

# ASP.NET Core MVC

## VB.NET完全ガイド

モダンWebアプリケーション開発のための  
包括的スライド教材

### 学習目標

- ✓ MVCパターンの理解と実装
- ✓ Entity Framework Core活用
- ✓ 認証・認可システム構築
- ✓ 本番環境デプロイメント

対象レベル: 初級〜中級開発者 (VB.NET基礎知識必須)

# 目次

学習フロー：概念理解 → 環境構築 → 実装 → デプロイ

- 1  ASP.NET Core MVC概要
- 2  開発環境セットアップ
- 3  プロジェクト構造とMVCパターン
- 4  コントローラーとアクション
- 5  Razorビューエンジン
- 6  モデルとデータバインディング
- 7  Entity Framework Core統合
- 8  認証・認可とセキュリティ
- 9  Web APIとサービス
- 10  デプロイメントと運用
- 11  よくある質問 (FAQ)
- 12  参考リンク・資料



推奨学習時間

詳細学習: 60-75分 / 概要把握: 30-40分



対象者

# ASP.NET Core MVC概要

## ★ フレームワークの特徴

- 🌐 クロスプラットフォーム対応  
Windows、Linux、macOSで動作
- 🦉 高性能・軽量  
最適化されたランタイムとメモリ効率
- 🔓 オープンソース  
透明性の高い開発とコミュニティ貢献
- 🔗 モジュラー設計  
必要な機能のみを組み込み可能

## </> VB.NETでの開発メリット

- ✓ 既存のVB.NET知識とコード資産を活用
- ✓ C#と同等の機能とパフォーマンス
- ✓ 自然言語に近い読みやすい構文

## 👥 MVCアーキテクチャパターン



**Model:** データとビジネスロジック  
**View:** ユーザーインターフェース  
**Controller:** リクエスト処理と制御

## 従来ASP.NETとの違い

### ✖ 従来ASP.NET

- Windows限定
- 重いランタイム
- Web.config設定

### ✓ ASP.NET Core

- クロスプラットフォーム
- 軽量高速
- JSON設定

# 開発環境セットアップ

## ☰ セットアップ手順

- 1 .NET SDK インストール  
最新版 .NET 8.0以降を推奨
- 2 Visual Studio 2022  
Community版で十分
- 3 SQL Server Express  
LocalDBまたはフル版
- 4 プロジェクトテンプレート  
ASP.NET Core Web アプリ (MVC)

### i 推奨構成

- RAM: 8GB以上
- ストレージ: 5GB以上の空き容量
- OS: Windows 10/11

## 📦 NuGetパッケージ管理

### 主要パッケージ:

- Microsoft.EntityFrameworkCore
- Microsoft.AspNetCore.Identity
- Microsoft.EntityFrameworkCore.SqlServer

# プロジェクト構造とMVCパターン

## ソリューション構造

```
MyWebApp/  
├── Controllers/ HomeController.vb  
├── Views/ Index.vbhtml  
├── Models/ Product.vb  
├── wwwroot/ css/, js/  
└── Program.vb
```

## Program.vbとStartup設定

主要設定:

✓ DI設定    ✓ ミドルウェア    ✓ ルーティング

## MVCフォルダの役割

### Controllers

- HTTPリクエスト処理
- ビジネスロジック制御
- ビューとモデルの橋渡し

### Views

- UI表示とレンダリング
- Razor構文でHTML生成
- レイアウト・部分ビュー

### Models

- データ構造定義
- ビジネスロジック
- バリデーションルール

### wwwroot

- 静的ファイル配置
- CSS、JavaScript、画像
- 直接アクセス可能

# コントローラーとアクション

## 🔧 VB.NETでのコントローラー実装

```
Public Class HomeController
    Inherits Controller

    Public Function Index() As IActionResult
        Return View()
    End Function

End Class
```

## ▶ アクションメソッドの定義

### 🕒 Viewアクション

ビューを返すアクション (Return View())

### </> JSONアクション

JSON形式でデータを返す (Return Json())

### ↪ Redirectアクション

別ページへリダイレクト (RedirectToAction)

## 🗺 ルーティングとURLマッピング

### 規約ベースルーティング

/Controller/Action/Id

### 🔗 HTTPメソッドとアトリビュート

GET <HttpGet>

POST <HttpPost>

PUT <HttpPut>

DELETE <HttpDelete>

## 🔗 パラメーター受け取り

主要なバインディング方法:

- ルートパラメーター (URL内)
- クエリストリング (?key=value)
- フォームデータ (POST)
- リクエストボディ (JSON)

# Razorビューエンジン

## </> Razor構文 (VB.NET版) の特徴

```
@Code
    Dim message As String = "Hello World"
    Dim count As Integer = 5
End Code

<h1>@message</h1>
<p>Count: @count</p>
```

### @Code...End Codeブロック

#### 変数定義

VB.NET構文でサーバーサイド変数を定義

#### 制御構造

If文、For文、While文等の制御フロー

#### Helper関数

再利用可能なRazorヘルパーメソッド

## レイアウトとセクション

### \_Layout.vbhtml

- 共通のHTMLテンプレート
- ヘッダー・フッター定義
- CSS・JS参照設定

#### セクション定義例:

```
@Section Scripts
<script>...</script>
End Section
```

## 部分ビューとコンポーネント

### 部分ビュー

@Html.Partial("\_UserInfo")

### ビューコンポーネント

@await Component.InvokeAsync("Menu")

### 強く型付けされたビュー

@model Product

# モデルとデータバインディング

## ViewModelパターンの実装

```
Public Class ProductViewModel
    Public Property Id As Integer
    Public Property Name As String
    Public Property Price As Decimal
End Class
```

## モデルバインディングの仕組み

### ↓ 自動バインディング

フォームデータを自動的にオブジェクトにマッピング

### ✏ 型変換

文字列から適切なデータ型に自動変換

### 🛡 バリデーション

データアノテーションによる入力検証

## ✔ データアノテーションによる検証

```
<Required>
Public Property Name As String

<Range(1, 100)>
Public Property Age As Integer

<EmailAddress>
Public Property Email As String
```

## ⚙ カスタムバリデーション

### ValidationAttribute継承

独自のバリデーション属性作成

### IValidatableObject実装

クラスレベルでの複雑な検証

### ModelState確認

コントローラーでの検証結果チェック

### 💡 ベストプラクティス

- ViewModelとEntityを分離
- 適切なバリデーション属性使用
- セキュリティを考慮した設計



# Entity Framework Core統合

## DbContextのVB.NET実装

```
Public Class ApplicationDbContext
    Inherits DbContext

    Public Property Products As DbSet(Of Product)

    Protected Overrides Sub OnModelCreating(
        modelBuilder As ModelBuilder)
        ' 設定コード
    End Sub
End Class
```

## Code Firstアプローチ

### モデル定義

VB.NETクラスでエンティティ定義

### マイグレーション

データベーススキーマの自動生成

### 設定

Fluent APIによる詳細設定

## LINQ to Entitiesクエリ

基本的なクエリパターン:

- ≡ 全件取得: context.Products.ToList()
- ▼ 条件絞込: Where(Function(p) p.Price > 100)
- ◆ 並び替え: OrderBy(Function(p) p.Name)

## マイグレーション管理

主要コマンド:

```
Add-Migration InitialCreate
```

```
Update-Database
```

```
Remove-Migration
```

### ⚠ 注意点

- 非同期メソッド推奨 (ToListAsync)
- N+1問題対策 (Include使用)
- 接続文字列のセキュリティ

# 認証・認可とセキュリティ

## ASP.NET Core Identity統合

### 👤 ユーザー管理

登録、ログイン、プロフィール管理

### 🔑 パスワード管理

ハッシュ化、複雑性チェック、リセット

### 👥 ロール管理

権限グループ、階層構造

## ⚙️ 認証ミドルウェアの設定

```
builder.Services.AddIdentity(Of IdentityUser, IdentityRole>()  
    .AddEntityFrameworkStores(Of ApplicationDbContext)()
```

## 🔒 ロールベースアクセス制御

認可属性の使用:

```
<Authorize>  
<Authorize(Roles:="Admin")>  
<Authorize(Policy:="CanEdit")>
```

## ⚙️ セキュリティベストプラクティス

### CSRF対策

アンチフォージェリトークン使用

### XSS対策

入力値エスケープ処理

### HTTPS強制

SSL/TLS暗号化通信

### セキュアヘッダー

Content Security Policy等

# Web APIとサービス

## RESTful API設計原則

GET /api/products → 商品一覧取得  
POST /api/products → 新規商品作成  
PUT /api/products/1 → 商品更新  
DELETE /api/products/1 → 商品削除

## JSONシリアライゼーション

```
<ApiController>
Public Class ProductsController
Function GetProducts() As ActionResult(Of List(Of Product))
Return Ok(products)
End Function
End Class
```

## 依存性注入 (DI)

### サービス登録:

```
builder.Services.AddScoped(Of IProductService,
ProductService)()
builder.Services.AddTransient(Of IEmailService,
EmailService)()
```

## サービスレイヤーの実装

ビジネスロジック分離  
コントローラーからロジックを分離

再利用性向上  
複数コントローラーで共通利用

テストビリティ  
モックを使った単体テスト

### サービスライフタイム

- Singleton: アプリ全体
- Scoped: リクエスト毎
- Transient: 呼出毎に新インスタンス

# デプロイメントと運用

## IISへのデプロイ手順

- 1 アプリケーションパブリッシュ  
dotnet publish -c Release
- 2 IISサイト作成  
アプリケーションプールとサイト設定
- 3 ファイル配置  
パブリッシュされたファイルをサーバーに配置
- 4 設定とテスト  
接続文字列等の本番設定適用

## Azure App Serviceへの展開

### Visual Studioから直接デプロイ

- 発行プロファイル作成
- Azure App Service選択
- 自動デプロイ設定

## 環境別設定管理

### 設定ファイル階層:

- appsettings.json (基本設定)
- appsettings.Development.json
- appsettings.Production.json

## ロギングと監視

### Application Insights

パフォーマンス監視・例外追跡

### 構造化ログ

Serilog等を使った高度なログ出力

### ヘルスチェック

アプリケーション状態監視

### ⚠ 重要な確認事項

- セキュリティ設定 (HTTPS強制)
- データベース移行の実行
- バックアップ・復旧計画

# よくある質問 (FAQ)

## </> VB.NET特有の注意点

### ❓ Q: Razorビューでのコードブロック

A: C#の{}ではなく、@Code...End Code、@If...End Ifを使用

### ❓ Q: 非同期プログラミング

A: Async Function、Await構文を使用。Task(Of T)を返す

### ❓ Q: ラムダ式の書き方

A: Function(x) x.Name、Sub(x) Console.WriteLine(x)

## ⇄ C#からの移行ガイド

```
C#
{ }
var
=>
VB.NET
Begin...End
Dim
Function()
```

## 🚀 パフォーマンス最適化

### 🗄️ EF Core最適化

- AsNoTracking()で読み取り専用クエリ
- Include()でN+1問題回避
- 非同期メソッド使用

### 🏠 メモリ効率化

- Using文でリソース管理
- 大量データの分割処理
- キャッシュ戦略の活用

### ⚡ レスポンス向上

- 静的ファイルの圧縮
- CDN活用
- バンドル最適化

## 🔧 トラブルシューティング

### よくあるエラー:

- 依存性注入の循環参照
- Migrationの競合
- ViewModelのnull参照
- セッション状態の問題

# 参考リンク・資料

## Microsoft Learn 学習パス



ASP.NET Core公式ドキュメント

[docs.microsoft.com/aspnet/core](https://docs.microsoft.com/aspnet/core)



VB.NET言語リファレンス

[docs.microsoft.com/dotnet/visual-basic](https://docs.microsoft.com/dotnet/visual-basic)



Entity Framework Core

[docs.microsoft.com/ef/core](https://docs.microsoft.com/ef/core)




Microsoft Learn

[learn.microsoft.com](https://learn.microsoft.com) ( 無料学習プラットフォーム )

## VB.NET開発者コミュニティ

 Stack Overflow: 技術的質問と回答

 GitHub: オープンソースプロジェクト

## サンプルプロジェクト

実践的サンプル:


- 商品管理システム・ユーザー認証
- RESTful API・Azure App Service配置

## → 次のステップ

### 応用学習推奨

- Blazor WebAssembly・ASP.NET Core Web API
- マイクロサービス設計・Docker コンテナ化

🌟 資格: Microsoft Certified Azure Developer

 学習完了おめでとうございます！

実際のプロジェクトで学んだ知識を活かしましょう