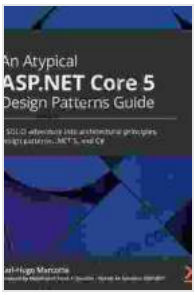
An Atypical ASP.NET Core Design Patterns Guide



An Atypical ASP.NET Core 6 Design Patterns Guide: A SOLID adventure into architectural principles and design patterns using .NET 6 and C# 10, 2nd Edition

by Carl-Hugo Marcotte

★★★★★ 5 out of 5

Language	: English
File size	: 15464 KB
Text-to-Speech	: Enabled
Screen Reader	: Supported
Enhanced typesetting	: Enabled
Print length	: 678 pages

ASP.NET Core is a popular open-source web framework for building modern, cloud-based applications. It provides a wide range of features and capabilities, including support for:

- Model-View-Controller (MVC) architecture
- Dependency injection
- Inversion of control
- Asynchronous programming
- Cross-platform development

In this article, we will explore some atypical design patterns that can be used to improve the design and maintainability of ASP.NET Core applications.

Layered Architectures

A layered architecture is a design pattern that divides an application into multiple layers, each with its own specific responsibilities. This helps to improve the modularity and maintainability of the application, as each layer can be developed and tested independently.

In a typical ASP.NET Core application, the following layers are commonly used:

- **Presentation layer:** This layer is responsible for handling user interaction and displaying data. It typically consists of controllers, views, and models.
- **Business logic layer:** This layer is responsible for implementing the business logic of the application. It typically consists of services and repositories.
- **Data access layer:** This layer is responsible for interacting with the database. It typically consists of data access objects (DAOs) and entity framework (EF) models.

By using a layered architecture, we can improve the separation of concerns in our application, making it easier to develop and maintain.

Dependency Injection

Dependency injection is a design pattern that allows us to create and pass dependencies to objects without having to create them ourselves. This helps to improve the testability and maintainability of our code, as we can easily swap out dependencies for testing or mocking purposes.

In ASP.NET Core, dependency injection is typically done using the built-in dependency injection container. We can register dependencies in the container using the `AddScoped`, `AddTransient`, or `AddSingleton` methods. For example:

csharp public void ConfigureServices(IServiceCollection services) {services.AddScoped(); }

Once we have registered our dependencies, we can inject them into our classes using the `[Inject]` attribute. For example:

csharp public class HomeController : Controller { [Inject] private IUserService _userService;

public IActionResult Index(){var users = _userService.GetAll(); return View(users); }}

By using dependency injection, we can improve the flexibility and testability of our code.

Inversion of Control

Inversion of control (IoC) is a design pattern that allows us to decouple the creation and management of objects from the code that uses them. This helps to improve the flexibility and maintainability of our code, as we can

easily change the way that objects are created and managed without having to modify the code that uses them.

In ASP.NET Core, IoC is typically achieved using a dependency injection container. By registering our dependencies in the container, we can allow the container to create and manage our objects for us. This gives us the flexibility to change the way that our objects are created and managed without having to modify the code that uses them.

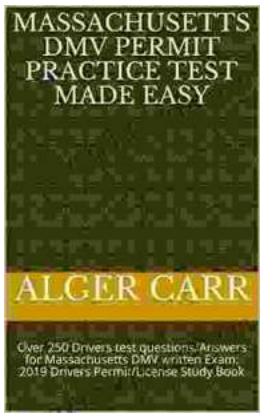
For example, we could use IoC to create a new instance of the `UserService` class every time it is needed, or we could use IoC to create a single instance of the `UserService` class that is shared across the entire application.

By using IoC, we can improve the flexibility and maintainability of our code.

In this article, we have explored some atypical design patterns that can be used to improve the design and maintainability of ASP.NET Core applications. These design patterns can help us to create applications that are more modular, testable, and flexible.

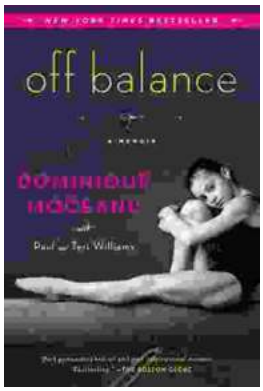
I encourage you to experiment with these design patterns in your own projects and see how they can help you to improve the quality of your code.

Text-to-Speech	: Enabled
Screen Reader	: Supported
Enhanced typesetting	: Enabled
Print length	: 678 pages



Ace Your Massachusetts DMV Written Exam: Over 250 Test Questions and Answers

Are you preparing to take the Massachusetts DMV written exam? If so, you're in luck! This article provides over 250 test questions and answers to help you...



Off Balance: Dominique Moceanu's Inspiring Memoir

A Heartfelt Account of a Champion's Journey and Advocacy In her gripping memoir, "Off Balance," former Olympic gymnast and vocal advocate...