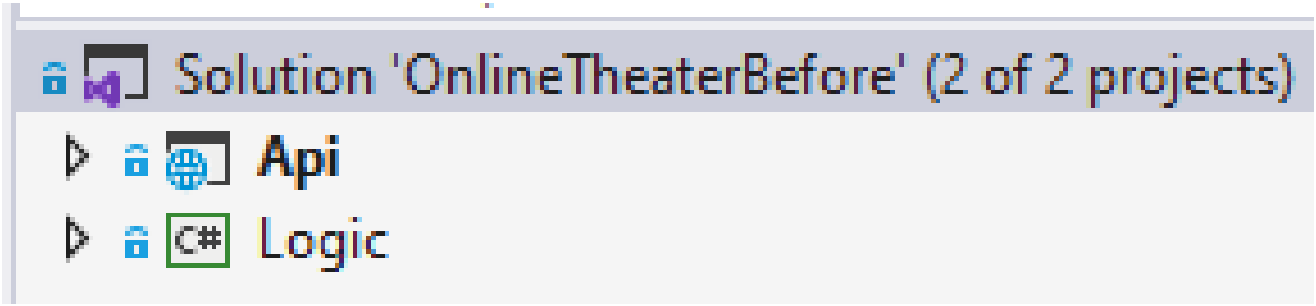
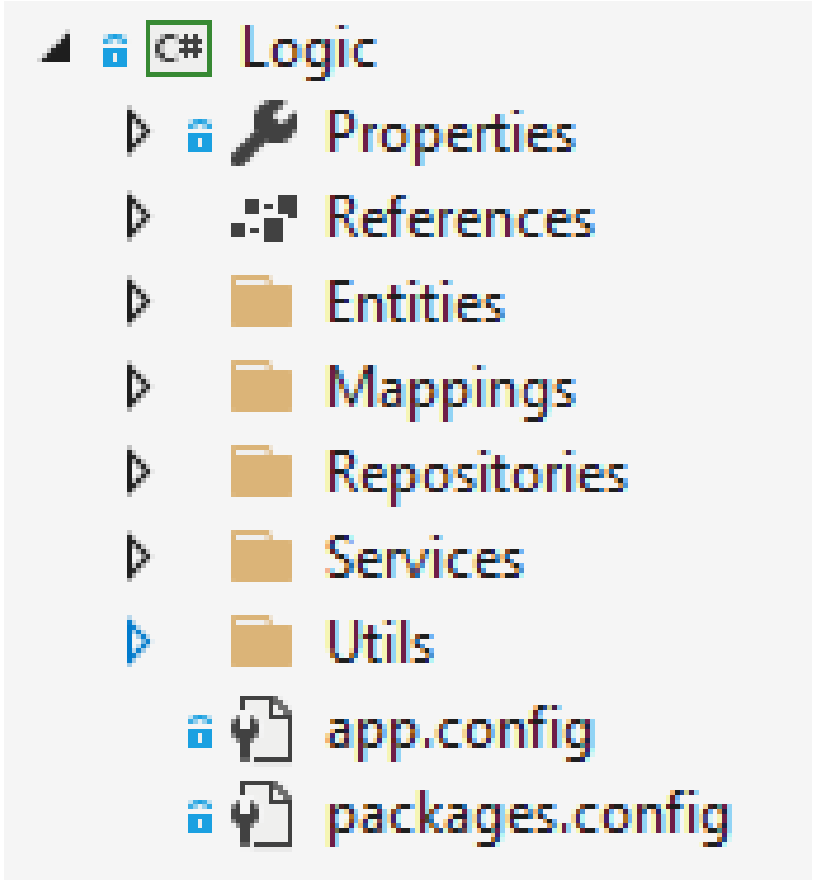
Anemic vs. Rich Domain Model

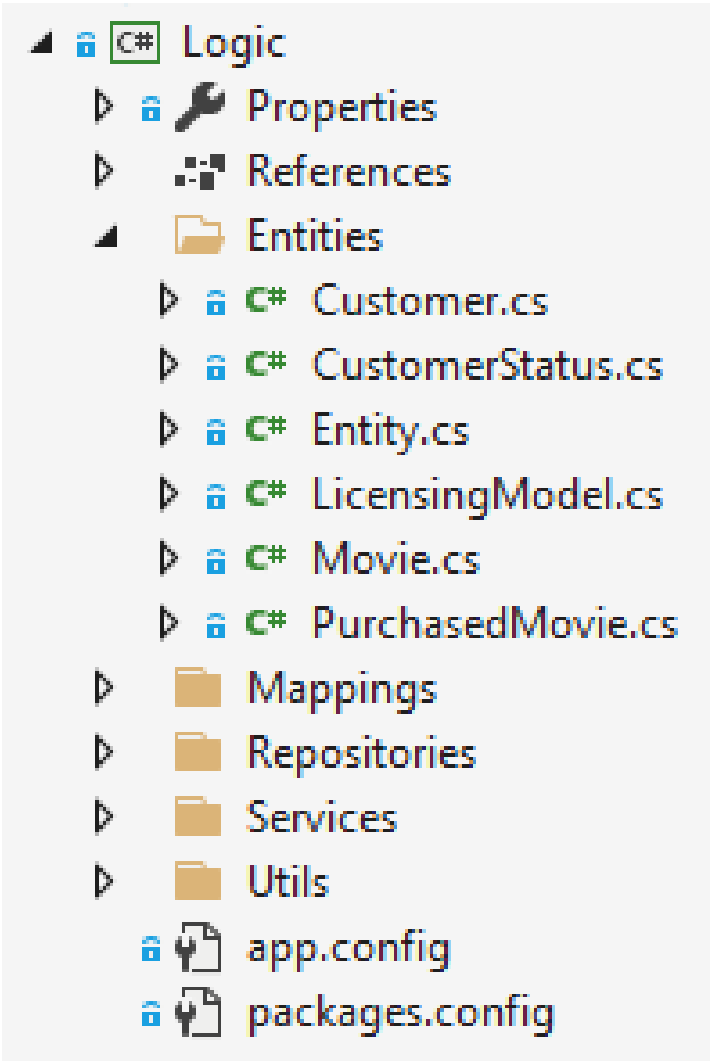
Anemic



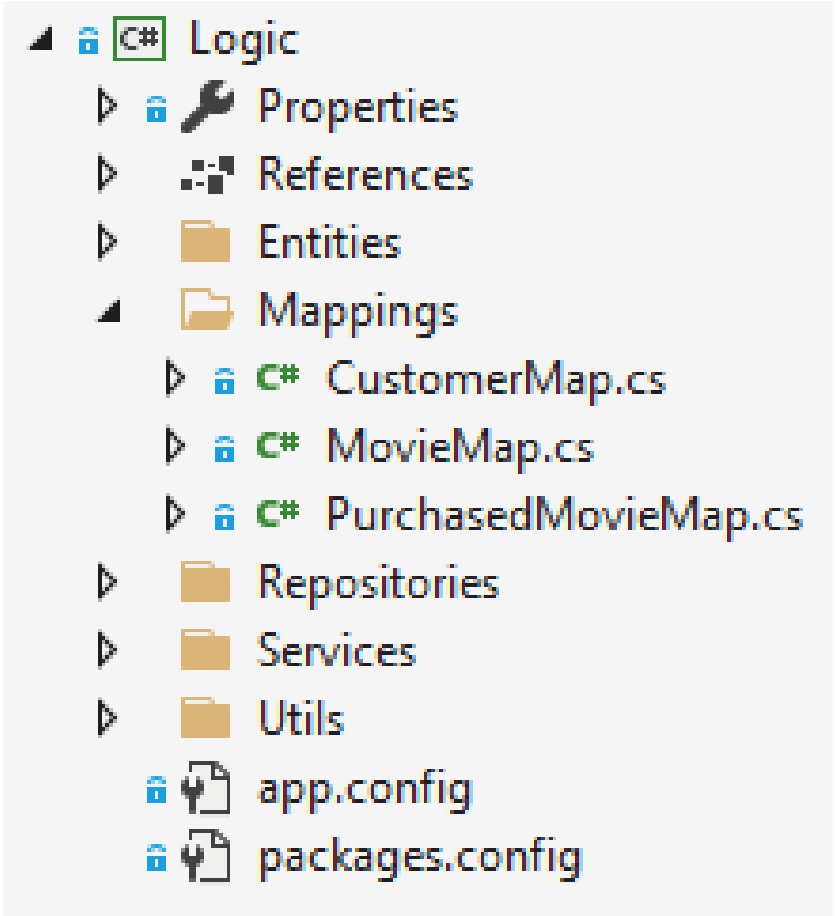
Anemic


Logic
Properties
References
Entities
Mappings
Repositories
Services
Utils
app.config
packages.config

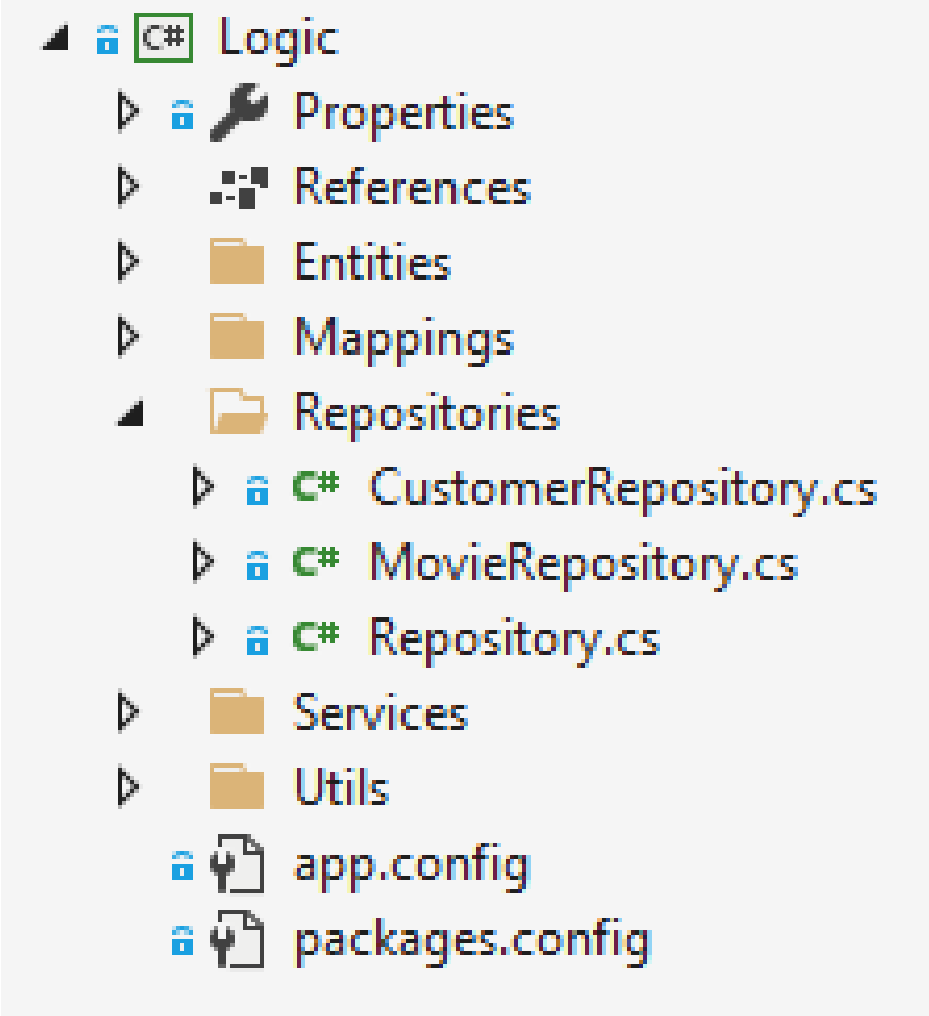
Anemic


Logic
Properties
References
Entities
Customer.cs
CustomerStatus.cs
Entity.cs
LicensingModel.cs
Movie.cs
PurchasedMovie.cs
Mappings
Repositories
Services
Utils
app.config
packages.config

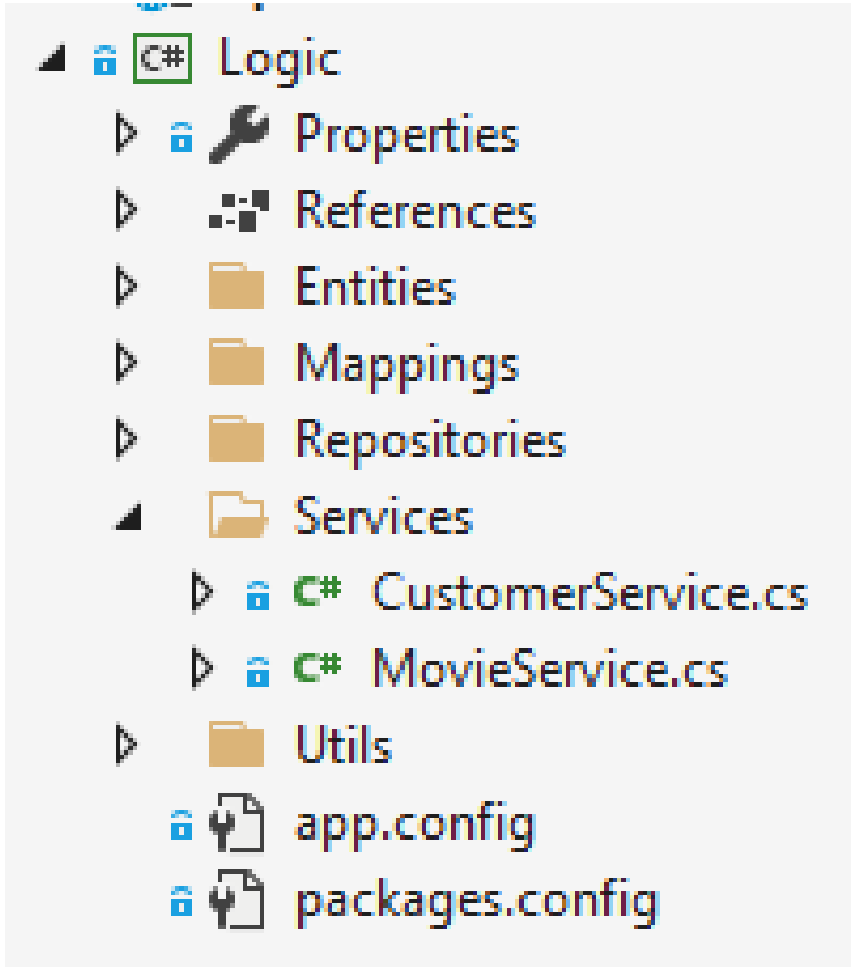
Anemic


Logic
Properties
References
Entities
Mappings
CustomerMap.cs
MovieMap.cs
PurchasedMovieMap.cs
Repositories
Services
Utils
app.config
packages.config

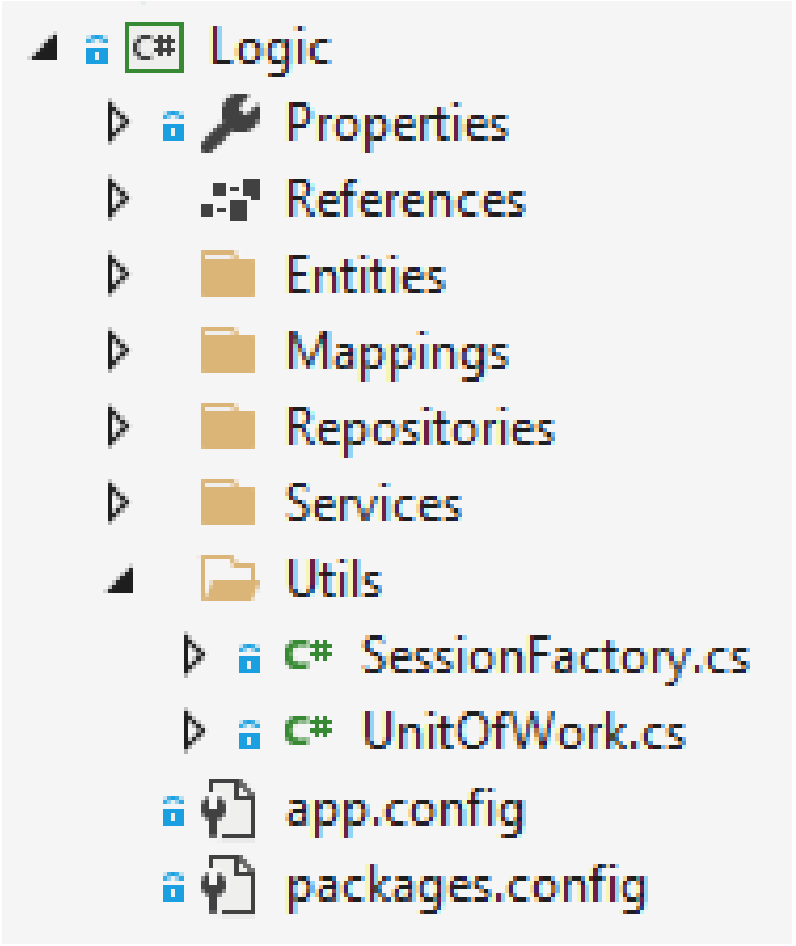
Anemic


Logic
Properties
References
Entities
Mappings
Repositories
CustomerRepository.cs
MovieRepository.cs
Repository.cs
Services
Utils
app.config
packages.config

Anemic


Logic
Properties
References
Entities
Mappings
Repositories
Services
CustomerService.cs
MovieService.cs
Utils
app.config
packages.config

Anemic


Logic
Properties
References
Entities
Mappings
Repositories
Services
Utils
SessionFactory.cs
UnitOfWork.cs
app.config
packages.config

Anemic

```
1  using System;
2  using NHibernate.Proxy;
3
4  namespace Logic.Entities
5  {
       9 references
6      public abstract class Entity
7      {
           12 references
8          public virtual long Id { get; set; }
9
           1 reference
10         public override bool Equals([CanBeNull]object obj)
11         {
12             var other = obj as Entity;
13
14             if (ReferenceEquals(other, null))
15                 return false;
16
17             if (ReferenceEquals(this, other))
18                 return true;
19
20             if (GetRealType() != other.GetRealType())
21                 return false;
22
23             if (Id == 0 || other.Id == 0)
24                 return false;
25
26             return Id == other.Id;
27         }
28
```

Anemic

```
                6 references
29              public static bool operator ==(Entity a, Entity b)
30              {
31                  if (ReferenceEquals(a, null) && ReferenceEquals(b, null))
32                      return true;
33
34                  if (ReferenceEquals(a, null) || ReferenceEquals(b, null))
35                      return false;
36
37                  return a.Equals(b);
38              }
39
                1 reference
40              public static bool operator !=(Entity a, Entity b)
41              {
42                  return !(a == b);
43              }
44
                0 references
↑0 45           public override int GetHashCode()
46              {
47                  return (GetRealType().ToString() + Id).GetHashCode();
48              }
49
                3 references
50              private Type GetRealType()
51              {
52                  return NHibernateProxyHelper.GetClassWithoutInitializingProxy(obj: this);
53              }
54          }
55      }
```

Anemic

```
1  namespace Logic.Entities
2  {
       6 references
3      public enum CustomerStatus
4      {
5          Regular = 1,
6          Advanced = 2
7      }
8  }
```

```
1  namespace Logic.Entities
2  {
       7 references
3      public enum LicensingModel
4      {
5          TwoDays = 1,
6          LifeLong = 2
7      }
8  }
```

Anemic

```
1   using Newtonsoft.Json;
2
3   namespace Logic.Entities
4   {
        7 references
5       public class Movie : Entity
6       {
            2 references
7           public virtual string Name { get; set; }
8
9           [JsonIgnore]
            3 references
10          public virtual LicensingModel LicensingModel { get; set; }
11      }
12  }
```

Anemic

```
1  using System;
2  using Newtonsoft.Json;
3
4  namespace Logic.Entities
5  {
       3 references
6      public class PurchasedMovie : Entity
7      {
8          [JsonIgnore]
           3 references
9          public virtual long MovieId { get; set; }
10
           1 reference
11         public virtual Movie Movie { get; set; }
12
13         [JsonIgnore]
           2 references
14         public virtual long CustomerId { get; set; }
15
           3 references
16         public virtual decimal Price { get; set; }
17
           2 references
18         public virtual DateTime PurchaseDate { get; set; }
19
           6 references
20         public virtual DateTime? ExpirationDate { get; set; }
21     }
22 }
```

Anemic

```
1  using System;
2  using Newtonsoft.Json;
3
4  namespace Logic.Entities
5  {
       3 references
6      public class PurchasedMovie : Entity
7      {
8          [JsonIgnore]
           3 references
9          public virtual long MovieId { get; set; }
10
           1 reference
11         public virtual Movie Movie { get; set; }
12
13         [JsonIgnore]
           2 references
14         public virtual long CustomerId { get; set; }
15
           3 references
16         public virtual decimal Price { get; set; }
17
           2 references
18         public virtual DateTime PurchaseDate { get; set; }
19
           6 references
20         public virtual DateTime? ExpirationDate { get; set; }
21     }
22 }
```

Anemic

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using Newtonsoft.Json;
5  using Newtonsoft.Json.Converters;
6
7  namespace Logic.Entities
8  {
       15 references
9      public class Customer : Entity
10     {
11         [Required]
12         [MaxLength(length: 100, ErrorMessage = "Name is too long")]
           3 references
13         public virtual string Name { get; set; }
14
15         [Required]
16         [RegularExpression(pattern: @"^(.+)@(.+)$", ErrorMessage = "Email is invalid")]
           4 references
17         public virtual string Email { get; set; }
18
19         [JsonConverter(typeof(StringEnumConverter))]
           5 references
20         public virtual CustomerStatus Status { get; set; }
21
           5 references
22         public virtual DateTime? StatusExpirationDate { get; set; }
23
           2 references
24         public virtual decimal MoneySpent { get; set; }
25
           6 references
26         public virtual IList<PurchasedMovie> PurchasedMovies { get; set; }
27     }
28 }
```

Anemic

```
using FluentNHibernate.Mapping;
using Logic.Entities;

namespace Logic.Mappings
{
    1 reference
    public class CustomerMap : ClassMap<Customer>
    {
        0 references
        public CustomerMap()
        {
            Id(memberExpression: x => x.Id);

            Map(memberExpression: x => x.Name);
            Map(memberExpression: x => x.Email);
            Map(memberExpression: x => x.Status).CustomType<int>();
            Map(memberExpression: x => x.StatusExpirationDate).Nullable();
            Map(memberExpression: x => x.MoneySpent);

            HasMany(memberExpression: x => x.PurchasedMovies);
        }
    }
}
```

Anemic

```
1  using FluentNHibernate.Mapping;
2  using Logic.Entities;
3
4  namespace Logic.Mappings
5  {
       1 reference
6      public class MovieMap : ClassMap<Movie>
7      {
           0 references
8          public MovieMap()
9          {
10             Id(memberExpression: x => x.Id);
11
12             Map(memberExpression: x => x.Name);
13             Map(memberExpression: x => x.LicensingModel).CustomType<int>();
14         }
15     }
16 }
```

Anemic

```
1  using FluentNHibernate.Mapping;
2  using Logic.Entities;
3
4  namespace Logic.Mappings
5  {
       1 reference
6      public class PurchasedMovieMap : ClassMap<PurchasedMovie>
7      {
           0 references
8          public PurchasedMovieMap()
9          {
10             Id(memberExpression: x => x.Id);
11
12             Map(memberExpression: x => x.Price);
13             Map(memberExpression: x => x.PurchaseDate);
14             Map(memberExpression: x => x.ExpirationDate).Nullable();
15             Map(memberExpression: x => x.MovieId);
16             Map(memberExpression: x => x.CustomerId);
17
18             References(memberExpression: x => x.Movie).LazyLoad(Laziness.False).ReadOnly();
19         }
20     }
21 }
```

Anemic

```
using Logic.Entities;
using Logic.Utils;

namespace Logic.Repositories
{
    5 references
    public abstract class Repository<T>
        where T : Entity
    {
        protected readonly UnitOfWork _unitOfWork;

        2 references
        protected Repository(UnitOfWork unitOfWork)
        {
            _unitOfWork = unitOfWork;
        }

        5 references
        public T GetById(long id)
        {
            return _unitOfWork.Get<T>(id);
        }

        1 reference
        public void Add(T entity)
        {
            _unitOfWork.SaveOrUpdate(entity);
        }

        4 references
        public void SaveChanges()
        {
            _unitOfWork.Commit();
        }
    }
}
```

Anemic

```
using System.Collections.Generic;
using System.Linq;
using Logic.Entities;
using Logic.Utils;

namespace Logic.Repositories
{
    4 references
    public class CustomerRepository : Repository<Customer>
    {
        0 references
        public CustomerRepository(UnitOfWork unitOfWork)
            : base(unitOfWork)
        {
        }

        1 reference
        public IReadOnlyList<Customer> GetList()
        {
            return _unitOfWork
                .Query<Customer>()
                .ToList()
                .Select(x =>
                {
                    x.PurchasedMovies = null;
                    return x;
                })
                .ToList();
        }

        1 reference
        public Customer GetByEmail(string email)
        {
            return _unitOfWork
                .Query<Customer>()
                .SingleOrDefault(x => x.Email == email);
        }
    }
}
```

Anemic

```
1  using System.Collections.Generic;
2  using System.Linq;
3  using Logic.Entities;
4  using Logic.Utils;
5
6  namespace Logic.Repositories
7  {
       4 references
8      public class MovieRepository : Repository<Movie>
9      {
           0 references
10         public MovieRepository(UnitOfWork unitOfWork)
11             : base(unitOfWork)
12         {
13         }
14
           0 references
15         public IReadOnlyList<Movie> GetList()
16         {
17             return _unitOfWork.Query<Movie>().ToList();
18         }
19     }
20 }
```

Anemic

```
1  using System;
2  using Logic.Entities;
3
4  namespace Logic.Services
5  {
       3 references
6      public class MovieService
7      {
           1 reference
8          public DateTime? GetExpirationDate(LicensingModel licensingModel)
9          {
10             DateTime? result;
11
12             switch (licensingModel)
13             {
14                 case LicensingModel.TwoDays:
15                     result = DateTime.UtcNow.AddDays(2);
16                     break;
17
18                 case LicensingModel.LifeLong:
19                     result = null;
20                     break;
21
22                 default:
23                     throw new ArgumentOutOfRangeException();
24             }
25
26             return result;
27         }
28     }
29 }
```

Anemic

```
1  using System;
2  using System.Linq;
3  using Logic.Entities;
4
5  namespace Logic.Services
6  {
       4 references
7      public class CustomerService
8      {
9          private readonly MovieService _movieService;
10
           0 references
11         public CustomerService(MovieService movieService)
12         {
13             _movieService = movieService;
14         }
15
```

Anemic

```
1 reference
private decimal CalculatePrice(CustomerStatus status, DateTime? statusExpirationDate, LicensingModel licensingModel)
{
    decimal price;
    switch (licensingModel)
    {
        case LicensingModel.TwoDays:
            price = 4;
            break;

        case LicensingModel.LifeLong:
            price = 8;
            break;

        default:
            throw new ArgumentOutOfRangeException();
    }

    if (status == CustomerStatus.Advanced && (statusExpirationDate == null || statusExpirationDate.Value >= DateTime.UtcNow))
    {
        price = price * 0.75m;
    }

    return price;
}
```

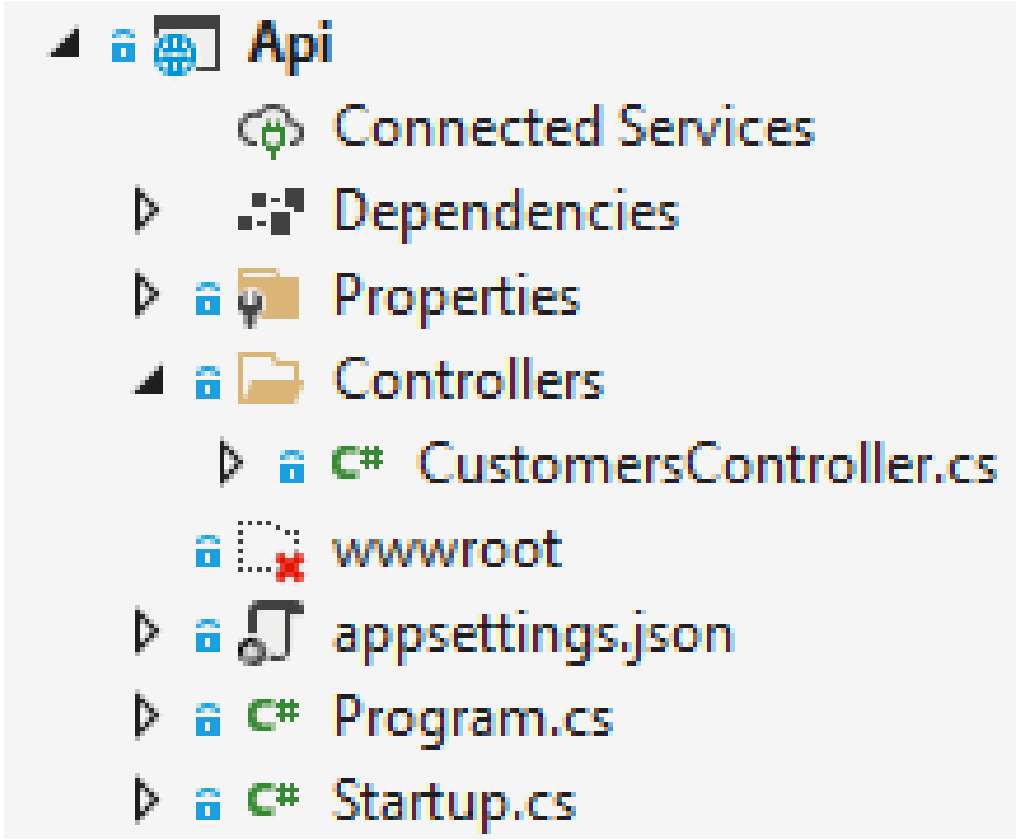
Anemic

```
1 reference
41  public void PurchaseMovie(Customer customer, Movie movie)
42  {
43      DateTime? expirationDate = _movieService.GetExpirationDate(movie.LicensingModel);
44      decimal price = CalculatePrice(customer.Status, customer.StatusExpirationDate, movie.LicensingModel);
45
46      var purchasedMovie = new PurchasedMovie
47      {
48          MovieId = movie.Id,
49          CustomerId = customer.Id,
50          ExpirationDate = expirationDate,
51          Price = price
52      };
53
54      customer.PurchasedMovies.Add(purchasedMovie);
55      customer.MoneySpent += price;
56  }
57
```

Anemic

```
1 reference
public bool PromoteCustomer(Customer customer)
{
    // at least 2 active movies during the last 30 days
    if (customer.PurchasedMovies.Count(x => x.ExpirationDate == null || x.ExpirationDate.Value >= DateTime.UtcNow.AddDays(-30)) < 2)
        return false;

    // at least 100 dollars spent during the last year
    if (customer.PurchasedMovies.Where(x => x.PurchaseDate > DateTime.UtcNow.AddYears(-1)).Sum(x => x.Price) < 100m)
        return false;

    customer.Status = CustomerStatus.Advanced;
    customer.StatusExpirationDate = DateTime.UtcNow.AddYears(1);

    return true;
}
```

Anemic


Api
Connected Services
Dependencies
Properties
Controllers
CustomersController.cs
wwwroot
appsettings.json
Program.cs
Startup.cs

Anemic

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using Logic.Entities;
5   using Logic.Repositories;
6   using Logic.Services;
7   using Microsoft.AspNetCore.Mvc;
8
9   namespace Api.Controllers
10  {
11      [Route(template:"api/[controller]")]

12      public class CustomersController : Controller
13      {
14          private readonly MovieRepository _movieRepository;
15          private readonly CustomerRepository _customerRepository;
16          private readonly CustomerService _customerService;
17

18          public CustomersController(MovieRepository movieRepository, CustomerRepository customerRepository, CustomerService customerService)
19          {
20              _customerRepository = customerRepository;
21              _movieRepository = movieRepository;
22              _customerService = customerService;
23          }
```

Anemic

```
25          [HttpGet]
26          [Route(template:"{id}")]
            0 references
27          public IActionResult Get(long id)
28          {
29              Customer customer = _customerRepository.GetById(id);
30              if (customer == null)
31              {
32                  return NotFound();
33              }
34
35              return Json(customer);
36          }
37
38          [HttpGet]
            0 references
39          public JsonResult GetList()
40          {
41              IReadOnlyList<Customer> customers = _customerRepository.GetList();
42              return Json(customers);
43          }
```

Anemic

```
45      [HttpPost]
        0 references
46      public IActionResult Create([FromBody] Customer item)
47      {
48          try
49          {
50              if (!ModelState.IsValid)
51              {
52                  return BadRequest(ModelState);
53              }
54
55              if (_customerRepository.GetByEmail(item.Email) != null)
56              {
57                  return BadRequest(error:"Email is already in use: " + item.Email);
58              }
59
60              item.Id = 0;
61              item.Status = CustomerStatus.Regular;
62              _customerRepository.Add(item);
63              _customerRepository.SaveChanges();
64
65              return Ok();
66          }
67          catch (Exception e)
68          {
69              return StatusCode(500, value:new { error = e.Message });
70          }
71      }
```

Anemic

```
73      [HttpPut]
74      [Route(template:"{id}")]
        0 references
75      public IActionResult Update(long id, [FromBody] Customer item)
76      {
77          try
78          {
79              if (!ModelState.IsValid)
80              {
81                  return BadRequest(ModelState);
82              }
83
84              Customer customer = _customerRepository.GetById(id);
85              if (customer == null)
86              {
87                  return BadRequest(error:"Invalid customer id: " + id);
88              }
89
90              customer.Name = item.Name;
91              _customerRepository.SaveChanges();
92
93              return Ok();
94          }
95          catch (Exception e)
96          {
97              return StatusCode(500, value:new { error = e.Message });
98          }
99      }
100
```

Anemic

```
101         [HttpPost]
102         [Route(template:"{id}/movies")]
            0 references
103         public IActionResult PurchaseMovie(long id, [FromBody] long movieId)
104         {
105             try
106             {
107                 Movie movie = _movieRepository.GetById(movieId);
108                 if (movie == null)
109                 {
110                     return BadRequest(error:"Invalid movie id: " + movieId);
111                 }
112
113                 Customer customer = _customerRepository.GetById(id);
114                 if (customer == null)
115                 {
116                     return BadRequest(error:"Invalid customer id: " + id);
117                 }
118
119                 if (customer.PurchasedMovies.Any(x => x.MovieId == movie.Id && (x.ExpirationDate == null || x.ExpirationDate.Value >= DateTime.UtcNow)))
120                 {
121                     return BadRequest(error:"The movie is already purchased: " + movie.Name);
122                 }
123
124                 _customerService.PurchaseMovie(customer, movie);
125
126                 _customerRepository.SaveChanges();
127
128                 return Ok();
129             }
130             catch (Exception e)
131             {
132                 return StatusCode(500, value:new { error = e.Message });
133             }
134         }
```

Anemic

```
136     [HttpPost]
137     [Route(template:"{id}/promotion")]
        0 references
138     public IActionResult PromoteCustomer(long id)
139     {
140         try
141         {
142             Customer customer = _customerRepository.GetById(id);
143             if (customer == null)
144             {
145                 return BadRequest(error:"Invalid customer id: " + id);
146             }
147
148             if (customer.Status == CustomerStatus.Advanced && (customer.StatusExpirationDate == null || customer.StatusExpirationDate.Value < DateTime.UtcNow))
149             {
150                 return BadRequest(error:"The customer already has the Advanced status");
151             }
152
153             bool success = _customerService.PromoteCustomer(customer);
154             if (!success)
155             {
156                 return BadRequest(error:"Cannot promote the customer");
157             }
158
159             _customerRepository.SaveChanges();
160
161             return Ok();
162         }
163         catch (Exception e)
164         {
165             return StatusCode(500, value:new { error = e.Message });
166         }
167     }
```

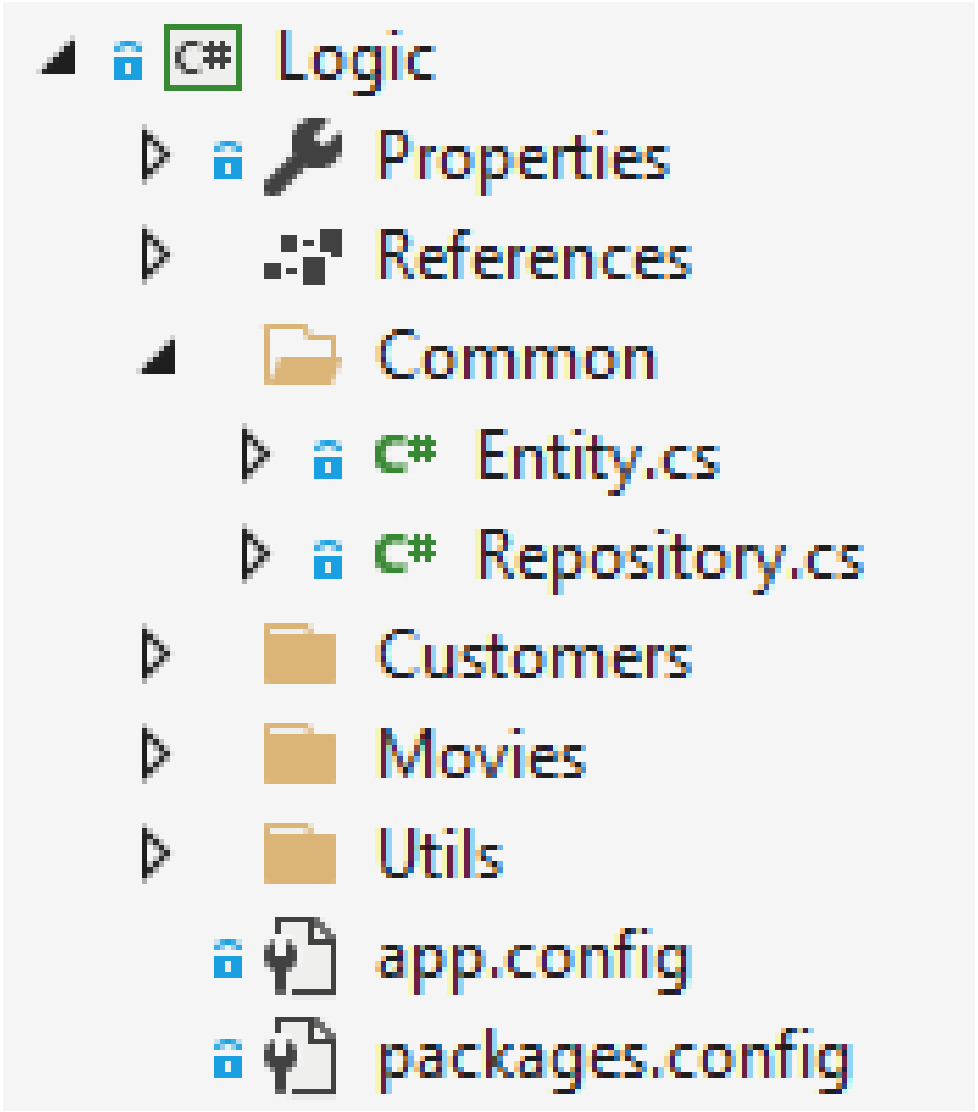
Anemic

```
using Logic.Repositories;
using Logic.Services;
using Logic.Utils;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace Api
{
    2 references
    public class Startup
    {
        0 references
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        2 references
        public IConfiguration Configuration { get; }

        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc();

            services.AddSingleton(new SessionFactory(Configuration["ConnectionString"]));
            services.AddScoped<UnitOfWork>();
            services.AddTransient<MovieRepository>();
            services.AddTransient<CustomerRepository>();
            services.AddTransient<MovieService>();
            services.AddTransient<CustomerService>();
        }

        0 references
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseMvc();
        }
    }
```

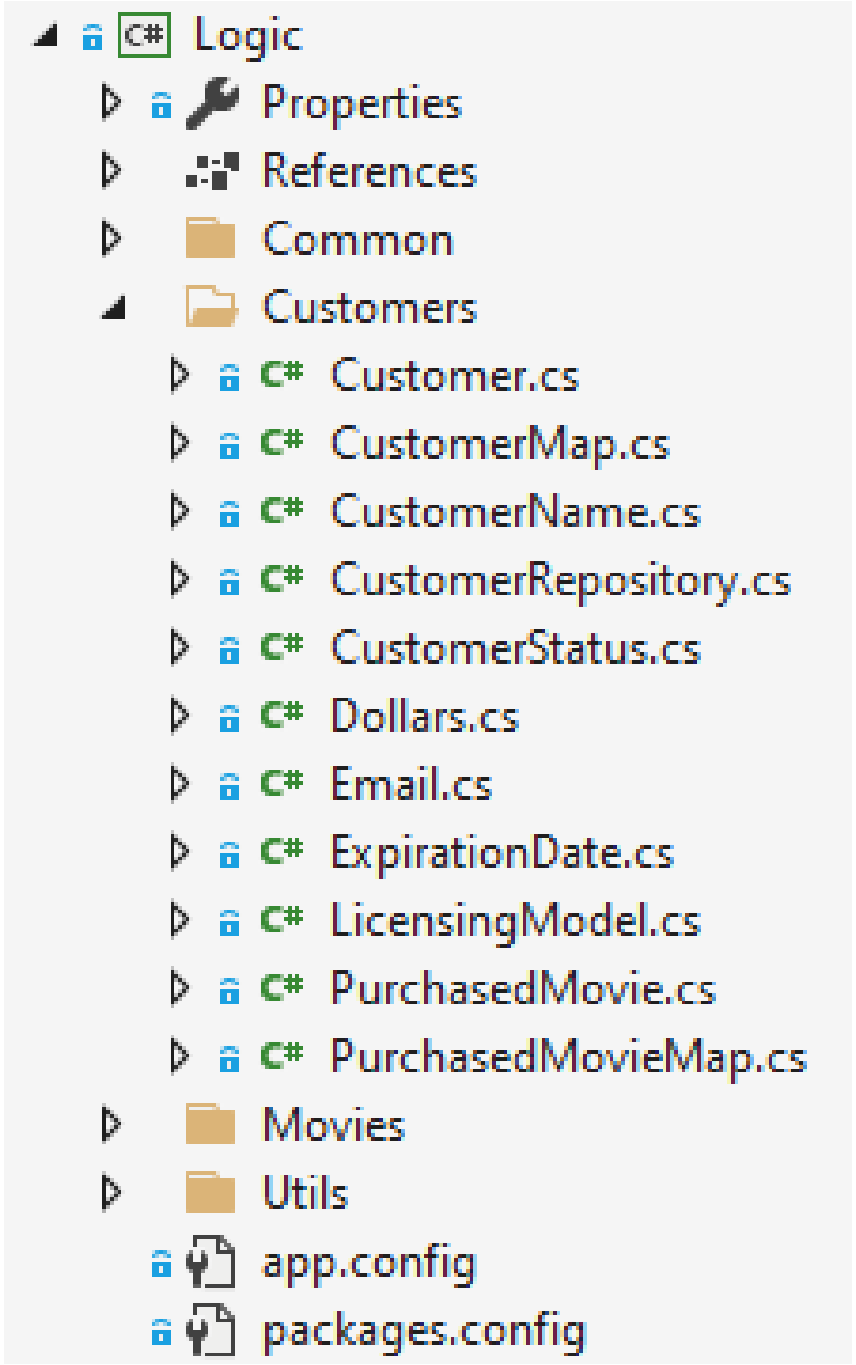
Anemic

```
1  using Microsoft.AspNetCore;
2  using Microsoft.AspNetCore.Hosting;
3
4  namespace Api
5  {
       0 references
6      public class Program
7      {
           0 references
8          public static void Main(string[] args)
9          {
10             BuildWebHost(args).Run();
11         }
12
           1 reference
13         public static IWebHost BuildWebHost(string[] args)
14         {
15             return WebHost.CreateDefaultBuilder(args)
16                 .UseStartup<Startup>()
17                 .Build();
18         }
19     }
20 }
```

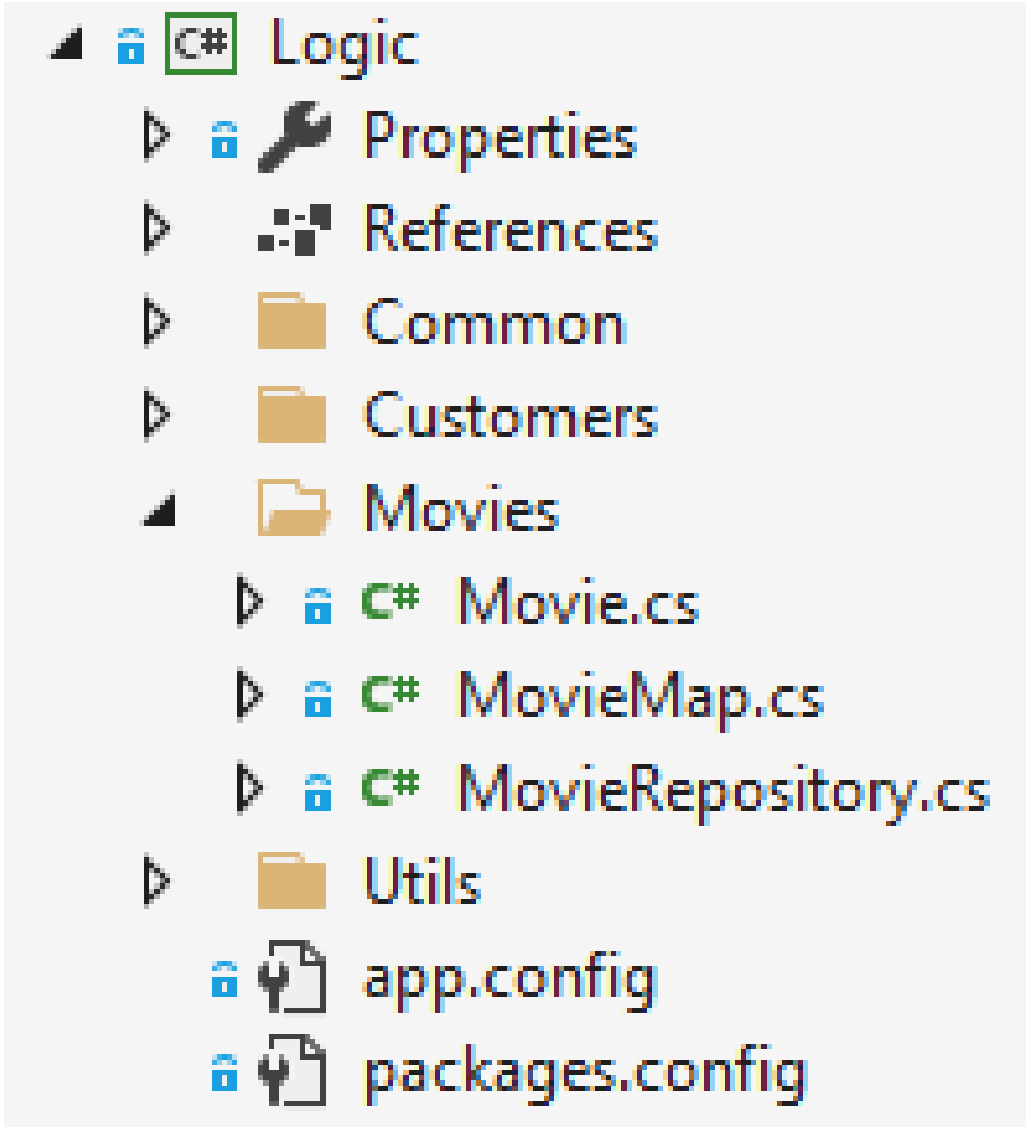
Rich


Logic
Properties
References
Common
Entity.cs
Repository.cs
Customers
Movies
Utils
app.config
packages.config

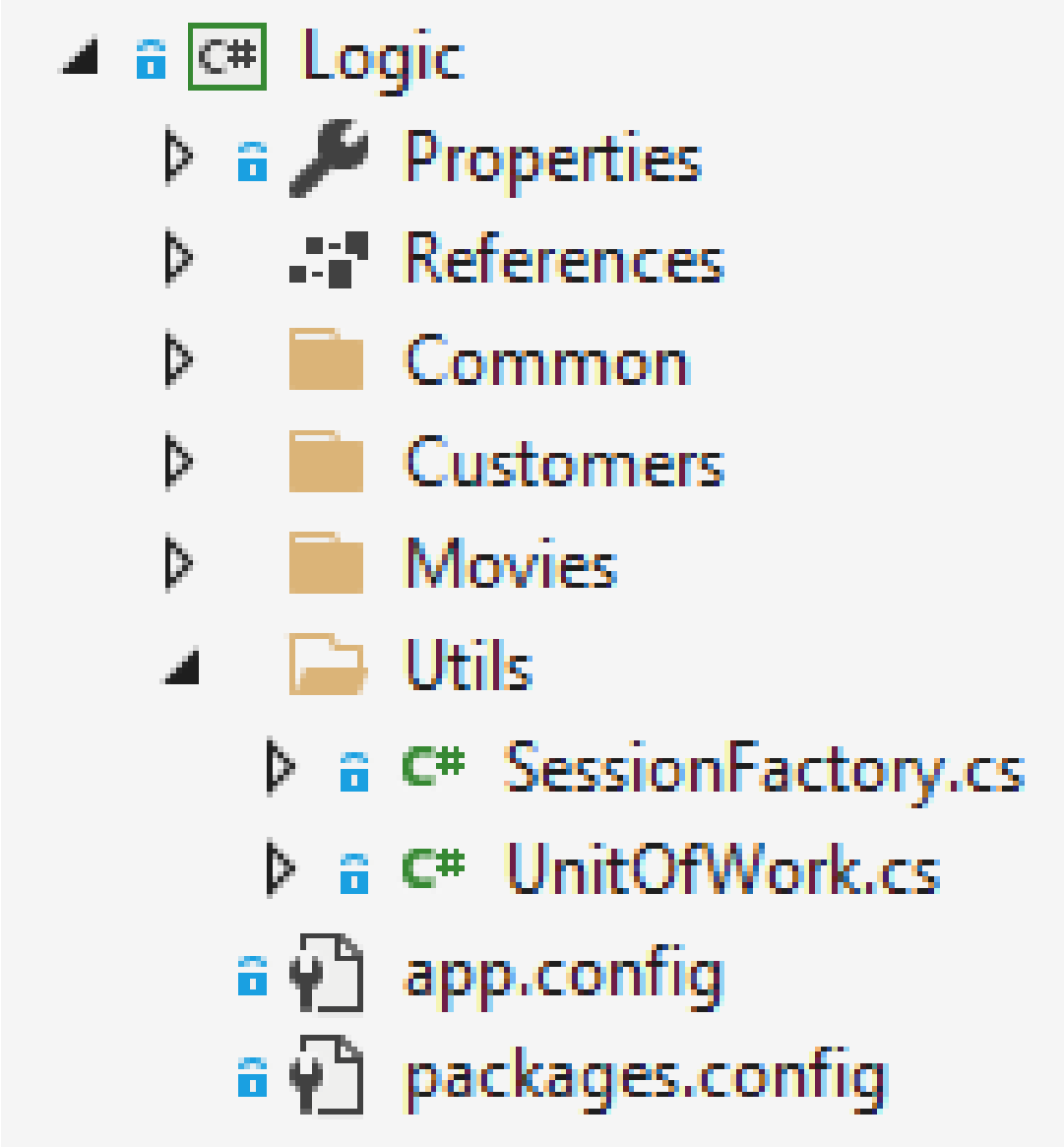
Rich


Logic
Properties
References
Common
Customers
Customer.cs
CustomerMap.cs
CustomerName.cs
CustomerRepository.cs
CustomerStatus.cs
Dollars.cs
Email.cs
ExpirationDate.cs
LicensingModel.cs
PurchasedMovie.cs
PurchasedMovieMap.cs
Movies
Utils
app.config
packages.config

Rich


Logic
Properties
References
Common
Customers
Movies
Movie.cs
MovieMap.cs
MovieRepository.cs
Utils
app.config
packages.config

Rich


Logic
Properties
References
Common
Customers
Movies
Utils
SessionFactory.cs
UnitOfWork.cs
app.config
packages.config

Rich

```
1  using System;
2  using CSharpFunctionalExtensions;
3
4  namespace Logic.Customers
5  {
       16 references
6      public class CustomerName : ValueObject<CustomerName>
7      {
           7 references
8          public string Value { get; }
9
           1 reference
10         private CustomerName(string value)
11         {
12             Value = value;
13         }
14
           3 references
15         public static Result<CustomerName> Create(string customerName)
16         {
17             customerName = (customerName ?? string.Empty).Trim();
18
19             if (customerName.Length == 0)
20                 return Result.Fail<CustomerName>(error: "Customer name should not be empty");
21
22             if (customerName.Length > 50)
23                 return Result.Fail<CustomerName>(error: "Customer name is too long");
24
25             return Result.Ok(new CustomerName(customerName));
26         }
```

Rich

```
                0 references
28              protected override bool EqualsCore(CustomerName other)
29              {
30                  return Value.Equals(other.Value, StringComparison.InvariantCultureIgnoreCase);
31              }
32
                0 references
33              protected override int GetHashCodeCore()
34              {
35                  return Value.GetHashCode();
36              }
37
38              public static implicit operator string(CustomerName customerName)
39              {
40                  return customerName.Value;
41              }
42
43              public static explicit operator CustomerName(string customerName)
44              {
45                  return Create(customerName).Value;
46              }
```

Rich

```
5 references
public enum CustomerStatusType
{
    Regular = 1,
    Advanced = 2
}
```

```
using System;
using CSharpFunctionalExtensions;

namespace Logic.Customers
{
    12 references
    public class CustomerStatus : ValueObject<CustomerStatus>
    {
        public static readonly CustomerStatus Regular = new CustomerStatus(CustomerStatusType.Regular, ExpirationDate.Infinite);

        8 references
        public CustomerStatusType Type { get; }

        private readonly DateTime? _expirationDate;
        7 references
        public ExpirationDate ExpirationDate => (ExpirationDate)_expirationDate;

        2 references
        public bool IsAdvanced => Type == CustomerStatusType.Advanced && !ExpirationDate.IsExpired;

        1 reference
        private CustomerStatus()
        {
        }

        2 references
        private CustomerStatus(CustomerStatusType type, ExpirationDate expirationDate)
            : this()
        {
            Type = type;
            _expirationDate = expirationDate ?? throw new ArgumentNullException(nameof(expirationDate));
        }
```

Rich

```
                1 reference
28              public decimal GetDiscount() => IsAdvanced ? 0.25m : 0m;
29
                1 reference
30              public CustomerStatus Promote()
31              {
32                  return new CustomerStatus(CustomerStatusType.Advanced, (ExpirationDate)DateTime.UtcNow.AddYears(1))
33              }
34
                0 references
35              protected override bool EqualsCore(CustomerStatus other)
36              {
37                  return Type == other.Type && ExpirationDate == other.ExpirationDate;
38              }
39
                0 references
40              protected override int GetHashCodeCore()
41              {
42                  return Type.GetHashCode() ^ ExpirationDate.GetHashCode();
43              }
44          }
```

Rich

```
1   using CSharpFunctionalExtensions;
2
3   namespace Logic.Customers
4   {
        30 references
5       public class Dollars : ValueObject<Dollars>
6       {
7           private const decimal MaxDollarAmount = 1_000_000;
8
            9 references
9           public decimal Value { get; }
10
            1 reference
11          public bool IsZero => Value == 0;
12
            3 references
13          private Dollars(decimal value)
14          {
15              Value = value;
16          }
17
            1 reference
18          public static Result<Dollars> Create(decimal dollarAmount)
19          {
20              if (dollarAmount < 0)
21                  return Result.Fail<Dollars>(error: "Dollar amount cannot be negative");
22
23              if (dollarAmount > MaxDollarAmount)
24                  return Result.Fail<Dollars>(error: "Dollar amount cannot be greater than " + MaxDollarAmount);
25
26              if (dollarAmount % 0.01m > 0)
27                  return Result.Fail<Dollars>(error: "Dollar amount cannot contain part of a penny");
28
29              return Result.Ok(new Dollars(dollarAmount));
30          }
```

Rich

```
        5 references
32      public static Dollars Of(decimal dollarAmount)
33      {
34          return Create(dollarAmount).Value;
35      }
36
        1 reference
37      public static Dollars operator *(Dollars dollars, decimal multiplier)
38      {
39          return new Dollars(dollars.Value * multiplier);
40      }
41
        1 reference
42      public static Dollars operator +(Dollars dollars1, Dollars dollars2)
43      {
44          return new Dollars(dollars1.Value + dollars2.Value);
45      }
46
        0 references
↑0  47  protected override bool EqualsCore(Dollars other)
48      {
49          return Value == other.Value;
50      }
51
        0 references
↑0  52  protected override int GetHashCodeCore()
53      {
54          return Value.GetHashCode();
55      }
56
57      public static implicit operator decimal(Dollars dollars)
58      {
59          return dollars.Value;
60      }
61  }
62  }
```

Rich

```
using System;
using System.Text.RegularExpressions;
using CSharpFunctionalExtensions;

namespace Logic.Customers
{
    16 references
    public class Email : ValueObject<Email>
    {
        8 references
        public string Value { get; }

        1 reference
        private Email(string value)
        {
            Value = value;
        }

        2 references
        public static Result<Email> Create(string email)
        {
            email = (email ?? string.Empty).Trim();

            if (email.Length == 0)
                return Result.Fail<Email>(error: "Email should not be empty");

            if(email.Length > 150)
                return Result.Fail<Email>(error: "Email is too long");

            if (!Regex.IsMatch(input: email, pattern: @"^(.+)@(.+)$"))
                return Result.Fail<Email>(error: "Email is invalid");

            return Result.Ok(new Email(email));
        }
```

Rich

```
                0 references
↑0   32         protected override bool EqualsCore(Email other)
     33         {
     34             return Value.Equals(other.Value, StringComparison.InvariantCultureIgnoreCase);
     35         }
     36
                0 references
↑0   37         protected override int GetHashCodeCore()
     38         {
     39             return Value.GetHashCode();
     40         }
     41
     42         public static explicit operator Email(string email)
     43         {
     44             return Create(email).Value;
     45         }
     46
     47         public static implicit operator string(Email email)
     48         {
     49             return email.Value;
     50         }
     51     }
     52 }
```

Rich

```
1  using System;
2  using CSharpFunctionalExtensions;
3
4  namespace Logic.Customers
5  {
6      public class ExpirationDate : ValueObject<ExpirationDate>
7      {
8          public static readonly ExpirationDate Infinite = new ExpirationDate(date: null);
9
10         public DateTime? Date { get; }
11
12         public bool IsExpired => this != Infinite && Date < DateTime.UtcNow;
13
14         private ExpirationDate(DateTime? date)
15         {
16             Date = date;
17         }
18
19         public static Result<ExpirationDate> Create(DateTime date)
20         {
21             return Result.Ok(new ExpirationDate(date));
22         }
23
```

Rich

```
                0 references
↑0   24         protected override bool EqualsCore(ExpirationDate other)
     25         {
     26             return Date == other.Date;
     27         }
     28
                0 references
↑0   29         protected override int GetHashCodeCore()
     30         {
     31             return Date.GetHashCode();
     32         }
     33
     34         public static explicit operator ExpirationDate(DateTime? date)
     35         {
     36             if (date.HasValue)
     37                 return Create(date.Value).Value;
     38
     39             return Infinite;
     40         }
     41
     42         public static implicit operator DateTime? (ExpirationDate date)
     43         {
     44             return date.Date;
     45         }
     46     }
     47 }
```

Rich

```
1  using System;
2  using Logic.Common;
3  using Logic.Customers;
4
5  namespace Logic.Movies
6  {
       12 references
7      public abstract class Movie : Entity
8      {
           3 references
9          public virtual string Name { get; protected set; }
           0 references
10         protected virtual LicensingModel LicensingModel { get; set; }
11
           3 references
12         public abstract ExpirationDate GetExpirationDate();
13
           1 reference
14         public virtual Dollars CalculatePrice(CustomerStatus status)
15         {
16             decimal modifier = 1 - status.GetDiscount();
17             return GetBasePrice() * modifier;
18         }
19
           3 references
20         protected abstract Dollars GetBasePrice();
21     }
```

Rich

```
1 reference
23  public class TwoDaysMovie : Movie
24  {
        2 references
25      public override ExpirationDate GetExpirationDate()
26      {
27          return (ExpirationDate)DateTime.UtcNow.AddDays(2);
28      }
29
        2 references
30      protected override Dollars GetBasePrice()
31      {
32          return Dollars.Of(4);
33      }
34  }
```

```
1 reference
36  public class LifeLongMovie : Movie
37  {
        2 references
38      public override ExpirationDate GetExpirationDate()
39      {
40          return ExpirationDate.Infinite;
41      }
42
        2 references
43      protected override Dollars GetBasePrice()
44      {
45          return Dollars.Of(8);
46      }
47  }
```

Rich

```
using FluentNHibernate;
using FluentNHibernate.Mapping;

namespace Logic.Movies
{
    1 reference
    public class MovieMap : ClassMap<Movie>
    {
        0 references
        public MovieMap()
        {
            Id(memberExpression: x => x.Id);

            DiscriminateSubClassesOnColumn("LicensingModel");

            Map(memberExpression: x => x.Name);
            Map(Reveal.Member<Movie>(name: "LicensingModel")).CustomType<int>();
        }
    }

    1 reference
    public class TwoDaysMovieMap : SubclassMap<TwoDaysMovie>
    {
        0 references
        public TwoDaysMovieMap()
        {
            DiscriminatorValue(1);
        }
    }

    1 reference
    public class LifeLongMovieMap : SubclassMap<LifeLongMovie>
    {
        0 references
        public LifeLongMovieMap()
        {
            DiscriminatorValue(2);
        }
    }
}
```

Rich

```
using System;
using Logic.Common;
using Logic.Movies;

namespace Logic.Customers
{
    7 references
    public class PurchasedMovie : Entity
    {
        5 references
        public virtual Movie Movie { get; protected set; }
        2 references
        public virtual Customer Customer { get; protected set; }

        private decimal _price;
        4 references
        public virtual Dollars Price
        {
            get => Dollars.Of(_price);
            protected set => _price = value;
        }

        4 references
        public virtual DateTime PurchaseDate { get; protected set; }

        private DateTime? _expirationDate;
        6 references
        public virtual ExpirationDate ExpirationDate
        {
            get => (ExpirationDate)_expirationDate;
            protected set => _expirationDate = value;
        }
```

Rich

```
0 references
protected PurchasedMovie()
{
}

1 reference
internal PurchasedMovie(Movie movie, Customer customer, Dollars price, ExpirationDate expirationDate)
{
    if (price == null || price.IsZero)
        throw new ArgumentException(message: nameof(price));
    if (expirationDate == null || expirationDate.IsExpired)
        throw new ArgumentException(message: nameof(expirationDate));

    Movie = movie ?? throw new ArgumentNullException(nameof(movie));
    Customer = customer ?? throw new ArgumentNullException(nameof(customer));
    Price = price;
    ExpirationDate = expirationDate;
    PurchaseDate = DateTime.UtcNow;
}
```

Rich

```
1  using System;
2  using FluentNHibernate.Mapping;
3
4  namespace Logic.Customers
5  {
       1 reference
6      public class PurchasedMovieMap : ClassMap<PurchasedMovie>
7      {
           0 references
8          public PurchasedMovieMap()
9          {
10             Id(memberExpression: x => x.Id);
11
12             Map(memberExpression: x => x.Price).CustomType<decimal>().Access.CamelCaseField(Prefix.Underscore);
13             Map(memberExpression: x => x.PurchaseDate);
14             Map(memberExpression: x => x.ExpirationDate).CustomType<DateTime?>().Access.CamelCaseField(Prefix.Underscore).Nullable();
15
16             References(memberExpression: x => x.Movie);
17             References(memberExpression: x => x.Customer);
18         }
19     }
20 }
```

Rich

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using CSharpFunctionalExtensions;
5  using Logic.Common;
6  using Logic.Movies;
7
8  namespace Logic.Customers
9  {
       17 references
10     public class Customer : Entity
11     {
12         private string _name;
               4 references
13         public virtual CustomerName Name
14         {
15             get => (CustomerName)_name;
16             set => _name = value;
17         }
18
19         private readonly string _email;
               4 references
20         public virtual Email Email => (Email)_email;
21
               10 references
22         public virtual CustomerStatus Status { get; protected set; }
23
24         private decimal _moneySpent;
               5 references
25         public virtual Dollars MoneySpent
26         {
27             get => Dollars.Of(_moneySpent);
28             protected set => _moneySpent = value;
29         }
30
31         private readonly IList<PurchasedMovie> _purchasedMovies;
               5 references
32         public virtual IReadOnlyList<PurchasedMovie> PurchasedMovies => _purchasedMovies.ToList();
```

Rich

```
33
                1 reference
34              protected Customer()
35              {
36                  _purchasedMovies = new List<PurchasedMovie>();
37              }
38
                1 reference
39              public Customer(CustomerName name, Email email) : this()
40              {
41                  _name = name ?? throw new ArgumentNullException(nameof(name));
42                  _email = email ?? throw new ArgumentNullException(nameof(email));
43
44                  MoneySpent = Dollars.Of(0);
45                  Status = CustomerStatus.Regular;
46              }
```

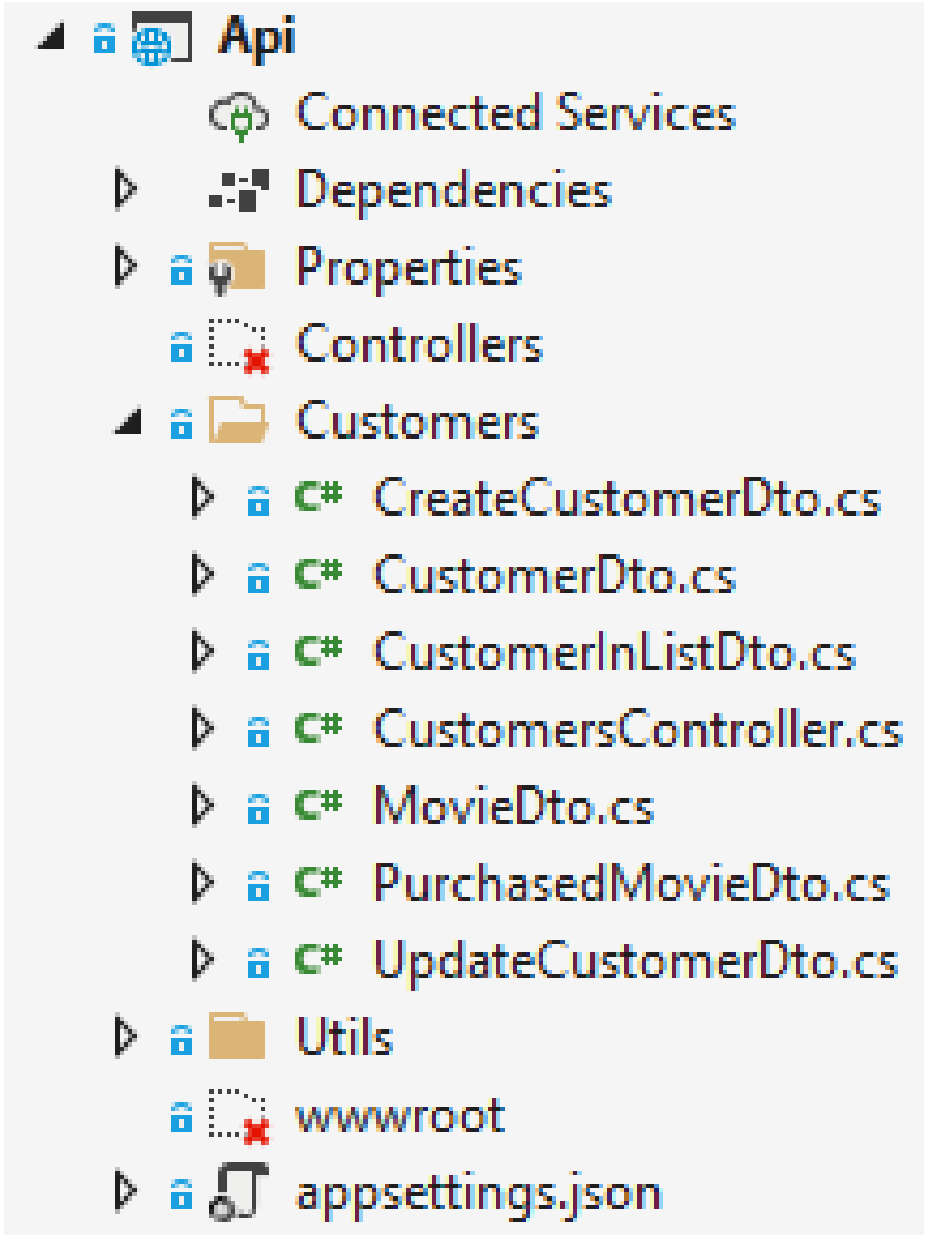
Rich

```
2 references
48  public virtual bool HasPurchasedMovie(Movie movie)
49  {
50      return PurchasedMovies.Any(x => x.Movie == movie && !x.ExpirationDate.IsExpired);
51  }
52
    1 reference
53  public virtual void PurchaseMovie(Movie movie)
54  {
55      if (HasPurchasedMovie(movie))
56          throw new Exception();
57
58      ExpirationDate expirationDate = movie.GetExpirationDate();
59      Dollars price = movie.CalculatePrice(Status);
60
61      var purchasedMovie = new PurchasedMovie(movie, customer: this, price, expirationDate);
62      _purchasedMovies.Add(purchasedMovie);
63
64      MoneySpent += price;
65  }
```

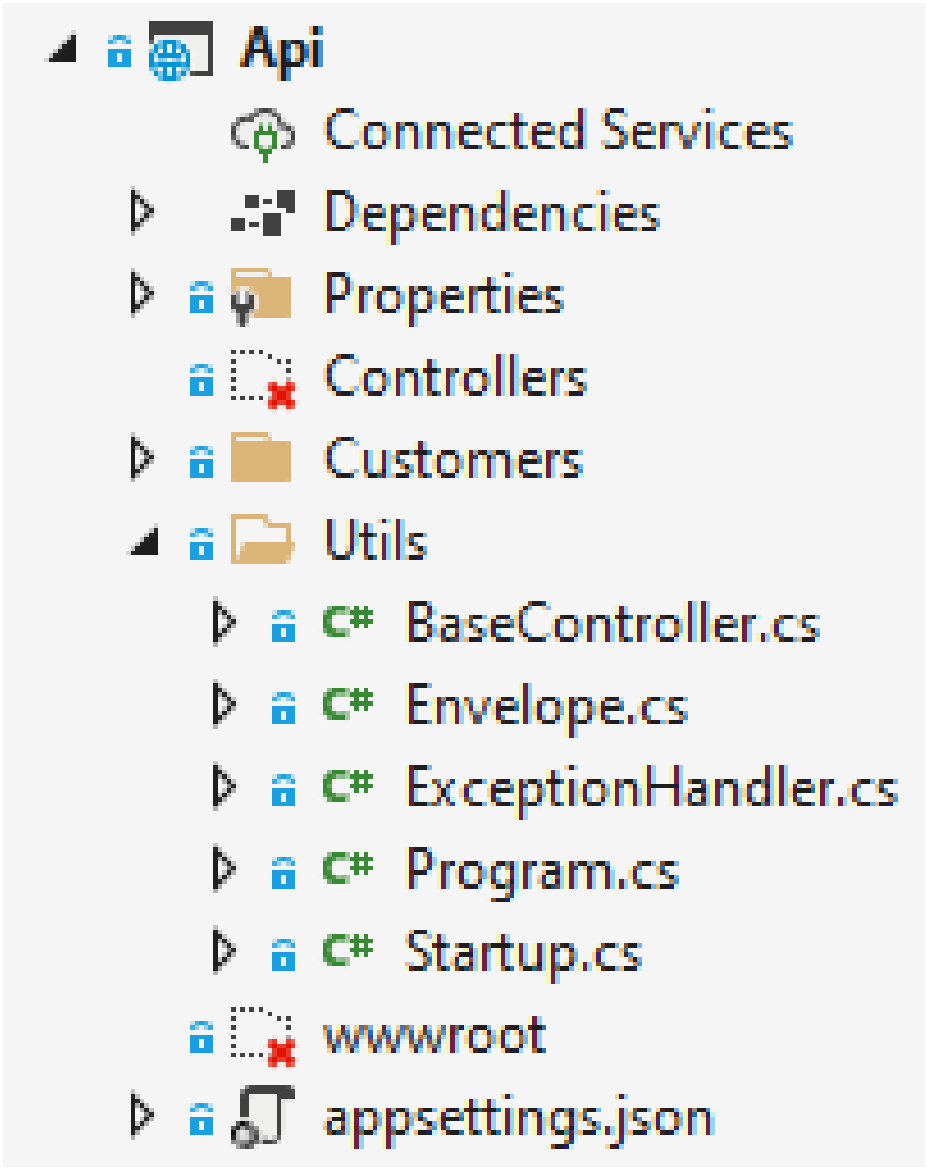
Rich

```
2 references
public virtual Result CanPromote()
{
    if (Status.IsAdvanced)
        return Result.Fail("The customer already has the Advanced status");

    if (PurchasedMovies.Count(x =>
        x.ExpirationDate == ExpirationDate.Infinite || x.ExpirationDate.Date >= DateTime.UtcNow.AddDays(-30)) < 2)
        return Result.Fail("The customer has to have at least 2 active movies during the last 30 days");

    if (PurchasedMovies.Where(x => x.PurchaseDate > DateTime.UtcNow.AddYears(-1)).Sum(x => x.Price) < 100m)
        return Result.Fail("The customer has to have at least 100 dollars spent during the last year");

    return Result.Ok();
}

1 reference
public virtual void Promote()
{
    if (CanPromote().IsFailure)
        throw new Exception();

    Status = Status.Promote();
}
```

Rich


Api
Connected Services
Dependencies
Properties
Controllers
Customers
CreateCustomerDto.cs
CustomerDto.cs
CustomerInListDto.cs
CustomersController.cs
MovieDto.cs
PurchasedMovieDto.cs
UpdateCustomerDto.cs
Utils
wwwroot
appsettings.json

Rich


Api
Connected Services
Dependencies
Properties
Controllers
Customers
Utils
BaseController.cs
Envelope.cs
ExceptionHandler.cs
Program.cs
Startup.cs
wwwroot
appsettings.json

Rich

```
namespace Api.Customers
{
    2 references
    public class MovieDto
    {
        1 reference
        public long Id { get; set; }
        1 reference
        public string Name { get; set; }
    }
}
```

```
using System;

namespace Api.Customers
{
    2 references
    public class PurchasedMovieDto
    {
        1 reference
        public MovieDto Movie { get; set; }
        1 reference
        public decimal Price { get; set; }
        1 reference
        public DateTime PurchaseDate { get; set; }
        1 reference
        public DateTime? ExpirationDate { get; set; }
    }
}
```

Rich

```
1  using System;
2  using System.Collections.Generic;
3
4  namespace Api.Customers
5  {
       1 reference
6      public class CustomerDto
7      {
           1 reference
8          public long Id { get; set; }
           1 reference
9          public string Name { get; set; }
           1 reference
10         public string Email { get; set; }
           1 reference
11         public string Status { get; set; }
           1 reference
12         public DateTime? StatusExpirationDate { get; set; }
           1 reference
13         public decimal MoneySpent { get; set; }
           1 reference
14         public List<PurchasedMovieDto> PurchasedMovies { get; set; }
15     }
16 }
```

Rich

```
namespace Api.Customers
{
    1 reference
    public class UpdateCustomerDto
    {
        1 reference
        public string Name { get; set; }
    }
}
```

```
namespace Api.Customers
{
    1 reference
    public class CreateCustomerDto
    {
        1 reference
        public string Name { get; set; }
        2 references
        public string Email { get; set; }
    }
}
```

```
using System;

namespace Api.Customers
{
    2 references
    public class CustomerInListDto
    {
        1 reference
        public long Id { get; set; }
        1 reference
        public string Name { get; set; }
        1 reference
        public string Email { get; set; }
        1 reference
        public string Status { get; set; }
        1 reference
        public DateTime? StatusExpirationDate { get; set; }
        1 reference
        public decimal MoneySpent { get; set; }
    }
}
```

Rich

```
1  using Logic.Utils;
2  using Microsoft.AspNetCore.Mvc;
3
4  namespace Api.Utils
5  {
       3 references
6      public class BaseController : Controller
7      {
8          private readonly UnitOfWork _unitOfWork;
9
               1 reference
10         public BaseController(UnitOfWork unitOfWork)
11         {
12             _unitOfWork = unitOfWork;
13         }
14
               4 references
15         protected new IActionResult Ok()
16         {
17             _unitOfWork.Commit();
18             return base.Ok(Envelope.Ok());
19         }
20
               2 references
21         protected IActionResult Ok<T>(T result)
22         {
23             _unitOfWork.Commit();
24             return base.Ok(Envelope.Ok(result));
25         }
26
               9 references
27         protected IActionResult Error(string errorMessage)
28         {
29             return BadRequest(Envelope.Error(errorMessage));
30         }
31     }
32 }
```

Rich

```
1  using System.Collections.Generic;
2  using System.Linq;
3  using Api.Utils;
4  using CSharpFunctionalExtensions;
5  using Logic.Customers;
6  using Logic.Movies;
7  using Logic.Utils;
8  using Microsoft.AspNetCore.Mvc;
9
10 namespace Api.Customers
11 {
12     [Route(template:"api/[controller]")]
       1 reference
13     public class CustomersController : BaseController
14     {
15         private readonly MovieRepository _movieRepository;
16         private readonly CustomerRepository _customerRepository;
17
           0 references
18         public CustomersController(UnitOfWork unitOfWork, MovieRepository movieRepository, CustomerRepository customerRepository)
19             : base(unitOfWork)
20         {
21             _customerRepository = customerRepository;
22             _movieRepository = movieRepository;
23         }
```

Rich

```
25      [HttpGet]
26      [Route(template:"{id}")]
        0 references
27      public IActionResult Get(long id)
28      {
29          Customer customer = _customerRepository.GetById(id);
30          if (customer == null)
31              return NotFound();
32
33          var dto = new CustomerDto
34          {
35              Id = customer.Id,
36              Name = customer.Name.Value,
37              Email = customer.Email.Value,
38              MoneySpent = customer.MoneySpent,
39              Status = customer.Status.Type.ToString(),
40              StatusExpirationDate = customer.Status.ExpirationDate,
41              PurchasedMovies = customer.PurchasedMovies.Select(x => new PurchasedMovieDto
42              {
43                  Price = x.Price,
44                  ExpirationDate = x.ExpirationDate,
45                  PurchaseDate = x.PurchaseDate,
46                  Movie = new MovieDto
47                  {
48                      Id = x.Movie.Id,
49                      Name = x.Movie.Name
50                  }
51              }).ToList()
52          };
53
54          return Ok(dto);
55      }
```

Rich

```
57          [HttpGet]
            0 references
58          public IActionResult GetList()
59          {
60              IReadOnlyList<Customer> customers = _customerRepository.GetList();
61
62              List<CustomerInListDto> dtos = customers.Select(x => new CustomerInListDto
63              {
64                  Id = x.Id,
65                  Name = x.Name.Value,
66                  Email = x.Email.Value,
67                  MoneySpent = x.MoneySpent,
68                  Status = x.Status.Type.ToString(),
69                  StatusExpirationDate = x.Status.ExpirationDate
70              }).ToList();
71
72              return Ok(dtos);
73          }
```

Rich

```
75          [HttpPost]
            0 references
76          public IActionResult Create([FromBody] CreateCustomerDto item)
77          {
78              Result<CustomerName> customerNameOrError = CustomerName.Create(item.Name);
79              Result<Email> emailOrError = Email.Create(item.Email);
80
81              Result result = Result.Combine(customerNameOrError, emailOrError);
82              if (result.IsFailure)
83                  return Error(result.Error);
84
85              if (_customerRepository.GetByEmail(emailOrError.Value) != null)
86                  return Error("Email is already in use: " + item.Email);
87
88              var customer = new Customer(customerNameOrError.Value, emailOrError.Value);
89              _customerRepository.Add(customer);
90
91              return Ok();
92          }
```

Rich

```
94          [HttpPut]
95          [Route(template:"{id}")]
            0 references
96          public IActionResult Update(long id, [FromBody] UpdateCustomerDto item)
97          {
98              Result<CustomerName> customerNameOrError = CustomerName.Create(item.Name);
99              if (customerNameOrError.IsFailure)
100                 return Error(customerNameOrError.Error);
101
102             Customer customer = _customerRepository.GetById(id);
103             if (customer == null)
104                 return Error("Invalid customer id: " + id);
105
106             customer.Name = customerNameOrError.Value;
107
108             return Ok();
109         }
```

Rich

```
111         [HttpPost]
112         [Route(template:"{id}/movies")]
            0 references
113         public IActionResult PurchaseMovie(long id, [FromBody] long movieId)
114         {
115             Movie movie = _movieRepository.GetById(movieId);
116             if (movie == null)
117                 return Error("Invalid movie id: " + movieId);
118
119             Customer customer = _customerRepository.GetById(id);
120             if (customer == null)
121                 return Error("Invalid customer id: " + id);
122
123             if (customer.HasPurchasedMovie(movie))
124                 return Error("The movie is already purchased: " + movie.Name);
125
126             customer.PurchaseMovie(movie);
127
128             return Ok();
129         }
```

Rich

```
131         [HttpPost]
132         [Route(template:"{id}/promotion")]
            0 references
133         public IActionResult PromoteCustomer(long id)
134         {
135             Customer customer = _customerRepository.GetById(id);
136             if (customer == null)
137                 return Error("Invalid customer id: " + id);
138
139             Result promotionCheck = customer.CanPromote();
140             if (promotionCheck.IsFailure)
141                 return Error(promotionCheck.Error);
142
143             customer.Promote();
144
145             return Ok();
146         }
```

Rich

```
1  using Logic.Customers;
2  using Logic.Movies;
3  using Logic.Utils;
4  using Microsoft.AspNetCore.Builder;
5  using Microsoft.Extensions.Configuration;
6  using Microsoft.Extensions.DependencyInjection;
7
8  namespace Api.Utils
9  {
       2 references
10     public class Startup
11     {
           0 references
12         public Startup(IConfiguration configuration)
13         {
14             Configuration = configuration;
15         }
16
           2 references
17         public IConfiguration Configuration { get; }
18
           0 references
19         public void ConfigureServices(IServiceCollection services)
20         {
21             services.AddMvc();
22
23             services.AddSingleton(new SessionFactory(Configuration["ConnectionString"]));
24             services.AddScoped<UnitOfWork>();
25             services.AddTransient<MovieRepository>();
26             services.AddTransient<CustomerRepository>();
27         }
28
           0 references
29         public void Configure(IApplicationBuilder app)
30         {
31             app.UseMiddleware<ExceptionHandler>();
32             app.UseMvc();
33         }
34     }
35 }
```

Спасибо за внимание!