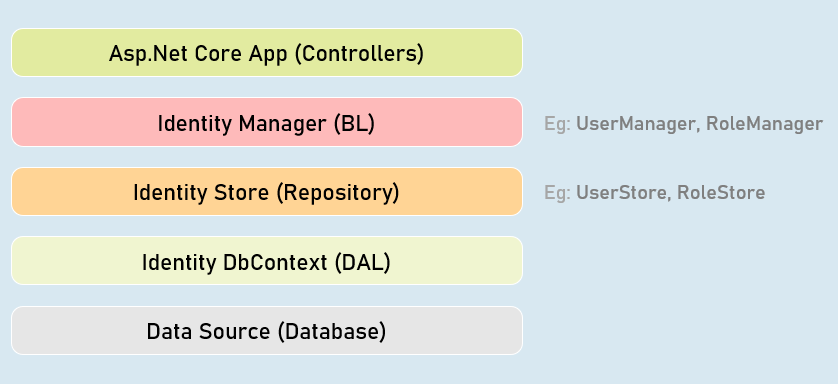
**Introduction to Identity**

It is an API that manages users, passwords, profile data, roles, tokens, email confirmation, external logins etc.

It is by default built on top of EntityFrameworkCore; you can also create custom data stores.



**IdentityUser<T>**

Acts as a base class for ApplicationUser class that acts as model class to store user details.

You can add additional properties to the ApplicationUser class.

**Built-in Properties:**

Id

UserName

PasswordHash

Email

PhoneNumber

IdentityRole<T>

Acts as a base class for ApplicationRole class that acts as model class to store role details. Eg: "admin"

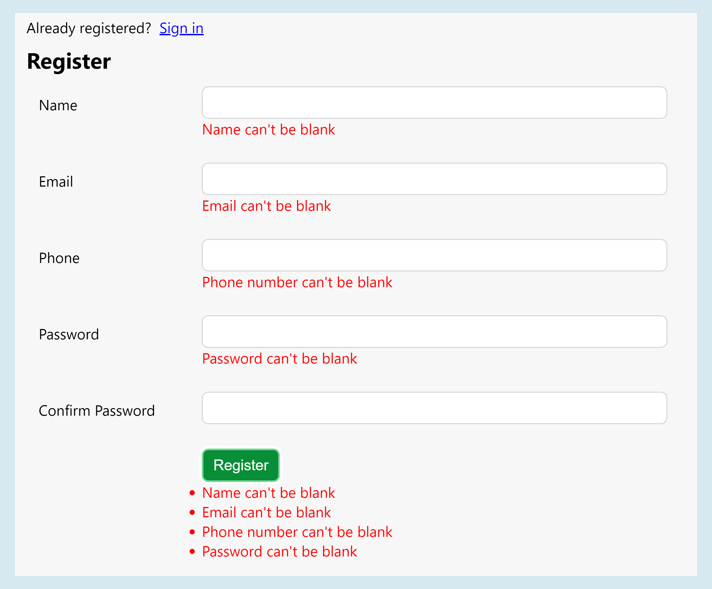
You can add additional properties to the ApplicationRole class.

**Built-in Properties:**

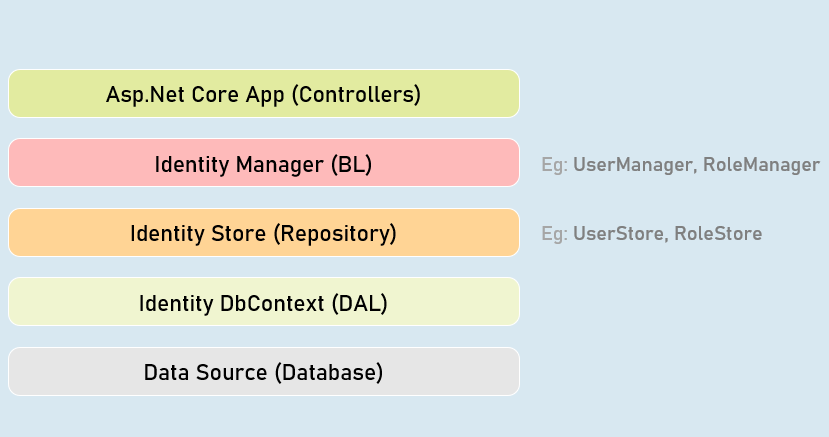
Id

Name

**Register View**



Managers



**UserManager**

Provides business logic methods for managing users.

It provides methods for creating, searching, updating and deleting users.

Methods:

CreateAsync()

DeleteAsync()

UpdateAsync()

IsInRoleAsync()FindByEmailAsync()

FindByIdAsync()

FindByNameAsync()

**SignInManager**

Provides business logic methods for sign-in and sign-in functionality of the users.

It provides methods for creating, searching, updating and deleting users.

Methods:

SignInAsync()

PasswordSignInAsync()

SignOutAsync()

IsSignedIn()

Password Complexity Configuration

services.AddIdentity<ApplicationUser, ApplicationRole>(options => {

options.Password.RequiredLength = 6; //number of characters required in password

options.Password.RequireNonAlphanumeric = true; //is non-alphanumeric characters (symbols)

required in password

options.Password.RequireUppercase = true; //is at least one upper case character required in password

options.Password.RequireLowercase = true; //is at least one lower case character required in password

options.Password.RequireDigit = true; //is at least one digit required in password

options.Password.RequiredUniqueChars = 1; //number of distinct characters required in password

})

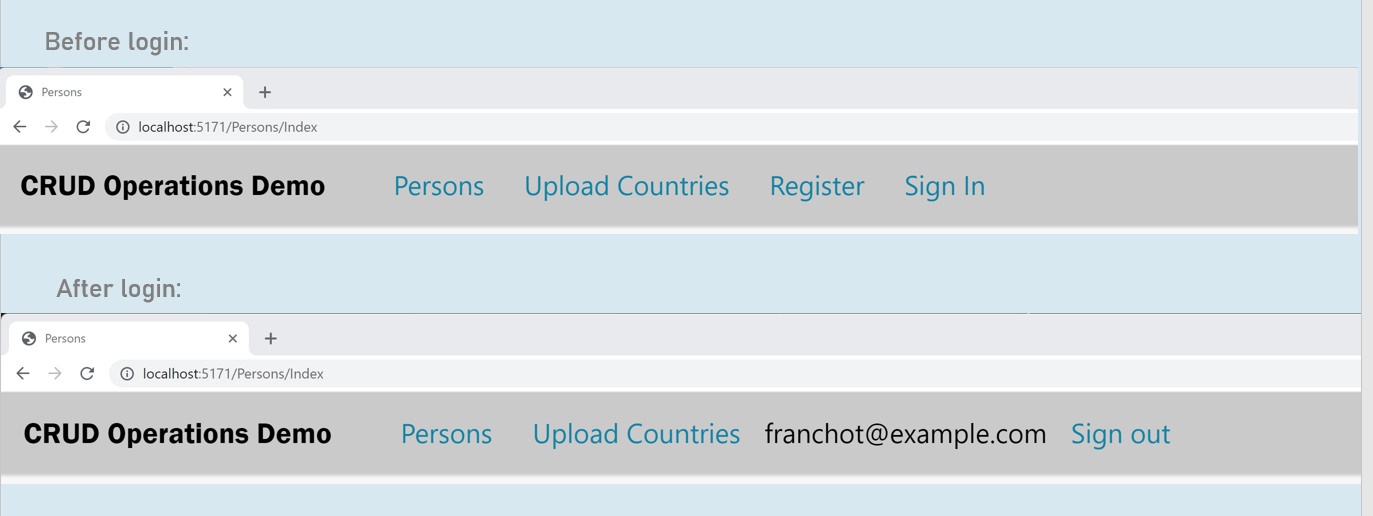
.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders()

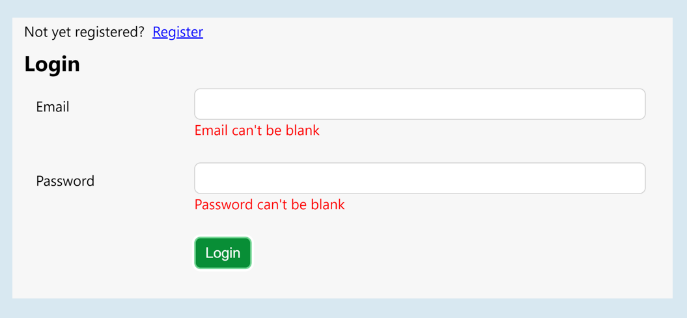
.AddUserStore<UserStore<ApplicationUser, ApplicationRole, ApplicationDbContext, Guid>>()

.AddRoleStore<RoleStore<ApplicationRole, ApplicationDbContext, Guid>>();

Login/Logout Buttons



Login View



Authorization Policy

services.AddAuthorization(options =>

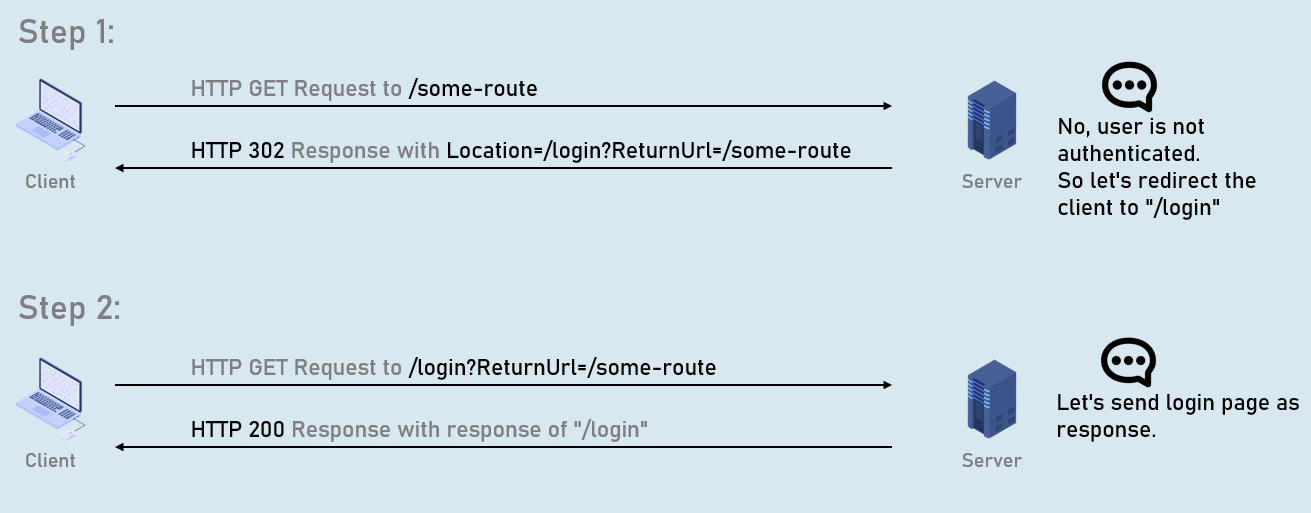
{

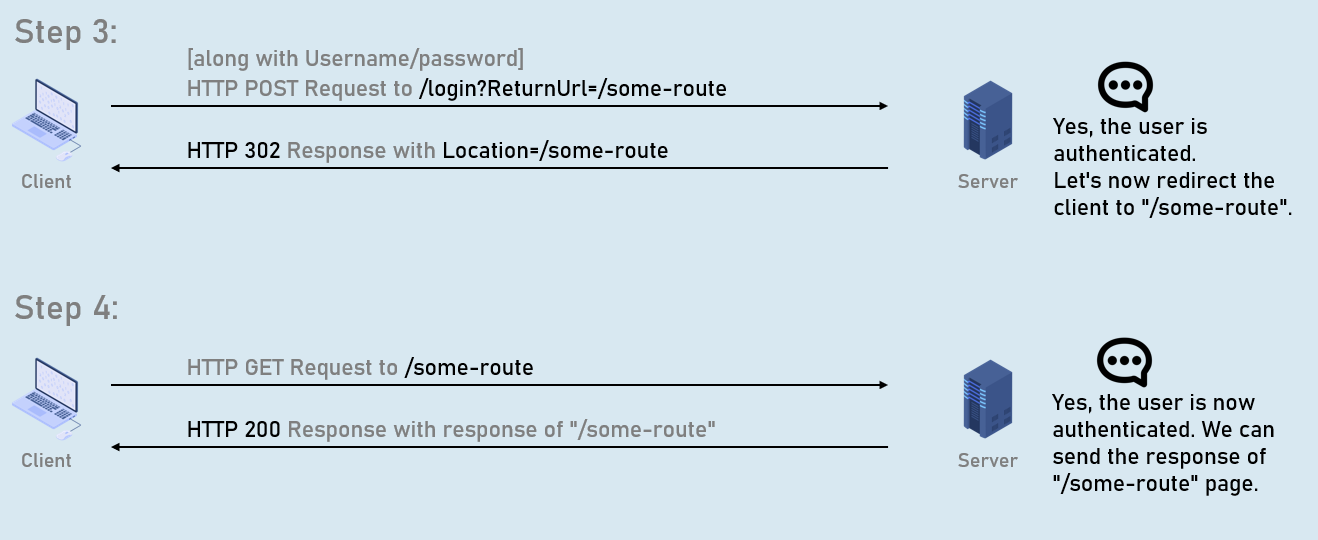
var policy = new AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();

options.FallbackPolicy = policy;

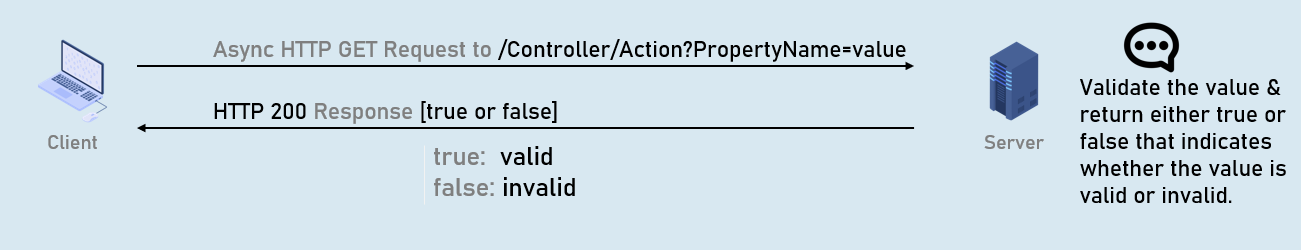
});

ReturnUrl





Remote Validation



**Model class**

public class ModelClassName

{

[Remote(action: "action name", controller: "controller name", ErrorMessage = "error message")]

public type PropertyName { get; set; }

}

Conventional Routing

Conventional routing is a type of routing system in asp.net core that defines route templates applied on all controllers in the entire application.

You can override this using attribute routing on a specific action method.

endpoints.MapControllerRoute(

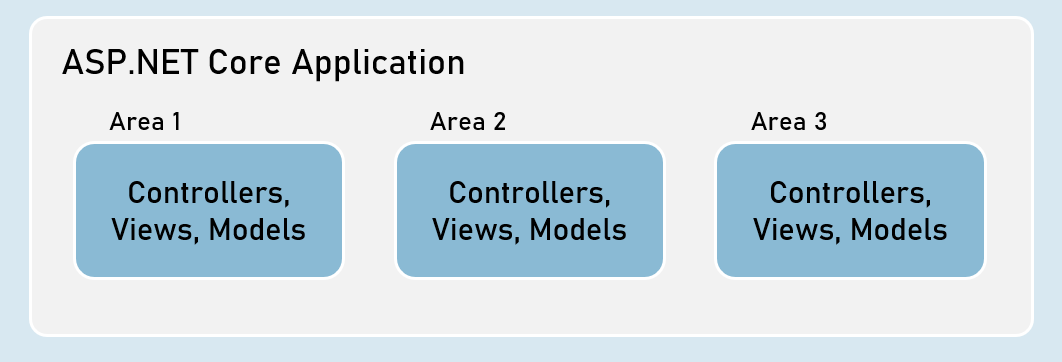
name: "default",

pattern: "{controller=Persons}/{action=Index}/{id?}"

);

Areas

Area is a group of related controllers, views and models that are related to specific module or specific user.



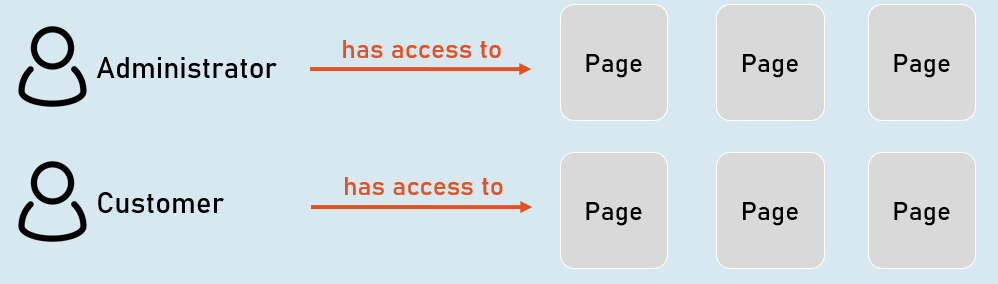
User Roles



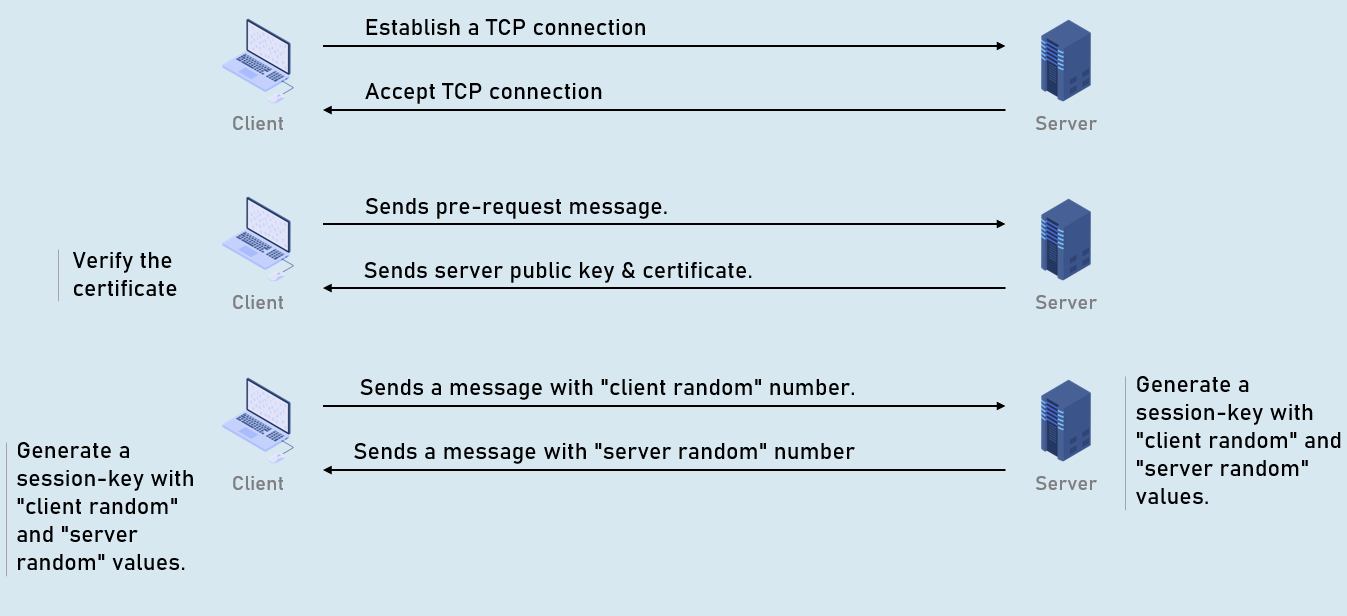
Role Based Authentication

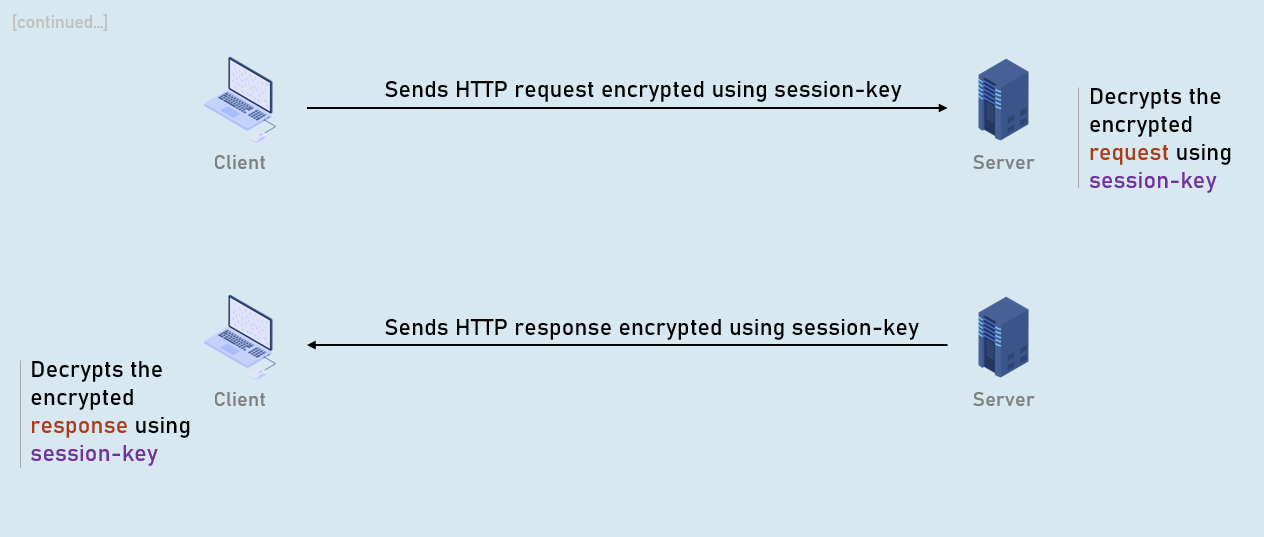
User-role defines type of the user that has access to specific resources of the application.

Examples: Administrator role, Customer role etc.



HTTPS



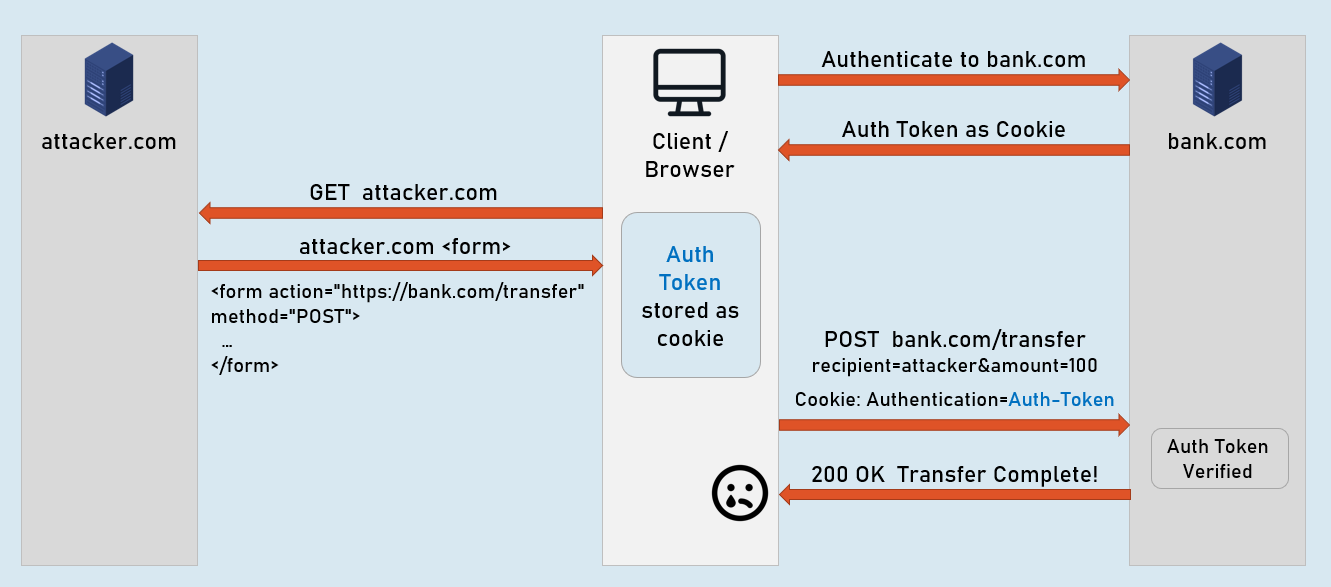


XSRF

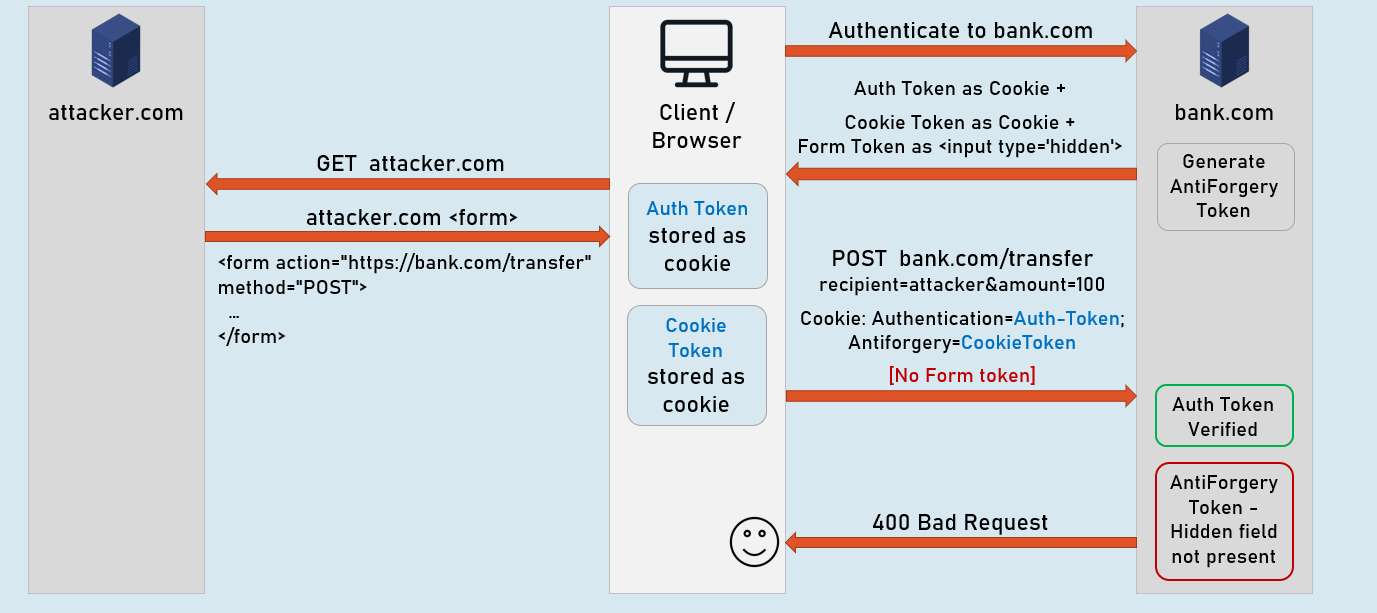
XSRF (Cross Site Request Forgery - CSRF) is a process of making a request to a web server from another domain, using an existing authentication of the same web server.

Eg: attacker.com creates a form that sends malicious request to original.com.

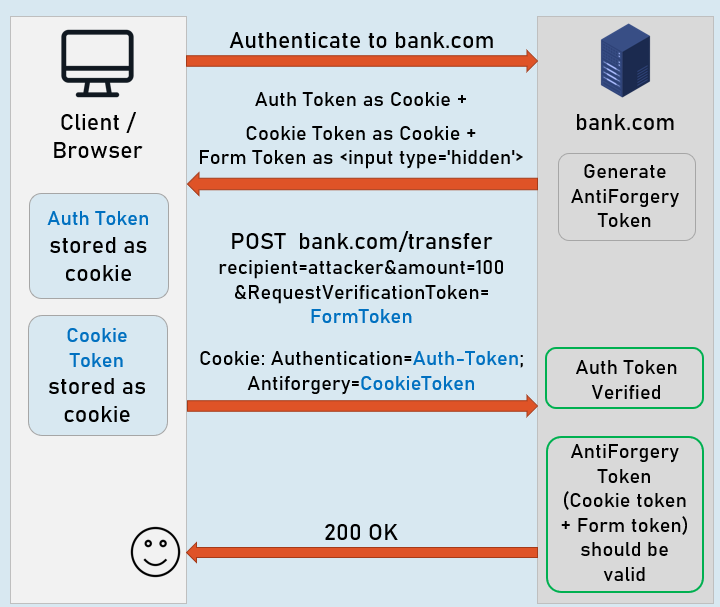
**Attacker's request without AntiForgeryToken**



**Attacker's request**



**Legit request [No attacker.com]**



**Introduction to Web API**

ASP.NET Core Web API is a component (part) of ASP.NET Core, which is used create HTTP-based RESTful services (also known as HTTP services) that can be consumed (invoked) by wide range of client applications such as single-page web applications, mobile applications etc.

Asp.Net Core:

Asp.Net Core MVC

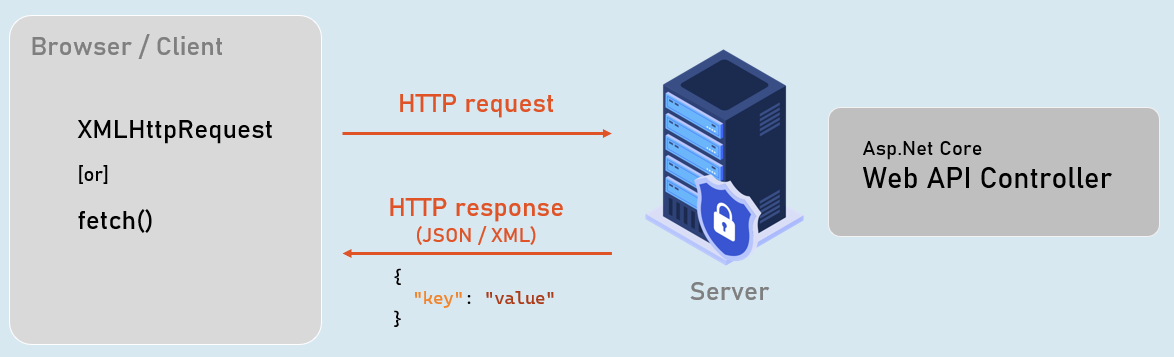
Asp.Net Core Web API

Asp.Net Core Blazor

Asp.Net Core Razor Pages

**RESTful / Web API Services**

RESTful services (Representational State Transfer) is an architecture style that defines to create HTTP services that receives HTTP GET, POST, PUT, DELETE requests; perform CRUD operations on the appropriate data source; and returns JSON / XML data as response to the client.



**Web API Controllers**

Should be either or both:

The class name should be suffixed with "Controller". Eg: ProductsController

The [ApiController] attribute is applied to the same class or to its base class.

**Controller**

**[ApiController]**

class ClassNameController

{

//action methods here

}

Optional:

Is a public class.

Inherited from Microsoft.AspNetCore.Mvc.ControllerBase. (Controller has access to view , etc.

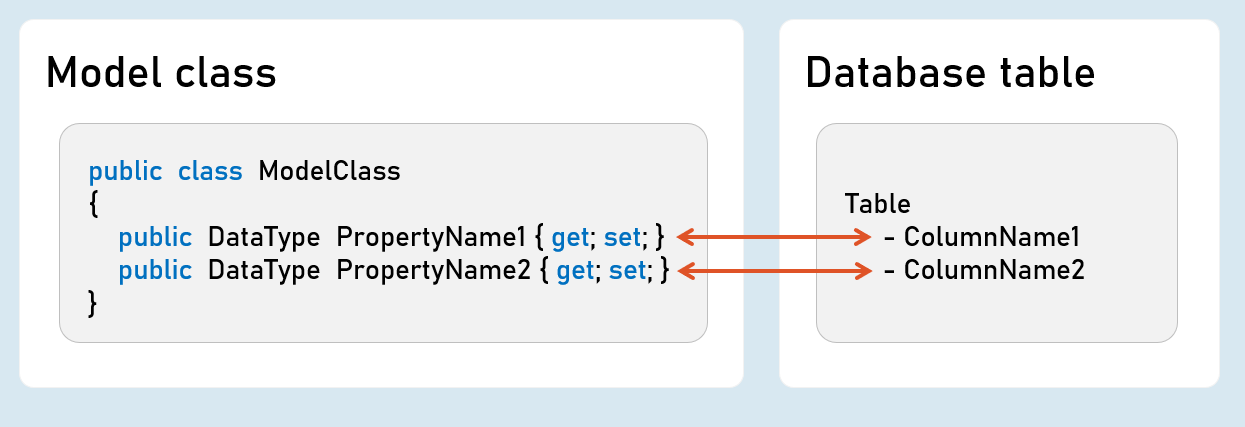
Introduction to EntityFrameworkCore

EntityFrameworkCore is light-weight, extensible and cross-platform framework for accessing databases in .NET applications.

It is the most-used database framework for Asp.Net Core Apps.



**EFCore Models**



Pros & Cons of EntityFrameworkCore

1. Shorter Code

The CRUD operations / calling stored procedures are done with shorter amount of code than ADO.NET.

2. Performance

EFCore performs slower than ADO.NET.

So ADO.NET or its alternatives (such as Dapper) are recommended for larger & high-traffic applications.

3. Strongly-Typed

The columns as created as properties in model class.

So the Intellisense offers columns of the table as properties, while writing the code.

Plus, the developer need not convert data types of values; it's automatically done by EFCore itself.

ProblemDetails

ProblemDetails

public class ProblemDetails

{

string? Type { get; set; } //URI references that identifies the problem type

string? Title { get; set; } //Summary of the problem type

int? Status { get; set; } //HTTP response status code

string? Detail { get; set; } //Explanation of the problem

}

ValidationProblemDetails

public class ValidationProblemDetails : ProblemDetails

{

string? Type { get; set; } //URI references that identifies the problem type

string? Title { get; set; } //Summary of the problem type

int? Status { get; set; } //HTTP response status code

string? Detail { get; set; } //Explanation of the problem

IDictionary<string, string[]> Errors { get; set; } //List of validation errors

}

IActionResult [vs] ActionResult

**IActionResult**

public interface IActionResult

{

Task ExecuteResultAsync(ActionContext context); //converts an object into response

}

**ActionResult<T>**

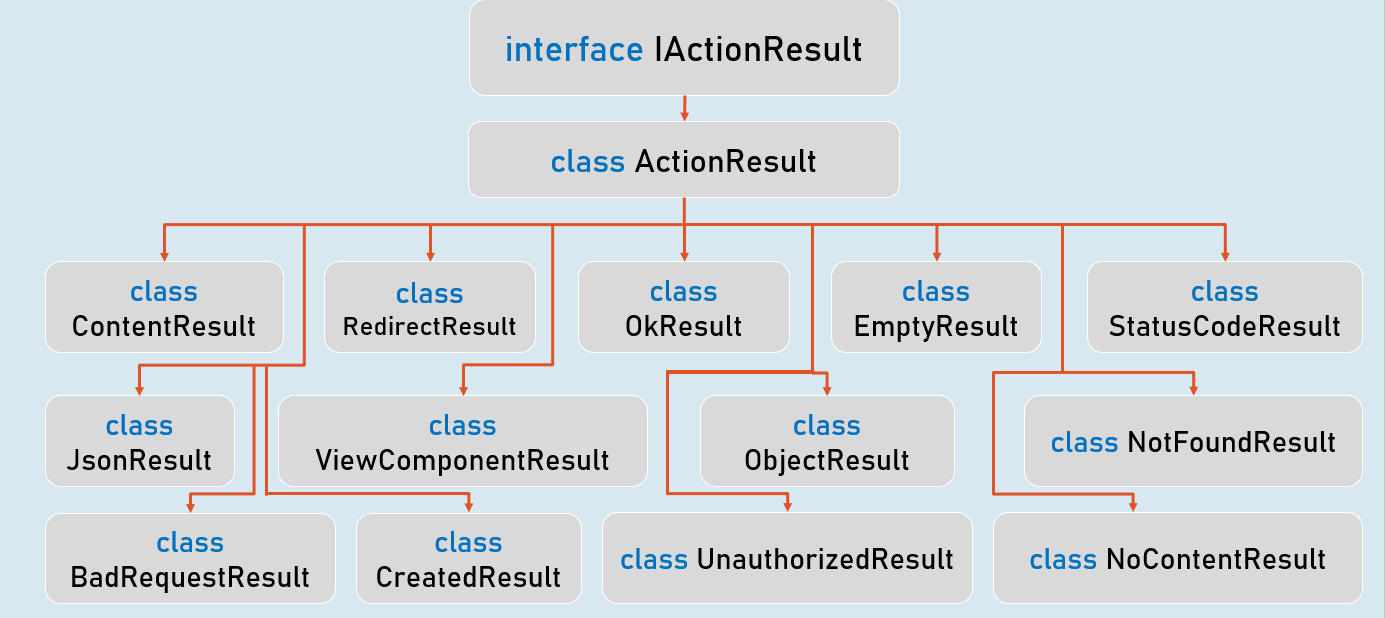
public sealed class ActionResult<T>

{

IActionResult Convert(); //converts the object into ObjectResult

}

IActionResult



**ObjectResult**

