Module Logic:
Microsoft.EntityFrameworkCore
Microsoft.EntityFrameworkCore.SqlServer
System.Linq.Dynamic.Core

Module ConApp:
Microsoft.EntityFrameworkCore.Tools
Microsoft.EntityFrameworkCore.Design

ConnectionString In der Klasse 'ProjectDbContext' muss der ConnctionString gesetzt werden: ConnectionString = "Data Source=(localdb)\MSSQLLocalDB;Database=SmartNQuickDb;Integrated Security=True";

Schritt 1:

Contracts -> Persistence -> Ordner für Entity erstellen -> Interface für Entity erstellen

Schritt 2:

Logic -> Entities -> Ordner für Entity erstellen -> Entity erstellen

Schritt 3:

Logic -> DataContext -> ProjectDbContextEx.cs -> DbSet für Entity erstellen

Logic -> DataContext -> ProjectDbContextEx.cs -> GetDbSet erstellen

```
partial void GetDbSet<C, E>(ref DbSet<E> dbset) where E : class
{

    if (typeof(C) == typeof(ClientCorntracts.Persistence.Creditcard.ICreditcard))
    {
        dbset = Creditcards as DbSet<E>;
    }
}
```

Logic -> DataContext -> ProjectDbContextEx.cs -> BeforeOnModelCreating erstellen

```
partial void BeforeOnModelCreating(ModelBuilder modelBuilder, ref bool handled)
{


    var creditcardBuilder = modelBuilder.Entity<Entities.Creditcard.Creditcard>();

    creditcardBuilder.HasKey(p => p.Id);
    creditcardBuilder.Property(p => p.RowVersion).IsRowVersion();
    creditcardBuilder.Property(p => p.CreditcardNumber).IsRequired();
    creditcardBuilder.HasIndex(p => p.CreditcardNumber).IsUnique();
}
```

Schritt 4:

Logic -> Controllers -> Persistence -> Ordner für Entity erstellen -> Controller für Entity erstellen

```
4 references
class CreditcardController : GenericPersistenceController<ICreditcard, Logic.Entities.Creditcard.Creditcard>
{
    1 reference
    public CreditcardController(ControllerObject other) : base(other)
    {
    }

    1 reference
    public CreditcardController(IContext context) : base(context)
    {
    }

    3 references
    protected override Entities.Creditcard.Creditcard BeforeInsert(Entities.Creditcard.Creditcard entity)
    {

        if(CreditcardLogic.CheckCreditcard(entity.CreditcardNumber.ToString()))
            return base.BeforeInsert(entity);
        throw new ArgumentException("Ich heiße Paul Pils und habe dieses Programm von Github kopiert!");
    }

    3 references
    protected override Entities.Creditcard.Creditcard BeforeUpdate(Entities.Creditcard.Creditcard entity)
    {
        if(CreditcardLogic.CheckCreditcard(entity.CreditcardNumber.ToString()))
            return base.BeforeUpdate(entity);
        throw new ArgumentException("Ich heiße Paul Pils und habe dieses Programm von Github kopiert!");
    }

}
```

Schritt 5:

Package Manager Console (links unten) -> Default Project -> Logic auswählen -> ConApp als StartUpPorject -> add-migration initDb -> update-database

Schritt 6:

Logic -> FactoryEx.cs -> erstellen

```
2 references
static partial void CreateController<C>(IContext context, ref IControllerAccess<C> controller) where C : IIdentifiable
{
    if (typeof(C) == typeof(ICreditcard))
    {
        controller = new Controllers.Persistence.Creditcard.CreditcardController(context) as IControllerAccess<C>;
    }
}
2 references
static partial void CreateController<C>(ControllerObject controllerObject, ref IControllerAccess<C> controller) where C : IIdentifiable
{
    controllerObject.CheckArgument(nameof(controllerObject));

    if (typeof(C) == typeof(ICreditcard))
    {
        controller = new Controllers.Persistence.Creditcard.CreditcardController(controllerObject) as IControllerAccess<C>;
    }
}
```

Schritt 7:

Logic -> Controllers -> Business -> <Entity>Controller.cs erstellen

```
public static class CreditcardLogic
{
    7 references | 5/5 passing
    public static bool CheckCreditcard(string ssn)
    {
        int odd = 0;
        int even = 0;
        int both = 0;
        if (ssn == null)
        {
            throw new ArgumentNullException();
        }
        if (ssn.Length != 16)
        {
            return false;
        }
        for (int i = 0; i < ssn.Length - 1; i++)
        {
            if (!Char.IsDigit(ssn[i]))
            {
                return false;
            }
            if (i % 2 == 0)
            {
                int quer = Convert.ToInt32(ssn[i].ToString()) * 2;
                if (quer >= 10)
                {
                    quer = quer.ToString().Sum(c => c - '0');
                }
                even += quer;

            }
            else
            {
                odd += Convert.ToInt32(ssn[i].ToString());
            }

        }
        both = even + odd;
        int i = both % 10;
```

Schritt 8:

WebApi -> Controllers -> Persistence -> Ordner für Entity erstellen -> <Entity>Controller.cs erstellen

```
[Route("api/[controller]")]
[ApiController]
0 references
public class CreditcardController : GenericController<ICreditcard, Transfer.Models.Persistence.Creditcard.Creditcard>
{

}
```

Schritt 9:

Transfer -> Models -> Persistence -> Ordner für Entity erstellen -> <Entity>.cs

```
1 reference
public partial class Creditcard : VersionModel, Contracts.Persistence.Creditcard.ICreditcard
{
    6 references
    public long CreditcardNumber { get; set; }

    5 references
    public void CopyProperties(ICreditcard other)
    {
        other.CheckArgument(nameof(other));

        Id = other.Id;
        RowVersion = other.RowVersion;
        CreditcardNumber = other.CreditcardNumber;
    }
}
```

Schritt 10:

AspMvc -> Models -> Ordner für Entity erstellen -> <Entity>.cs

```
public class Creditcard : VersionModel, ICreditcard
{
    7 references
    public long CreditcardNumber { get; set; }

    5 references
    public void CopyProperties(ICreditcard other)
    {
        other.CheckArgument(nameof(other));

        Id = other.Id;
        RowVersion = other.RowVersion;
        CreditcardNumber = other.CreditcardNumber;
    }
}
```

Schritt 11:

AspMvc -> Controllers -> rechts Klick -> Add -> Controller -> Empty

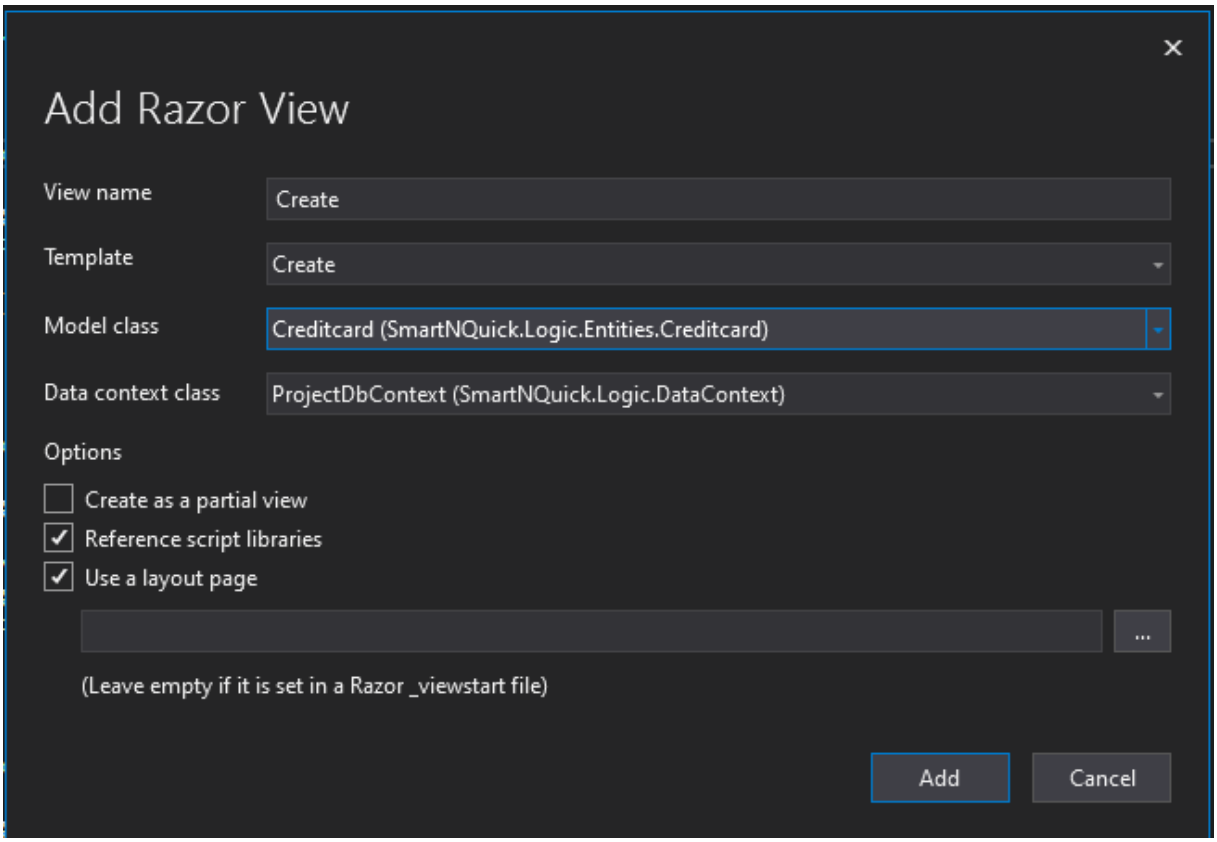
```
0 references
public class CreditcardController : GenericModelController<ICreditcard, AspMvc.Models.Creditcard.Creditcard>
{
    [HttpGet]
    0 references
    public async Task<IActionResult> Create()
    {
        using var ctrl = Logic.Factory.Create<Contracts.Persistence.Creditcard.ICreditcard>();
        var entity = await ctrl.CreateAsync().ConfigureAwait(false);

        return View(ToModel(entity));
    }

    [HttpPost]
    0 references
    public async Task<IActionResult> Insert(Creditcard model)
    {
        using var ctrl = Logic.Factory.Create<ICreditcard>();

        await ctrl.InsertAsync(model).ConfigureAwait(false);
        await ctrl.SaveChangesAsync().ConfigureAwait(false);
        return RedirectToAction("Index");
    }

    [HttpGet]
    0 references
    public async Task<IActionResult> Edit(int id)
    {
        using var ctrl = Logic.Factory.Create<ICreditcard>();
        var entity = await ctrl.GetByIdAsync(id).ConfigureAwait(false);

        return View(ToModel(entity));
    }
```

```
[HttpPost]
0 references
public async Task<IActionResult> Update(Creditcard model)
{
    using var ctrl = Logic.Factory.Create<ICreditcard>();
    var entity = await ctrl.GetByIdAsync(model.Id).ConfigureAwait(false);

    if (entity != null)
    {
        entity.CreditcardNumber = model.CreditcardNumber;
        await ctrl.UpdateAsync(entity).ConfigureAwait(false);
        await ctrl.SaveChangesAsync().ConfigureAwait(false);
    }
    return RedirectToAction("Index");
}
[HttpGet]
0 references
public async Task<IActionResult> Delete(int id)
{
    using var ctrl = Logic.Factory.Create<ICreditcard>();
    var entity = await ctrl.GetByIdAsync(id).ConfigureAwait(false);

    return View(ToModel(entity));
}

0 references
public async Task<IActionResult> DeleteEntity(int id)
{
    using var ctrl = Logic.Factory.Create<ICreditcard>();

    await ctrl.DeleteAsync(id).ConfigureAwait(false);
    await ctrl.SaveChangesAsync().ConfigureAwait(false);

    return RedirectToAction("Index");
}
```

Schritt 12:

AspMvc -> Controllers -> <Entity>Controller.cs -> Create() -> rechts klick -> Add View -> Razor View -> Add

bei create nochmal view erstellen! (index) 4.Auswahl als List



Schritt 13:

AspMvc -> Views -> in den Entity Ordner gehen (wurde davor erstellt) -> 1. Zeile -> Pfad ändern

```
@model SmartNQuick.AspMvc.Models.Creditcard.Creditcard
```

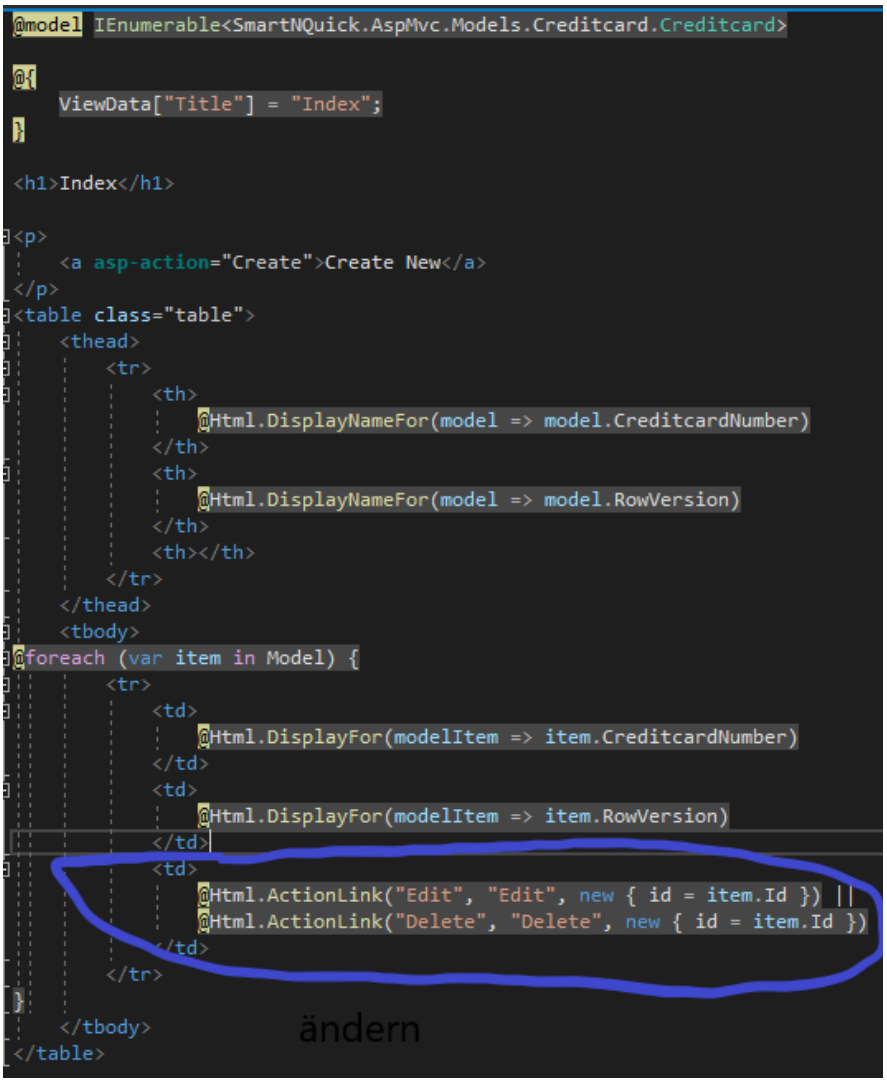
Schritt 14:

AspMvc -> Views -> in den Entity Ordner gehen (wurde davor erstellt) -> Create/Delete/Edit/Index -> ändern

```
@model SmartNQuick.AspMvc.Models.Creditcard.Creditcard

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

<h1>Create</h1>

<h4>Creditcard</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Insert" method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="CreditcardNumber" class="control-label"></label>
                <input asp-for="CreditcardNumber" class="form-control" />
                <span asp-validation-for="CreditcardNumber" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

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

```
@model SmartNQuick.AspMvc.Models.Creditcard.Creditcard

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

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Creditcard</h4>
    <hr />
    <dl class="row">
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.CreditcardNumber)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.CreditcardNumber)
        </dd>
        <dt class = "col-sm-2">
            @Html.DisplayNameFor(model => model.RowVersion)
        </dt>
        <dd class = "col-sm-10">
            @Html.DisplayFor(model => model.RowVersion)
        </dd>
    </dl>

    form asp-action="DeleteEntity">
        <input type="hidden" id="Id" asp-for="Id" value="@Model.Id" />
        <input                                                    />
        <a asp-action="Index">Back to List</a>
    </form>
</div>
```

```
@model SmartNQuick.AspMvc.Models.Creditcard.Creditcard

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

<h1>Edit</h1>

<h4>Creditcard</h4>
<hr />
<div class="row">
    <div
        <form asp-action="Update" method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="CreditcardNumber" class="control-label"></label>
                <input asp-for="CreditcardNumber" class="form-control" />
                <span asp-validation-for="CreditcardNumber" class="text-danger"></span>
            </div>
            <input type="hidden" asp-for="Id" />
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

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



Schritt 15:

AspMvc -> Views -> Shared -> _Layout -> ändern

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - SmartNQuick.AspMvc</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">SmartNQuick.AspMvc</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Genre" asp-action="Index">Genre</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Creditcard" asp-action="Index">Creditcard</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <footer class="border-top footer text-muted">
```

ADDEN

Schritt 16:

Neues Projekt -> Unit Test Project (.NET Core) -> Klasse erstellen

```
namespace SmartNQuick.Logic.UnitTests
{
    [TestClass]
    0 references
    public class BookLibraryBusinessTest
    {
        [TestMethod]
        0 references
        public void Validate_SetNull_False()
        {
            var expected = false;
            var actual = Business.ISBNBusiness.validateISBN(null);

            Assert.AreEqual(expected, actual);

        }

        [TestMethod]
        0 references
        public void Validate_ToShortNumber_False()
        {
            var expected = false;
            var actual = Business.ISBNBusiness.validateISBN("1234");

            Assert.AreEqual(expected, actual);
        }

        [TestMethod]
        0 references
        public void Validate_OnlyDigit_True()
        {
            var expected = true;
            var actual = Business.ISBNBusiness.validateISBN("1234567890");

            Assert.AreEqual(expected, actual);
        }
    }
}
```