ASP.NET MVC Guide

This guide provides an overview of key ASP.NET MVC concepts, including Code First and Database First approaches, view navigation, data passing, LINQ expressions, model relationships, configurations, and common problem-solving strategies.

1. Code First vs. Database First Approaches

Code First Approach

- **Description**: Start with C# classes that represent your domain model.
- **Steps**:
  1. Define classes that represent tables.
  2. Use `DbContext` to manage the database.
  3. Use `Migrations` to create and update the database schema.

Example

```
public class Product
{
    public int ProductId { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
}

public class AppDbContext : DbContext
{
    public DbSet<Product> Products { get; set; }
}
```

- **Commands**:
  - `Add-Migration InitialCreate`
  - `Update-Database`

Database First Approach

- **Description**: Start with an existing database and generate the model classes.
- **Steps**:
  1. Right-click on the project, select `Add > New Item > ADO.NET Entity Data Model`.
  2. Choose `EF Designer from Database` and follow the wizard.

2. Navigation Between Views

HTML Link-Based Navigation

```
<a asp-controller="Home" asp-action="About">Go to About</a>
```

Action-Based Navigation

```
public IActionResult NavigateToAbout()
{
    return RedirectToAction("About", "Home");
}
```

Passing Data Between Actions and Views

Using Route Parameters

Controller:

```
public IActionResult Details(int id)
{
    var product = db.Products.Find(id);
    return View(product);
}
```

View:

```
<a asp-controller="Products" asp-action="Details" asp-route-
        id="@item.ProductId">View Details</a>
```

Using `TempData`

Controller:

```
TempData["SuccessMessage"] = "Product saved successfully!";
return RedirectToAction("Index");
```

View:

```
<p>@TempData["SuccessMessage"]</p>
```

Using Query Strings

Controller:

```
public IActionResult Search(string query)
{
```

```
    var results = db.Products.Where(p =>
        p.Name.Contains(query)).ToList();
    return View(results);
}
```

View:

```
<a href="/Products/Search?query=Example">Search for Example</a>
```

3. HTML Elements in ASP.NET MVC Views

Tables

```
<table class="table">
    <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Price</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.ProductId</td>
                <td>@item.Name</td>
                <td>@item.Price</td>
            </tr>
        }
    </tbody>
</table>
```

Input Fields

Text Input

```
<input type="text" name="ProductName" value="@Model.Name" />
```

Checkbox

```
<input type="checkbox" name="IsAvailable"
        checked="@Model.IsAvailable" />
```

Radio Button

```
<input type="radio" name="Category" value="Electronics" />
        Electronics
<input type="radio" name="Category" value="Clothing" /> Clothing
```

Dropdown List

```
<select name="Category">
    <option value="Electronics">Electronics</option>
    <option value="Clothing">Clothing</option>
</select>
```

Checkbox Example

Controller:

```
public IActionResult UpdateProduct()
{
    return View();
}

[HttpPost]
public IActionResult UpdateProduct(bool isAvailable)
{
    ViewBag.AvailabilityStatus = isAvailable ? "Product is
        available" : "Product is not available";
    return View();
}
```

View:

```
<form method="post" action="/Products/UpdateProduct">
    <label for="isAvailable">Is Available</label>
    <input type="checkbox" name="isAvailable" />
    <button type="submit">Submit</button>
</form>
<p>@ViewBag.AvailabilityStatus</p>
```

4. Passing Data Between Views

Using `ViewData`

```
ViewData["Message"] = "Hello, World!";
return View();
```

```
<p>@ViewData["Message"]</p>
```

Using ViewBag

```
ViewBag.Message = "Welcome!";
return View();

<p>@ViewBag.Message</p>
```

Using Strongly Typed Models

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
}

return View(new Product { Id = 1, Name = "Laptop" });

<p>@Model.Name</p>
```

Using TempData Between Redirects

Controller:

```
TempData["Status"] = "Operation completed!";
return RedirectToAction("Index");
```

Index View:

```
<p>@TempData["Status"]</p>
```

5. Basic and Advanced LINQ Expressions

Filtering Data

```
var expensiveProducts = db.Products.Where(p => p.Price >
        100).ToList();
```

Filtering with Multiple Conditions

```
var filteredProducts = db.Products.Where(p => p.Price > 50 &&
        p.Category == "Electronics").ToList();
```

Searching with Case Insensitivity

```
var searchResults = db.Products.Where(p =>
        p.Name.ToLower().Contains("laptop")).ToList();
```

Sorting Data

```
var sortedProducts = db.Products.OrderBy(p => p.Name).ToList();
```

Sorting Descending

```
var sortedDescending = db.Products.OrderByDescending(p =>
        p.Price).ToList();
```

Selecting Specific Columns

```
var productNames = db.Products.Select(p => p.Name).ToList();
```

Grouping Data

```
var grouped = db.Products.GroupBy(p => p.Category).Select(g =>
        new
{
    Category = g.Key,
    Count = g.Count()
}).ToList();
```

Including Related Data

```
var orders = db.Orders.Include(o => o.Customer).ToList();
```

Joins

```
var query = from order in db.Orders
            join customer in db.Customers on order.CustomerId
        equals customer.CustomerId
            select new { order.OrderId, customer.Name };
```

6. Model Relationships and Configurations

One-to-Many Relationship

```
public class Customer
{
    public int CustomerId { get; set; }
    public string Name { get; set; }
```

```
    public ICollection<Order> Orders { get; set; }
}

public class Order
{
    public int OrderId { get; set; }
    public string ProductName { get; set; }
    public int CustomerId { get; set; }
    public Customer Customer { get; set; }
}
```

Many-to-Many Relationship

```
public class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }
    public ICollection<Course> Courses { get; set; }
}

public class Course
{
    public int CourseId { get; set; }
    public string Title { get; set; }
    public ICollection<Student> Students { get; set; }
}
```

Fluent API Configuration

```
protected override void OnModelCreating(ModelBuilder
        modelBuilder)
{
    modelBuilder.Entity<Order>()
        .HasOne(o => o.Customer)
        .WithMany(c => c.Orders)
        .HasForeignKey(o => o.CustomerId);
}
```

7. Common Problems and Solutions

Problem: Null Reference Exception in Views

- **Cause**: Accessing a property on a null model.
- **Solution**: Check if the model or property is null before using it.

```
@if (Model?.Name != null)
{
    <p>@Model.Name</p>
}
```

Problem: Entity Framework Lazy Loading Not Working

- **Cause**: Lazy loading requires virtual navigation properties.
- **Solution**: Ensure properties are declared as `virtual`.

```
public virtual Customer Customer { get; set; }
```

Problem: Migration Errors (Pending Changes)

- **Solution**: Run `Add-Migration` and `Update-Database`.

Problem: Multiple Enumeration of IEnumerable

- **Solution**: Use `.ToList()` to materialize the query.

```
var products = db.Products.ToList();
```

8. T-SQL Basics

Selecting Data

```
SELECT * FROM Products;
```

Filtering Data

```
SELECT * FROM Products WHERE Price > 100;
```

Sorting Data

```
SELECT * FROM Products ORDER BY Name ASC;
```

Grouping Data

```
SELECT Category, COUNT(*) AS Count FROM Products GROUP BY
        Category;
```

Joining Tables

```
SELECT o.OrderId, c.Name FROM Orders o
INNER JOIN Customers c ON o.CustomerId = c.CustomerId;
```

Subqueries

```
SELECT * FROM Products WHERE Price > (SELECT AVG(Price) FROM
        Products);
```

Using Functions

```
SELECT UPPER(Name) AS UpperName FROM Products;
```