ASP.NET Core CRUD with ViewModel - Complete Documentation

Table of Contents

Initial Setup

1. Install Required NuGet Packages

Tools → NuGet Package Manager → Manage NuGet Packages for Solution

Install the following packages:

- `Microsoft.EntityFrameworkCore.Tools`
- `Microsoft.EntityFrameworkCore`
- `Microsoft.EntityFrameworkCore.SqlServer`

Models & ViewModels

2. Create Models

Location: `ProjectName → Models → Item.cs`

```
namespace CRUD.Models
{
    public class Item
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
    }
}
```

3. Create ViewModels

Location: `ProjectName → ViewModels → ItemViewModel.cs`

```
using System.ComponentModel.DataAnnotations;

namespace CRUD.ViewModels
{
    public class ItemViewModel
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Name is required")]
        [StringLength(100, ErrorMessage = "Name cannot exceed 100
characters")]
        public string Name { get; set; }
```

```
        [Required(ErrorMessage = "Description is required")]
        [StringLength(500, ErrorMessage = "Description cannot exceed 500
characters")]
        public string Description { get; set; }

        [Required(ErrorMessage = "Price is required")]
        [Range(0.01, 999999.99, ErrorMessage = "Price must be between 0.01
and 999999.99")]
        public decimal Price { get; set; }
    }
}
```

Why Use ViewModels?

- Separation of concerns (presentation logic vs. data model)
- Add validation attributes without modifying database models
- Control what data is exposed to views
- Combine data from multiple models if needed

Database Configuration

4. Create Connection String

Location: `ProjectName → appsettings.json`

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Server=.;Database=CRUDDb;User
Id=sa;Password=1qa2ws3ed!!!;Encrypt=False;"
  }
}
```

5. Create DbContext

Location: `ProjectName → Data → CRUDDbContext.cs`

```
using Microsoft.EntityFrameworkCore;
using CRUD.Models;

namespace CRUD.Data
{
    public class CRUDDbContext : DbContext
    {
        public CRUDDbContext(DbContextOptions<CRUDDbContext> options) :
base(options)
        {
        }

        public DbSet<Item> Items { get; set; }
    }
}
```

6. Register DbContext in Program.cs

Location: `Program.cs`

```
using CRUD.Data;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);
```

```
// Add services to the container
builder.Services.AddControllersWithViews();

// Register DbContext
builder.Services.AddDbContext<CRUDDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnec
tion")));

var app = builder.Build();

// Configure the HTTP request pipeline
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

7. Create and Apply Migration

Open Package Manager Console: Press `Ctrl + ~` or go to **Tools → NuGet Package Manager → Package Manager Console**

```
# Create Migration
Add-Migration InitialCreate

# Apply Migration (Creates Database)
Update-Database
```

Controllers

8. Create Items Controller with ViewModel Support

Location: `ProjectName → Controllers → ItemsController.cs`

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using CRUD.Data;
using CRUD.Models;
using CRUD.ViewModels;

namespace CRUD.Controllers
{
    public class ItemsController : Controller
    {
        private readonly CRUDDbContext _context;

        public ItemsController(CRUDDbContext context)
        {
            _context = context;
        }

        // GET: Items
        public async Task<IActionResult> Index()
        {
            var items = await _context.Items.ToListAsync();

            // Map Model to ViewModel
            var itemViewModels = items.Select(item => new ItemViewModel
```

```
            {
                Id = item.Id,
                Name = item.Name,
                Description = item.Description,
                Price = item.Price
            }).ToList();

            return View(itemViewModels);
        }

        // GET: Items/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var item = await _context.Items.FirstOrDefaultAsync(m => m.Id ==
id);

            if (item == null)
            {
                return NotFound();
            }

            // Map Model to ViewModel
            var itemViewModel = new ItemViewModel
            {
                Id = item.Id,
                Name = item.Name,
                Description = item.Description,
                Price = item.Price
            };

            return View(itemViewModel);
        }

        // GET: Items/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: Items/Create
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(ItemViewModel viewModel)
        {
            if (ModelState.IsValid)
            {
                // Map ViewModel to Model
                var item = new Item
                {
                    Name = viewModel.Name,
                    Description = viewModel.Description,
                    Price = viewModel.Price
                };

                _context.Add(item);
                await _context.SaveChangesAsync();

                TempData["SuccessMessage"] = "Item created successfully!";
                return RedirectToAction(nameof(Index));
            }

            return View(viewModel);
        }

        // GET: Items/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
```

```
            var item = await _context.Items.FindAsync(id);

            if (item == null)
            {
                return NotFound();
            }

            // Map Model to ViewModel
            var itemViewModel = new ItemViewModel
            {
                Id = item.Id,
                Name = item.Name,
                Description = item.Description,
                Price = item.Price
            };

            return View(itemViewModel);
        }

        // POST: Items/Edit/5
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, ItemViewModel
viewModel)
        {
            if (id != viewModel.Id)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    // Map ViewModel to Model
                    var item = await _context.Items.FindAsync(id);

                    if (item == null)
                    {
                        return NotFound();
                    }

                    item.Name = viewModel.Name;
                    item.Description = viewModel.Description;
                    item.Price = viewModel.Price;

                    _context.Update(item);
                    await _context.SaveChangesAsync();

                    TempData["SuccessMessage"] = "Item updated
successfully!";
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!ItemExists(viewModel.Id))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }

                return RedirectToAction(nameof(Index));
            }

            return View(viewModel);
        }

        // GET: Items/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
```

```
                return NotFound();
            }

            var item = await _context.Items.FirstOrDefaultAsync(m => m.Id ==
id);

            if (item == null)
            {
                return NotFound();
            }

            // Map Model to ViewModel
            var itemViewModel = new ItemViewModel
            {
                Id = item.Id,
                Name = item.Name,
                Description = item.Description,
                Price = item.Price
            };

            return View(itemViewModel);
        }

        // POST: Items/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            var item = await _context.Items.FindAsync(id);

            if (item != null)
            {
                _context.Items.Remove(item);
                await _context.SaveChangesAsync();

                TempData["SuccessMessage"] = "Item deleted successfully!";
            }

            return RedirectToAction(nameof(Index));
        }

        private bool ItemExists(int id)
        {
            return _context.Items.Any(e => e.Id == id);
        }
    }
}
```

Views

9. Create Views for CRUD Operations

Index View (List)

Location: `Views → Items → Index.cshtml`

```
@model IEnumerable<CRUD.ViewModels.ItemViewModel>

@{
    ViewData["Title"] = "Items List";
}

<div class="container mt-4">
    <h2>Items List</h2>

    @if (TempData["SuccessMessage"] != null)
    {
        <div class="alert alert-success alert-dismissible fade show"
role="alert">
            @TempData["SuccessMessage"]
            <button type="button" class="btn-close" data-bs-dismiss="alert"
aria-label="Close"></button>
```

```
        </div>
    }

    <p>
        <a asp-action="Create" class="btn btn-primary">Create New Item</a>
    </p>

    <table class="table table-striped table-hover">
        <thead class="table-dark">
            <tr>
                <th>@Html.DisplayNameFor(model => model.Name)</th>
                <th>@Html.DisplayNameFor(model => model.Description)</th>
                <th>@Html.DisplayNameFor(model => model.Price)</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@Html.DisplayFor(modelItem => item.Name)</td>
                    <td>@Html.DisplayFor(modelItem => item.Description)</td>
                    <td>@string.Format("{0:C}", item.Price)</td>
                    <td>
                        <a asp-action="Edit" asp-route-id="@item.Id"
class="btn btn-sm btn-warning">Edit</a>
                        <a asp-action="Details" asp-route-id="@item.Id"
class="btn btn-sm btn-info">Details</a>
                        <a asp-action="Delete" asp-route-id="@item.Id"
class="btn btn-sm btn-danger">Delete</a>
                    </td>
                </tr>
            }
        </tbody>
    </table>
</div>
```

Create View

Location: `Views → Items → Create.cshtml`

```
@model CRUD.ViewModels.ItemViewModel

@{
    ViewData["Title"] = "Create Item";
}

<div class="container mt-4">
    <h2>Create New Item</h2>
    <hr />

    <div class="row">
        <div class="col-md-6">
            <form asp-action="Create" method="post">
                <div asp-validation-summary="ModelOnly" class="text-
danger"></div>

                <div class="mb-3">
                    <label asp-for="Name" class="form-label"></label>
                    <input asp-for="Name" class="form-control" />
                    <span asp-validation-for="Name" class="text-
danger"></span>
                </div>

                <div class="mb-3">
                    <label asp-for="Description" class="form-label"></label>
                    <textarea asp-for="Description" class="form-control"
rows="4"></textarea>
                    <span asp-validation-for="Description" class="text-
danger"></span>
                </div>

                <div class="mb-3">
                    <label asp-for="Price" class="form-label"></label>
```

```
                    <input asp-for="Price" class="form-control" type="number"
step="0.01" />
                    <span asp-validation-for="Price" class="text-
danger"></span>
                </div>

                <div class="mb-3">
                    <button type="submit" class="btn btn-
primary">Create</button>
                    <a asp-action="Index" class="btn btn-secondary">Back to
List</a>
                </div>
            </form>
        </div>
    </div>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Edit View

Location: `Views → Items → Edit.cshtml`

```
@model CRUD.ViewModels.ItemViewModel

@{
    ViewData["Title"] = "Edit Item";
}

<div class="container mt-4">
    <h2>Edit Item</h2>
    <hr />

    <div class="row">
        <div class="col-md-6">
            <form asp-action="Edit" method="post">
                <div asp-validation-summary="ModelOnly" class="text-
danger"></div>

                <input type="hidden" asp-for="Id" />

                <div class="mb-3">
                    <label asp-for="Name" class="form-label"></label>
                    <input asp-for="Name" class="form-control" />
                    <span asp-validation-for="Name" class="text-
danger"></span>
                </div>

                <div class="mb-3">
                    <label asp-for="Description" class="form-label"></label>
                    <textarea asp-for="Description" class="form-control"
rows="4"></textarea>
                    <span asp-validation-for="Description" class="text-
danger"></span>
                </div>

                <div class="mb-3">
                    <label asp-for="Price" class="form-label"></label>
                    <input asp-for="Price" class="form-control" type="number"
step="0.01" />
                    <span asp-validation-for="Price" class="text-
danger"></span>
                </div>

                <div class="mb-3">
                    <button type="submit" class="btn btn-primary">Save
Changes</button>
                    <a asp-action="Index" class="btn btn-secondary">Back to
List</a>
                </div>
            </form>
        </div>
    </div>
```

```
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Details View

Location: `Views → Items → Details.cshtml`

```
@model CRUD.ViewModels.ItemViewModel

@{
    ViewData["Title"] = "Item Details";
}

<div class="container mt-4">
    <h2>Item Details</h2>
    <hr />

    <div class="card">
        <div class="card-body">
            <dl class="row">
                <dt class="col-sm-3">@Html.DisplayNameFor(model =>
model.Name)</dt>
                <dd class="col-sm-9">@Html.DisplayFor(model =>
model.Name)</dd>

                <dt class="col-sm-3">@Html.DisplayNameFor(model =>
model.Description)</dt>
                <dd class="col-sm-9">@Html.DisplayFor(model =>
model.Description)</dd>

                <dt class="col-sm-3">@Html.DisplayNameFor(model =>
model.Price)</dt>
                <dd class="col-sm-9">@string.Format("{0:C}",
Model.Price)</dd>
            </dl>
        </div>
    </div>

    <div class="mt-3">
        <a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-
warning">Edit</a>
        <a asp-action="Index" class="btn btn-secondary">Back to List</a>
    </div>
</div>
```

Delete View

Location: `Views → Items → Delete.cshtml`

```
@model CRUD.ViewModels.ItemViewModel

@{
    ViewData["Title"] = "Delete Item";
}

<div class="container mt-4">
    <h2>Delete Item</h2>
    <hr />

    <div class="alert alert-danger">
        <h4>Are you sure you want to delete this item?</h4>
    </div>

    <div class="card">
        <div class="card-body">
            <dl class="row">
                <dt class="col-sm-3">@Html.DisplayNameFor(model =>
model.Name)</dt>
                <dd class="col-sm-9">@Html.DisplayFor(model =>
model.Name)</dd>
```

```
                <dt class="col-sm-3">@Html.DisplayNameFor(model =>
model.Description)</dt>
                <dd class="col-sm-9">@Html.DisplayFor(model =>
model.Description)</dd>

                <dt class="col-sm-3">@Html.DisplayNameFor(model =>
model.Price)</dt>
                <dd class="col-sm-9">@string.Format("{0:C}",
Model.Price)</dd>
            </dl>
        </div>
    </div>

    <form asp-action="Delete" method="post" class="mt-3">
        <input type="hidden" asp-for="Id" />
        <button type="submit" class="btn btn-danger">Delete</button>
        <a asp-action="Index" class="btn btn-secondary">Cancel</a>
    </form>
</div>
```

Complete Implementation

Project Structure

```
CRUD/
├── Controllers/
│   └── ItemsController.cs
├── Data/
│   └── CRUDDbContext.cs
├── Migrations/
│   └── (Generated files)
├── Models/
│   └── Item.cs
├── ViewModels/
│   └── ItemViewModel.cs
├── Views/
│   └── Items/
│       ├── Index.cshtml
│       ├── Create.cshtml
│       ├── Edit.cshtml
│       ├── Details.cshtml
│       └── Delete.cshtml
├── appsettings.json
└── Program.cs
```

Key Benefits of Using ViewModels

1. **Validation:** Add validation rules without modifying database models
2. **Security:** Control what data is exposed to views
3. **Flexibility:** Combine multiple models into one view
4. **Maintainability:** Separate presentation logic from business logic
5. **Data Transformation:** Format data specifically for display

Common Commands

```
# Add new migration
Add-Migration MigrationName

# Update database
Update-Database

# Remove last migration
Remove-Migration

# Revert to specific migration
Update-Database -Migration MigrationName

# Generate SQL script
Script-Migration
```

Testing Your Application

1. Run the application (F5 or Ctrl+F5)
2. Navigate to `/Items` in your browser
3. Test all CRUD operations:
    - Create new items
    - View item list
    - Edit existing items
    - View item details
    - Delete items

Additional Resources

- Entity Framework Core Documentation
- ASP.NET Core MVC Documentation
- Data Annotations

Documentation Version: 1.0
Last Updated: November 2025

GIT Repository: https://github.com/Jandelocks/CRUD.git