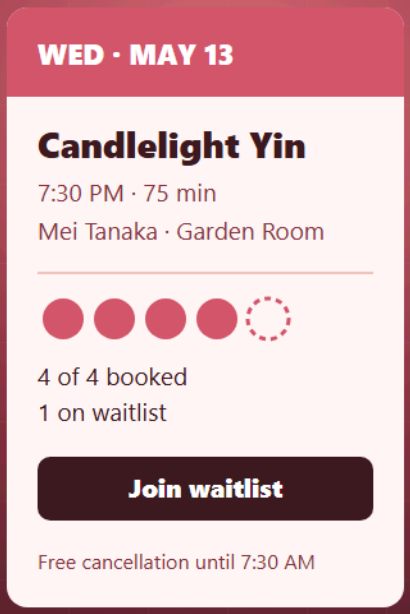
routes/web.php

```php
Route::middleware('auth')
  ->group(function () {
    Route::post('/classes/{session}
      /bookings', [BookingController::class,
      'store'])->name('bookings.store');
});
```

```php
DB::transaction(function () {
  $session->lockForUpdate();
  return $this->promoteFromWaitlist(...);
});
```




```
$ php artisan test
✓ promotes the first waitlisted member                PASS
✓ forfeits the credit on a late cancellation          PASS
```

A COMPLETE, PRACTICAL GUIDE

Laravel from Zero to Production

Build, test, and deploy modern PHP web applications

Routing · Blade · Eloquent · Auth · Policies · Queues · Mail
Sanctum APIs · Pest · Caching · Docker · Nginx · Deployment

BUILD · TEST · SECURE · DEPLOY

THENEXBOOKS

Laravel
from Zero to Production

Build, Test, and Deploy Modern PHP Web Applications
From your first route to a tested, secured, deployed booking platform — built step by step

First Edition

THENEXBOOKS

Laravel from Zero to Production
Build, Test, and Deploy Modern PHP Web Applications

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Laravel, PHP, and the packages and services described in this book change over time; commands, generated files, default settings, and features can differ between versions. Always consult the official documentation for the versions you use, and test changes before deploying them to production.

Laravel is a trademark of Taylor Otwell. PHP is a trademark of The PHP Group. Docker is a trademark of Docker, Inc. Nginx is a trademark of F5, Inc. MySQL is a trademark of Oracle and/or its affiliates. PostgreSQL is a trademark of the PostgreSQL Community Association. Redis is a trademark of Redis Ltd. Ubuntu is a trademark of Canonical Ltd. GitHub is a trademark of GitHub, Inc. Amazon Web Services and AWS are trademarks of Amazon.com, Inc. or its affiliates. Windows is a trademark of the Microsoft group of companies. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by the Laravel project, Taylor Otwell, or The PHP Group.

Willowbend Yoga, the studio used as the running example in this book, is fictional, as are its instructors, members, classes, bookings, payments, and all sample data. FakePay is a fictional payment gateway created for teaching; no real payment provider's API is described. Email addresses and domains use reserved example and test domains.

Preface

Writing a small PHP script is easy. Writing a web application that a real business depends on is a different job. Members expect to log in securely, book a class in two clicks, get an email confirmation, and receive a reminder the day before. Staff need an admin area with the right permissions and no more. Payments must never be taken twice, the last spot in a popular class must never be sold to two people at once, and when something goes wrong at 6 a.m. you need logs, tests, and a deployment process you trust.

Laravel exists to make that job manageable. It gives you routing, controllers, templates, a powerful database layer, validation, authentication, authorization, queues, mail, notifications, scheduling, file storage, API tooling, and first-class testing support, all designed to work together. Its conventions let you build quickly, and its documentation and ecosystem mean you are rarely the first person to meet a problem.

This book teaches Laravel from the first `laravel new` to a production deployment. We do not tour features in isolation. We build one real application step by step, introducing each feature at the moment it solves a problem we actually have, and we explain what Laravel does under the hood so its conveniences never feel like magic you cannot debug.

Who This Book Is For

- **PHP developers** who know the basics of the language — variables, functions, arrays, classes, and Composer — and want to build complete web applications with a modern framework.
- **Developers coming from other frameworks or languages** who need a practical, structured path into Laravel and its ecosystem.
- **Developers maintaining existing Laravel applications** who want to understand the framework properly: the container, Eloquent, queues, testing, security, and deployment.
- **Freelancers and small teams** who build booking, membership, and e-commerce style apps for clients and need to ship them reliably.

You do not need prior framework experience. If you are new to PHP itself, our companion book *Modern PHP from Zero to Production* covers the language from first principles.

What You Will Learn

- How a Laravel application is structured, how a request flows through it, and how configuration, service providers, and the service container fit together.
- Routing, controllers, Blade templates and components, and safe output by default.
- Databases with migrations, factories, and seeders; Eloquent models, relationships, eager loading, casts, enums, scopes, and collections.

- Transactions and row locking to enforce real business rules: class capacity, a waitlist with automatic promotion, and a cancellation window.
- Forms and validation, authentication, authorization with gates and policies, file uploads, and interactive front ends with Vite, Livewire, and Inertia.
- Events, queues and jobs, mail and notifications, task scheduling, payments through a gateway interface, and verified webhooks.
- JSON APIs with resources and Sanctum tokens, rate limiting, and testing with Pest and PHPUnit — including fakes and time travel.
- Caching, performance, security hardening, configuration across environments, Sail and Docker, a full VPS deployment with Nginx and PHP-FPM, managed hosting options, and monitoring.

How This Book Is Organized

Part I — Laravel Foundations (Chapters 1–5) sets up your environment and covers application structure, routing, controllers, the service container, and Blade. **Part II — Data with Eloquent** (Chapters 6–10) covers migrations, seeders, Eloquent models and relationships, advanced Eloquent features, and the transactional booking engine at the heart of the application. **Part III — Building the Web Application** (Chapters 11–15) adds forms and validation, authentication, authorization, file storage, and interactive front ends. **Part IV — Background Work and Integrations** (Chapters 16–19) covers events, queues, mail and notifications, scheduling, payments, and webhooks. **Part V — APIs, Testing, and Performance** (Chapters 20–24) builds and secures a JSON API, tests the whole application, and makes it fast. **Part VI — Production** (Chapters 25–28) hardens security, prepares environments and containers, deploys to a VPS, and compares managed hosting options before a final case study. The appendices provide a cheat sheet, templates and reference material, a production checklist, and a glossary.

Each chapter ends with exercises and a **Key Takeaways** box for quick review.

The Running Example: Willowbend Yoga

Throughout the book we build the booking platform for **Willowbend Yoga**, a fictional studio with two rooms, three instructors — Priya Nair, Jonah Reyes, and Mei Tanaka — and members who buy class passes or unlimited memberships. Members browse the weekly schedule, book classes, join a waitlist when a class is full, and cancel within the studio's rules: free cancellation up to 12 hours before class, a forfeited credit after that, and automatic promotion of the next person on the waitlist. Payments go through **FakePay**, a fictional gateway that lets us practice charges, refunds, idempotency, and signed webhooks safely. Reminders go out automatically, staff check members in from a roster, and a JSON API serves a future mobile app. The sample data is small enough to check by hand — one week of classes in May 2026 — so every number in the book can be verified.

Versions and Conventions

Examples target **recent versions of Laravel** running on **PHP 8.3 or newer**. Laravel releases a new major version roughly once a year, and each release can change installer prompts, generated files, and defaults; where details may differ, a Note says so and points you to the official documentation for your version. We use the streamlined application structure of recent Laravel releases, Composer 2, Vite, SQLite for local development, and MySQL 8 in production (PostgreSQL works equally well). Command-line examples use Unix-style shells; on Windows, use Laravel Herd, WSL, or Docker. Command output is illustrative — your version numbers and timings will differ.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, version differences, or a useful aside.

> **WARNING**
> A common mistake or something that can cause data loss, errors, or security problems.

Code appears in `monospace`; a caption above each listing names the file it belongs to, such as `app/Services/BookingService.php` or `routes/web.php`. Long lines are wrapped to fit the page. Money is always stored as integer cents in US dollars, and all timestamps are stored in UTC and displayed in the studio's time zone, America/Chicago.

Contents

Preface

PART I — LARAVEL FOUNDATIONS

1 Why Laravel? The Framework, the Ecosystem, and Your Setup

2 Anatomy of a Laravel Application: Structure, Configuration, and the Request Lifecycle

3 Routing: Routes, Parameters, Groups, and Named Routes

4 Controllers, Requests, Responses, and the Service Container

5 Blade Templates: Layouts, Components, and Safe Output

PART II — DATA WITH ELOQUENT

6 Databases, Migrations, Factories, and Seeders

7 Eloquent Models: Querying, Creating, Updating, and Deleting

8 Eloquent Relationships and Eager Loading

9 Advanced Eloquent: Scopes, Casts, Enums, Accessors, and Collections

10 Transactions and Concurrency: Bookings, Capacity, and the Waitlist

PART III — BUILDING THE WEB APPLICATION

11 Forms and Validation: Form Requests, Errors, and Old Input

12 Authentication: Starter Kits, Sessions, and Password Resets

13 Authorization: Gates, Policies, and Roles

14 File Storage and Uploads

15 Interactive Front Ends: Vite, Livewire, and Inertia

PART IV — BACKGROUND WORK AND INTEGRATIONS

16 Events, Listeners, and Observers

17 Queues and Jobs: Background Work Done Right

18 Mail and Notifications: Confirmations, Reminders, and Receipts

19 Scheduling, Payments, and Webhooks with a Fake Gateway

PART V — APIS, TESTING, AND PERFORMANCE

20 Building a JSON API: Resources, Pagination, and Error Handling

21 API Authentication with Sanctum and Rate Limiting

22 Testing Fundamentals with Pest and PHPUnit

23 Testing in Depth: HTTP, Database, Fakes, and Time Travel

24 Caching, Performance, and Debugging

PART VI — PRODUCTION

25 Security Hardening for Laravel Applications

26 Environments and Containers: Configuration, Sail, and Docker

27 Deploying to a VPS: Nginx, PHP-FPM, Workers, and Zero-Downtime Releases

28 Managed Hosting, Monitoring, and the Final Case Study

APPENDICES

A Laravel Cheat Sheet

B Templates and Reference

C Production Checklist

D Glossary

PART I

Laravel Foundations

Setting up a productive Laravel environment, understanding how an application is structured and how a request travels through it, and building Willowbend Yoga's first pages with routes, controllers, the service container, and Blade templates that are safe by default.

1 · Why Laravel?

2 · Anatomy of a Laravel Application

3 · Routing

4 · Controllers, Requests, and the Container

5 · Blade Templates

CHAPTER 1 · PART I — LARAVEL FOUNDATIONS

Why Laravel? The Framework, the Ecosystem, and Your Setup

Before we write a single route, we want to understand what a framework actually does for us, why Laravel makes the choices it makes, and how to get a working Laravel application running on your machine. By the end of this chapter you will have a fresh project called `willowbend` on your screen and a clear picture of the yoga studio we are going to build together.

What a Framework Gives You

You already know enough PHP to build a web page. You can read `$_GET`, open a database connection with PDO, run a query, and echo some HTML. So why reach for a framework at all? The honest answer is that every real web application ends up needing the same two dozen pieces of plumbing, and writing them yourself is slow, repetitive, and surprisingly easy to get wrong.

Consider a tiny piece of the application we will build in this book: showing the details of one yoga class. Written by hand, with no framework, it might look like this.

public/class.php (hand-rolled, no framework)

```
<?php

declare(strict_types=1);

$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);
if ($id === false || $id === null) {
    http_response_code(404);
    exit('Class not found');
}

$pdo = new PDO('mysql:host=127.0.0.1;dbname=willowbend', 'app', 'secret', [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
]);
$stmt = $pdo->prepare('SELECT * FROM class_sessions WHERE id = ?');
$stmt->execute([$id]);
$session = $stmt->fetch(PDO::FETCH_ASSOC);

if ($session === false) {
    http_response_code(404);
    exit('Class not found');
}
?>
<h1><?= htmlspecialchars($session['starts_at'], ENT_QUOTES, 'UTF-8') ?></h1>
```

There is nothing wrong with this code as far as it goes. It validates the input, uses a prepared statement, escapes the output, and returns a 404 when the class does not exist. But look at everything it does *not* do. The URL is `/class.php?id=4` instead of something clean like `/classes/4`. The database password is hard-coded. There is no layout, so every page repeats its own HTML head. There is no notion of a logged-in member, no protection against cross-site request forgery for the booking form that will sit on this page, no logging, no error page, no tests. Each of those is another file of plumbing you would write, debug, and maintain yourself.

Here is the same page in the Laravel application we will have by the end of Chapter 7.

routes/web.php and app/Http/Controllers/ClassSessionController.php (excerpt)

```
Route::get('/classes/{session}', [ClassSessionController::class, 'show'])
    ->name('sessions.show');

public function show(ClassSession $session): View
{
    return view('sessions.show', ['session' => $session]);
}
```

The route gives us a clean URL and a name we can use to generate links. Route model binding turns the `{session}` segment into a database lookup and answers with a 404 automatically when no row matches. The database credentials live in an environment file outside version control. The view is a Blade template that escapes everything by default and inherits a shared layout. None of that is magic in the sense of being unknowable; every piece is ordinary PHP you can open and read in the `vendor` folder. It is simply plumbing that somebody else has already written, documented, and tested against thousands of real applications.

The building blocks you get

Over the course of this book we will use nearly every major part of Laravel. Here is the list, with the chapter where each one first matters for our studio.

Building block	What it does for you	Where we use it
Routing	Maps URLs and HTTP verbs to code	Chapter 3
Controllers and the container	Organize request handling, inject dependencies	Chapter 4
Blade templates	Layouts, components, safe output	Chapter 5
Migrations and Eloquent	Version-controlled schema, an object-relational mapper	Chapters 6–10
Validation	Declarative rules, error messages, old input	Chapter 11
Authentication and authorization	Logins, sessions, gates, policies	Chapters 12–13
Queues, events, mail	Background work, decoupling, email and notifications	Chapters 16–18
Scheduler	Cron-like tasks defined in PHP	Chapter 19

API tooling	JSON resources, tokens, rate limiting	Chapters 20–21
Testing helpers	HTTP tests, fakes, time travel	Chapters 22–23

Laravel's Philosophy

Laravel is one of the most widely used PHP frameworks, and much of its appeal comes from a small set of design values that show up everywhere once you know to look for them.

Convention over configuration. If you name a model `Booking`, Laravel assumes its table is `bookings`, its primary key is `id`, and a `user_id` column points at the `users` table. If you name a policy `BookingPolicy` and put it in `app/Policies`, Laravel finds it on its own. You can override every one of these conventions, but when you follow them you write far less setup code. Throughout the book we follow the conventions unless there is a good reason not to, and we point out the convention each time so you are never surprised.

Expressive, readable APIs. Laravel code is meant to read almost like a sentence: `Schedule::command('studio:send-reminders')->everyFifteenMinutes()`, or `$session->bookings()->where('status', 'confirmed')->count()`. This style relies heavily on fluent method chaining and, in places, on *facades* — static-looking class names that forward calls to real objects behind the scenes. Facades are sometimes described as magic. We will open the hood on them in Chapter 2 and show that they are a thin, testable layer over the service container.

Batteries included, but replaceable. A new Laravel application can send mail, run queued jobs, cache data, and store files without installing anything extra. Each of those features talks to a *driver* that you choose in configuration: the cache might live in the database today and in Redis tomorrow, with no change to the code that calls `Cache::remember()`.

Developer experience matters. The framework ships with an interactive command-line tool called Artisan, generators for nearly every kind of class, a REPL called Tinker, helpful error pages during development, and testing helpers that make writing tests feel cheap. The idea is that good habits should also be the easy habits.

The Laravel Ecosystem

Beyond the framework itself, the Laravel team maintains a family of first-party packages and services. You do not need any of them to build a good application, and we will only install the ones our studio actually needs. Still, it helps to know the names, because you will see them in documentation, job descriptions, and forum answers. The list below is neutral; none of these is mandatory, and each has third-party alternatives.

Name	What it is	In this book
Sail	Docker-based local development environment	Chapter 26

Herd	Native local environment for macOS and Windows	This chapter
Sanctum	API tokens and SPA cookie authentication	Chapter 21
Livewire	Interactive components written in PHP	Chapter 15
Horizon	Dashboard and supervisor for Redis queues	Chapter 17
Telescope	Local debugging assistant	Chapter 24
Pulse	Production performance dashboard	Chapters 24, 28
Pint	Code style fixer, ships with the skeleton	Chapter 24
Octane	Runs the app on long-lived application servers	Chapter 24
Reverb	WebSocket server for real-time broadcasting	Mentioned
Forge, Envoyer	Server management and zero-downtime deploys	Chapter 28
Vapor, Laravel Cloud	Serverless and managed hosting	Chapter 28

Release Rhythm and Support

Laravel ships a new major version roughly once a year. Between majors, the team releases frequent minor and patch versions that add features and fix bugs without breaking your code. Each major version receives bug fixes for a limited period and security fixes for a longer one. The exact windows are published on the official support policy page in the Laravel documentation; check that page when you plan a new project or an upgrade, because the dates move forward every year.

For this book we target **recent versions of Laravel** running on **PHP 8.3 or newer**. We deliberately avoid tying explanations to a specific major version number. The core ideas — routes, controllers, Eloquent, Blade, queues — have been stable for many years, and the code in this book uses mainstream APIs that you will find in any recent release. Where something changed noticeably in recent versions, such as the streamlined application structure that moved middleware and routing configuration into `bootstrap/app.php`, we call it out so that you can recognize older projects too.

> **TIP**
>
> When you search the web for Laravel help, check the date and version of the answer. A highly ranked answer from several years ago may reference files like `app/Http/Kernel.php` that no longer exist in new applications. The official documentation has a version selector at the top of every page; make sure it matches the version you installed.

What You Need on Your Machine

A Laravel application has a short list of requirements:

- **PHP 8.3 or newer**, with a handful of common extensions. The documentation lists them; they include extensions such as `ctype`, `curl`, `dom`, `fileinfo`, `filter`, `hash`, `mbstring`, `openssl`, `pcre`, `pdo`, `session`, `tokenizer`, and `xml`, plus the PDO driver for your database (`pdo_sqlite` for local development in this book, `pdo_mysql` for production). Most PHP distributions enable these by default.
- **Composer**, the PHP dependency manager, which downloads Laravel and every package it depends on.
- **Node.js and npm**, used by Vite to build the CSS and JavaScript assets. You can postpone this until Chapter 5, but installing it now saves a detour.
- **A database.** New applications use SQLite by default, which needs no server at all: the whole database is a single file. We switch to MySQL 8 for production later in the book, and PostgreSQL works equally well.
- **Git**, because every project deserves version control from the first minute.

You can check what you already have from a terminal:

Terminal

```
$ php -v
$ php -m
$ composer --version
$ node -v
$ git --version
```

The first command should report PHP 8.3 or newer. The second lists loaded extensions. You should see something like `PHP 8.3.x (cli)` from the first line, followed by build details; the exact text depends on how PHP was installed.

Choosing an Installation Path

There are several good ways to get PHP, Composer, and Laravel onto your machine. Pick one; they all lead to the same kind of project.

Laravel Herd (macOS and Windows)

Herd is an official native application that bundles PHP, Composer, the Laravel installer, and a small web server. After installing it, every folder inside Herd's sites directory is automatically served at a `.test` domain, so a project named `willowbend` appears at `http://willowbend.test` with no extra configuration. Herd is the quickest route for most readers on macOS or Windows, and it lets you switch between PHP versions per project. Some features, such as additional database services, belong to a paid tier; the free version is enough for this book.

PHP and Composer installed directly

On Linux, and on macOS with Homebrew, you can install PHP and Composer through your package manager. Then install the Laravel installer globally with Composer:

Terminal

```
$ composer global require laravel/installer
```

Make sure Composer's global `vendor/bin` directory is on your `PATH` so that the `laravel` command is found. Composer prints the exact location if you run `composer global config bin-dir --absolute`.

Composer only

If you prefer not to install the Laravel installer, Composer can create a project directly from the skeleton package. This skips the interactive questions and gives you the default setup:

Terminal

```
$ composer create-project laravel/laravel willowbend
```

Laravel Sail and Docker

Sail is a lightweight command-line wrapper around Docker Compose. It runs PHP, your database, Redis, and a local mail catcher in containers, so you do not need any of them installed on the host. It is a great fit when your team wants identical environments or when you already work in Docker every day. We cover Sail in detail in Chapter 26 alongside a production Docker image.

Windows: Herd or WSL

On Windows you have two comfortable options. Herd runs natively and is the simplest. Alternatively, the Windows Subsystem for Linux (WSL) gives you a real Linux environment where every command in this book works exactly as printed. All commands in this book are written in Unix style with a `$` prompt; if you use plain PowerShell, a few of them, such as file permission commands in the deployment chapters, will need translating.

> **WARNING**
> Avoid mixing installation methods in the same project. If you create a project with Herd's PHP and later run commands with a different PHP binary from your system path, you may see confusing errors about missing extensions or a different PHP version. Run `which php` (or `where php` on Windows) when something behaves strangely.

Creating the Willowbend Project

With the installer in place, create the project. Change into the folder where you keep your code (Herd users: the Herd sites folder) and run:

Terminal

```
$ laravel new willowbend
```

The installer asks a few interactive questions. The exact wording and the list of options change between versions, so check the official documentation for the version you install, but in recent versions the questions cover three topics.

1. **A starter kit.** Recent versions of Laravel offer official starter kits in React, Vue, and Livewire flavors that come with authentication pages and a dashboard already built. Choose **none** for now. We want to understand each layer before we let a generator write it for us, and in Chapter 12 we build a minimal authentication flow by hand before comparing it to what a starter kit provides.
2. **A testing framework.** Recent installers offer Pest or PHPUnit. Choose **Pest**. It runs on top of PHPUnit, so everything you learn transfers, and Chapter 22 shows the PHPUnit equivalent for each test.
3. **A database.** Choose **SQLite**. The installer creates `database/database.sqlite` and runs the default migrations for you. We move to MySQL 8 for production later, but SQLite keeps the first chapters free of database servers.

The installer may also ask whether to initialize a Git repository or whether to run `npm install` and build the front-end assets. Saying yes to both is fine. You should see something like this when it finishes:

Terminal

```
   INFO  Application ready in [willowbend]. You can start your local
   development using:

  ➜ cd willowbend
  ➜ composer run dev
```

If you used `composer create-project` instead, you get the defaults without the questions. Recent skeletons create the SQLite file and run the initial migrations as part of that command; if yours did not, run `touch database/database.sqlite` and then `php artisan migrate` inside the project.

Starting the development server

Move into the project and start PHP's built-in development server through Artisan:

Terminal

```
$ cd willowbend
$ php artisan serve

   INFO  Server running on [http://127.0.0.1:8000].

  Press Ctrl+C to stop the server
```

Open `http://127.0.0.1:8000` in your browser and you should see Laravel's welcome page. Herd users can skip `serve` entirely and open `http://willowbend.test`. Sail users, once Sail is set up, use `http://localhost`.

Recent application skeletons also include a `composer run dev` script. It starts several processes together in one terminal: the development server, a queue listener, a log viewer, and the Vite development server for front-end assets. You do not need the queue or Vite yet, so plain `php artisan serve` is enough for the next few chapters; from Chapter 15 onward, `composer run dev` becomes the more convenient choice. If your skeleton does not include this script, nothing is lost — you can run the same commands in separate terminals.

> **NOTE**
>
> The built-in server started by `php artisan serve` is for development only. It handles one request at a time and is not designed for real traffic. In production we put Nginx and PHP-FPM in front of the application (Chapter 27) or use a managed platform (Chapter 28).

Asking Laravel about itself

Artisan has a command that summarizes the current application: its environment, versions, drivers, and cache status. Run it now; it is the fastest way to confirm that everything is wired up correctly.

Terminal

```
$ php artisan about
```

You should see something like the following, with your own version numbers and paths:

Terminal (illustrative output)

```
  Environment .....................................................
  Application Name ............................................ Laravel
  Laravel Version ................................................ x.y.z
  PHP Version ..................................................... 8.3.x
  Composer Version ............................................... 2.x.y
  Environment ..................................................... local
  Debug Mode ...................................................... ENABLED
  URL ............................................... localhost:8000
  Maintenance Mode ................................................ OFF

  Cache ...........................................................
  Config ...................................................... NOT CACHED
  Routes ...................................................... NOT CACHED
  Views ....................................................... NOT CACHED

  Drivers .........................................................
  Cache ....................................................... database
  Database ..................................................... sqlite
```

```
  Mail ..................................................... log
  Queue .................................................. database
  Session ................................................ database
```

Two details are worth noticing already. First, by default new applications use the database for the cache, the queue, and the session store, and the `log` driver for mail, which means emails are written to the log file instead of being sent. That is exactly what we want during development. Second, the application name is still "Laravel". We fix that in Chapter 2 when we tour the `.env` file and set `APP_NAME="Willowbend Yoga"`.

Saving your starting point

If the installer did not already create a Git repository, create one now and commit the untouched skeleton. Having a clean first commit makes it easy to see exactly what each chapter changes.

Terminal

```
$ git init
$ git add .
$ git commit -m "Fresh Laravel skeleton"
```

Setting Up Your Editor

Any editor that understands PHP will do, but a little setup pays off quickly in a Laravel project, where much of the code refers to classes by name and to views and routes by string.

- **Visual Studio Code** is free and popular. Install a PHP language extension for autocompletion and navigation, and consider the official Laravel extension, which adds completion for route names, view names, configuration keys, and Blade syntax highlighting. Extension names and features change over time, so search the marketplace for "Laravel" and "PHP" and read the descriptions.
- **PhpStorm** is a commercial IDE with deep PHP support and a Laravel plugin that understands facades, Eloquent, and Blade.
- Whatever you choose, configure it to run **Pint**, the code style fixer that ships with the skeleton, or simply run `./vendor/bin/pint` before each commit. Consistent formatting makes diffs far easier to review.

Meet Willowbend Yoga

A framework is easiest to learn with a real problem in front of you. Throughout this book we build one complete application for one fictional business: **Willowbend Yoga**, a small studio with two rooms, three instructors, and a growing list of members.

The owner, **Rosa Delgado**, currently takes bookings through a mix of phone calls, text messages, and a paper sign-up sheet at the front desk. Classes overbook, members forget whether they still have credits on their pass, and nobody remembers who was on the waitlist for Wednesday's candlelight class. Rosa wants a web application where members can see the schedule, book a spot in a couple of clicks, join a waitlist when a class is full, receive reminders, pay online, and cancel without calling the studio. Her front-desk manager wants a simple admin area. Later, she would like a mobile app, which means a JSON API.

The people

Person	Role in the app	Notes
Rosa Delgado	admin	Owner; full access
Sam Whitaker	staff	Front desk; manages classes and check-ins
Priya Nair	instructor	Teaches Vinyasa and Power
Jonah Reyes	instructor	Teaches Power
Mei Tanaka	instructor	Teaches Yin and Hatha
Lena, Omar, Grace, Ben, Aisha, Chloe, Noah	member	Our seven sample members

The members are a deliberately varied bunch. Lena Park and Aisha Bello hold 10-Class Passes, Grace Kim and Chloe Martin hold 5-Class Passes, and Omar Haddad has an Unlimited Monthly membership. Ben Carter has no membership at all and pays per class. Noah Fischer's unlimited membership expired at the end of March. Each of them exists to exercise a different business rule, and you will get to know them well.

Rooms, classes, and plans

The studio has two rooms: the **Sunrise Room**, which holds 20 people, and the smaller **Garden Room**, which holds 12. It offers four kinds of class.

Class type	Length	Level	Drop-in price
Vinyasa Flow	60 minutes	all	$22.00
Power Yoga	60 minutes	intermediate	$22.00
Gentle Hatha	60 minutes	beginner	$22.00
Candlelight Yin	75 minutes	all	$22.00

Members can pay per class (a drop-in) or buy a plan.

Plan	Price	Credits	Valid for
5-Class Pass	$95.00	5	60 days

10-Class Pass	$170.00	10	90 days
Unlimited Monthly	$129.00	unlimited	30 days

A quick sanity check shows why members care about the plans. A class on the 5-Class Pass costs $19.00, and on the 10-Class Pass it costs $17.00, which saves $50.00 compared with ten drop-ins at $220.00. The Unlimited Monthly plan beats paying per class from the sixth class in 30 days: six drop-ins cost $132.00, more than the $129.00 membership, while five cost only $110.00. Every amount in the application is stored as an integer number of cents in US dollars — 2200 for a drop-in, 17000 for the 10-Class Pass — a decision we explain in Chapter 6.

The rules, in brief

Rosa's studio runs on a handful of rules that will drive most of the interesting code in this book. We implement them precisely in Chapter 10; for now, here is the plain-language version.

- **Booking window.** Booking for a class opens 14 days before it starts and closes when it starts.
- **One booking per member per class.** You cannot book the same class twice.
- **How a class is paid for.** An active unlimited membership covers it first; otherwise the class pass that expires soonest gives up one credit; otherwise the member pays a $22.00 drop-in through the payment gateway.
- **Capacity and waitlist.** When a class is full, members with a valid pass or membership can join a waitlist of up to five people. Drop-in-only members cannot join a waitlist. Nobody is charged while waitlisted.
- **Cancellation.** Cancelling at least 12 hours before the start is free and returns the credit or refunds the drop-in. Later cancellations are "late": the credit is forfeited and there is no refund.
- **Waitlist promotion.** When a spot frees up, the first person on the waitlist is promoted automatically, unless the class starts within the next 60 minutes.
- **Reminders and attendance.** Members receive a reminder within 24 hours of class. Staff check members in; anyone still marked as booked after the class ends becomes a no-show.

Time zones from day one

Willowbend Yoga is located in the **America/Chicago** time zone. The application stores and compares every timestamp in **UTC** and converts to Chicago time only when it displays something to a person. Storing local times invites daylight saving bugs that appear only in March and November. During our sample week, Chicago observes daylight saving time, so local time is UTC minus five hours.

The sample week

The database seeder we write in Chapter 6 creates a realistic week of classes, from Monday 11 May to Sunday 17 May 2026. Most examples in the book refer to these seven sessions by their ids.

Id	Local start (Chicago)	Class	Instructor	Room, capacity