

#DMVCFramework MASTERY

BUILDING PRODUCTION-GRADE DELPHI APPLICATIONS FROM BEGINNER TO EXPERT



RESTful APIs & Routing

Design and build robust
web services



Security & Authentication

Protect your applications
and users



MVCActiveRecord ORM

Powerful, flexible,
and easy to use



Dependency Injection

Write clean, modular,
and testable code



Testing, Deployment & More

Test with confidence
and deploy with ease

```
app := TWebApp.Create;  
app.Use(TMVCMiddleware.Create);  
app.Use(TAuthenticationMiddleware.Create);  
app.Use(TMVCDispatcher.Create);  
app.Run;  
}
```



PRACTICAL. COMPLETE. PRODUCTION-READY.
Everything you need to build real-world applications
with DMVCFramework

MAYBACH REISIG

DMVCFramework Mastery

Building Production-Grade Delphi Applications from
Beginner to Expert

Steve Publications

This book is available at <https://leanpub.com/dmvcframeworkmastery>

This version was published on 2026-08-31



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Steve Publications

Contents

Building Production-Grade Delphi Applications from Beginner to Expert	1
Introduction	2
What You Will Learn	2
How This Book Is Organized	3
Prerequisites	4
Code Examples	4
Conventions Used	5
A Note on Versions	5
Getting Help	5
Chapter 1: Getting Started with DMVCFramework	7
What Is DMVCFramework and Why It Exists	7
Installing the Framework: Package Manager, Git, and Manual Methods	8
Your First Project: From Empty Form to Running Server	11
Understanding the Default Project Structure	15
How DMVCFramework Fits into the Delphi Ecosystem	16
Summary	17
Chapter 2: MVC Architecture Fundamentals in DMVCFramework	18
The Model-View-Controller Pattern: Origins and Intent	18
Controllers: Request Handling, Actions, and Lifecycle	19
Views: Rendering Responses, Templates, and Formats	23
Models: Data Representation and Business Logic Separation	26
How a Request Flows Through the Framework	29
Summary	31
Chapter 3: Routing and URL Design	32
Basic Route Configuration and Conventions	32
Route Parameters and Dynamic Segments	32
HTTP Method Routing and RESTful Design	33

CONTENTS

Advanced Routing: Constraints, Groups, and Prefixes	33
API Versioning Strategies with Routing	34
Summary	35
Chapter 4: Controllers in Depth	36
Controller Classes, Naming Conventions, and Registration	36
Action Methods: Signatures, Attributes, and Return Types	36
Request Data Access: Parameters, Headers, Body Parsing	37
Filters: Before, After, and Around Execution	37
Thin Controllers: Delegating to Services	38
Summary	39
Chapter 5: Views and Response Handling	40
Response Types: JSON, XML, HTML, and Binary	40
Template Engines: Mustache, Handlebars, and Custom Templates	40
Content Negotiation and Accept Headers	41
Serialization and Deserialization with DMVCFramework	42
Streaming Responses and Large Payloads	42
Summary	43
Chapter 6: Models, Services, and Repositories	44
Domain Models vs DTOs vs Entity Classes	44
The Repository Pattern: Abstraction Over Data Access	44
Service Layer Design and Business Logic Placement	45
Dependency Injection with TInjector	46
Building a Layered Architecture	46
Summary	47
Chapter 7: Database Integration and ORM Patterns	48
Connecting to Databases with FireDAC and Alternatives	48
Structuring Data Access: Repositories and Mappers	48
Transaction Management and Concurrency Patterns	49
Building a Generic Repository Base Class	49
Database Migration Strategies	49
Summary	50
Chapter 8: Authentication, Authorization, and Security	51
Authentication Mechanisms: Sessions, JWT, and API Keys	51
Implementing Role-Based Access Control (RBAC)	51
Input Validation and Sanitization Patterns	52

CONTENTS

CORS Configuration and Cross-Origin Security	52
Security Hardening: HTTPS, Headers, and Common Vulnerabilities . .	53
Summary	53
Chapter 9: Middleware, Error Handling, and Logging	54
The Middleware Pipeline: Concept and Configuration	54
Building Custom Middleware Components	54
Centralized Error Handling and Exception Filters	55
Structured Logging with Multiple Providers	55
Health Checks and Readiness Probes	56
Summary	56
Chapter 10: File Operations, Uploads, and Downloads	57
Handling File Uploads and Multipart Requests	57
Serving Static Files and Assets	57
Streaming File Downloads with Proper Headers	58
Security Considerations for File Operations	58
Integrating Cloud Storage: S3 and Azure Blob	59
Summary	59
Chapter 11: Background Processing and Async Patterns	60
Thread Management in Delphi Web Applications	60
Background Jobs and Task Queues	60
Scheduled Tasks and Cron-Like Scheduling	61
Integrating Message Brokers: RabbitMQ and Redis	61
Async/Await Patterns with Modern Delphi	61
Summary	62
Chapter 12: Caching, Performance, and Optimization	63
In-Memory Caching Strategies and Implementations	63
Distributed Caching with Redis	63
Database Query Optimization and Indexing	64
Connection Pooling and Resource Management	64
Profiling, Benchmarking, and Performance Tuning	65
Summary	65
Chapter 13: Testing DMVCFramework Applications	66
Unit Testing Controllers, Services, and Repositories	66
Mocking Dependencies with TInjector	66
Integration Testing with Real Database Connections	67

End-to-End API Testing Strategies	67
Continuous Integration Setup for Delphi Projects	67
Summary	68
Chapter 14: Deployment and Production Operations	69
Build Configuration and Environment-Specific Settings	69
Deploying to Linux, Windows Servers, and Containers	69
Docker and Kubernetes for DMVCFramework Applications	70
Monitoring, Alerting, and Observability in Production	70
Scaling Strategies: Horizontal, Vertical, and Microservices	71
Summary	71
Chapter 15: Advanced Patterns and Real-World Architecture	73
Clean Architecture and Hexagonal Design in DMVCFramework	73
Event-Driven Patterns and Domain Events	73
Microservice Communication and API Gateway	74
Building a Production-Grade Application: Complete Example	74
Framework Limitations and When to Extend or Replace	75
Summary	75
Conclusion	76
References	77

Building Production-Grade Delphi Applications from Beginner to Expert

DMVCFramework is the most popular open-source web framework for Delphi, trusted by developers worldwide since 2010. This book takes you from your first RESTful endpoint to designing scalable, secure, production-ready applications. Through code examples, architectural deep-dives, and real-world patterns, you will master controllers, routing, middleware, authentication, the MVCActiveRecord ORM, dependency injection, testing, deployment, and everything between. Whether you are building a simple API or a complex microservice architecture, this guide gives you the practical knowledge to do it right.

Introduction

In 2010, Daniele Teti released the first version of what would become DelphiMVCFramework on GitHub. At the time, Delphi developers who wanted to build modern web APIs had limited options: WebBroker was aging and clunky, ISAPI modules were painful to develop, and third-party frameworks were either expensive or incomplete. DMVCFramework filled that gap with a clean MVC architecture, attribute-based routing, automatic JSON serialization, and a developer experience that felt natural in Object Pascal.

Fifteen years later, it is still the most popular Delphi project on GitHub, with over 1400 stars, hundreds of forks, and thousands of developers using it in production systems worldwide. It has grown from a simple REST framework into a comprehensive platform supporting JSON-RPC 2.0, WebSocket communication, Server-Sent Events, JWT authentication, an ActiveRecord ORM, dependency injection, OpenAPI documentation generation, and TLS 1.3 encryption.

This book exists because DMVCFramework deserves thorough documentation that matches its capabilities. The official resources are excellent but scattered across a GitBook site, GitHub repositories, Patreon content, blog posts, and video tutorials. Many developers learn the framework piecemeal, copying samples without understanding the underlying architecture. Others never move beyond basic CRUD endpoints because they do not know how to structure larger applications properly.

This book fills that gap. It is organized as a complete learning journey from absolute beginner to advanced professional. Every chapter builds on previous ones, with code examples that grow progressively more sophisticated. By the end, you will have the knowledge to design, implement, test, secure, optimize, and deploy robust Delphi web applications using DMVCFramework.

What You Will Learn

After finishing this book, you will be able to:

- Set up a complete DMVCFramework development environment from scratch
- Design and implement RESTful APIs following Richardson Maturity Model level 3 principles
- Build controllers with proper separation of concerns and thin controller patterns
- Configure complex routing with parameters, constraints, groups, and API versioning
- Handle all response types: JSON, XML, HTML templates, files, and streaming data
- Structure applications using the repository pattern, service layer, and dependency injection
- Integrate databases efficiently using MVCActiveRecord or custom FireDAC-based repositories
- Implement secure authentication with JWT tokens, sessions, and role-based authorization
- Configure CORS, input validation, error handling, and security headers properly
- Build middleware components for logging, rate limiting, analytics, and custom processing
- Manage file uploads, static file serving, and cloud storage integration
- Implement background processing, scheduled tasks, and message queue integration
- Optimize performance with caching strategies, query tuning, and profiling techniques
- Write comprehensive unit tests, integration tests, and end-to-end API tests
- Deploy applications to Windows services, Linux daemons, Docker containers, and Kubernetes clusters
- Design microservice architectures with proper communication patterns

How This Book Is Organized

The book is divided into fifteen chapters organized in four progressive sections.

Foundation (Chapters 1-3): These chapters introduce the framework, help you install it, walk through your first application, explain MVC architecture

fundamentals, and cover routing in depth. By the end of Chapter 3, you will understand how requests flow through the framework and how to design clean URL structures.

Core Development (Chapters 4-9): This is where the bulk of development happens. You will master controller design, response handling, business logic architecture with services and repositories, database integration, security including authentication and authorization, and cross-cutting concerns like middleware, error handling, and logging. Each chapter includes complete, working code examples you can use as templates for your own projects.

Advanced Features (Chapters 10-13): These chapters cover specialized topics: file operations, background processing, caching and performance optimization, and testing strategies. You will learn how to handle real-world requirements like large file uploads, distributed caching with Redis, asynchronous task execution, and comprehensive test coverage.

Production Readiness (Chapters 14-15): The final chapters focus on deployment, operations, and advanced architectural patterns. You will learn how to configure environment-specific settings, containerize your applications, set up monitoring and observability, scale horizontally, and design microservice architectures. Chapter 15 also discusses the framework's limitations and when you might need to extend it or choose alternatives.

Prerequisites

This book assumes you have:

- Working knowledge of Object Pascal/Delphi including classes, interfaces, generics, RTTI, and attributes
- Access to Delphi 10.2 Tokyo or later (Delphi 12 Athens or 13 Florence recommended for the latest features)
- Basic understanding of HTTP, REST concepts, and JSON
- Familiarity with relational databases (PostgreSQL is used in examples, but the patterns apply to any database)

No prior experience with DMVCFramework or web development frameworks is required. If you know Delphi, this book will teach you everything else you need.

Code Examples

Every code example in this book is complete and compilable. I avoid abbreviated snippets that leave out necessary units or context. When an example spans multiple units or files, all of them are shown with clear explanations of how they fit together. Database schemas required for examples are provided where needed.

The examples follow a progressive pattern: simple concepts are introduced first in isolation, then combined and expanded in later chapters. A basic user management API that appears early in the book evolves into a production-grade system with authentication, validation, caching, and proper architecture by the final chapters.

Conventions Used

- Code is formatted in monospace blocks with line numbers where helpful
- File paths are shown relative to the project root unless otherwise noted
- URLs in examples use `http://localhost:8080` as the default base address
- Delphi language features specific to understanding DMVCFramework code are explained inline when they first appear
- Version-specific behavior is clearly labeled (e.g., “Available since version 3.4.2-magnesium”)

A Note on Versions

DMVCFramework follows a semantic versioning scheme with descriptive codenames: hydrogen, lithium, beryllium, boron, carbon, and so on through the latest releases. This book targets version 3.4.x as the baseline, with notes about features introduced in later versions where relevant. The core architecture has been stable since version 3.0, so most patterns described here will remain valid for future releases.

When a feature or behavior differs between major versions, I explicitly call it out. If you are working with an older version (before 3.4.2), some of the newer features like the dependency injection container and server-side session management will not be available, but the fundamental MVC patterns and routing concepts remain the same.

Getting Help

While reading this book, you may encounter questions or issues that go beyond what is covered here. The DMVCFramework community is active and helpful. Key resources include:

- The official GitHub repository at <https://github.com/danieleteti/delphimvcframework> with issue tracking and discussions
- The DelphiMVCFramework Facebook group where hundreds of developers share knowledge
- Daniele Teti's blog at <https://www.danieleteti.it> for framework updates and advanced tutorials
- Stack Overflow with the `delphi-mvc-framework` tag

Now let us begin. Chapter 1 will get you up and running with your first DMVCFramework application in minutes.

Chapter 1: Getting Started with DMVCFramework

Before you can build anything with DMVCFramework, you need to install it and understand how it fits into the Delphi ecosystem. This chapter covers everything from downloading the framework to running your first web server, including installation methods, IDE configuration, project creation, and a tour of the default project structure. By the end, you will have a working application responding to HTTP requests and the foundation for everything that follows.

What Is DMVCFramework and Why It Exists

DMVCFramework is an open-source web framework for Delphi that implements the Model-View-Controller (MVC) architectural pattern. Created by Daniele Teti and released under the Apache 2.0 license, it has been in active development since 2010 and remains the most popular Delphi project on GitHub with over 1400 stars and a thriving community of contributors worldwide.

At its core, DMVCFramework solves a simple problem: making it easy to build modern web applications and RESTful APIs in Delphi without wrestling with low-level HTTP handling or legacy frameworks. Before DMVCFramework existed, Delphi developers had three main options for building web services:

1. **WebBroker:** Built into Delphi since version 4, WebBroker provides basic HTTP request handling but lacks modern conventions, automatic serialization, and clean routing. It feels like building a web application in the late 1990s.
2. **ISAPI modules:** These run inside IIS or Apache and offer good performance, but development is painful. You handle every aspect of request parsing, response formatting, and error handling manually. Debugging ISAPI modules requires attaching to a running server process.
3. **Third-party frameworks:** Before DMVCFramework, options were limited. Some required expensive licenses, others had steep learning curves, and many lacked the comprehensive feature set modern applications demand.

DMVCFramework combines the best aspects of these approaches while adding modern conveniences:

- It runs as a standalone console application (easy to debug) but can also be deployed as an ISAPI module, Apache module, or Windows/Linux service
- Routing is declarative using Delphi attributes, not configuration files
- JSON serialization happens automatically through RTTI inspection
- The MVC pattern enforces clean separation between request handling, business logic, and data access
- Built-in middleware support handles cross-cutting concerns like authentication, logging, and CORS
- OpenAPI/Swagger documentation is generated automatically from your code

The framework supports multiple deployment models out of the box. You can start with a simple console application during development and deploy to production as a Windows service, Linux daemon, Docker container, or web server module without changing your application code. This flexibility is one of DMVCFramework's greatest strengths.

Current stable versions (3.4.x) support Delphi 10.2 Tokyo through Delphi 13 Florence on Windows and Linux platforms. Android support exists in experimental form. The framework integrates with major databases including PostgreSQL, MySQL, SQL Server, Oracle, Firebird, SQLite, and InterBase through its built-in MVCAccessRecord ORM or direct FireDAC connections.

Installing the Framework: Package Manager, Git, and Manual Methods

There are three ways to install DMVCFramework in your development environment. I recommend the release zip method for most developers because it is straightforward and avoids potential issues with incomplete Git submodules. However, all three methods are covered here so you can choose what works best for your workflow.

Method 1: Release Zip (Recommended)

This is the simplest and most reliable installation method. It gives you a complete, tested release without worrying about submodule initialization or build order.

1. Go to the GitHub releases page at <https://github.com/danieleteti/delphimvcframework>
2. Download the latest stable release zip file (for example, `dmvcframework-3.4.3.zip`). As of this writing, version 3.4.3 is the current stable release.
3. Extract the archive to a permanent location on your system. I recommend `C:\DMVC` on Windows or `/opt/DMVC` on Linux, but any path works. Avoid spaces in the path if possible to prevent issues with some build tools.
4. Open RAD Studio and navigate to the appropriate packages folder for your Delphi version:
 - Delphi 13 Florence: `packages/d130/`
 - Delphi 12 Athens: `packages/d120/`
 - Delphi 11 Alexandria (11.3): `packages/d113/`
 - Delphi 10.4 Sydney: `packages/d104/`
 - Delphi 10.3 Rio: `packages/d103/`
 - Delphi 10.2 Tokyo: `packages/d102/`
5. Open the group project file named `dmvcframework_group.groupproj`
6. Right-click on the group project and select **Build All** from the context menu
7. After all packages build successfully, right-click on the design-time package named `dmvcframeworkDT` and select **Install**
8. If installation succeeds, you will see a confirmation dialog. Click **OK** to continue.
9. Configure library paths by going to **Tools > Options > Environment Options > Delphi Options > Library** and adding these paths (adjusting for your installation directory):
 - `C:\DMVC\sources\`
 - `C:\DMVC\lib\dmustache\`
 - `C:\DMVC\lib\loggerpro\`
10. Click **OK** to save the settings

The framework is now installed and ready to use. You should see a new `DelphiMVCFramework` category in the **File > New > Other** dialog for creating projects with the built-in wizard.

Method 2: Git Clone (For Contributors or Latest Features)

If you want access to the latest development features or plan to contribute to the framework, cloning from Git gives you the most up-to-date code. However, this method requires more steps and care to ensure all dependencies are properly initialized.

1. Install Git if you do not already have it (<https://git-scm.com>)
2. Open a terminal and clone the repository with submodules:

```
git clone --recursive https://github.com/danieleteti/delphimvcframework.git
```
3. Navigate to the cloned directory and ensure all submodules are initialized:

```
cd delphimvcframework git submodule update --init --recursive
```
4. Follow steps 5-10 from Method 1 above to build packages, install the design-time package, and configure library paths

Note that master branch code may contain breaking changes or incomplete features not present in stable releases. For production work, always use tagged releases rather than the development branch.

Method 3: Delphi Package Manager (DPM)

The Delphi Package Manager is a third-party tool inspired by NuGet that can manage DMVCFramework as a dependency. This method is useful if you already use DPM for other packages or want automated dependency management across multiple projects.

1. Install DPM from <https://github.com/DelphiPackageManager/DPM> if not already installed
2. Open your Delphi project and access the Package Manager view
3. Search for delphimvcframework in the package search
4. Install the latest stable version using the Install button
5. DPM will automatically configure library paths and add required references to your project

The DPM method is convenient but may lag behind official releases since it depends on someone maintaining the package listing. For most developers, the direct release zip installation remains the best choice.

Troubleshooting Installation Issues

If you encounter problems during installation, check these common issues:

- **“Unit not found” errors:** Verify that all three library paths (sources, dmustache, loggerpro) are added to your Delphi library path settings. The order matters: sources should come before lib folders.
- **IDE Expert does not appear:** The IDE Expert requires Delphi Enterprise or Architect edition because it uses the Open Tools API. Professional edition users can still use DMVCFramework but must create projects manually instead of using the wizard.
- **Package build failures:** Ensure you are opening the correct package group for your Delphi version. Opening a d120 group project in Delphi 11 will fail with compiler errors.
- **Access violations at runtime:** Check that all required DLLs (especially Indy and FireDAC components) are available in your application’s deployment path, particularly on Linux where shared library paths must be configured correctly.

Your First Project: From Empty Form to Running Server

Now that DMVCFramework is installed, let us create your first application. I will show you two approaches: using the IDE wizard (if available) and creating a project manually. Both produce functionally identical results.

Using the IDE Wizard

If you have Delphi Enterprise or Architect edition and successfully installed the IDE Expert, creating a new DMVCFramework project takes seconds:

1. Select File > New > Other from the main menu
2. In the New Items dialog, expand the DelphiMVCFramework category
3. Select DelphiMVCFramework Project and click OK
4. The wizard presents several configuration options:
 - **Controller Class Name:** Enter a name like TMainController (this is just the initial controller)

- Server type: Choose Console Application for development (simplest to debug)
- Protocol: HTTP for now; we will add HTTPS later
- Port: 8080 is the default and works fine

5. Click OK to generate the project

The wizard creates a complete project with a DPR file, WebModule unit, controller unit, and configuration files. Run the application by pressing F9. You should see console output indicating the server started on port 8080.

Open your browser and navigate to <http://localhost:8080/api/system/describeserver.in>. You will see JSON information about the running server, including framework version, available controllers, and configuration details. This system endpoint is automatically provided by DMVCFramework and useful for debugging.

Creating a Project Manually

If you do not have access to the IDE wizard (for example, using Delphi Professional), creating a DMVCFramework project manually is straightforward. Follow these steps:

1. Create a new WebBroker Application (File > New > Other > WebBroker Application)
2. Select Console Application as the server type and click OK
3. Open the generated WebModule unit and modify it to include DMVCFramework:

```
1  unit WebModuleUnit;  
2  
3  interface  
4  
5  uses  
6      System.SysUtils, System.Classes, Web.HTTPApp,  
7      MVCFramework;  
8  
9  type  
10     TWebModule1 = class(TWebModule)  
11     procedure WebModuleCreate(Sender: TObject);  
12     procedure WebModuleDestroy(Sender: TObject);  
13 private
```

```

14     MVC: TMVCEngine;
15     public
16     end;
17
18     var
19         WebModuleClass: TComponentClass = TWebModule1;
20
21     implementation
22
23     {$R *.dfm}
24
25     uses
26         MainControllerU;
27
28     procedure TWebModule1.WebModuleCreate(Sender: TObject);
29     begin
30         MVC := TMVCEngine.Create(Self);
31
32         // Set default content type to JSON
33         MVC.Config[TMVCCConfigKey.DocumentRoot] :=
34             ↪ 'C:\DMVC\samples\hello_world\public';
35         MVC.Config[TMVCCConfigKey.DefaultContentType] := 'application/json';
36
37         // Enable pretty-printed JSON for development (disable in production)
38         MVC.Config[TMVCCConfigKey.PrettyJSON] := True;
39
40         MVC.Setup;
41     end;
42
43     procedure TWebModule1.WebModuleDestroy(Sender: TObject);
44     begin
45         FreeAndNil(MVC);
46     end;
47
48     initialization
49         WebReqMgr.InitServer('8080', nil, 'http://localhost');
50     end.

```

4. Create a new unit called MainControllerU and add your first controller:

```

1  unit MainControllerU;
2
3  interface
4
5  uses
6      MVCFramework;
7
8  type
9      [MVCPath('/api')]
10     TMainController = class(TMVCController)
11     public
12         [MVCPath('/hello')]
13         [MVCHTTPMethod([httpGET])]
14         function HelloWorld: string;
15
16         [MVCPath('/hello/($name)')]
17         [MVCHTTPMethod([httpGET])]
18         function HelloName(const name: string): string;
19     end;
20
21 implementation
22
23 function TMainController.HelloWorld: string;
24 begin
25     Result := 'Hello, World!';
26 end;
27
28 function TMainController.HelloName(const name: string): string;
29 begin
30     Result := Format('Hello, %s!', [name]);
31 end;
32
33 end.

```

5. Run the application with F9

6. Test your endpoints:

- `http://localhost:8080/api/hello` returns “Hello, World!”
- `http://localhost:8080/api/hello/Alice` returns “Hello, Alice!”

Congratulations. You have just created your first DMVCFramework application with two working REST endpoints. Let me explain what is happening here because understanding these fundamentals is crucial for everything that follows.

The controller class `TMainController` inherits from `TMVCController`, the base class for all DMVCFramework controllers. The `[MVCPath('/api')]` attribute

on the class defines the base URL path for all actions in this controller. Each public method decorated with `[MVCPath]` becomes a route endpoint.

The first action `HelloWorld` has the path `/hello`, which combines with the controller path to form `/api/hello`. The `[MVCHttpMethod([httpGET])]` attribute restricts this endpoint to HTTP GET requests only. When called, it returns a string that DMVCFramework automatically serializes to JSON and sends as the response body.

The second action `HelloName` introduces route parameters. The syntax `($name)` in the path captures whatever value appears at that position in the URL and passes it as a method parameter. This is called strongly typed routing: DMVCFramework uses RTTI to match URL parameter names to method parameter names and performs automatic type conversion for supported types including `Integer`, `Int64`, `Boolean`, `TDateTime`, `Single`, `Double`, and `string`.

Understanding the Default Project Structure

Whether you used the wizard or created the project manually, understanding the file structure helps you navigate larger applications. A typical DMVCFramework project contains these key files:

- **ProjectName.dpr**: The main Delphi program file that initializes the WebBroker server and starts listening for requests
- **WebModuleUnit.pas**: Contains the `TWebModule` descendant where `TMVCEngine` is created and configured. This is the framework's entry point where you configure middleware, routing options, serialization settings, and other global behavior
- **Controller units** (e.g., `MainControllerU.pas`): Each controller lives in its own unit. Controllers handle incoming requests, validate input, call business logic services, and return responses
- **.env file** (optional but recommended): Environment-specific configuration loaded by the DotEnv system. Contains settings like database connection strings, API keys, port numbers, and feature flags without hardcoding them into source files

For larger applications, I recommend organizing your code into folders following a layered architecture pattern:

```
1 MyProject/
2 |— Controllers/          # All controller units
3 |— Services/             # Business logic services
4 |— Repositories/         # Data access layer
5 |— Models/               # Domain entities and DTOs
6 |— Middleware/           # Custom middleware components
7 |— Configuration/       # Config classes and helpers
8 |— Tests/                # Unit and integration tests
```

This structure scales well as your application grows. Each layer has a clear responsibility, making the codebase easier to navigate, test, and maintain. We will explore this architecture in detail starting in Chapter 6.

How DMVCFramework Fits into the Delphi Ecosystem

DMVCFramework does not replace other Delphi technologies; it builds on top of them. Understanding these relationships helps you make informed decisions about when and how to use each tool.

WebBroker: DMVCFramework uses WebBroker as its underlying HTTP handling layer in most deployment scenarios. The TMVCEngine is created inside a TWebModule and intercepts all incoming requests. You benefit from WebBroker’s mature request processing while getting DMVCFramework’s modern MVC abstractions on top.

Indy: For console application deployments, DMVCFramework uses Indy (TIdHTTPServer) as the HTTP server instead of WebBroker. This gives you a standalone executable that does not require IIS or Apache. The framework abstracts away these differences so your controller code works identically regardless of the underlying server.

FireDAC: For database access, DMVCFramework’s MVCActiveRecord ORM is built on FireDAC, Delphi’s universal data access library. You can also use FireDAC directly in custom repositories if you need more control than ActiveRecord provides. The framework never forces a specific data access approach; it gives you tools and lets you choose.

RTTI: DMVCFramework relies heavily on Delphi’s Runtime Type Information for automatic JSON serialization, route parameter binding, validation, and Swagger documentation generation. Understanding RTTI basics helps you customize these features when needed.

Comparison with alternatives: The main competitors in the Delphi web ecosystem are:

- **mORMot:** A high-performance framework by Arnaud Bouchez that combines ORM, REST, and JSON-RPC in a single library. It is extremely fast but has a steeper learning curve and less conventional architecture than DMVCFramework.
- **WebBroker/ISAPI:** Built-in Delphi options that give you full control but require handling everything manually. No automatic serialization, no routing conventions, no middleware system.
- **TMS Web Core/API:** Commercial frameworks from TMS Software with visual designers and additional features, but at a licensing cost.

DMVCFramework occupies a sweet spot: modern conventions and rich features without sacrificing the familiar Object Pascal patterns Delphi developers already know. It is production-proven in systems handling millions of requests daily while remaining approachable enough for beginners to create their first API in minutes.

Summary

In this chapter, you learned what DMVCFramework is and why it exists in the Delphi ecosystem. You installed the framework using one of three methods, created your first project with working REST endpoints, and explored the default project structure. You now have a running server responding to HTTP requests and understand the basic building blocks: controllers, actions, routing attributes, and automatic serialization.

The next chapter dives into MVC architecture fundamentals, explaining how models, views, and controllers interact within DMVCFramework, how requests flow through the framework pipeline, and why proper separation of concerns matters for building maintainable applications.

Chapter 2: MVC Architecture

Fundamentals in DMVCFramework

Understanding how DMVCFramework implements the Model-View-Controller pattern is essential for writing clean, maintainable code. This chapter explains each component's role, lifecycle, and interactions within the framework. You will learn how a request travels from the HTTP server to your controller action and back to the client as a response, what responsibilities each layer should have, and common mistakes that violate MVC principles. By the end, you will have the architectural foundation needed to design well-structured applications rather than collections of ad-hoc endpoints.

The Model-View-Controller Pattern: Origins and Intent

The MVC pattern originated in the 1970s as part of Smalltalk's user interface architecture. It was designed to solve a specific problem: keeping user interface code separate from application logic and data representation so that changes in one area do not ripple through the entire system. Over decades, MVC has been adapted for web applications where the “view” is typically HTML or JSON returned to a browser rather than a graphical widget.

In its purest form, MVC divides an application into three interconnected components:

- **Model:** Represents the data and business rules of the application. Models know nothing about how data is displayed or how user requests arrive. They encapsulate domain logic, validation rules, and relationships between entities.
- **View:** Responsible for presenting data to the user. In web applications, views can be HTML templates rendered server-side, JSON responses consumed by JavaScript frameworks, or any format appropriate for the client. Views should contain minimal logic; their job is formatting and presentation only.

- **Controller:** Acts as an intermediary between models and views. Controllers receive incoming requests, interpret user intent, interact with models to retrieve or modify data, and select the appropriate view for the response. Controllers coordinate but do not perform core business operations themselves.

The key insight behind MVC is separation of concerns. When each component has a single responsibility, you can change how data is displayed without touching business logic, modify validation rules without affecting routing, or swap out database implementations without changing controllers. This modularity makes applications easier to understand, test, and evolve over time.

DMVCFramework implements MVC with some adaptations suited to RESTful API development:

- Controllers are Delphi classes inheriting from `TMVCController` with actions defined as public methods decorated with routing attributes
- Views are response renderers that convert objects into JSON, XML, HTML templates, or binary formats
- Models can be simple data classes (DTOs), domain entities using `MVCActiveRecord`, or any object representing application data

Unlike traditional web MVC frameworks where views are template files in a directory structure, DMVCFramework's view system is more flexible. For REST APIs, the "view" is often just automatic JSON serialization of whatever object your controller action returns. For server-rendered HTML applications, Mustache or Sempare templates serve as views in the traditional sense.

Controllers: Request Handling, Actions, and Lifecycle

Controllers are the entry point for all requests in a DMVCFramework application. Every HTTP request that matches a route is dispatched to a controller action, which processes the request and produces a response. Understanding controller behavior is fundamental because controllers appear in every endpoint you write.

Controller Class Structure

Every controller in DMVCFramework inherits from `TMVCController`, defined in `MVCFramework.pas`. The base class provides access to the current request context, rendering methods for responses, session data, and utility functions. A minimal controller looks like this:

```

1  unit UsersControllerU;
2
3  interface
4
5  uses
6      MVCFramework;
7
8  type
9      [MVCPATH('/users')]
10     TUsersController = class(TMVCController)
11     public
12         [MVCPATH('')]
13         [MVCHTTPMethod([httpGET])]
14         procedure GetAll(ctx: TWebContext);
15     end;
16
17 implementation
18
19 procedure TUsersController.GetAll(ctx: TWebContext);
20 begin
21     // Controller logic here
22 end;
23
24 end.
```

The class-level `[MVCPATH('/users')]` attribute defines the base path for all actions in this controller. Individual action paths are relative to this base, so an action with `[MVCPATH('')]` responds at `/users` while `[MVCPATH('/123')]` would respond at `/users/123`.

Controller Lifecycle

Understanding when controllers are created and destroyed is important for managing resources correctly. In DMVCFramework, a new controller instance is created for each incoming request. This per-request lifecycle has several implications:

- **Thread safety:** Each request runs on its own thread with its own controller instance, so you do not need to synchronize access to controller properties between requests
- **No state persistence:** Do not store data in controller fields expecting it to survive across requests. Use sessions or external storage for that purpose
- **Resource cleanup:** Any resources allocated in the controller should be freed before the action completes, typically using try-finally blocks

The framework manages this lifecycle automatically through TMVCEngine. When a request arrives and matches a route, the engine:

1. Creates a new instance of the target controller class
2. Injects any dependencies marked with [MVCInject] attributes
3. Executes middleware hooks (OnBeforeRouting, OnBeforeControllerAction)
4. Calls the matched action method
5. Executes post-action middleware hooks (OnAfterControllerAction)
6. Serializes the return value or uses Render methods to produce the response
7. Frees the controller instance

This lifecycle means controllers are lightweight coordination objects. Heavy initialization work should happen in singleton services, not in controllers.

Action Methods and Return Types

Action methods can be procedures or functions, and their return type determines how responses are generated:

- **String return:** The string is returned as-is with the configured content type (usually text/plain or application/json)
- **Object return:** Any Delphi object is automatically serialized to JSON using RTTI. Nested objects, arrays, and collections are handled recursively
- **TJSONObject/TJSONArray:** Returned directly as JSON without additional serialization
- **TDataSet descendants:** Serialized as JSON arrays of records with field names as keys

- **Stream return:** Binary streams (images, PDFs) are returned with appropriate content type headers
- **Procedure (no return):** The action must use Render methods explicitly to produce a response

Here is an example showing multiple return types:

```

1  type
2    [MVCPath('/api/items')]
3    TItemsController = class(TMVController)
4    public
5      // Returns JSON object automatically serialized from the function result
6      [MVCPath('/')]
7      [MVCHTTPMethod([httpGET])]
8      function GetItems: TObjectList<TItem>;
9
10     // Returns a single item by ID
11     [MVCPath('/($id)')]
12     [MVCHTTPMethod([httpGET])]
13     function GetItem(const id: Integer): TItem;
14
15     // Returns custom JSON structure directly
16     [MVCPath('/summary')]
17     [MVCHTTPMethod([httpGET])]
18     function GetSummary: TJSONObject;
19
20     // Uses Render for procedural control
21     [MVCPath('/dataset')]
22     [MVCHTTPMethod([httpGET])]
23     procedure GetAsDataset(ctx: TWebContext);
24   end;

```

When an action returns an object, DMVCFramework uses its serialization system to convert it to the appropriate format based on the client's Accept header and the action's [MVCProduces] attribute. By default, JSON is used. We will explore serialization in depth in Chapter 5.

Accessing Request Data

Controllers need access to incoming request data: query parameters, URL path segments, headers, cookies, and body content. DMVCFramework provides several ways to access this data:

Route parameters: As shown earlier, (*\$paramname*) syntax in *MVCPath* captures URL segments and passes them as strongly typed method parameters. This is the cleanest approach for path-based parameters.

Query string parameters: Access via *Context.Request.Params['paramname']* or by declaring them as action parameters with *[MVCFromQueryString]* attribute:

```
1 [MVCPath('/search')]
2 [MVCHTTPMethod([httpGET])]
3 function Search(
4     [MVCFromQueryString] const query: string;
5     [MVCFromQueryString('page')] const page: Integer = 1): TObjectList<TResult>;
```

Request body: For POST/PUT requests with JSON bodies, use *[MVCFromBody]* to have the framework deserialize automatically:

```
1 [MVCPath('')]
2 [MVCHTTPMethod([httpPOST])]
3 procedure CreateItem(const item: TNewItem);
```

The framework deserializes the JSON body into the specified type using RTTI. If deserialization fails or required fields are missing, appropriate error responses are generated automatically when validation is enabled.

Headers and cookies: Access via *Context.Request.Headers['HeaderName']* and *Context.Request.Cookies['CookieName']*. You can also use attributes:

```
1 [MVCPath('/test')]
2 [MVCHTTPMethod([httpGET])]
3 procedure Test(
4     [MVCFromHeader('Authorization')] const authHeader: string;
5     [MVCFromCookie('session_id')] const sessionId: string);
```

Views: Rendering Responses, Templates, and Formats

In *DMVCFramework*, views are the mechanism for converting data into HTTP responses. The framework supports multiple view types depending on your application's needs: JSON for APIs, HTML templates for server-rendered pages, XML for legacy integrations, and binary streams for file downloads.

Automatic Response Rendering

The simplest form of view in DMVCFramework requires no explicit code at all. When a controller action returns an object, the framework automatically serializes it to JSON (or another format based on configuration) and sends it as the response body with appropriate headers. This automatic rendering is why DMVCFramework feels so productive: you focus on business logic, not boilerplate serialization code.

```
1 function TUsersController.GetUser(const id: Integer): TUser;
2 begin
3     Result := FUserService.FindById(id);
4     // Automatically serialized to JSON and sent as response
5 end;
```

Behind the scenes, TMVCEngine calls the appropriate renderer based on the return type and content negotiation rules. The renderer handles setting Content-Type headers, HTTP status codes, and character encoding automatically.

Explicit Rendering with Render Methods

Sometimes you need more control over responses than automatic rendering provides. Every controller has access to Render methods through the inherited TMVCController base class:

```
1 procedure TItemsController.DeleteItem(const id: Integer);
2 begin
3     if not FItemService.Delete(id) then
4     begin
5         Context.Response.StatusCode := 404;
6         Render('Item not found');
7         Exit;
8     end;
9     Context.Response.StatusCode := 204; // No Content
10 end;
```

The Render method has many overloads supporting different data types:

- Render(String): Sends a plain text or JSON string response

- `Render(TObject)`: Serializes an object to the configured format
- `Render(TJSONObject)`: Sends JSON directly without additional serialization
- `Render(TStream, FreeStream)`: Streams binary content (files, images) with optional auto-free
- `Render(TDataSet)`: Serializes dataset contents as JSON array

You can also set HTTP status codes explicitly via `Context.Response.StatusCode` before rendering. By default, successful responses use 200 OK. For RESTful APIs, using correct status codes matters: 201 Created for new resources, 204 No Content for deletions, 400 Bad Request for validation errors, 401 Unauthorized for authentication failures, 403 Forbidden for authorization denials, 404 Not Found for missing resources, and 500 Internal Server Error for unexpected server problems.

HTML Templates with Mustache and Sempare

For applications that render HTML on the server (rather than returning JSON to a JavaScript frontend), DMVCFramework integrates with template engines. Mustache is included by default; Sempare was added in version 3.4.2-magnesium as an additional option.

Mustache templates are logic-less: they contain placeholders for data but no conditional logic or loops embedded in code. This keeps presentation concerns separate from business logic, consistent with MVC principles.

To use Mustache templates:

1. Create a templates directory in your project (or configure the path via `TMVCConfigKey.DocumentRoot`)
2. Add template files with `.mustache` extension:

```

1 <!-- templates/user.mustache -->
2 <!DOCTYPE html>
3 <html>
4 <head><title>{{Name}}</title></head>
5 <body>
6   <h1>{{Name}}</h1>
7   <p>Email: {{Email}}</p>
8   <ul>
9     {{#Orders}}
10    <li>{{ProductName}} - ${{Price}}</li>
11    {{/Orders}}
12  </ul>
13 </body>
14 </html>

```

3. Render the template from a controller action:

```

1 [MVCPath('/profile')]
2 [MVCHTTPMethod([httpGET])]
3 [MVCProduces('text/html')]
4 procedure TUserController.ShowProfile;
5 var
6   User: TUser;
7 begin
8   User := FUserService.GetCurrentUser;
9   RenderMustache('user', User);
10 end;

```

The `RenderMustache` method looks for `templates/user.mustache`, fills in placeholders with data from the `User` object, and returns the result as an HTML response. Mustache supports sections (`{{...}}{/Section}}`) for arrays and conditional display, inverted sections (`{{^Section}}...{/Section}}`) for empty checks, and partials (`{{> partialName}}`) for reusable template fragments.

Sempare templates work similarly but offer more advanced features including custom helpers, filters, and a different syntax style. Choose based on your team's preferences and existing familiarity.

Models: Data Representation and Business Logic Separation

Models in DMVCFramework are simply Delphi classes that represent data or domain concepts. The framework does not enforce a specific model structure;

you design models according to your application's needs and architectural patterns.

Types of Models

In practice, three types of model classes appear in most applications:

DTOs (Data Transfer Objects): Simple classes used to transfer data between layers or over the network. DTOs contain properties but minimal logic. They are ideal for API request/response payloads because you control exactly what data is exposed without leaking internal implementation details.

```
1  type
2    TUserDTO = class
3      private
4        FId: Integer;
5        FName: string;
6        FEmail: string;
7      published
8        property Id: Integer read FId write FId;
9        property Name: string read FName write FName;
10       property Email: string read FEmail write FEmail;
11     end;
```

Note the use of published properties. DMVCFramework's RTTI-based serialization inspects published properties by default, so only published members appear in JSON output. Private and protected fields are excluded unless you use specific serialization attributes.

Domain entities: Classes representing core business concepts with behavior attached. Unlike DTOs, domain entities may contain methods that enforce business rules, calculate derived values, or manage relationships. When using MVCActiveRecord, domain entities inherit from TMVCActiveRecord and include persistence logic.

```

1  type
2    TOrder = class(TMVCActiveRecord)
3      private
4        FOrderId: Integer;
5        FCustomerId: Integer;
6        FTotalAmount: Currency;
7        FStatus: TOrderStatus;
8      published
9        [MVCTableField('order_id', [foPrimaryKey, foAutoGenerated])]
10       property OrderId: Integer read FOrderId write FOrderId;
11
12       [MVCTableField('customer_id')]
13       property CustomerId: Integer read FCustomerId write FCustomerId;
14
15       [MVCTableField('total_amount')]
16       property TotalAmount: Currency read FTotalAmount write FTotalAmount;
17
18       [MVCTableField('status')]
19       property Status: TOrderStatus read FStatus write FStatus;
20     end;

```

ViewModels: Classes specifically designed for a particular view or response format. ViewModels combine data from multiple sources into a structure optimized for presentation. They are useful when the data needed by a client does not match any single domain entity.

```

1  type
2    TUserProfileViewModel = class
3      published
4        property Id: Integer;
5        property DisplayName: string;
6        property Email: string;
7        property MemberSince: TDateTime;
8        property OrderCount: Integer;
9        property TotalSpent: Currency;
10       property RecentOrders: TObjectList<TOrderSummary>;
11     end;

```

Where Business Logic Belongs

A common mistake in MVC applications is putting all business logic into controllers. This creates fat controllers that are hard to test, understand, and maintain. Proper MVC architecture places business logic in models or dedicated service classes, leaving controllers as thin coordinators.

The rule of thumb: if a piece of logic does not involve HTTP concerns (reading request parameters, setting response headers, choosing status codes), it probably belongs outside the controller. Controllers should:

- Parse and validate incoming request data
- Call appropriate services or repositories
- Handle HTTP-level error responses
- Select the appropriate view or serialization format

They should NOT:

- Execute complex business calculations
- Manage database transactions directly
- Implement authentication algorithms
- Contain duplicate logic shared across multiple endpoints

We will explore proper service layer design and repository patterns in Chapter 6, where you will learn how to structure business logic for maximum maintainability.

How a Request Flows Through the Framework

Understanding the complete request lifecycle helps you debug problems, write middleware correctly, and appreciate where each component fits. Let us trace what happens when a client sends an HTTP GET request to `http://localhost:8080/api/users/42`.

Step 1: HTTP Server receives the request. Depending on your deployment mode, this is either Indy's `TIdHTTPServer` (console application), `WebBroker` (ISAPI/Apache module), or another handler. The raw HTTP request is parsed into method, path, headers, and body.

Step 2: TMVCEngine intercepts the request. The engine is configured in your `WebModule`'s `OnCreate` event and handles all incoming requests. It creates a `TWebContext` object containing the request and response objects, which will be passed through the pipeline.

Step 3: Middleware executes (OnBeforeRouting). Each registered middleware component gets a chance to inspect and modify the request before

routing occurs. CORS middleware adds appropriate headers here. Authentication middleware might validate tokens. Custom logging middleware records request details. Any middleware can short-circuit the pipeline by setting `Handled := True`, preventing further processing and sending an immediate response.

Step 4: Routing matches the request to a controller action. The router examines all registered controllers and their actions, looking for a match based on HTTP method and URL path. It uses the `MVCPATH` attributes on both the controller class and individual actions, combining them to form complete routes. Route parameters in the URL are extracted and stored for later use.

If no matching route is found, the engine returns a 404 Not Found response with details about available routes (when configured). If multiple routes match ambiguously, a configuration error is raised.

Step 5: Controller instance is created. A new instance of the matched controller class is instantiated. If the controller or any action parameters have `[MVCInject]` attributes, the dependency injection container resolves and injects the required services at this point.

Step 6: Middleware executes (OnBeforeControllerAction). Middleware components get another hook before the action runs. This is where authentication middleware typically validates user credentials and attaches user information to the context. Authorization checks happen here as well.

Step 7: Action method executes. The matched action method is called with any route parameters, query string values, headers, or deserialized body content passed as arguments based on attribute declarations. Your business logic runs: services are called, data is retrieved or modified, and a result is produced.

If the action returns a value, it will be serialized automatically. If it is a procedure, you must call `Render` methods explicitly to produce a response. Any unhandled exceptions bubble up to the error handling layer.

Step 8: Middleware executes (OnAfterControllerAction). After the action completes but before the response is sent, middleware runs one final time. This is useful for response transformation, compression, adding security headers, or logging response details and timing information.

Step 9: Response is serialized and sent. If an object was returned from the action, the appropriate renderer converts it to JSON, XML, HTML, or another

format based on content negotiation rules. HTTP headers are set, including Content-Type, Content-Length, and any custom headers added by middleware or the action. The complete response is written to the output stream and sent back to the client.

Step 10: Controller is freed. The per-request controller instance is destroyed, releasing any resources it held. The thread returns to the thread pool awaiting the next request.

This pipeline architecture is powerful because each stage has a single responsibility and can be customized independently. Middleware adds cross-cutting behavior without modifying controllers. Routing decouples URLs from implementation details. Rendering separates data from presentation format. Together they create a flexible foundation for building applications of any complexity.

Summary

This chapter established the architectural foundation for working with DMVCFramework effectively. You learned how MVC principles are implemented in the framework, with controllers as request handlers created per-request, views as response renderers supporting multiple formats, and models as flexible data representations ranging from simple DTOs to ActiveRecord entities.

You traced the complete request lifecycle from HTTP server through middleware, routing, controller execution, and back out as a response. Understanding this flow is crucial because every feature we cover in subsequent chapters fits into this pipeline somewhere. Controllers coordinate, views present, models represent data, and middleware handles cross-cutting concerns.

The next chapter focuses on routing, the mechanism that connects URLs to controller actions. You will learn advanced routing patterns, RESTful URL design principles, parameter handling, route constraints, API versioning strategies, and how to structure your routes for clarity and maintainability.

Chapter 3: Routing and URL Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Basic Route Configuration and Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

The MVCPath Attribute

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Path Segments and Slashes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Multiple Routes for One Action

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Route Parameters and Dynamic Segments

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Capturing URL Segments

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Supported Parameter Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Optional Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Named Parameters vs Positional Binding

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

HTTP Method Routing and RESTful Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Restricting HTTP Methods

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

RESTful CRUD Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

When to Deviate from REST

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Advanced Routing: Constraints, Groups, and Prefixes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Route Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Route Groups and Prefixes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Wildcard Routes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

API Versioning Strategies with Routing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

URL Path Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Header-Based Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Content Negotiation Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Versioning Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 4: Controllers in Depth

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Controller Classes, Naming Conventions, and Registration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Naming Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Controller Registration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Controller Inheritance Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Action Methods: Signatures, Attributes, and Return Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Method Signature Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Parameter Attributes in Detail

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Return Type Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Request Data Access: Parameters, Headers, Body Parsing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Accessing Query Parameters Directly

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Reading Raw Request Body

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Handling File Uploads in Controllers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Filters: Before, After, and Around Execution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

OnBeforeControllerAction Filter

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

OnAfterControllerAction Filter

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

OnException Filter

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Thin Controllers: Delegating to Services

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Why Fat Controllers Are Bad

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

The Service Layer Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Service Interface Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

When Controllers Can Do More

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 5: Views and Response Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Response Types: JSON, XML, HTML, and Binary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

JSON as the Default

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

XML Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

HTML Template Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Binary and File Responses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Template Engines: Mustache, Handlebars, and Custom Templates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Mustache Templates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Sempare Templates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Custom Template Engines

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Content Negotiation and Accept Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

How Content Negotiation Works

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Configuring Content Negotiation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Handling 406 Not Acceptable

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Serialization and Deserialization with DMVCFramework

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

JSON Serialization Basics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Serialization Attributes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Custom Type Serialization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Deserialization of Request Bodies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Streaming Responses and Large Payloads

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Streaming Large Files

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Streaming JSON Arrays

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chunked Transfer Encoding

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 6: Models, Services, and Repositories

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Domain Models vs DTOs vs Entity Classes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Domain Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

DTOs (Data Transfer Objects)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Entity Classes (ActiveRecord)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

When to Use Each Type

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

The Repository Pattern: Abstraction Over Data Access

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Repository Interface Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Repository Implementation with FireDAC

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Generic Repository Base Class

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Service Layer Design and Business Logic Placement

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Service Responsibilities

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Service Interface and Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Single Responsibility in Services

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Dependency Injection with TInjector

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Why Dependency Injection Matters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Registering Services with the Container

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Injecting Dependencies into Controllers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Resolving Services Manually

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Building a Layered Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Recommended Project Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 7: Database Integration and ORM Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Connecting to Databases with FireDAC and Alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

FireDAC Connection Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Connection Pooling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Alternative Data Access Options

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Structuring Data Access: Repositories and Mappers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Using MVCActiveRecord for Simple CRUD

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Advanced Queries with MVCActiveRecord

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Transaction Management and Concurrency Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Basic Transaction Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Optimistic Concurrency Control

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Transaction Isolation Levels

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Building a Generic Repository Base Class

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Database Migration Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Versioned Migration Scripts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

ORM-Based Migrations with MVCActiveRecord

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Migration Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 8: Authentication, Authorization, and Security

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Authentication Mechanisms: Sessions, JWT, and API Keys

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Session-Based Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

JWT (JSON Web Token) Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

API Key Authentication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Implementing Role-Based Access Control (RBAC)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Defining Roles and Permissions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Authorization Middleware

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Attribute Definitions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Input Validation and Sanitization Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Using Validation Attributes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Manual Validation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Input Sanitization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

CORS Configuration and Cross-Origin Security

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Security Hardening: HTTPS, Headers, and Common Vulnerabilities

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

HTTPS/TLS Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Security Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Protecting Against Common Vulnerabilities

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 9: Middleware, Error Handling, and Logging

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

The Middleware Pipeline: Concept and Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

How the Pipeline Works

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Registering Middleware

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Building Custom Middleware Components

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Request Logging Middleware

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Request ID Middleware for Distributed Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Rate Limiting Middleware

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Centralized Error Handling and Exception Filters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Global Exception Handler

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Domain-Specific Exception Types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Controller-Level Exception Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Structured Logging with Multiple Providers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

LoggerPro Integration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Log Levels and When to Use Each

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Structured Logging for Machine Parsing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Health Checks and Readiness Probes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Liveness Probe

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Readiness Probe

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 10: File Operations, Uploads, and Downloads

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Handling File Uploads and Multipart Requests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Basic File Upload Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Accessing Upload Metadata

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Multiple File Uploads

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Upload with Form Fields

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Serving Static Files and Assets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Configuring Static File Middleware

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Multiple Static File Directories

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Security Considerations for Static Files

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Streaming File Downloads with Proper Headers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Basic File Download Endpoint

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Range Requests for Resumable Downloads

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Security Considerations for File Operations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Filename Sanitization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

File Type Validation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Storage Security

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Integrating Cloud Storage: S3 and Azure Blob

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

AWS S3 Integration Example

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Storage Abstraction Layer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 11: Background Processing and Async Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Thread Management in Delphi Web Applications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Request Thread Model

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Never Block Request Threads

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Using TThread for Background Work

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Using TTask for Modern Async Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Background Jobs and Task Queues

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

In-Memory Job Queue (Simple Applications)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Scheduled Tasks and Cron-Like Scheduling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Simple Interval-Based Scheduler

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Integrating Message Brokers: RabbitMQ and Redis

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

RabbitMQ Integration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Redis as a Job Queue

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Async/Await Patterns with Modern Delphi

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Using TTask for Chained Operations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Parallel Processing with TTask

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Important Threading Considerations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 12: Caching, Performance, and Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

In-Memory Caching Strategies and Implementations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Simple Thread-Safe Cache Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Cache Key Design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Cache Invalidation Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Distributed Caching with Redis

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Redis Cache Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Multi-Level Caching

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Database Query Optimization and Indexing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Identifying Slow Queries

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Common Query Anti-Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Indexing Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Pagination Optimization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Connection Pooling and Resource Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

FireDAC Connection Pool Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Resource Leak Prevention

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Profiling, Benchmarking, and Performance Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Application-Level Profiling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Load Testing with Realistic Traffic

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Optimization Priority

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 13: Testing DMVCFramework Applications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Unit Testing Controllers, Services, and Repositories

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Setting Up DUnitX

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Writing Effective Unit Tests

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Testing Controllers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Mocking Dependencies with TInjector

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Creating Mock Implementations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Using DI Container for Test Dependency Injection

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Integration Testing with Real Database Connections

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Test Database Setup

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Integration Test Example

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

End-to-End API Testing Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Starting the Server for Testing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Continuous Integration Setup for Delphi Projects

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

CI Pipeline Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

GitHub Actions Example

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Test Coverage Goals

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 14: Deployment and Production Operations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Build Configuration and Environment-Specific Settings

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Using DotEnv for Configuration Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Conditional Compilation for Build Variants

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Deploying to Linux, Windows Servers, and Containers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Console Application Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Windows Service Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Linux Systemd Service

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Docker and Kubernetes for DMVCFramework Applications

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Creating a Dockerfile

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Docker Compose for Local Development

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Kubernetes Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Monitoring, Alerting, and Observability in Production

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Application Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Log Aggregation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Alerting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Scaling Strategies: Horizontal, Vertical, and Microservices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Vertical Scaling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Horizontal Scaling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Microservice Decomposition

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Chapter 15: Advanced Patterns and Real-World Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Clean Architecture and Hexagonal Design in DMVCFramework

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

The Dependency Rule

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Implementing Hexagonal Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Event-Driven Patterns and Domain Events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Domain Events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Event Publisher and Subscribers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Microservice Communication and API Gateway

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Synchronous Communication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Asynchronous Communication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

API Gateway Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Building a Production-Grade Application: Complete Example

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Project Structure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Dependency Registration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Key Architectural Decisions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Framework Limitations and When to Extend or Replace

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Known Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

When DMVCFramework Is a Great Choice

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

When to Consider Alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Extending DMVCFramework

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.

References

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/dmvcframeworkmastery>.