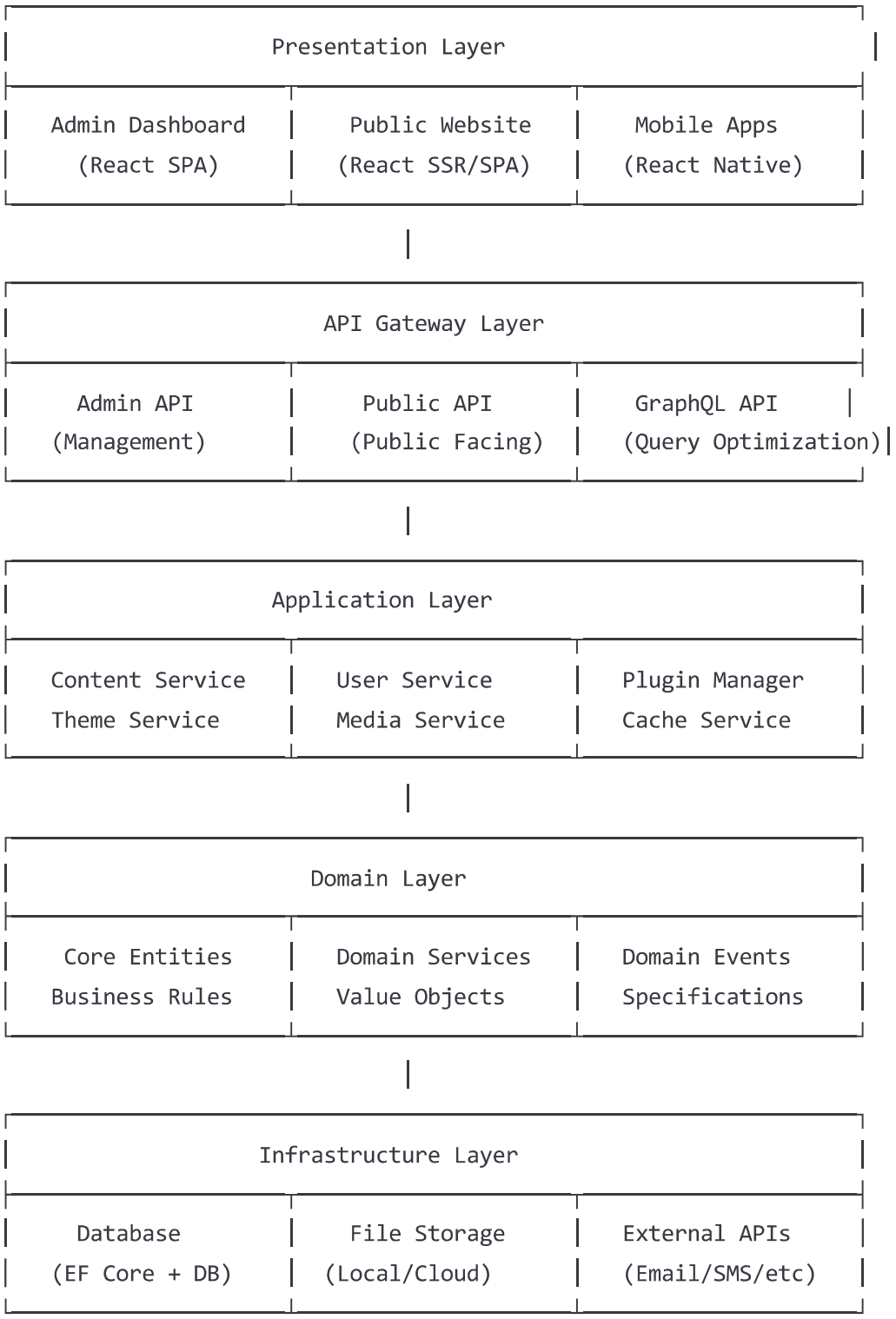
# WordPress.NET - System Architecture Design

## 📋 Overall Architecture Overview




## 🧬 Detailed Layered Architecture

### 1. Presentation Layer

#### 1.1 Admin Dashboard

```
WordPress.Admin/
├── src/
│   ├── components/          # React Components
│   │   ├── Dashboard/       # Dashboard Components
│   │   ├── Posts/           # Post Management
│   │   ├── Pages/           # Page Management
│   │   ├── Media/           # Media Library
│   │   ├── Users/           # User Management
│   │   ├── Themes/          # Theme Management
│   │   ├── Plugins/         # Plugin Management
│   │   └── Settings/        # System Settings
│   ├── hooks/               # Custom Hooks
│   ├── services/            # API Services
│   ├── store/               # State Management (Redux/Zustand)
│   ├── utils/               # Utility Functions
│   └── types/               # TypeScript Type Definitions
├── public/
└── package.json
```

### 1.2 Public Website

```
WordPress.Web/
├── src/
│   ├── components/          # Shared Components
│   │   ├── Layout/          # Layout Components
│   │   ├── Post/            # Post Related
│   │   ├── Comment/         # Comment System
│   │   ├── Widget/          # Widgets
│   │   └── Theme/           # Theme Components
│   ├── pages/               # Page Components
│   ├── hooks/               # Custom Hooks
│   ├── services/            # API Services
│   ├── store/               # State Management
│   └── themes/              # Theme Directory
├── public/
└── next.config.js           # Next.js Configuration
```

## 2. API Layer

### 2.1 Main API Project Structure

```
WordPress.API/
├── Controllers/
│   ├── Admin/              # Admin Controllers
│   │   ├── PostsController.cs
│   │   ├── UsersController.cs
│   │   ├── ThemesController.cs
│   │   └── PluginsController.cs
│   └── Public/             # Public Controllers
│       ├── BlogController.cs
│       ├── PageController.cs
│       └── CommentController.cs
├── Middleware/             # Middleware
│   ├── AuthenticationMiddleware.cs
│   ├── CachingMiddleware.cs
│   └── ThemeMiddleware.cs
├── Filters/                # Filters
├── Hubs/                   # SignalR Hubs
└── Program.cs
```

## 3. Application Layer

```
WordPress.Application/
├── Services/               # Application Services
│   ├── ContentService.cs
│   ├── UserService.cs
│   ├── ThemeService.cs
│   ├── PluginService.cs
│   └── MediaService.cs
├── DTOs/                   # Data Transfer Objects
├── Commands/               # CQRS Commands
├── Queries/                # CQRS Queries
├── Handlers/               # Command/Query Handlers
├── Interfaces/             # Interface Definitions
├── Validators/             # Data Validation
└── Mappers/                # Object Mapping
```

## 4. Domain Layer

```
WordPress.Domain/
├── Entities/               # Entity Classes
│   ├── User.cs
│   ├── Post.cs
│   ├── Category.cs
│   ├── Comment.cs
│   └── Media.cs
├── ValueObjects/           # Value Objects
├── Enums/                  # Enumerations
├── Events/                 # Domain Events
├── Services/               # Domain Services
├── Specifications/         # Specification Pattern
└── Interfaces/             # Repository Interfaces
```

### 5. Infrastructure Layer

```
WordPress.Infrastructure/
├── Data/                   # Data Access
│   ├── Context/
│   │   └── WordPressDbContext.cs
│   ├── Repositories/       # Repository Implementations
│   ├── Configurations/     # EF Configurations
│   └── Migrations/         # Database Migrations
├── Services/               # Infrastructure Services
│   ├── EmailService.cs
│   ├── FileStorageService.cs
│   └── CacheService.cs
├── Extensions/             # Plugin Extensions
└── External/               # External Service Integration
```

## 🔌 Plugin System Architecture

### Plugin Interface Design

csharp

```csharp
public interface IPlugin
{
    string Name { get; }
    string Version { get; }
    string Description { get; }

    Task Initialize(IServiceProvider serviceProvider);
    Task<PluginResult> Execute(PluginContext context);
    Task Shutdown();
}

public interface ITheme
{
    string Name { get; }
    string Version { get; }
    ThemeManifest Manifest { get; }

    string RenderTemplate(string templateName, object model);
    IEnumerable<string> GetAssets();
}
```

### Plugin Manager

csharp

```csharp
public class PluginManager
{
    private readonly Dictionary<string, IPlugin> _loadedPlugins;
    private readonly IServiceProvider _serviceProvider;

    public async Task LoadPlugin(string pluginPath);
    public async Task UnloadPlugin(string pluginName);
    public async Task<T> ExecuteHook<T>(string hookName, T data);
}
```

## 🎨 Theme System Architecture

### Theme Structure

```
themes/
├── default/
│   ├── templates/           # Template Files
│   │   ├── layout.html
│   │   ├── post.html
│   │   ├── page.html
│   │   └── archive.html
│   ├── assets/              # Static Assets
│   │   ├── css/
│   │   ├── js/
│   │   └── images/
│   ├── components/          # React Components
│   ├── theme.json           # Theme Configuration
│   └── functions.js         # Theme Functions
└── custom-theme/
    └── ...
```

### Theme Configuration Example

json

```
{
  "name": "Default Theme",
  "version": "1.0.0",
  "description": "WordPress.NET Default Theme",
  "author": "WordPress.NET Team",
  "templates": {
    "home": "home.html",
    "post": "post.html",
    "page": "page.html",
    "archive": "archive.html"
  },
  "customizer": {
    "colors": {
      "primary": "#0073aa",
      "secondary": "#23282d"
    },
    "fonts": {
      "body": "Arial, sans-serif",
      "heading": "Georgia, serif"
    }
  }
}
```

## 📊 Database Design

### Core Table Structure

```sql
-- Users Table
Users (Id, Username, Email, PasswordHash, Role, CreatedAt, UpdatedAt)

-- Posts Table
Posts (Id, Title, Slug, Content, AuthorId, Status, Type, CreatedAt, PublishedAt)

-- Categories Table
Categories (Id, Name, Slug, Description, ParentId)

-- Tags Table
Tags (Id, Name, Slug, Description)

-- Comments Table
Comments (Id, PostId, AuthorId, Content, Status, CreatedAt, ParentId)

-- Media Table
Media (Id, FileName, FilePath, MimeType, Size, Alt, UserId, UploadedAt)

-- Options Table
Options (Id, Name, Value, Autoload)

-- Metadata Tables
PostMeta (Id, PostId, MetaKey, MetaValue)
UserMeta (Id, UserId, MetaKey, MetaValue)
```

## 🚀 Technology Stack

### Backend Technology Stack

- **Framework**: ASP.NET Core 8.0
- **ORM**: Entity Framework Core 8.0
- **Database**: PostgreSQL/MySQL/SQL Server
- **Caching**: Redis + MemoryCache
- **Authentication**: JWT + Identity
- **API**: RESTful + GraphQL
- **Real-time**: SignalR
- **Background Jobs**: Hangfire
- **Logging**: Serilog
- **Testing**: xUnit + Moq

### Frontend Technology Stack

- **Framework**: React 18 + TypeScript
- **Routing**: React Router v6
- **State Management**: Redux Toolkit / Zustand
- **UI Library**: Ant Design / Material-UI
- **Build Tools**: Vite / Next.js
- **Styling**: Tailwind CSS + Styled Components
- **Editor**: TinyMCE / Monaco Editor
- **Testing**: Jest + React Testing Library

### DevOps & Deployment

- **Containerization**: Docker + Kubernetes
- **CI/CD**: GitHub Actions / Azure DevOps
- **Monitoring**: Application Insights / Prometheus
- **CDN**: Azure CDN / CloudFlare
- **Storage**: Azure Blob / AWS S3

## 🔒 Security Considerations

### Security Measures

1. **Authentication**: JWT Token + Refresh Token
2. **Authorization**: Role-Based Access Control (RBAC)
3. **Data Validation**: Input Validation + XSS Protection
4. **SQL Injection Protection**: Parameterized Queries + ORM
5. **HTTPS**: Enforced SSL/TLS
6. **CORS**: Cross-Origin Resource Sharing Configuration
7. **CSP**: Content Security Policy
8. **Audit Logging**: Operation Log Recording

## 📈 Performance Optimization

### Caching Strategy

1. **Memory Cache**: Hot Data Caching
2. **Distributed Cache**: Redis Cluster
3. **HTTP Cache**: ETags + Last-Modified

4. **CDN Cache**: Static Asset Distribution
5. **Database Cache**: Query Result Caching

### Performance Monitoring

1. **Application Performance Monitoring**: APM Tools
2. **Database Performance**: Slow Query Monitoring
3. **Server Monitoring**: CPU/Memory/Disk
4. **User Experience**: Core Web Vitals

## 🔄 Scalability Design

### Horizontal Scaling

- **Load Balancing**: Nginx/HAProxy
- **Database Separation**: Read/Write Splitting
- **Microservices Architecture**: Independent Module Deployment
- **Message Queue**: RabbitMQ/Azure Service Bus

### Plugin Extensions

- **Hook System**: Event-Driven Architecture
- **Dependency Injection**: Plugin Service Registration
- **Dynamic Loading**: Runtime Plugin Management
- **API Extensions**: Plugin Custom Endpoints

## 📦 Project Structure

### Solution Structure

```
WordPress.NET/
├── src/
│   ├── WordPress.Domain/              # Domain Layer
│   ├── WordPress.Application/         # Application Layer
│   ├── WordPress.Infrastructure/      # Infrastructure Layer
│   ├── WordPress.API/                 # API Layer
│   ├── WordPress.Admin/               # Admin Dashboard
│   └── WordPress.Web/                 # Public Website
├── plugins/                           # Plugins Directory
│   ├── sample-plugin/
│   └── seo-plugin/
├── themes/                           # Themes Directory
│   ├── default/
│   └── custom-theme/
├── tests/                           # Test Projects
│   ├── WordPress.Domain.Tests/
│   ├── WordPress.Application.Tests/
│   └── WordPress.API.Tests/
├── docker/                           # Docker Configuration
├── scripts/                          # Build Scripts
└── docs/                            # Documentation
```

## 🎯 Key Features

### Content Management

- **Rich Text Editor**: Advanced WYSIWYG editor
- **Media Library**: File upload and management
- **SEO Optimization**: Built-in SEO tools
- **Multilingual Support**: Internationalization
- **Revision History**: Content version control

### User Management

- **Role-Based Permissions**: Flexible role system
- **User Profiles**: Extended user information
- **Social Login**: OAuth integration
- **Two-Factor Authentication**: Enhanced security

### Customization

- **Plugin System**: Extensible plugin architecture
- **Theme Engine**: Customizable themes

- **Widget System**: Draggable widgets
- **Custom Fields**: Meta data management
- **Hook System**: Event-driven customization

### Performance & SEO

- **Caching Layers**: Multi-level caching
- **CDN Integration**: Static asset optimization
- **SEO-Friendly URLs**: Clean URL structure
- **Schema Markup**: Structured data support
- **Performance Monitoring**: Real-time metrics

This architecture provides a modern, scalable, and high-performance WordPress alternative built entirely on the .NET technology stack, supporting plugin development and theme customization with enterprise-grade features.