ASP.NET MVC Course

Section 1: Getting Started

MVC Architectural Pattern

- Model View Controller
- Widely adopted as an architecture for web applications
- Better separation of concerns

Model

- Application data and behavior in terms of its problem domain
- Independent of the UI
- A class that includes the application data and rules
- Independent of persistence (can be stored in any databases)

View

- HTML markup displayed to users

Controller

- Responsible for handling HTTP requests

Router

- Selects the right controller to handle the request

Action

- A method in a controller responsible for handling a request

NuGet

- A package manager similar to `NPM` and `Bower`
- Used to manage dependencies of app
- Used to upgrade existing dependencies when newer versions are available

Plugins

- Productivity Power Tools
- Web Essentials
- ReSharper

Shortcuts

- `ctrl + F5` -> Start without debugging
- `F2` -> rename selected
- `ctrl + m + m` -> collapse selected code block / html tag
- `Alt + Enter` -> access **ReSharper** shortcuts
- `F9` -> Add breakpoint
- `F5` -> Start with debugging
- `Shift + F5` -> Quit debugging

- `Ctrl + w` -> Select code block
- `Ctrl + Shift + w` -> Deselect code block

Snippets

- `prop` -> `public TYPE Type { get; set; }`
- `ctor` -> create constructor
- `mvcaction4` -> create action template

Files

- `App_Start` -> `BundleConfig.cs`
  - Defines various bundles and client side assets
- `App_Start` -> `FilterConfig.cs`
  - Handles global filters

C# Syntax

- Add `?` after argument data type to make it nullable:
  `public void functionName(int? intVar){}`
- Write `C#` code within a `.cshtml` file by prefixing with an `@`
- `0` is the default value for numeric properties
- `(byte) varName` -> Cast variable as a `byte`

Function	Definition
`Enumerable.Any()`	Determines whether a sequence contains any elements.
`Enumerable.SingleOrDefault()`	Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

URL Encoded Characters

Character	Encoded
backspace	%08
tab	%09
linefeed	%0A
creturn	%0D
space	%20
!	%21
"	%22
#	%23
$	%24
%	%25
&	%26

'	%27
(	%28
)	%29
*	%2A
+	%2B
,	%2C
-	%2D
.	%2E
/	%2F

IEnumerable, ICollection, List

Interface	Scenario
IEnumerable	You only want to iterate over the elements in a collection
IEnumerable< T >	You only need read-only access to that collection
ICollection	You want to modify the collection or you care about its size
ICollection< T >	
IList	You want to modify the collection and you care about the ordering and / positioning of elements in that collection
IList< T >	
List	Since in object oriented design you want to depend on abstractions instead of implementations, you should never have a member of your own implementations with the concrete type List/List.

Examples

A request to /customers/delete/1 by convention is handled by:
`CustomersController.Delete(int id)`

Conventions

- Name `partial views` with an `_` prefix
- Name `ViewModels` with a `ViewModel` suffix
- `EntityFramework` recognizes foreign key convention `[MODEL NAME]Id`
- Name `DbContext`s with a `_` prefix

Misc

- ASP.NET uses `bootstrap` by default as its front end framework
- Use `bootswatch.com` for bootstrap templates
- `Navigation Properties` allow for navigating between / linking multiple model types and objects together

Course Layout

- ASP.NET MVC Fundamentals
- Entity Framework (Code-first)
- Forms
- Validation
- Build RESTful Services
- Client-side Development
- Authentication and Authorization
- Performance Optimization
- Building a Feature Systematically
- Deployment

Section 2: ASP.NET MVC Fundamentals

Action Results

```
namespace Vidly.Controllers
{
    public class MoviesController : Controller
    {
        // GET: Movies
        public ActionResult Random()
        {
            var movie = new Movie() { Name = "Shrek!" };

            return View(movie);
        }
    }
```

- Actions performed within controller methods in response to HTTP requests

Type	Helper Method
ViewResult	`View()`
PartialViewResult	`PartialView()`
ContentResult	`Content()`
RedirectResult	`Redirect()`
RedirectToRouteResult	`RedirectToAction()`
JsonResult	`Json()`
FileResult	`File()`
HttpNotFoundResult	`HttpNotFound()`
EmptyResult	`HttpNotFound()`

Action Parameters

- Parameter passed to controller from request can be linked to a controller action with the same name

Parameter Sources

- In the URL: `/movies/edit/1`
- In the query string: `/movies/edit?id=1`
- In the form data: `id=1`

Convention-based Routing

- `routes.MapRoute()` order matters, so specify from most-to-least granular
- `routes.MapRoute([Route Name], [URL Pattern], [Defaults])`

```
routes.MapRoute(
    "MoviesByReleaseDate",
    "movies/released/{year}/{month}",
    new { controller = "Movies", action = "ByReleaseDate" }
    );
```

Limiting Route Parameters

```
routes.MapRoute(
    "MoviesByReleaseDate",
    "movies/released/{year}/{month}",
    new { controller = "Movies", action = "ByReleaseDate" },
    new { year = @"\d{4}", month = @"\d{2}"}
    );
```

- Fourth argument to `MapRoute` is for parameter constraints

Attribute Routing

- Activate Attribute Routes in `RouteConfig.cs`:
  `routes.MapMvcAttributeRoutes();`
- Add route template to controller: `[Route("movies/released/{year}/{month:regex(\\d{2}):range(1, 12)}")]`
  - Apply a constraint using `:`

Constraints

- min
- max
- minlength
- maxlength
- int
- float
- guid

Passing Data to Views

- Avoid using `ViewData` or `ViewBag` when possible
- Use `controller`'s `view` method:
  `return view(data)`

View Models

- Used to import and access multiple models within one view

Razor Views

- Can use `<text></text>` tags to render markup with no enclosing tags
- Use `@{}` to write C# code blocks within views
- Add comments with:
  `@* comment text on multiple lines *@`

Partial Views

- Not only for re-using markup, but also for breaking up large views into manageable chunks
- Access partials with `@Html.Partial("_PartialName")`
- Unless specified, model passed to parent view will be passed to partial view
  - To specify, pass a second parameter to `@Html.Partial()`

Section 3: Working with Data

Entity Framework

- Object Relational Mapper (O/RM)
- Maps data from Relational DB into usable objects
- `DbContext` -> Class representing Database
- `DbSet` -> Classes representing tables

LINQ

- Used to query `DbSet`s and automatically generate `SQL` queries to DB on runtime
- Entity Framework keeps track of add / modify / deletes on DbSet and automatically generates `SQL` queries reflecting changes

Workflows

- Database First
- Code First

DbFirst vs CodeFirst

DbFirst
[Domain Classes] <-- [Entity Framework] <-- [Database]

CodeFirst
[Domain Classes] --> [Entity Framework] --> [Database]

- Increases productivity
- Full versioning of database
- Much easier to build an integration test db

CodeFirst Myths

- Does not give you full control over the Database

Code-first Migrations

Package Manager Console Commands

- `enable-migrations`
- `add-migration [MIGRATION MODEL NAME]`

- `add-migration [MIGRATION MODEL NAME] - force`
  - Overwrite last migration
  - Treat like `git` commits for Db model
- `update-database`
  - Run migration and generate database
- Allows for a consistent, trackable way to version database through migrations

Changing the Model

- Aim for small migrations
  - Many developers struggle with code-first development because they push 'atomic commits' that are far too large

Seeding the Database

- In `code-first` workflow, data should not be added to DB directly, but through `migrations`
- Exception is for arbitrary test data that serves no integral function in app
- `Sql("INSERT INTO ...")`

Overriding Conventions

- Use `DataAnnotations` to override property requirements:
  `public class Customer { public int Id { get; set; } [Required] //DataAnnotation [StringLength(255)] //DataAnnotation public string Name { get; set; } public bool IsSubscribedToNewsletter { get; set; } public MembershipType MembershipType { get; set; } // Navigation property public byte MembershipTypeId { get; set; } }`
- Requires `using System.ComponentModel.DataAnnotations;`

Data Annotations

- `[Display(Name = "[DISPLAY TEXT]")]`
- `[Required]`
- `[StringLength(255)]`
- `[Range(1, 10)]`
- `[Compare(["OtherProperty"])]`
- `[Phone]`
- `[EmailAddress]`
- `[Url]`
- `[RegularExpression("...")]`
- Used for both server-side and client-side validation

Querying Objects

- Controllers must have access to `DbContext`:

```
  private ApplicationDbContext _context;

      public CustomersController()
      {
         _context = new ApplicationDbContext();
      }</code></pre></li>
```

- Remember to dispose of DbContext when done:

```
      protected override void Dispose(bool disposing)
             {
                 _context.Dispose();
```

```
            }
```

- `var customers = _context.Customers;`
    - Entity Framework does **NOT** immediately query database after this line
    - DB query occurs when iterating over `customers`
    - Use `var customers = _context.Customers.ToList();` to execute db query

LINQ Extension Methods

- `_context.Movies.Where(m => m.GenreId == 1)`
- `_context.Movies.Single(m => m.Id == 1);`
- `_context.Movies.SingleOrDefault(m => m.Id == 1);`
- `_context.Movies.ToList();`

Eager Loading

- When a `DbSet` is joined with another, Entity framework does not query linked DbSet by default -- solve with `Eager Loading`
- `_context.Movies.Include(m => m.Genre);`
- Requires `using System.Data.Entity;`

Section 4: Building Forms

Create Form

```
@using (Html.BeginForm("Save", "Customers"))
{
    <div class="form-group">
        @Html.LabelFor(m => m.Customer.Name)
        @Html.TextBoxFor(m => m.Customer.Name, new { @class = "form-control" })
    </div>
    <button type="submit" class="btn btn-primary">Save</button>
}
```

HTML Helper Methods

- `Html.BeginForm`
- `Html.LabelFor`
- `Html.TextBoxFor`
    - Inherits constraints from `model` (e.g. `StringLength`, `Required`)
- `Html.CheckBoxFor`
- `Html.DropDownListFor`
- `Html.HiddenFor`
- `Html.Hidden`
- `Html.ValidationMessageFor`
- `Html.ValidationSummary()`
- `Html.AntiForgeryToken()`

Form Dropdowns

```
<div class="form-group">
    @Html.LabelFor(m => m.Customer.MembershipTypeId, "MembershipType")
    @Html.DropDownListFor(m => m.Customer.MembershipTypeId, new SelectList(Model.MembershipTypes, "Id", "MembershipTypeTit
</div>
```

Model Binding

- Use `[HttpPost]` before an action to handle `POST` requests

```
[HttpPost]
    public ActionResult Create(Customer customer)
    {
        ...
    }
```

Saving Data

```
_context.Customers.Add(customer);
_context.SaveChanges();
```

- Changes to `_context` do not modify database until `.SaveChanges()`
- When `.SaveChanges()` is called, `SQL` statements for all modifications to `_context` are generated and DB is modified

Edit Form

Microsoft Suggests

```
var customerInDb = _context.Customers.Single(c => c.Id == customer.Id);
TryUpdateModel(customerInDb);
```

- Issues with this approach:
  - Security vulnerabilities. What if user should not have permission to update **All** customer attributes?

Microsoft Workaround

```
var customerInDb = _context.Customers.Single(c => c.Id == customer.Id);
TryUpdateModel(customerInDb, "", new string[]{ "Name", "Email" });
```

- Why this is worse
  - If attribute name changes, code breaks

Best Solution

```
customerInDb.Name = customer.Name;
customerInDb.Birthdate = customer.Birthdate;
customerInDb.MembershipTypeId = customer.MembershipTypeId;
customerInDb.IsSubscribedToNewsletter = customer.IsSubscribedToNewsletter;
```

- Can be done automatically with `AutoMapper`
  - Maps property names of source and target and maps them by convention
  - `Mapper.Map(customer, customerInDb)`

Addressing Security Risk

- Can create a `dto` class (Data Transfer Object) with only 'update-safe' properties:
  - `UpdateCustomerDto`

Section 5: Validation

Adding Validation

- `ASP.NET MVC` uses **Data Annotations** to validate **Action Parameters**
- Use `ModelState` object to gain access to validation data

Steps

1. Add **Data Annotations** to entities:
   `[Required]`
2. Use `ModelState.IsValid` to verify validity of form data
3. Add validation messages to form:
   `@Html.ValidationMessageFor(m => m.Customer.Name)`

Custom Validation Error Message

- Has default messages based on **Data Annotations**
- Can be overwritten:
  `[Required(ErrorMessage = "Please enter customer's name.")]`

Custom Validation

1. Add a validation model
   `public class Min18YearsIfAMember : ValidationAttribute { protected override ValidationResult IsValid(object value, ValidationContext validationContext) { if(valid) return ValidationResult.Success; else return new ValidationResult("error message") }`
2. Add validation model as **Data Annotation**
   `[Min18YearsIfAMember] public DateTime? Birthdate { get; set; }`

Validation Summary

- `Html.ValidationSummary()`
- Displays summary of validation errors

Client-side Validation

Benefits

- Immediate feedback
- No waste of server-side resources
- *Not* enabled by default in `ASP.NET` applications

- Must enable `jquery.validate` scripts
- `ASP.NET Razor` recognizes default **Data Annotations**, *NOT* custom ones

```
@section scripts {
@Scripts.Render("~/bundles/jqueryval")}
```

Anti-forgery Tokens

CSRF

- Cross-site Request Forgery
- Forged request to different w eb domain on behalf of user

`Html.AntiForgeryToken()`

- Creates a token stored in form and user cookies (encrypted) and compares the tw o w hen `POST` request is made
- Means user must be on form page to access hidden token field
- Add `[ValidateAntiForgeryToken]` attribute to `action` along w ith `Html.AntiForgeryToken()` in view s

Section 6: Building RESTful Services with ASP.NET Web API

- `Razor Engine` generates markup on server-side and sends markup to client

Benefits of generating markup on client

- Less server resources (improve scalability)
- Less bandw idth (improves performance)
- Support for a broad range of clients

RESTful Convention

1. Create an ApiController:

```
public class CustomersController : ApiController
    {
        // GET /api/customers
        public IHttpActionResult GetCustomers() {}

       // POST /api/customers
       [HttpPost]
       public IHttpActionResult CreateCustomer(CustomerDto customer) {}

       // PUT /api/customers/{id}
       [HttpPut]
       public IHttpActionResult UpdateCustomer(int id, CustomerDto customer) {}

       // DELETE /api/customers/{id}
       [HttpDelete]
       public IHttpActionResult DeleteCustomer(int id) {}
    }</code></pre></li>
```

- Api Actions w ith `Post` prefix are automatically configured to handle `POST` routes

- However, avoid this convention when possible and stick to `[HttpPost]` decoration

Data Transfer Objects

- APIs shouldn't send or receive domain objects
  - As these objects change and grow with the application, chances of breaking APIs increase
  - With direct access to domain objects, hackers can modify properties they should not have access to
- DTOs allow for decoupling domain objects

AutoMapper

- Maps properties between source and destination models based on property names (convention-based mapping tool)

1. Install through `Package Manager Console`:
   `install-package automapper`
2. Create Dtos folter and Dto
3. Add Mapping Profile class to `App_Start`
4. Initialize mapper during application startup in `Application_Start()` function within `Global.asax.cs`:
   `Mapper.Initialize(c => c.AddProfile<MappingProfile>());`
5. To map objects:
   `var customerDto = Mapper.Map<Customer, CustomerDto>(customer);`
   Or map to existing object:
   `Mapper.Map(customer, customerDto)`

To Ignore properties when mapping:
`Mapper.CreateMap<Movie, MovieDto>().ForMember(m => m.Id, opt => opt.Ignore());`

Using Camel Notation

- In `App_Start/WebApiConfig.cs`:
  `var settings = config.Formatters.JsonFormatter.SerializerSettings; settings.ContractResolver = new CamelCasePropertyNamesContractResolver(); settings.Formatting = Formatting.Indented;`

IHttpActionResults

- Allows for improved control over Http Responses (e.g., Sending a `201 Created` status code)

ApiController Helper Methods

- `NotFound()`
- `Ok()`
- `Created()`
- `Unauthorized()`
- `BadRequest()`

Section 7: Client-side Development

Calling an API Using jQuery

```
$(document).ready(function() {
            $("#customers").on("click", ".js-delete",
                function () {
                    bootbox.confirm("Are you sure you want to delete this customer?",
                        (result) => {
                            if (result) {
```

```
                                $.ajax({
                                    url: `/api/customers/${$(this).attr("data-customer-id")}`,
                                    method: "DELETE",
                                    success: () => {
                                        $(this).parents("tr").remove();
                                    }
                                });
                            }
                        });
                });
        });
```

Bootbox

- An abstraction over `Bootstrap`

DataTables Plugin

- A jQuery plugin for paginating, sorting, and filtering table data

DataTables - Zero Configuration

```
$("#customers").DataTable();
```

DataTables with AJAX Source

```
$("#customers").DataTable({
                ajax: {
                    url: "/api/customers",
                    dataSrc: ""
                },
                columns: [
                    {
                        data: "name",
                        render: (data, type, customer) => ("<a href='/customers/edit/" +
                            customer.id +
                            "'>" +
                            customer.name +
                            "</a>")
                    },
                    {
                        data: "name",
                    },
                    {
                        data: "id",
                        render: (data) => ("<button class='btn-link js-delete' data-customer-id=" + data + ">Delete</butto
                    }
                ]
            });
```

Hierarchical Data

1. Create DTO for referenced domain model:
   `public class MembershipTypeDto { public byte Id { get; set; } public string MembershipTypeTitle { get; set; } }`
2. Update Mapping Profile:
   `Mapper.CreateMap<MembershipType, MembershipTypeDto>();`
3. Update ControllerAPI to include referenced model:
   `using System.Data.Entity;`
   `var customerDtos = _context.Customers .Include(c => c.MembershipType) .ToList() .Select(Mapper.Map<Customer,`

`CustomerDto>);`

DataTables: Removing Records

```
var table = $(...).DataTable(...);
table.row($(this).parents("tr")).remove().draw();
```

Single Page Applications

Benefits

- Smoother
- Faster

Section 8: Authentication and Authorization

- Done using `ASP.NET Identity` Framework

Authentication Options

- No Authentication
- Individual User Accounts
- Organizational Accounts
- Windows Authentication

ASP.Net Identity

Domains

- `IdentityUser`
- `Role`

Api / Service

- `UserManager`
- `RoleManager`
- `SignInManager`
- ...

Persistence

- `UserStore`
- `RoleStore`

Restricting Access

```
[Authorize] // Action Attribute / Filter
public ActionResult Index()
{
    ...
}
```

- Can be applied to entire `controller`, or `globally` not just `actions`
- Use `[AllowAnonymous]` on a `controller` or `action` to override higher-level filters

Seeding Users and Roles

- *Best Practice:* Name user roles after actual permissions (i.e. `CanManageMovies` not `StoreManager`)

Creating a Manager Role

1. Create a new RoleStore
   `var roleStore = new RoleStore<IdentityRole>(new ApplicationDbContext());`
2. Create a new RoleManager for that RoleStore:
   `var roleManager = new RoleManager<IdentityRole>(roleStore);`
3. Create a new Role within that RoleManager:
   `await roleManager.CreateAsync(new IdentityRole("CanManageMovies"));`
4. Add New user to that Role
   `await UserManager.AddToRoleAsync(user.Id, "CanManageMovies");`

Seeding a DB with Guest and Admin Roles

1. Create a new migration called `SeedUsers` or some variation
2. Run the following queries:
   - Insert Guest user in `AspNetUsers` table
   - Insert Admin user `AspNetUsers` table
   - Insert Admin / Manager Role `AspNetRoles` table
   - Link Admin user and Role in `AspNetUserRoles` table

- Avoid using the `Seed` method of the configuration class to seed new users:
  - This requires running `update-database` on production, as this is how `seed` method is called
  - This requires changing connection string in `Web.config`, which is risky

Working with Roles

- To prevent large nest of `if / else` blocks in views for tool access, often best practice is to create new views based on user roles:

```
if (User.IsInRole("CanManageMovies"))
    return View("List");
return View("readOnlyList");
```

- Apply `[Authorize(Roles = "ROLE NAME")]` to actions to override `global` filters and remedy security holes

- Apply `filters.Add(new AuthorizeAttribute());` to `FilterConfig.cs` to declaratively require authorization globally

User Filter Data Annotations

- `[Authorized]`
- `[Authorize]`
- `[Authorize(Roles = "ROLE NAME")]`
- `[AllowAnonymous]`

Adding Profile Data

1. Update `IdentityUser` model with new property
   `[Required] [StringLength(255)] public string DrivingLicense { get; set; }`

2. Add migration, since a domain model was modified:
   `add-migration AddDriverLicenseToApplicationUser`
3. Update databse
   `Update-Database`
4. Update View Model to include new property:
   `[Required] public string DrivingLicense { get; set; }`
5. Add new property to form view:
   `¨K139K`
6. Ensure domain object includes new property:
   `var user = new ApplicationUser { UserName = model.Email, Email = model.Email, DrivingLicense = model.DrivingLicense };`

OAuth

- Open Authorization

Under the Hood
[App] -- {key, secret} --> [Google OAuth]
[App] <-- {authorization="" token}="" --="" [google="" oauth]="" [app]="" {token,="" key,="" secret}=""> [Google OAuth]
[App] <-- {access token} -- [Google OAuth]

Using Social Logins

- Enable SSL (for secure communication)
- Register App with OAuth Service (Google, Facebook, LinkedIn, Twitter, etc…)

1. Enable `SSL` within project
   - Project Properties -> SSL Enabled = True
   - Properties -> Web -> Project Url = [SSL URL]
2. Add `filters.Add(new RequireHttpsAttribute());` to `FilterConfig`
   - Requires HTTPS connections
3. Register app with external Authentication provider to get a key/secret
4. In `App_Start/Startup.Auth.cs`, remove comment for the corresponding providers and add key/secret

Section 9: Authentication and Authorization

> Premature optimization is the root of all evils
> ~ Donald Knuth

- Optimize only when necessary
- Otherwise, development and maintenance costs increase with little to no benefit

Three-tier Architecture

Data

- SQL Server

Application

- IIS

Client

- Browser

- Usually, performance optimization has the greatest observable gain on the `Data Tier`

Mosh's Optimization Rules

- Do not sacrifice maintainability of code to premature optimization
- Be realistic and think like an "engineer"
- Be pragmatic and ensure your efforts have observable results and give value

Data Tier

- Schema and Queries

Schema Issues

- Include Primary Keys
- Foreign Keys / Relationships
- Index columns that are used for filtering records in queries
- Avoid EAV Pattern
  - EAV - Entity Attribute Value
  - No O/RMs
  - Long, tedious queries (must be written manually since no O/RMs)
  - Very slow

Optimizing Queries

- Use Execution Plan in SQL Server
- Create a "read" database (`CQRS`)
  - A separate DB optimized for reading data
- Use caching
  - Run query and store results in memory

Glimpse

- `install-package glimpse.mvc5`
- `install-package glimpse.ef6`
- Adds `/glimpse.axd` endpoint to application
- Only accessible locally
- Used to monitor activity on web application
  - SQL Queries executed
  - AJAX requests made
  - …
- Avoid **Lazy Loading** when possible
  - Opposite of **Eager Loading**
  - Done by adding `virtual` to model properties:
    `public virtual MembershipType MembershipType { get; set; }`
  - Loads object only if one or more navigation properties of an object are touched
  - Should be avoided, as data sent to client should be known ahead of time
  - Lazy loading sends multiple round-trips to DB
  - Causes `N + 1` issue

Output Cache

- Used for caching HTML
- Add `[OutputCache]` action filter above action or controller:

```
[OutputCache(Duration = 50, Location = OutputCacheLocation.Server, VaryByParam = "genre")]
    public ActionResult Index()
    {
        return View();
    }
```

- Caches the view served by the `Index` action for 50 seconds
- Caches on the server, not client
- Stores separate cache based on action parameter (in this case, the query string `genre`)

Downside of Caching

- May display 'stale' data to customers
- `[OutputCache(Duration = 0, VaryByParam = "*", NoStore = true)]`
  - Disables caching rendered HTML on a given action

Data Caching

- Should be limited to actions that are responsible for *displaying* data, *not modifying* it

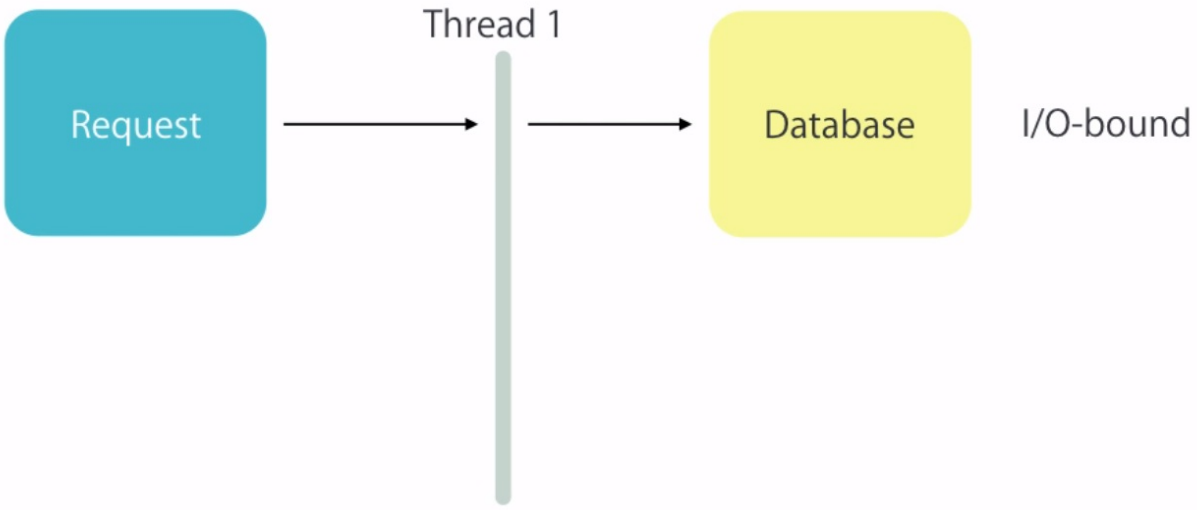
```
if (MemoryCache.Default["Genres"] == null)
{
   MemoryCache.Default["Genres"] = _context.Genres.ToList();
}
var genres = MemoryCache.Default["Genres"] as IEnumerable<Genre>;
```

- Avoid caching until significant performance profiling has been performed
  - Increases memory consumption of application
  - Increases complexity at architectural and code level

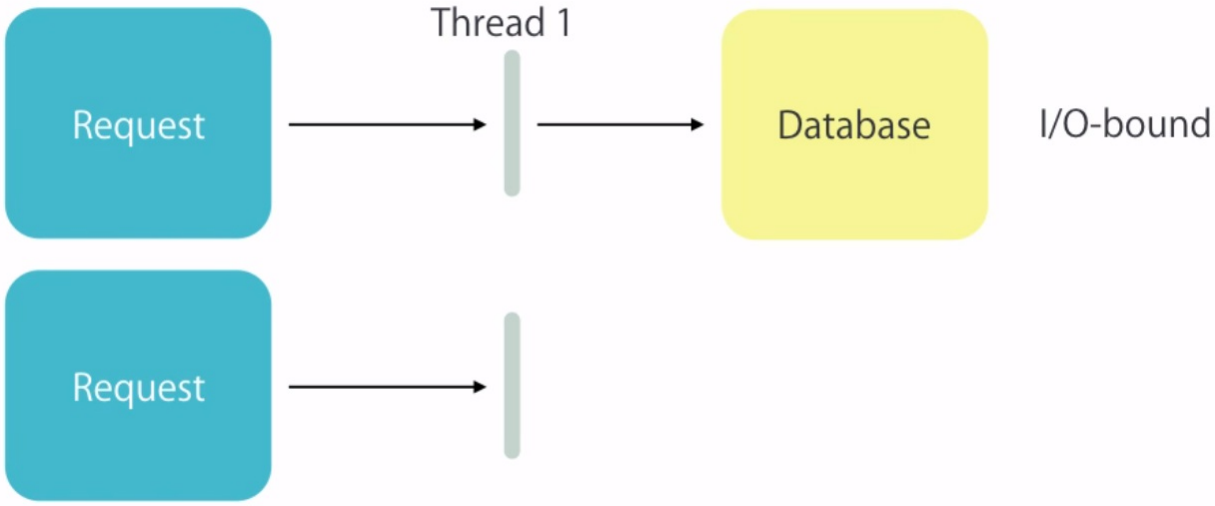
Async

- Async does *not* necessarily improve the performance of an application
  - Using `async` functions frees the thread to process new requests, but does not modify the time necessary for the initial request
- Provides better scalability, not performance
  - (when used with SQL Cluster, NoSQL, SQL Azure vs a single instance of SQL Server)

Non-Async:



Async:



Release Builds

- Switch to release build mode to compile application quicker for deployment
  - Slightly smaller / faster assemblies

Disabling Session

- **Session** - A piece of memory in the web server allocated to each user
- More users, more memory required on web server
- Kills scalability
- Scalable web apps should be stateless

Client Tier

Optimization

- DTO
- JS

- CSS
- Image

Section 10: Building a Feature End-to-End Systematically

- Gather Use Case from Client
- Determine Inputs / Outputs
- Implement in Front End and Back End