQL with PDO**
2. **Prepared statements practice**
3. **CRUD mini tasks**

4. **Simple validation**

Run exercises inside container:

php /var/www/html/../exercises/FILENAME.php

🎯 What You Should Know by the End of Month 03

By the end of this month, you will confidently understand:

✔ How relational databases store information
✔ How to use SELECT, INSERT, UPDATE, DELETE
✔ How PHP connects to MySQL using PDO
✔ How to prevent SQL injection with prepared statements
✔ How to build a real CRUD application
✔ How to organize database-powered PHP scripts

This prepares you perfectly for **Month 04 (Laravel Basics)**, where Laravel automates much of what they did manually here.

Cheatsheet3

📄 MONTH 03 — PHP + MySQL CHEAT SHEET (PRINTABLE)

A quick reference for beginners learning databases + PDO

🧰 Docker Commands

Start the environment:

```
cd month03-php-mysql/docker
docker compose up -d
docker compose exec php bash
```

Access MySQL:

```
mysql -h mysql -u root -p
```

Password: **password**

📦 Basic MySQL Commands

Select (Read)

```
SELECT * FROM students;
SELECT name FROM students;
```

Insert (Create)

```
INSERT INTO students (name) VALUES ('Keen');
```

Update

```
UPDATE students SET name='New Name' WHERE id=1;
```

Delete

DELETE FROM students WHERE id=1;

🔌 Connecting with PDO

Create connection

$pdo = new PDO("mysql:host=mysql;dbname=bootcamp", "root", "password");

Basic safety tip:

Always use **prepared statements** to prevent SQL injection.

🧪 PDO Prepared Statements

SELECT

$stmt = $pdo->prepare("SELECT * FROM students");
$stmt->execute();
$rows = $stmt->fetchAll();

INSERT

$stmt = $pdo->prepare("INSERT INTO students (name) VALUES (:name)");
$stmt->execute(['name' => $name]);

UPDATE

$stmt = $pdo->prepare("UPDATE students SET name=:name WHERE id=:id");
$stmt->execute(['name' => $name, 'id' => $id]);

DELETE

$stmt = $pdo->prepare("DELETE FROM students WHERE id=:id");
$stmt->execute(['id' => $id]);

🔧 CRUD Page Structure (Simple PHP App)

create.php → form → insert
list.php → table of rows
edit.php → load 1 row → update
delete.php → remove row

This is the foundation of all data-driven apps.

📁 Example Table Structure

`students` table:

id INT AUTO_INCREMENT PRIMARY KEY
name VARCHAR(100)
created_at TIMESTAMP

🌱 Mini Project: Treatment Log System

Two tables:

clients

- id
- name
- phone

treatments

- id
- client_id
- notes
- created_at

You will build:

✔ Add a treatment
✔ View list
✔ Edit
✔ Delete

🎯 At the End of Month 03 You Should Know

✔ SQL basics
✔ How PHP connects to MySQL
✔ PDO prepared statements
✔ How to run CRUD apps
✔ How data flows from form → PHP → DB → page
✔ Foundation for Laravel (Month 04)

Month 4

📘 MONTH 04 — LARAVEL BASICS (Beginner-Friendly Curriculum)

Duration: 20 days (4 weeks)
Time per day: 30–60 minutes
Goal: Learn how Laravel works: routing, controllers, Blade views, models, migrations, and your first Laravel application.

🌿 Overview

Month 04 is the moment the learner transitions from raw PHP to a professional, modern framework used in real software companies.

This month focuses on:

- Understanding the MVC pattern
- Creating pages using routes + controllers + views
- Using Blade templates
- Running migrations & working with databases
- Building their first Laravel application
- Keeping everything simple and digestible

All lessons are written for a learner with **zero technical background**, building confidence step-by-step.

🧰 Required Setup

Folder:
`month04-laravel-basics/`

1. Navigate to Laravel app

Example (blog-app):

cd month04-laravel-basics/blog-app

2. Install dependencies

composer install

3. Copy .env

cp .env.example .env
php artisan key:generate

4. Start Docker (if applicable)

If using Laravel Sail:

./vendor/bin/sail up -d

5. Visit local site

Default:

http://localhost

📅 WEEK 1 — Understanding Laravel & Routing

Goal: Learn how Laravel handles webpages.

Day	Topic	What You Do

Day 61	What is a framework? What is Laravel?	Understand MVC. Explore folder structure.
Day 62	Routes basics	Create simple routes in `routes/web.php`.
Day 63	Controllers	Create controllers using `php artisan make:controller`.
Day 64	Passing data to views	Send variables from controller → Blade.
Day 65	Review	Build small “Hello You” page using routes + controller + view.

📅 WEEK 2 — Blade Views & Layouts

Goal: Build dynamic, reusable pages.

Day	Topic	What You Do
Day 66	Blade syntax	Use `{{ }}` and simple directives.
Day 67	Layouts	Create a main layout file using `@yield` and `@section`.
Day 68	Components	Make reusable parts like navigation/header.
Day 69	Forms in Blade	Create a simple form and send to controller.
Day 70	Review	Build a page with layout + component + form.

📅 WEEK 3 — Models, Migrations, Database

Goal: Learn how Laravel stores data.

Day	Topic	What You Do

Day 71	Models	Create a model (e.g., Post).
Day 72	Migrations	Create database tables using Laravel migrations.
Day 73	Eloquent basics	Insert, update, delete using Laravel ORM.
Day 74	Eloquent queries	Learn `where()`, `find()`, `get()`.
Day 75	Review	Build CRUD using Eloquent directly (no views yet).

📅 WEEK 4 — Mini Project: “Journal App”

Goal: Build your first full Laravel application.

Your project will include:

- A JournalEntry model
- A migration for journal entries
- Routes for listing and creating entries
- A form for adding entries
- A page for reading all past entries

Day	Task
Day 76	Design the database + create migration
Day 77	Build form to create journal entries
Day 78	Insert entries using controller + model
Day 79	Build list view to display entries
Day 80	Final review + polish UI

🧪 Included Exercises in Repo

Inside:
`month04-laravel-basics/notes`
`month04-laravel-basics/blog-app`
`month04-laravel-basics/journal-app`

Exercises include:

- Creating routes
- Blade practice
- Simple controllers
- First CRUD example

🎯 What You Should Know by the End of Month 04

By the end of this month, you will understand:

✔ What Laravel does and why it's powerful
✔ How MVC works in Laravel
✔ How to create routes, controllers, and Blade views
✔ How to build and run migrations
✔ How to use Eloquent ORM
✔ How to build a simple working Laravel project
✔ How to organize Laravel code properly

This prepares you perfectly for **Month 05 — Laravel Advanced**, where they learn APIs, relationships, validation, authentication, and more.

Cheatsheet4

📄 MONTH 04 — LARAVEL BASICS CHEAT SHEET (PRINTABLE)

Beginner-friendly quick reference for MVC, routes, controllers, Blade, models, and migrations

---

🚀 Artisan Commands (Core Essentials)

Create controller

php artisan make:controller NameController

Create model

php artisan make:model Post

Create model + migration

php artisan make:model Post -m

Run migrations

php artisan migrate

Serve the app

php artisan serve

🌐 Routing Basics (web.php)

Simple Route

```
Route::get('/', function () {
    return 'Hello World';
});
```

Route → Controller

Route::get('/posts', [PostController::class, 'index']);

Route with parameter

Route::get('/post/{id}', [PostController::class, 'show']);

📦 Controller Basics

```
class PostController extends Controller {
    public function index() {
        return view('posts.index');
    }
}
```

🎨 Blade Templates

Blade Syntax

{{ $variable }}

If Statement

```
@if($count > 0)
  There are posts
@endif
```

Loop

```
@foreach($posts as $post)
  {{ $post->title }}
@endforeach
```

📁 Blade Layout Structure

layout.blade.php

```
<!DOCTYPE html>
<html>
  <body>
    @yield('content')
  </body>
</html>
```

child view

```
@extends('layout')
@section('content')
  <h1>Hello</h1>
@endsection
```

🛠️ Models & Eloquent (ORM)

Basic Model

```
class Post extends Model {
    protected $fillable = ['title', 'body'];
}
```

Insert

```
Post::create([
  'title' => 'Today',
  'body' => 'A good day'
]);
```

Get All

```
$posts = Post::all();
```

Find One

```
$post = Post::find($id);
```

🗄️ Migration Basics

Migration Structure

```
public function up() {
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->text('body');
        $table->timestamps();
    });
}
```

Run:

```
php artisan migrate
```

📝 Forms in Blade

```
<form method="POST" action="/posts">
  @csrf
  <input type="text" name="title" />
  <textarea name="body"></textarea>
  <button>Save</button>
</form>
```

🌱 Mini Project (End of Month 04)

Journal App

Requires:

✔ Model + migration (`JournalEntry`)
✔ Form to create journal entries
✔ Controller to store data
✔ Page to list entries

🎯 End-of-Month Skills

By the end of Month 04, you can:

✔ Build routes
✔ Create controllers and views
✔ Use Blade templates
✔ Make models & migrations
✔ Use Eloquent ORM
✔ Build your first real Laravel CRUD app

Month 5

📘 MONTH 05 — LARAVEL ADVANCED (Beginner-Friendly Curriculum)

Duration: 20 days (4 weeks)
Time per day: 30–60 minutes
Goal: Learn APIs, validation, Eloquent relationships, API resources, middleware, and build an advanced Laravel project.

🌿 Overview

Month 05 builds on the foundations from Month 04 and introduces powerful professional concepts:

- Building a real API
- Handling validation safely
- Protecting routes
- Modeling real-world relationships (1:many, many:many)
- Using API Resources to format JSON
- Authentication (simple version)
- Error handling
- Structuring Laravel projects

By the end of Month 05, the learner builds their first **real backend API**:
Habit Tracker API — similar to what junior devs build in the real world.

🧰 Required Setup

Folder:
`month05-laravel-advanced/habit-tracker-api`

1. Navigate to project

cd month05-laravel-advanced/habit-tracker-api

2. Install dependencies

composer install

3. Copy .env and set up database

cp .env.example .env
php artisan key:generate

Make sure DB in `.env` matches your Docker database.

4. Run migrations

php artisan migrate

5. Start the app

php artisan serve

📅 WEEK 1 — APIs & Validation

Goal: Learn how Laravel handles API routes, controllers, and validation.

Day	Topic	What You Do
Day 81	What is an API?	Learn routes/api.php vs routes/web.php.
Day 82	API routes	Create JSON-returning routes.
Day 83	API controllers	`php artisan make:controller HabitController --api`
Day 84	Request validation	Learn `$request->validate([])` for safe input.

Day 85	Review Day	Build a simple API endpoint with validation.

📅 WEEK 2 — Models, Migrations, and Relationships

Goal: Model real data using proper Eloquent relationships.

Day	Topic	What You Do
Day 86	One-to-Many	`User hasMany Habits`, `Habit belongsTo User`
Day 87	Many-to-Many	Example: tags for habits
Day 88	Migrations (advanced fields)	Add booleans, enums, timestamps.
Day 89	Eloquent queries	Filtering, ordering, pagination.
Day 90	Review Day	Build relationship between `Habit` and `HabitLog`.

📅 WEEK 3 — API Resources & Middleware

Goal: Learn how to format JSON and protect API routes.

Day	Topic	What You Do
Day 91	API Resources	Use `HabitResource` to format responses.
Day 92	Collections	Create `HabitResource::collection($habits)`
Day 93	Middleware	Add simple auth-like middleware.
Day 94	Authentication (basic)	Use Laravel's token-based guard.

Day 95	Review Day	Apply resource formatting to your Habit API.

📅 WEEK 4 — Mini Project: “Habit Tracker API”

Goal: Build a full Laravel API that tracks habits and logs.

Your API includes:

Tables

- **users**
- **habits**
- **habit_logs**

Features

✔ Create a habit
✔ List all habits
✔ Log a habit
✔ View logs per habit
✔ Update habit
✔ Delete habit
✔ Protected routes

Day	Task
Day 96	Build habit migration + model
Day 97	Add Habit CRUD endpoints
Day 98	Build HabitLog model + endpoints
Day 99	Add protected API routes + simple auth
Day 100	Final review + polish + JSON testing in Postman

🧪 Included Exercises in Repo

Inside:
`month05-laravel-advanced/notes`
`month05-laravel-advanced/habit-tracker-api`

Exercises include:

- API route practice
- Building a JSON controller
- Validation drills
- Using API Resources
- Simple authentication
- Relationship exercises

🎯 What You Should Know by the End of Month 05

By the end of Month 05, you will confidently understand:

✔ How APIs work
✔ How to create and protect API routes
✔ How to validate user input
✔ How API controllers differ from web controllers
✔ How to structure real backend Laravel code
✔ How relationships work (1:M, M:M)
✔ How to build JSON APIs using resources
✔ How to create a real habit-tracking API

This prepares you perfectly for **Month 06 — Vue Basics**, where they will build the frontend.

Cheatsheet5

📄 MONTH 05 — LARAVEL ADVANCED CHEAT SHEET (PRINTABLE)

Quick reference for APIs, validation, relationships, resources, and middleware

---

🌐 API Essentials

API Routes (routes/api.php)

```
Route::get('/habits', [HabitController::class, 'index']);
Route::post('/habits', [HabitController::class, 'store']);
```

API routes automatically include `/api` prefix.

🧰 API Controller Essentials

Create API controller:

```
php artisan make:controller HabitController --api
```

Basic structure:

```
class HabitController extends Controller
{
    public function index() {
        return Habit::all();
    }

    public function store(Request $request) {
        $validated = $request->validate([
            'title' => 'required|min:3'
        ]);
        return Habit::create($validated);
    }
```

```
}
```

🛡️ Validation

```
$request->validate([
    'title' => 'required|min:3',
    'frequency' => 'required|in:daily,weekly'
]);
```

Common rules: `required`, `email`, `min`, `max`, `in`, `numeric`, `boolean`.

📦 Models + Relationships

One-to-Many

User → Habits

```
class User extends Model {
    public function habits() {
        return $this->hasMany(Habit::class);
    }
}

class Habit extends Model {
    public function user() {
        return $this->belongsTo(User::class);
    }
}
```

Many-to-Many

```
class Habit extends Model {
    public function tags() {
        return $this->belongsToMany(Tag::class);
    }
}
```

🗄️ Migrations (Advanced Fields)

```
$table->string('title');
$table->enum('frequency', ['daily', 'weekly']);
$table->boolean('active')->default(true);
$table->timestamps();
```

Run:

```
php artisan migrate
```

🔄 Eloquent Queries

Get All

```
$habits = Habit::all();
```

Find One

```
$habit = Habit::find($id);
```

Filter

```
Habit::where('active', true)->get();
```

🎨 API Resources (Formatting JSON)

Create resource:

```
php artisan make:resource HabitResource
```

Usage:

```
return new HabitResource($habit);
```

Or for lists:

return HabitResource::collection(Habit::all());

Inside resource:

```
return [
    'id' => $this->id,
    'title' => $this->title,
    'frequency' => $this->frequency,
];
```

🚧 Middleware (Protect API Routes)

Assign middleware to routes:

```
Route::middleware('auth:sanctum')->group(function () {
    Route::post('/habits', [HabitController::class, 'store']);
});
```

🔑 Basic API Authentication (Token-Based)

Issue token (example):

```
$token = $user->createToken('api')->plainTextToken;
```

Send in requests:

```
Authorization: Bearer YOUR_TOKEN
```

📘 Habit Tracker API (Mini Project)

Includes:

✔ Habit model + migration
✔ HabitLog model + migration
✔ Full CRUD endpoints
✔ Relationship: Habit → HabitLog

✔ Protected routes
✔ JSON responses via resources

🎯 End-of-Month Skills

By the end of Month 05, you should understand:

✔ API routes vs web routes
✔ API controllers & JSON responses
✔ Input validation
✔ One-to-many & many-to-many relationships
✔ Migrations with advanced field types
✔ API Resources
✔ Middleware & simple authentication
✔ How to build a real backend API

Month 6

📘 MONTH 06 — VUE.JS BASICS (Beginner-Friendly Curriculum)

Duration: 20 days (4 weeks)
Time per day: 30–60 minutes
Goal: Understand components, props, events, reactive data, watchers, computed properties, and build your first Vue SPA.

🌿 Overview

Month 06 introduces the learner to **frontend development** using Vue.js — one of the most intuitive JavaScript frameworks for beginners.

This month focuses on:

- Understanding what reactive UI means
- Building simple components
- Passing data between components
- Managing local state
- Using the Vue Router to switch pages
- Building a small, fully working app
- Staying calm and avoiding JavaScript overwhelm

Everything is explained in **plain English**, designed for a learner with zero technical background.

🧰 Required Setup

Folder:
`month06-vue-basics/vue-sandbox`

1. Navigate to the project

cd month06-vue-basics/vue-sandbox

2. Install dependencies

npm install

3. Start the development server

npm run dev

4. Open app in browser

http://localhost:5173

📅 WEEK 1 — Getting Comfortable With Vue

Goal: Understand the structure and mindset of a Vue app.

Day	Topic	What You Do
Day 101	What is Vue? What is a component?	Explore project structure (main.js, App.vue, components folder).
Day 102	Template syntax	Display variables with `{{ }}`.
Day 103	Data() function	Create reactive variables and show them on screen.
Day 104	Methods	Add buttons and event handlers.
Day 105	Review Day	Build a small counter component.

📅 WEEK 2 — Props, Events, Computed, Watch

Goal: Learn how components communicate and react to changes.

Day	Topic	What You Do
Day 106	Props (inputs into components)	Pass data from parent → child.
Day 107	Emit events	Send events from child → parent.
Day 108	Computed properties	Create reactive calculated values.
Day 109	Watchers	Watch for changes in variables.
Day 110	Review Day	Build a “Daily Mood Component” with props + computed.

📅 WEEK 3 — Vue Router Basics

Goal: Build a multi-page app.

Day	Topic	What You Do
Day 111	Install Vue Router	Create `/home` and `/about` pages.
Day 112	Router links	Switch pages using `<router-link>`.
Day 113	Route params	Create dynamic pages like `/journal/5`.
Day 114	Nested routes	Add sub-pages.
Day 115	Review Day	Build a 3-page micro-SPA.

📅 WEEK 4 — Mini Project: “Meditation Timer SPA”

Goal: Build your first real Vue application.

Your app includes:

Features:

- Countdown timer
- List of meditation presets
- Page for daily mood
- Page for session history
- Simple UI styling

Day	Task
Day 116	Set up basic pages using Vue Router
Day 117	Create meditation presets component
Day 118	Build countdown timer (JS interval)
Day 119	Add mood tracker + simple storage
Day 120	Final review + UI polish

🧪 Included Exercises in Repo

Inside:
`month06-vue-basics/notes`
`month06-vue-basics/exercises`
`month06-vue-basics/vue-sandbox`

Exercises include:

- Basic component creation
- Props & emits practice
- Computed & watchers
- Vue Router practice
- Mini UI components

🎯 What You Should Know by the End of Month 06

By the end of Month 06, you will understand:

✔ How Vue components work
✔ How to pass data using props
✔ How to send data upward using events
✔ How to manage reactive UI
✔ How to use computed and watch
✔ How to build multi-page Vue apps
✔ How to structure a frontend project
✔ How to build a simple SPA

This prepares you for **Month 07 — Vue Advanced**, where they will learn state management, reusable patterns, and API consumption.

Cheatsheet6

📄 MONTH 06 — VUE.JS BASICS CHEAT SHEET (PRINTABLE)

Quick reference for components, props, events, state, computed, watch, and routing

---

🚀 Vue Project Basics

Start Dev Server

npm run dev

Project Structure

src/
  main.js
  App.vue
  components/
  router/

🔧 Component Basics

Create component

`src/components/Counter.vue`

```
<template>
  <button @click="count++">Count: {{ count }}</button>
</template>

<script>
export default {
  data() {
    return { count: 0 }
  }
```

```
}
</script>
```

Use component

```
<Counter />
```

Import inside parent:

```
import Counter from './components/Counter.vue'
```

📥 Props (Parent → Child)

Child

```
<script>
export default {
  props: ['title']
}
</script>
```

Parent

```
<MyCard title="Hello" />
```

📤 Emit Events (Child → Parent)

Child

```
<button @click="$emit('selected', item)">Choose</button>
```

Parent

```
<MyItem @selected="handleSelect" />
```

🧠 Computed Properties

Used for **calculated values** that update automatically.

```
computed: {
  fullName() {
    return this.first + ' ' + this.last
  }
}
```

👀 Watchers

Used to “watch” variables for changes.

```
watch: {
  mood(newVal) {
    console.log("Mood changed:", newVal)
  }
}
```

🎛️ Methods

Actions triggered by events.

```
methods: {
  add() {
    this.total++
  }
}
```

🔄 Two-Way Binding

```
<input v-model="message" />
```

🎨 Template Syntax

- Print variable: `{{ message }}`
- Conditional:

Welcome!

- Loop:

<li v-for="item in items" :key="item.id">{{ item.name }}

🌐 Vue Router Basics

Install Router

npm install vue-router

Define routes

```
import { createRouter, createWebHistory } from 'vue-router'
import Home from './pages/Home.vue'

export default createRouter({
  history: createWebHistory(),
  routes: [
    { path: '/', component: Home },
  ]
})
```

Use in app

createApp(App).use(router).mount('#app')

Links

<router-link to="/about">About</router-link>

🧘 Mini Project Structure (Meditation Timer SPA)

Pages:

- `/` — Home
- `/timer` — Countdown timer
- `/mood` — Daily mood tracker
- `/history` — Simple log screen

Components:

- TimerComponent
- MoodSelector
- PresetList

🎯 End-of-Month Skills

By the end of Month 06 you will understand:

✔ Components
✔ Props & emits
✔ Computed & watch
✔ State & v-model
✔ Router basics
✔ How to build a simple Vue SPA

Month 7

📘 MONTH 07 — VUE.JS ADVANCED (Beginner-Friendly Curriculum)

Duration: 20 days (4 weeks)
Time per day: 30–60 minutes
Goal: Learn state management, API integration, advanced components, reusable patterns, and build a full-featured advanced Vue application.

🌿 Overview

Month 07 takes the learner from basic Vue components into **professional front-end development** skills.

This month teaches:

- State management with **Pinia**
- Organizing components into patterns
- Advanced component communication
- Making API calls (GET/POST/PUT/DELETE)
- Async/await + error handling
- Reusable UI patterns
- Managing loading states, forms, errors
- More advanced Vue Router features

By the end of this month, the learner will build a real-world frontend that connects to a Laravel backend API (from Month 05).

🧰 Required Setup

Folder:
`month07-vue-advanced/`

Your project folders for the month:

- `component-patterns/`
- `vue-state-management/`
- `vue-router-demo/`
- `api-consumption-demo/`

To install and run each:

npm install
npm run dev

📅 WEEK 1 — Component Patterns & Reusability

Goal: Build more flexible, reusable components.

Day	Topic	What You Do
Day 121	Slot basics	Build a `<Card>` component with slots.
Day 122	Named slots	Add header/footer slots.
Day 123	Dynamic components	Use `<component :is="">` for swappable UI.
Day 124	Provide/Inject	Share data without props/emits.
Day 125	Review Day	Build a reusable UI component set (Card, Modal).

📅 WEEK 2 — Pinia (State Management)

Goal: Learn global state to manage cross-component data.

Day	Topic	What You Do
Day 126	Install Pinia	Add store to project.
Day 127	State + getters	Create global state for habits or tasks.
Day 128	Actions	Add functions to update global state.
Day 129	Using stores in components	Replace props drilling.
Day 130	Review Day	Build “Global Mood Tracker” with Pinia store.

📅 WEEK 3 — API Calls & Async Programming

Goal: Connect Vue to your Laravel API (from Month 05).

Day	Topic	What You Do
Day 131	Axios installation	Make GET request to fetch data.
Day 132	POST request	Create new items through an API endpoint.
Day 133	PUT + DELETE	Update + delete API records.
Day 134	Error handling + loading states	Show loading spinners, error UI.
Day 135	Review Day	Build “Habit List” UI powered by real API data.

📅 WEEK 4 — Mini Project: “Wellness Dashboard Frontend”

Goal: Build a polished Vue frontend connected to the Laravel API.

Pages:

- Daily Mood
- Habit List
- Habit Detail
- Meditation Timer
- Progress Summary

Features:

- Fetch habits from API
- Add new habit
- Log habit activity
- Display charts (optional)
- Use Pinia for global state
- Routing between pages

Day	Task
Day 136	Setup pages + router
Day 137	Integrate Pinia stores for habits
Day 138	Connect to Laravel API (GET/POST)
Day 139	Build UI: lists, forms, detail pages
Day 140	Final review + UX polish

🧪 Included Exercises in Your Repo

Inside:

`month07-vue-advanced/component-patterns`
`month07-vue-advanced/vue-state-management`
`month07-vue-advanced/vue-router-demo`
`month07-vue-advanced/api-consumption-demo`

Exercises cover:

- Component composition patterns
- Global state + getters + actions
- API calls with Axios
- Route params, nested routes
- Handling asynchronous data

🎯 What You Should Know by the End of Month 07

By the end of Month 07, you will confidently understand:

✔ How to write reusable components (slots, dynamic components)
✔ How to manage global state with Pinia
✔ How to communicate across components efficiently
✔ How to fetch, create, update, delete API data
✔ How to handle loading, errors, empty states
✔ How to structure a scalable Vue project
✔ How to build an advanced SPA frontend

This prepares you perfectly for **Month 08 — Fullstack Projects**, where Laravel + Vue join together.

Cheatsheet7

📄 MONTH 07 — VUE.JS ADVANCED CHEAT SHEET

State Management • Component Patterns • API Calls • Advanced Router

---

🔷 1. Component Patterns (Reusable Components)

Slots (Replaceable Content Inside a Component)

```
<slot></slot>              <!-- default slot -->
<slot name="header"></slot> <!-- named slot -->
```

Use Slots

```
<Card>
  <template #header>Title here</template>
  Main content
</Card>
```

Dynamic Components

Swap components dynamically:

```
<component :is="currentComponent" />
```

Provide / Inject

Share data without props:

```
provide('theme', 'dark')
inject('theme')
```

🔷 2. Pinia — State Management

Setup

```
import { createPinia } from 'pinia'
app.use(createPinia())
```

Create Store

```
export const useHabitStore = defineStore('habit', {
  state: () => ({ list: [] }),
  actions: {
    add(habit) {
      this.list.push(habit)
    }
  }
})
```

Use in Component

```
const store = useHabitStore()
store.list
store.add('Exercise')
```

🔷 3. API Calls (Axios)

Install Axios

```
npm install axios
```

GET Request

```
const res = await axios.get('/api/habits')
```

POST

```
await axios.post('/api/habits', { name: 'Meditate' })
```

PUT

```
await axios.put('/api/habits/1', { name: 'Run' })
```

DELETE

```
await axios.delete('/api/habits/1')
```

Loading & Errors

```
loading = true
try {
  await apiCall()
} catch (e) {
  error = e.message
} finally {
  loading = false
}
```

🔷 4. Advanced Vue Router

Use Params

```
{ path: "/habit/:id", component: HabitDetail }
```

Access Route Params

```
const route = useRoute()
route.params.id
```

Programmatic Navigation

```
const router = useRouter()
router.push('/dashboard')
```

Nested Routes

```
{
  path: "/settings",
```

```
  children: [
    { path: "profile", component: Profile },
    { path: "account", component: Account }
  ]
}
```

🔷 5. Best Practices Cheat Sheet

✔ Use **Pinia** instead of prop-drilling
✔ Use **async/await** for all API calls
✔ Show **loading** and **error messages**
✔ Keep components small and reusable
✔ Group files by feature (not by type)
✔ Always use **router params** for detail views
✔ Use **computed** + **getters** for derived data
✔ Use **actions** instead of mutating state manually

🔷 6. Month 07 Mini-Project Checklist

Build a "Wellness Dashboard Frontend" featuring:

- ☐ Home page
- ☐ Habit list (GET)
- ☐ Add new habit (POST)
- ☐ Edit habit (PUT)
- ☐ Delete habit (DELETE)
- ☐ Mood tracker (Pinia store)
- ☐ Navigation via Vue Router
- ☐ Reusable components (Card, Modal)

- ☐ Loading states
- ☐ Error handling

Month 8

Here is your **Month 08 — Fullstack Projects (Google Docs–ready version)**.
You can **copy & paste directly into Google Docs** and it will paste cleanly with headings, tables, and structure — matching the format of your earlier months.

📘 MONTH 08 — FULLSTACK PROJECTS (Laravel + Vue + Docker)

Duration: 20 days (4 weeks)
Time per day: 30–60 minutes
Goal: Build full production-style fullstack apps combining PHP Laravel backend + Vue frontend + Docker environment.

This is the "capstone month"—the adult learner will now stitch everything together:
Backend, Frontend, APIs, Authentication, State Management, Docker workflows, and deployment preparation.

🌿 Overview

Month 08 focuses on **real-world developer skills**:

- Building fullstack apps from scratch
- Connecting Vue frontend ↔ Laravel backend
- Using Docker for local development
- API authentication (Laravel Sanctum)
- CRUD across fullstack
- File uploads, validation, and UX
- Creating reusable frontend architecture
- Deployment prep mindset

By the end of Month 08, the learner will produce **2 portfolio-grade apps**.

🧰 Projects for Month 08

Inside:

```
month08-fullstack-projects/
  laravel-vue-todo/
    backend/
    frontend/
    docker/
  wellness-journal-full/
    backend/
    frontend/
    docker/
```

Each app uses:

- **Laravel API backend**
- **Vue.js 3 SPA**
- **Pinia for state**
- **Axios for API calls**
- **Docker Compose** to run PHP, MySQL, and Node containers

Commands:

```
docker-compose up -d
npm install
npm run dev
composer install
php artisan serve
```

📅 WEEK 1 — Fullstack Foundation + API Authentication

Goal: Learn how to connect Laravel + Vue securely.

Day	Topic	What You Do
Day 141	Set up Laravel API	New Laravel backend + routes setup
Day 142	Sanctum Authentication	Register + Login endpoints
Day 143	Vue Auth UI	Login form → GET token from backend
Day 144	Authenticated requests	Send Authorization headers
Day 145	Review Day	Build a working login/logout flow

📅 WEEK 2 — Fullstack CRUD (Laravel + Vue)

Goal: Perform CRUD from frontend to backend with validation.

Day	Topic	What You Do
Day 146	API Resources + Controllers	Build Todo or Habit endpoints
Day 147	Vue CRUD list	Display items from backend
Day 148	Add item	Vue form → POST to Laravel
Day 149	Edit/Delete items	PUT + DELETE w/ UI
Day 150	Review Day	Full CRUD UI + API complete

📅 WEEK 3 — The Wellness Journal App (Capstone Project)

Goal: Develop a 5-page fullstack application.

Day	Page	What You Build
Day 151	Dashboard	Summary, daily logs, counts
Day 152	Mood Tracker	POST/GET mood logs
Day 153	Habits Page	Connect to Pinia store + API
Day 154	Journal Page	CRUD journal entries
Day 155	Review Day	UX polish + validation

📅 WEEK 4 — Docker, Deployment Mindset & Final Polish

Goal: Learn how real developers run containers and prep apps for deployment.

Day	Topic	What You Do
Day 156	Docker Compose	Run PHP, MySQL, Node containers
Day 157	Environment Variables	Configure `.env` for Docker
Day 158	Production Builds	Create optimized Vue + Laravel builds
Day 159	Debugging & Logs	How to debug containers
Day 160	Final Review	Package project for portfolio

🧩 Skills You Will Master This Month

✔ Connecting Vue frontend to Laravel backend
✔ Working with APIs end-to-end
✔ Using Sanctum for authentication
✔ Using Axios for fetch/update/delete
✔ State management with Pinia
✔ Building reusable UI patterns

✔ Docker environment setup
✔ Debugging fullstack applications
✔ Preparing projects for real-world deployment

This month transforms the learner from **beginner to full-stack developer**.

🌟 Capstone Projects Checklist

Project 1: Laravel + Vue Todo App

- ☐ Login/logout + authentication
- ☐ See list of todos
- ☐ Add new todo
- ☐ Edit todo
- ☐ Delete todo
- ☐ Docker environment working
- ☐ Clean UI with reusable components

Project 2: Wellness Journal (Full Featured)

- ☐ Daily mood logging
- ☐ Habit tracking
- ☐ Journal CRUD
- ☐ Dashboard summary
- ☐ Global state with Pinia
- ☐ API-driven UI
- ☐ Docker environment

- ☐ Final polish / responsive layout

🎉 Congratulations — You're Fullstack!

By the end of Month 08, you will have:

- 2 real-world apps
- A complete fullstack workflow
- A strong GitHub portfolio
- Experience connecting frontend + backend
- Enough skill to qualify for **junior developer interviews**

Cheatsheet8

📄 MONTH 08 — FULLSTACK PROJECTS CHEAT SHEET

Laravel API • Vue Frontend • Pinia • Axios • Docker Compose

🔷 1. Fullstack Directory Structure

```
month08-fullstack-projects/
│
├── laravel-vue-todo/
│   ├── backend/      ← Laravel API
│   ├── frontend/     ← Vue SPA
│   └── docker/
│       └── docker-compose.yml
│
└── wellness-journal-full/
    ├── backend/
    ├── frontend/
    └── docker/
```

🔷 2. Laravel API Essentials

Create a New API Endpoint

```
Route::get('/habits', [HabitController::class, 'index']);
```

Controller Structure

```
public function index() {
    return HabitResource::collection(Habit::all());
}
```

Sanctum Auth

```
composer require laravel/sanctum
php artisan vendor:publish --tag=sanctum-config
php artisan migrate
```

Protect Routes

```
Route::middleware('auth:sanctum')->get('/user', function() {});
```

🔷 3. Vue Frontend Essentials

GET Data

```
const res = await axios.get('/api/habits')
```

POST Data

```
await axios.post('/api/habits', { name: habit })
```

Use Router Params

```
const route = useRoute()
```

Navigate Programmatically

```
router.push('/dashboard')
```

🔷 4. Pinia State Management

```
export const useHabitStore = defineStore('habits', {
  state: () => ({ items: [] }),
  actions: {
    add(habit) { this.items.push(habit) }
  }
})
```

🔷 5. Docker Commands

Start Containers

docker-compose up -d

Stop Containers

docker-compose down

Rebuild

docker-compose up --build

🔷 6. API Error + Loading Handling

```
loading = true
try {
  await action()
} catch (e) {
  error = e.message
} finally {
  loading = false
}
```

🔷 7. Capstone Project Must-Haves

Laravel + Vue Todo App

- Login/logout
- Todo CRUD
- Pinia global state

- Axios API calls
- Docker environment

Wellness Journal App

- Mood tracker
- Habit list + detail
- Journal CRUD
- Dashboard
- Auth + protected routes
- Clean UI components

🔷 8. Deployment Mindset

✓ Use `.env` variables
✓ Build Vue with `npm run build`
✓ Use Laravel `php artisan optimize`
✓ Test API + frontend separately
✓ Check Docker logs for debugging

🎯 Final Outcome

After Month 08, you can:

✔ Build a fullstack app from scratch
✔ Use Docker confidently
✔ Connect frontend ↔ backend APIs
✔ Implement authentication

✔ Handle async data and state
✔ Build portfolio-ready projects

Prerequisites

Visual Studio Code

What is VSCode?

📘 INTRODUCTION: What Is Visual Studio Code (VS Code) and Why Do We Use It?

A simple explanation for complete beginners

Visual Studio Code — usually called **VS Code** — is a **free code editor** where you write, edit, organize, and run your programming projects. It's the most widely used editor by developers all over the world.

Think of VS Code like:

- **Microsoft Word — but designed for coding**
- **A workspace** where all your project files live
- **A smart assistant** that helps you write good code
- **A toolbox** with everything you need to build software

VS Code makes learning to code easier and more enjoyable for beginners.

🌟 Why Do Developers Use VS Code? (Beginner-Friendly)

✔ 1. It organizes all your files in one place

You open your folder (your GitHub repo), and VS Code shows all files and months neatly on the left.

✔ 2. It highlights mistakes automatically

If you type something incorrectly, VS Code shows red underlines—just like a spell checker.

✔ 3. It gives smart suggestions while you type

When writing code, VS Code pops up suggestions to help you complete commands.

✔ 4. It works perfectly with PHP, Laravel, Vue.js, JavaScript, Docker, and Git

All the tools used in your 8-month bootcamp.

✔ 5. It has a built-in terminal

You never need to open a separate terminal window.
You can run:

```
php artisan serve
npm run dev
docker-compose up -d
```

directly inside VS Code.

✔ 6. It's used by real professional developers

If someone becomes a junior developer, they will almost certainly use VS Code at work.

🔷 What VS Code Actually Does (Simple Explanation)

VS Code helps you:

- Create files
- Edit code
- Preview results
- Run commands
- Install helpful extensions
- Manage entire projects in one window

- Connect to GitHub
- Debug errors

Everything your student builds—from Month 01 to Month 08—will be written inside VS Code.

🔷 What You See When VS Code Opens

VS Code shows:

1. File Explorer (left side)

Your bootcamp folders:

month01-php-basics
month02-php-oop
month03-php-mysql
...
month08-fullstack-projects

2. Main Editor (middle)

Where you type your code.

3. Terminal (bottom)

Where you run commands (PHP, Laravel, Docker, Vue).

🔷 Why VS Code Is Required for This Bootcamp

Your student uses VS Code because it:

✔ Makes coding easier
✔ Helps spot mistakes early

✔ Works with all the technologies in the program
✔ Simplifies file management
✔ Provides real-world developer experience
✔ Prevents confusion when working with multiple languages
✔ Is safe, stable, and beginner-friendly

VS Code is the **home base** for your entire learning journey.

🔷 Simple Beginner Explanation (Easy to Add in Google Docs)

Here's the simplest one-sentence version you can paste as a summary:

> **"Visual Studio Code is the main tool where you write and organize your code — like a smart notebook designed for programmers."**

Installing VSCode

📘 TECH NOTES — How to Install Visual Studio Code (VS Code)

Required for all coding lessons (Months 01–08)

Visual Studio Code — commonly called **VS Code** — is the main tool you will use to write, edit, and organize code. It is the world's most popular code editor because it is simple, powerful, and works on Windows, macOS, and Linux.

🌟 What Is Visual Studio Code (Explained Simply)

Think of VS Code like:

✔ **Microsoft Word — but for writing code instead of essays**
✔ **A workspace for coding**, where you open folders and edit files
✔ A tool that helps you spot mistakes with red underlines
✔ A tool that can run your code right inside it
✔ A tool that provides "smart suggestions" while typing
✔ A tool used by **real professional developers every day**

VS Code is the #1 recommended editor for **PHP, Laravel, Vue.js, JavaScript, HTML, SQL, and Docker** — all technologies in your bootcamp.

🔷 Step-by-Step: Install VS Code on macOS

1. Go to the download page

👉 https://code.visualstudio.com/

2. Click “Download for macOS”

3. When the download finishes, open the ZIP file

4. Drag the “Visual Studio Code” app into your Applications folder

5. Open VS Code from Applications

If a pop-up says “Are you sure you want to open this?” → click **Open**.

6. Add to Dock (optional)

Right-click the VS Code icon → **Options** → **Keep in Dock**

🔷 Step-by-Step: Install VS Code on Windows

1. Go to the download page

👉 https://code.visualstudio.com/

2. Click “Download for Windows”

3. Run the installer

Click **Next** through the prompts.

4. Select “Add to PATH” (recommended)

5. Finish installation and launch VS Code

🔷 Step-by-Step: Install VS Code on Linux (Ubuntu example)

sudo snap install code --classic

Or download the `.deb` file from the VS Code website.

🔷 After Installing: Recommended Extensions

Open VS Code → left sidebar → Extensions (square icon) → search and install:

For PHP & Laravel

- **PHP Intelephense**
- **Laravel Artisan**
- **Laravel Blade Snippets**
- **Laravel Extra Intellisense**

For Vue.js

- **Vue Language Features (Volar)**
- **ESLint**
- **Prettier**

For Docker

- **Docker Extension by Microsoft**

For Git

- **GitLens — Supercharge Git**

These help students write cleaner code with fewer errors.

🔷 How to Open Your Bootcamp Repository in VS Code

After downloading or cloning your repo:

Step 1 — Open VS Code

Step 2 — Click “File → Open Folder”

Step 3 — Select the folder:

bootcamp-fullstack-php-laravel-vue-docker

Step 4 — Click “Open”

You should now see all your months:

- month01-php-basics
- month02-php-oop
- month03-php-mysql
- month04-laravel-basics
- month05-laravel-advanced
- month06-vue-basics
- month07-vue-advanced
- month08-fullstack-projects

🔷 Why VS Code Is Required for This Bootcamp

✔ Used by professionals everywhere
✔ Easy for beginners
✔ Perfect for PHP, Laravel, Vue, Docker, and JavaScript
✔ Lets you run code in a built-in terminal
✔ Helps you auto-format code
✔ Helps you spot mistakes as you type
✔ Works perfectly with GitHub

VS Code will be the student's **main coding environment** from Month 01 to Month 08.

Install Docker

What is Docker?

📘 INTRODUCTION: What Is Docker and Why Do Developers Use It?

Beginner-Friendly Explanation for Non-Technical Students

Docker is a tool that allows you to run software in a **clean, isolated environment**—like having a mini-computer inside your computer.

Think of Docker like:

- A **portable kitchen** where all ingredients and tools are already prepared
- A **sealed container** that always works the same way
- A way to avoid the “it works on my machine but not yours” problem

Docker ensures your code runs exactly the same on **every computer**, every time.

🌟 Why Does Docker Exist? (Simple Explanation)

Without Docker:

- Installing PHP, MySQL, Node, Composer, and Laravel can be confusing
- Software versions can conflict
- A mistake installing something can break your whole system
- Setup takes hours and is easy to get wrong

With Docker:

- Everything is prebuilt

- You avoid installation problems
- Your system stays clean
- Your project “just works” with one command

Docker removes frustration and lets you **focus on learning**, not debugging your computer.

🔷 What Does Docker Actually Do?

Docker creates **containers**, which are like isolated “boxes” that contain everything your project needs:

✔ PHP
✔ MySQL database
✔ Node.js
✔ Composer
✔ Laravel
✔ Vue.js
✔ Web server
✔ All required libraries

Each container is:

- Self-contained
- Predictable
- Safe
- Easy to start/stop

You run them with simple commands like:

```
docker-compose up -d
docker-compose down
```

This launches your entire development environment instantly.

🔷 Real-World Developer Benefits

1. Everything Works Everywhere

Your project behaves exactly the same on:

- Your computer
- Another student's computer
- A production server
- A future laptop you buy

2. No More Installation Nightmares

You don't need to install:

- PHP
- MySQL
- Node
- Composer
- Nginx/Apache

Docker provides them **for you**.

3. Clean System

Your computer stays clean because all software runs *inside* containers, not directly on your machine.

4. Easy to Restart or Fix

If something breaks, you can literally:

```
docker-compose down
docker-compose up --build
```

and everything rebuilds fresh.

5. Matches Real Company Environments

Most modern companies use Docker:

- Netflix
- Spotify
- Amazon
- Meta
- Stripe

So learning Docker early gives you career-ready skills.

🔷 Why Docker Is Required in This Bootcamp

Your 8-month curriculum uses Docker because it:

✔ Simplifies setup
✔ Helps beginners avoid installation issues
✔ Keeps projects consistent
✔ Allows Laravel + Vue to run correctly
✔ Provides MySQL without manual installation
✔ Mirrors real developer workflows

With Docker, the student can run an entire fullstack app with:

```
docker-compose up -d
```

This turns your computer into a professional coding environment.

🔷 What a Student Needs to Know (Beginner Summary)

Here’s the simplest way to explain Docker to your student:

> “Docker is like a toolbox that gives your project its own mini-computer so everything runs smoothly, the same way, every time.”

They don't need deep technical understanding—just the ability to run the commands.

Installing Docker

📘 TECH NOTES — How to Install Docker (Beginner-Friendly Instructions)

Required for Month 03, Month 04, Month 05, Month 06, Month 07, and Month 08

Docker lets you run PHP, MySQL, and Node.js without installing them manually.
Instead, Docker creates **containers** that give you a clean development environment.

Follow the steps below to install Docker Desktop.

🔷 Installing Docker on macOS (Recommended for Mac users)

Step 1 — Visit the official download page

👉 https://www.docker.com/products/docker-desktop/

Step 2 — Click “Download for Mac”

Choose one of these versions:

- **Apple Silicon (M1/M2/M3)**
- **Intel Chip (Older Macs)**

If unsure:
Click the Apple icon → **About This Mac** → check your chip type.

Step 3 — Open the downloaded file

You will see a window with a whale icon.

Step 4 — Drag the Docker icon into Applications folder

Step 5 — Open Docker from Applications

You may need to approve security permissions.

Step 6 — Wait for the whale icon to turn stable

This may take 1–3 minutes on first launch.

Step 7 — Verify installation

Open Terminal and type:

docker --version

You should see something like:

Docker version 27.x.x

🔷 Installing Docker on Windows

⚠ **Important:** Windows Home needs **WSL2** enabled.
Docker will help you install it if needed.

Step 1 — Download Docker Desktop

👉 https://www.docker.com/products/docker-desktop/

Step 2 — Run the installer

Click **Next** through the prompts.

Step 3 — Enable WSL2 when asked

This is required for containers.

Step 4 — Restart your computer

Step 5 — Verify installation

Open PowerShell:

docker --version

If you see a version number, it’s installed correctly.

🔷 Installing Docker on Linux (Ubuntu example)

Only if student uses Linux.

```
sudo apt update
sudo apt install docker.io -y
sudo systemctl enable docker
sudo systemctl start docker
```

Check version:

```
docker --version
```

🔷 After Docker Install — Important Settings

1. Make sure Docker is running

The **whale icon** should appear:

- Top menu bar (Mac)
- Taskbar (Windows)

If not running → launch Docker Desktop manually.

2. Enable these stable settings (recommended)

Open Docker Desktop →
Settings → **Resources** → **Advanced**:

- CPUs: **4**
- Memory: **4 GB**
- Swap: **1 GB**

These are safe for beginners.

🔷 Test Your Installation Using Docker

To confirm everything works, run:

docker run hello-world

You should see:

Hello from Docker!

This proves Docker is successfully installed.

🔷 Why You Need Docker for This Bootcamp

Docker eliminates setup headaches:

✔ No need to install PHP manually
✔ No need to install MySQL manually
✔ No need to install Node.js manually
✔ Your environment matches real developer setups
✔ All Laravel + Vue apps in this repo are pre-configured for Docker

With Docker, a beginner can run:

- Laravel
- MySQL
- phpMyAdmin
- Node.js/Vite
- Vue.js
- Redis (if needed later)

with one command:

docker-compose up -d

GitHub Repository

What is GitHub?

📘 INTRODUCTION: What Is a GitHub Repository? (Beginner-Friendly)

A **GitHub repository** — often called a **repo** — is simply a **folder on the internet where your code lives**.

Think of it like:

- A **Google Drive folder**, but for code
- A place to **store, organize, and back up** all your projects
- A place where you can **share your work** with others
- A place where you can **track every change** you ever make
- A place where future employers can see your **coding portfolio**

A GitHub repo allows you to work like a real software developer, even as a beginner.

🌟 Why Do Developers Use GitHub?

Here are the main reasons:

✔ Backup

Your code is safely stored online.
Even if your computer breaks, nothing is lost.

✔ Version History

GitHub remembers **every version** of your files.
You can always:

- Undo mistakes

- Restore old versions
- See what changed over time

✔ Sharing Your Work

You can easily send someone a link to your repo.
Perfect for:

- Mentors
- Hiring managers
- Classmates
- Clients

✔ Portfolio Building

A GitHub profile is like a **LinkedIn for developers**.
Your projects, activity, and skills are visible to employers.

✔ Works With Tools Like VS Code & Docker

GitHub connects directly with:

- Visual Studio Code
- Docker
- Laravel
- Vue.js

This makes development smooth and professional.

🗂️ What's Inside a GitHub Repository?

A repo usually contains:

- Project folders
- Code files
- Notes
- Docker files
- Configuration files
- A README introduction

For example, your bootcamp repo contains:

month01-php-basics
month02-php-oop
month03-php-mysql
month04-laravel-basics
month05-laravel-advanced
month06-vue-basics
month07-vue-advanced
month08-fullstack-projects
README.md

Each folder represents a month in your learning journey.

🔧 How Do You Get the Repo Onto Your Computer?

You can:

Option A — Download as ZIP

Click → **Code** → **Download ZIP**

Option B — Clone using Git

git clone https://github.com/iKeenRosal/bootcamp-fullstack-php-laravel-vue-docker.git

Once downloaded, you can open it in Visual Studio Code and begin coding.

🌱 Why You Need a Repo in This Bootcamp

The repo contains:

- All lesson materials
- All exercises
- All sample code
- All Docker environments
- All Laravel & Vue starter apps
- All monthly cheat sheets

It's the **home of your entire 8-month program**.

It also becomes your **developer portfolio** when you finish.

Installing a Repository

📘 TECH NOTES — How to Download the GitHub Repository (Prerequisite for All Lessons)

For students with no technical background — simple step-by-step

Before starting Month 01, you must download the full learning repository called:

`bootcamp-fullstack-php-laravel-vue-docker`

This repository contains:

- Your lesson folders
- Starter code
- Exercises
- Docker files
- Laravel + Vue projects
- Cheat sheets
- Everything needed for the entire 8-month program

Follow the instructions below to download it safely to your computer.

🔷 OPTION A — Download as a ZIP (Easiest for Beginners)

Step 1 — Go to the GitHub link

Open this URL in your browser:

👉 https://github.com/iKeenRosal/bootcamp-fullstack-php-laravel-vue-docker

Step 2 — Click the green “Code” button

It’s located near the top-right of the repository.

Step 3 — Click “Download ZIP”

A file named something like:

bootcamp-fullstack-php-laravel-vue-docker-main.zip

will download to your computer.

Step 4 — Unzip the file

- On Mac: double-click the ZIP
- On Windows: right-click ➜ "Extract All"

You will now have a folder **on your computer** called:

bootcamp-fullstack-php-laravel-vue-docker

Step 5 — Move the folder somewhere easy

Recommended locations:

- Desktop
- Documents
- Or a folder named “Coding Bootcamp”

Just make sure it’s easy to find.

🔷 OPTION B — Use Git (For Students Comfortable With Terminal)

If you later want to update your repo easily, you can use Git.

Step 1 — Install Git

If you don't have Git installed:

👉 https://git-scm.com/downloads

Step 2 — Clone the repository

Open a terminal and type:

git clone https://github.com/iKeenRosal/bootcamp-fullstack-php-laravel-vue-docker.git

This will create a folder with the full project.

Step 3 — Enter the folder

cd bootcamp-fullstack-php-laravel-vue-docker

🔷 FOLDER YOU SHOULD SEE AFTER DOWNLOAD

Inside the repo, you should see:

month01-php-basics/
month02-php-oop/
month03-php-mysql/
month04-laravel-basics/
month05-laravel-advanced/
month06-vue-basics/
month07-vue-advanced/
month08-fullstack-projects/
hackerank-solutions/
README.md

These will be used in order.

🔷 HOW TO CHECK THAT EVERYTHING DOWNLOADED CORRECTLY

Open the repo and confirm:

✔ You see **all 8 month folders**
✔ Each month contains **notes**, **exercises**, and **project folders**
✔ Month 04–08 include **Laravel or Vue project folders**
✔ Docker folders exist where expected
✔ The README.md is present

If anything is missing, re-download the ZIP or try cloning again.

🔷 WHY YOU MUST DOWNLOAD THE REPO BEFORE STARTING

This repo contains:

- All your practice code
- All exercises
- Starter boilerplate for each month
- Pre-built Laravel/Vue folders
- Your Docker environments
- Sample database scripts
- Your learning roadmap & cheat sheets

This ensures every lesson is ready to run with **zero setup confusion**.