HoGent

BEDRIJF
EN
ORGANISATIE

Hoofdstuk 8: ASP.NET MVC Core

https://github.com/WebIII/08thBeerhallMvcCRUD.git

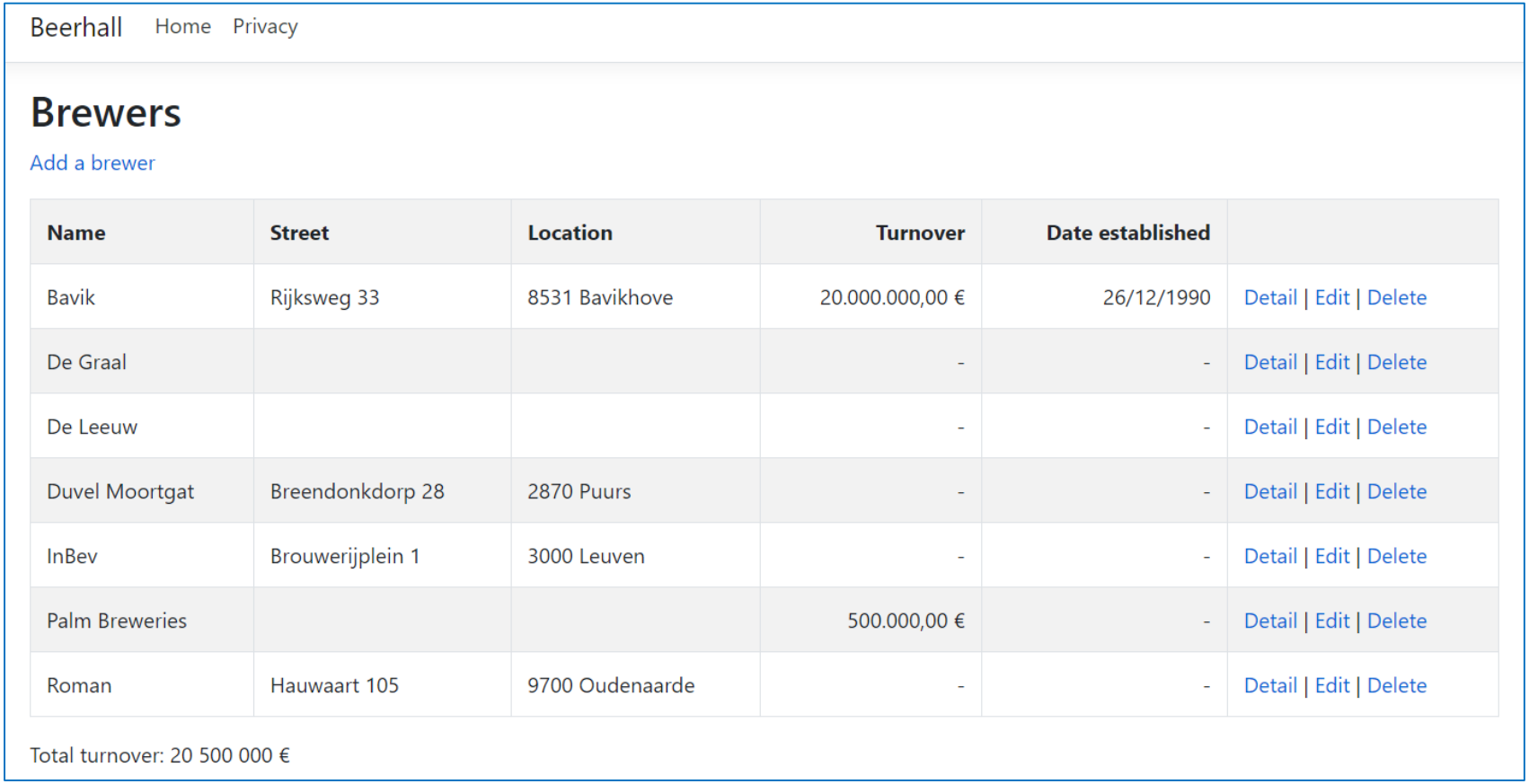
Hoofdstuk 8: ASP.NET MVC Core

1. Inleiding
2. Domain Model
3. DAL : Repository pattern
4. De MVC Applicatie Bierhalle
   - De Controller
   - Index: Controller
   - Index: View
   - Index: Extra vraag van de gebruiker
   - Create GET/POST: Controller
   - Create GET/POST: View
   - Edit GET/POST: Controller
   - Edit GET/POST: View
   - Delete GET/POST: Controller
   - Delete GET/POST: View
5. Layouts
6. Exceptionhandling in MVC
7. Unit testen van de Controller
   - Depencency injection
   - Mocking
8. Oefeningen
9. Enkele extra’s
10. Referenties

1. Inleiding

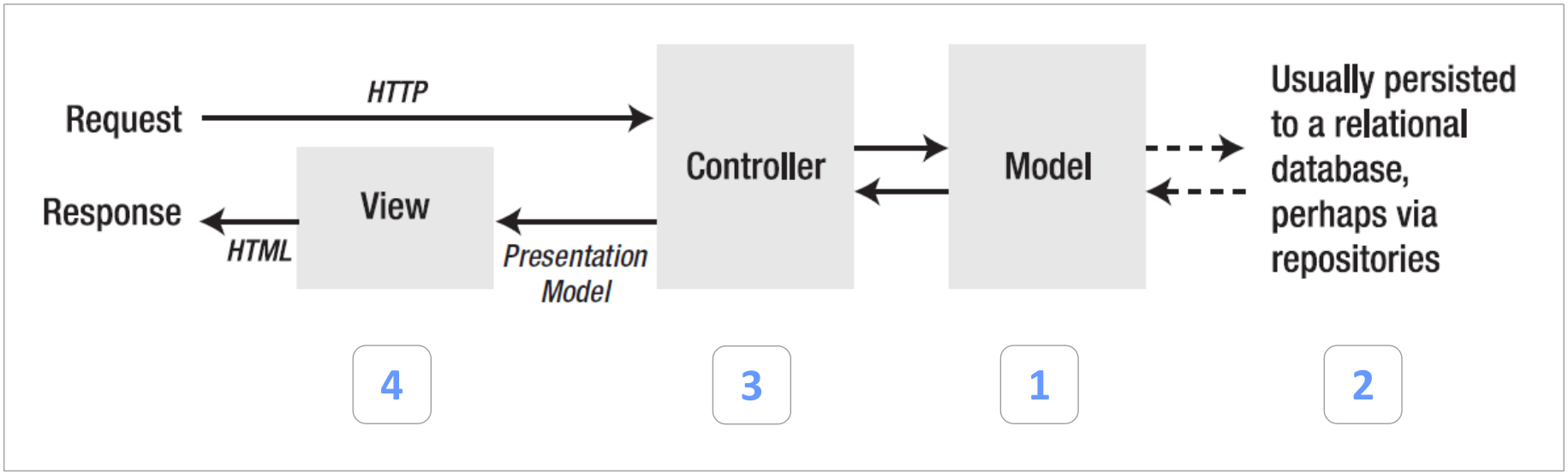
- Een eerste **data-driven** MVC applicatie
- Bedoeling
  - Volledige **CRUD** voor brouwers
    - TDD van het **Domein** (zoals in hoofdstuk 3)
    - **Controller**
      - Interacties met de databank gebruik makend van EF en Linq
    - Aanmaken van de **Views**
      - Web-helpers
      - Communicatie View – Controller via Viewmodels
  - Unit testen van Controller: **DI en Mocking**

1. Inleiding


Beerhall
Home
Privacy
Brewers
Add a brewer
Name
Street
Location
Turnover
Date established
Bavik
Rijksweg 33
8531 Bavikhove
20.000.000,00 €
26/12/1990
Detail | Edit | Delete
De Graal
-
-
Detail | Edit | Delete
De Leeuw
-
-
Detail | Edit | Delete
Duvel Moortgat
Breendonkdorp 28
2870 Puurs
-
-
Detail | Edit | Delete
InBev
Brouwerijplein 1
3000 Leuven
-
-
Detail | Edit | Delete
Palm Breweries
500.000,00 €
-
Detail | Edit | Delete
Roman
Hauwaart 105
9700 Oudenaarde
-
-
Detail | Edit | Delete
Total turnover: 20 500 000 €

1. Inleiding

- ASP.NET MVC framework



1. Inleiding: Use case

- Use Case

Use Case : Beheer brouwers

Actor : administrator

Precondities : actor is ingelogd als administrator. Brouwer gegevens zijn beschikbaar.

Postcondities : administrator heeft gegevens brouwers bekeken, eventueel de gegevens van een brouwer aangepast, een brouwer toegevoegd of een brouwer verwijderd.

Normale verloop:

1. Systeem geeft een overzicht van de brouwers (naam, gemeente, omzet) en de mogelijkheid om een brouwer te editeren, te verwijderen of een nieuwe brouwer toe te voegen
2. Herhaal
    1. Administrator kiest om brouwer te editeren
    2. Systeem geeft de gegevens van de brouwer weer : brouwernr, naam, straat, postcode, gemeente, omzet
    3. Administrator wijzigt de gegevens van de brouwer
    4. Systeem valideert de gegevens en slaat deze op in de database
    5. Systeem geeft melding dat de gegevens gewijzigd zijn

Ho

1. Inleiding: Use case

Alternatieve scenario's :

2.1.1a Administrator wenst een brouwer te creëren

1. Systeem biedt de mogelijkheid om te gegevens in te geven : naam, straat, postcode, omzet
2. Administrator geeft de gegevens in
3. Naar 2.1.4

2.3.a Administrator wenst de gegevens niet te wijzigen

1. Terug naar 2

2.3.b De administrator wenst een brouwer te verwijderen

1. Systeem vraagt om bevestiging
2. Actor bevestigt
3. Systeem verwijdert de brouwer uit de database
4. Systeem geeft melding. Terug naar 1

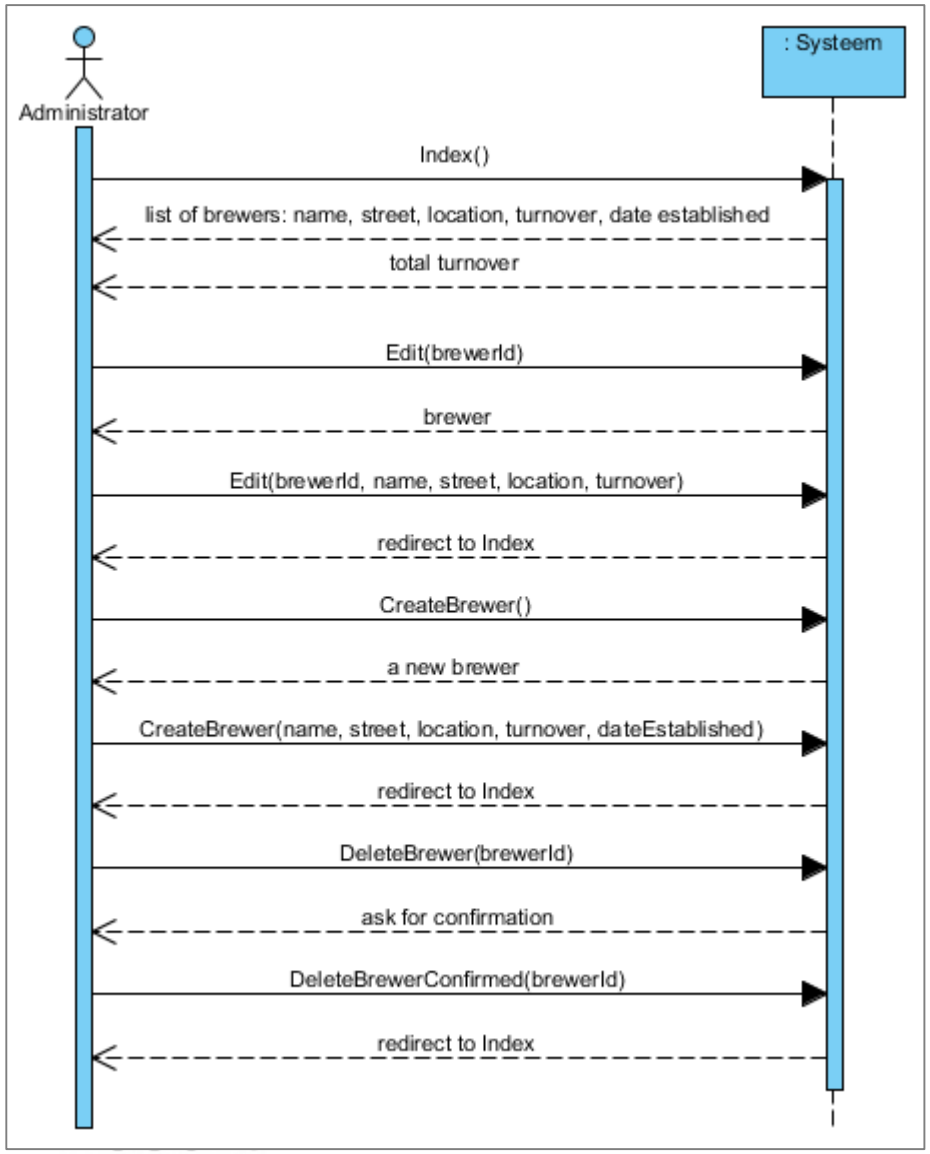
2.3.b.1.a. Systeem detecteert dat brouwer bieren bevat.

1. Systeem geeft melding dat de brouwer niet kan verwijderd worden zolang hij bieren heeft. Terug naar 2

2.4.a Systeem detecteert dat niet alle gegevens correct ingevuld zijn : naam verplicht, omzet > 0 als ingevuld.)

1. Systeem geeft melding. Terug naar 2.3

1. Inleiding: SSD - BrouwerController



BrewerController
Class

Methods
- Create() : IActionResult
- Create(BrewerEditViewModel brewerEditViewModel) : IActionresult
- Delete(int id) : IActionResult
- DeleteConfirmed(int id) : IActionResult
- Edit(BrewerEditViewModel brewerEditViewModel) : IActionResult
- Edit(int id) : IActionResult
- Index() : IActionResult

Inleiding

WELCOME TO THE

HoGent

1. Inleiding

- Als administrator wil ik…
  - Brouwers kunnen toevoegen
  - Brouwers kunnen wijzigen
  - Brouwers kunnen verwijderen
  - Brouwers kunnen raadplegen
- Als klant wil ik…
  - Alle bieren kunnen raadplegen
  - Bieren kunnen toevoegen aan mijn winkelmandje
  - De inhoud van mijn winkelmandje kunnen bestellen

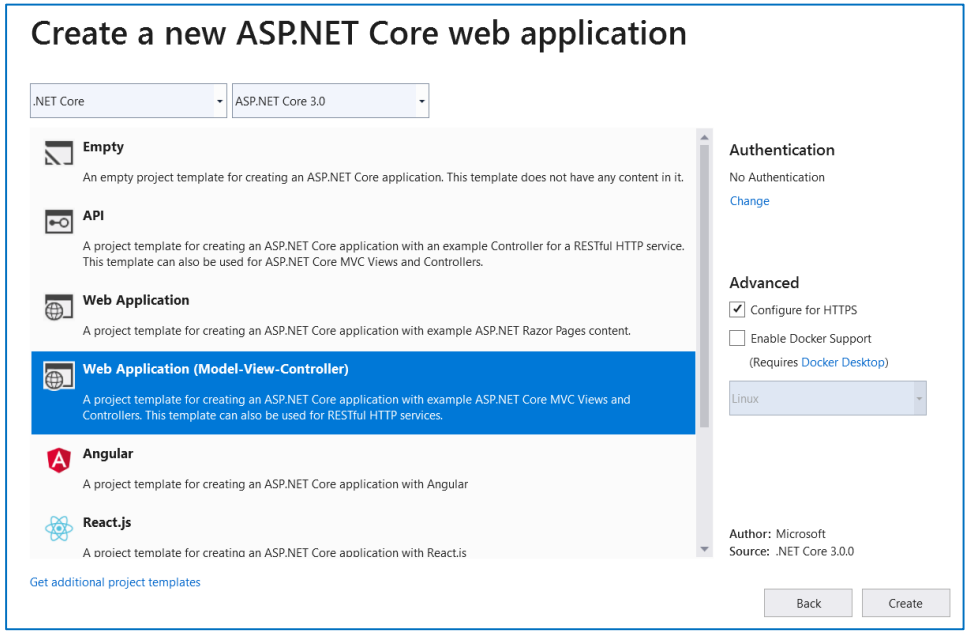
1. Inleiding

- Als administrator wil ik…
  - Brouwers kunnen creëren en toevoegen
  - Brouwers kunnen wijzigen
  - Brouwers kunnen verwijderen
  - Brouwers kunnen raadplegen

  CRUD
- Als klant wil ik…
  - Alle bieren kunnen raadplegen
  - Bieren kunnen toevoegen aan mijn winkelmandje
  - De inhoud van mijn winkelmandje kunnen bestellen

1. Inleiding

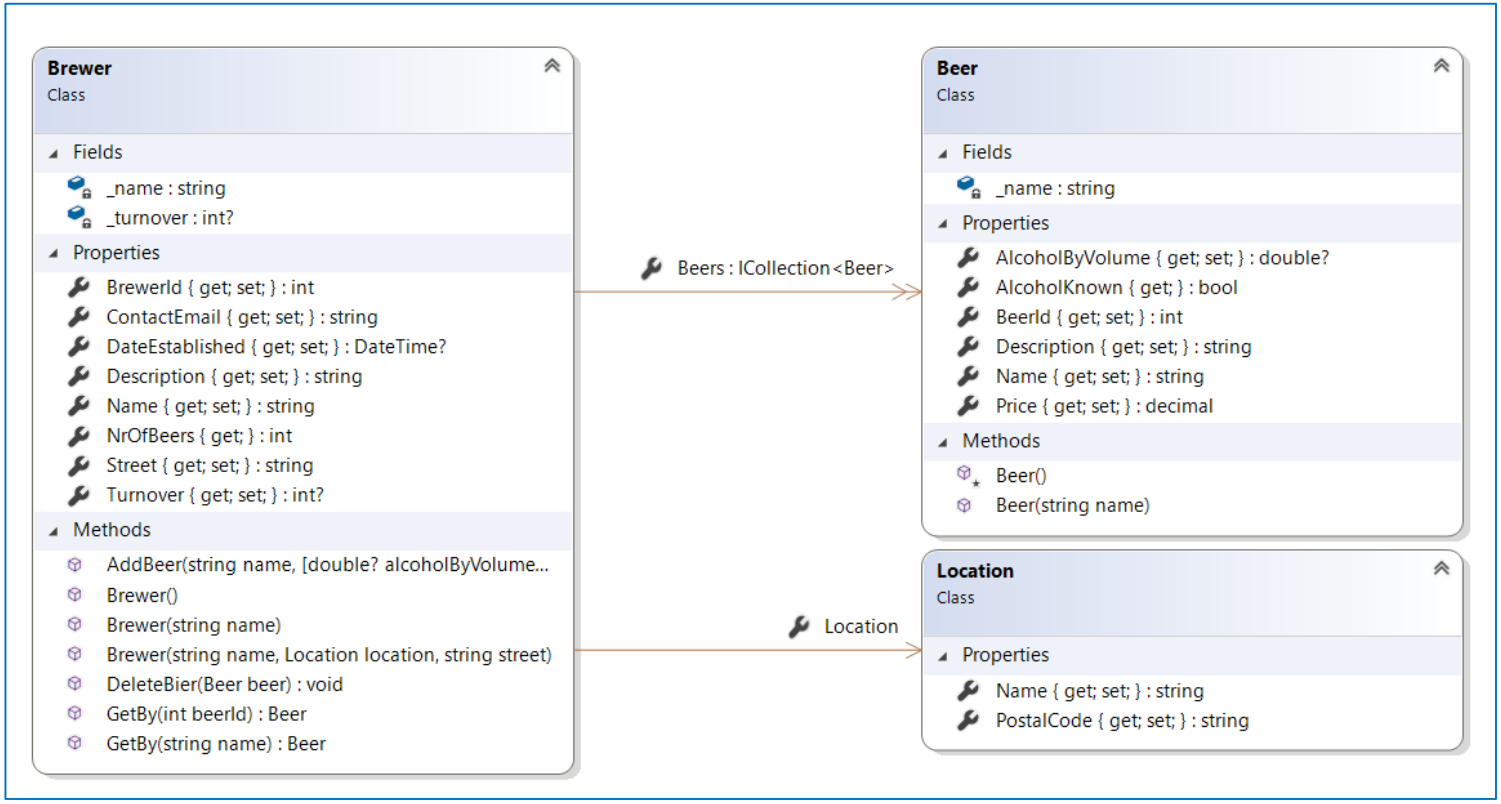
- De start: aanmaken van een nieuw project
  - Template: Visual C# - .NET Core
  - ASP.NET Core Web Application > Web Application (Model-View-Controller) > “Beerhall”
    - Merk op in dit project gebruiken we **geen authenticatie**



HoGent

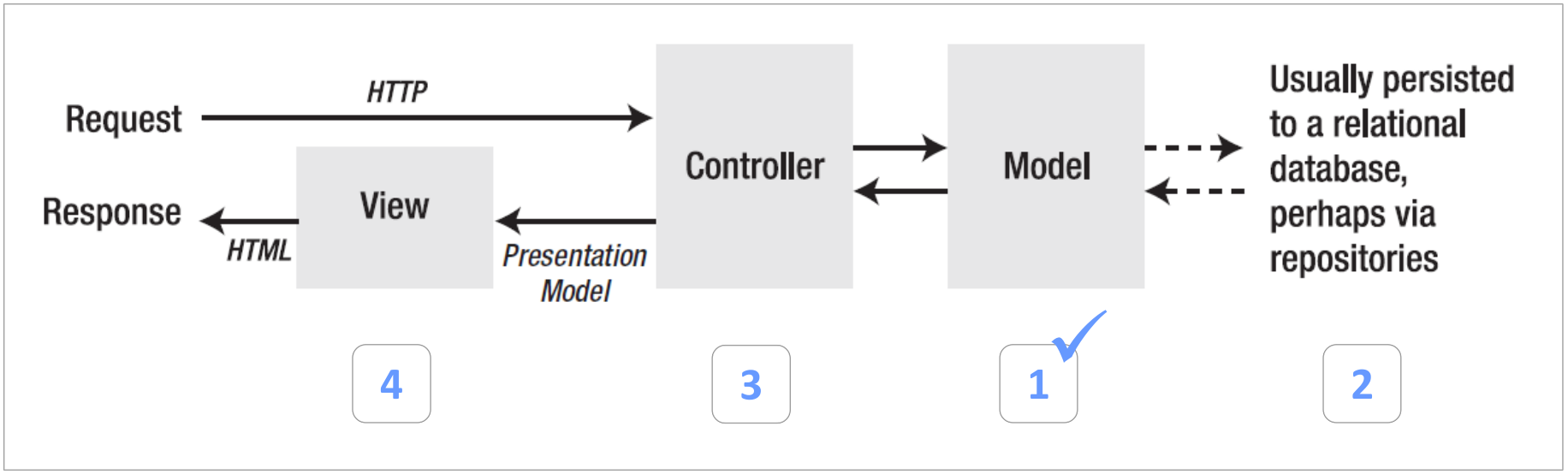
2. Het domein

- We nemen een deel van de domeinlaag over uit H7 Entity Framework
- Klassen toevoegen in Beerhall - Models - Domain



2. Het domein

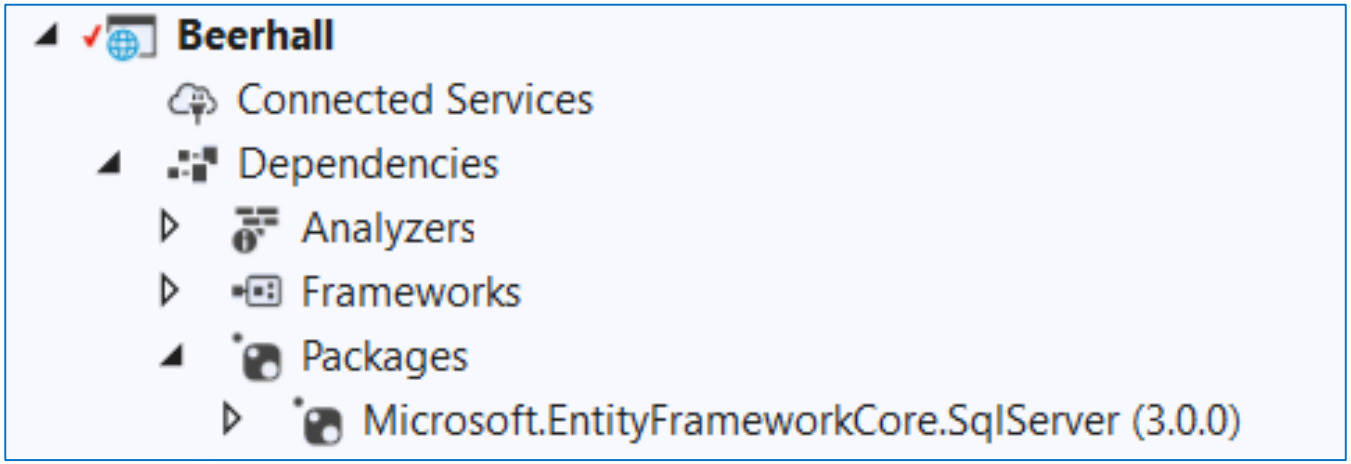
- Inspecteer de domeinklassen Brewer, Beer, Location
- Inspecteer de testklassen BrewerTest, BeerTest
  - **run de testen…**



HoGent

3. De Datalaag

- We maken gebruik van **EF Core** als ORM-tool.
- Installeer de nuget package **Microsoft.EntityFrameworkCore.SqlServer**



3. De Datalaag

- **Stap 2 – DbContext klasse & Initializer**
  - **ApplicationDbContext.cs**
    - erft van DbContext
    - DbSets voor Brewer en Location
      - Beer zullen we enkel via Brewer benaderen, enkel voor de aggregate roots maken we DbSets aan
    - Mappers voor Brewer, Beer en Location

Merk op

- in de console applicatie in hfstk 07 werd in deze klasse de configuratie verzorgd.
- deze ApplicationDbContext klasse bevat **geen** override voor de methode OnConfiguring, de **configuratie gebeurt nu in de StartUp** klasse (zie verderop)
- deze manier van werken vereist wel een constructor in onze ApplicationDbContext die de constructor van de base klasse aanroept:

```
public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
{
}
```

A provider can be configured by overriding the DbContext.OnConfiguring method or by using AddDbContext on the application service provider. If AddDbContext is used, then also ensure that your DbContext type accepts a DbContextOptions<TContext> object in its constructor and passes it to the base constructor for DbContext.

3. De Datalaag

- **Stap 2 – DbContext klasse & Initializer**
  - **BeerhallDataInitializer.cs**
    - het initializeren van de DB gebeurt **telkens bij de opstart van de applicatie**
    - we voorzien een constructor dewelke de ApplicationDbContext binnenkrijgt
    - methode voor initialisatie: InitializeData(), deze maakt gebruik van de ApplicationDbContext
      - we maken gebruik van de drop create strategy, de DB wordt gedropped en gecreëerd bij elke run:

public virtual bool **EnsureDeleted**()
Member of Microsoft.EntityFrameworkCore.Infrastructure.DatabaseFacade

Summary:
Ensures that the database for the context does not exist. If it does not exist, no action is taken. If it does exist then the database is deleted.

Warning: The entire database is deleted an no effort is made to remove just the datab

Returns:
True if the database is deleted, false if it did not exist.

public virtual bool **EnsureCreated**()
Member of Microsoft.EntityFrameworkCore.Infrastructure.DatabaseFacade

Summary:
Ensures that the database for the context exists. If it exists, no action is taken. If it does not exist then the database and all its schema are created. If the database exists, then no effort is made to ensure it is compatible with the model for this context.

Note that this API does not use migrations to create the database. In addition, the database that is created cannot be later updated using migrations. If you are targeting a relational database and using migrations, you can use the DbContext.Database.Migrate() method to ensure the database is created and all migrations are applied.

Returns:
True if the database is created, false if it already existed.

3. De Datalaag

- **Stap 2 – DbContext klasse & Initializer**
  - **BeerhallDataInitializer.cs… vervolg**
    - InitializeData()
      - bevat de seeding: statements om data toe te voegen aan de context/DB

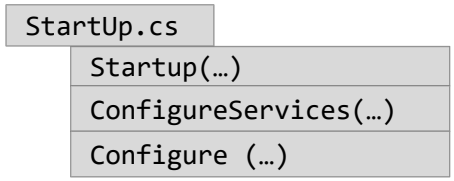
```
public static void InitializeData() {
            _dbContext.Database.EnsureDeleted();
            if (_dbContext.Database.EnsureCreated()) {
                Location bavikhove = new Location { Name = "Bavikhove", PostalCode = "8531" };
                Location roeselare = new Location { Name = "Roeselare", PostalCode = "8800" };
                Location puurs = new Location { Name = "Puurs", PostalCode = "2870" };
                Location leuven = new Location { Name = "Leuven", PostalCode = "3000" };
                Location oudenaarde = new Location { Name = "Oudenaarde", PostalCode = "9700" };
                Location affligem = new Location { Name = "Affligem", PostalCode = "1790" };
                Location[] locations =
                    new Location[] { bavikhove, roeselare, puurs, leuven, oudenaarde, affligem };
                _dbContext.Locations.AddRange(locations);
                _dbContext.SaveChanges();

                 ....
```

Merk op

- deze Drop/Create strategie is handig tijdens development daar je bij het opstarten van de applicatie steeds werkt met een verse databank die de gewenste data bevat; zo kunnen vooropgestelde scenario's steeds gemakkelijk getest worden

3. De Datalaag

- **Stap 3 – Configuratie**
  - **Program.cs**
    - entry point voor de applicatie, bevat de **Main** methode
    - in die Main methode wordt
      - een WebHost gebouwd en de Run() methode aangeroepen om de webapplicatie te runnen
      - tijdens het bouwen van de WebHost wordt, onder andere, **appsettings.json** toegevoegd als bron voor configuratie
        - de configuratie wordt doorgegeven aan de **StartUp** klasse



3. De Datalaag

StartUp.cs
Startup(…)
ConfigureServices(…)
Configure (…)

- **Stap 3 – Configuratie**
  - **StartUp.cs**
    - tijdens constructie wordt de configuratie doorgegeven
    - het framework roept de methode **ConfigureServices** aan
      - configuratie van services die nodig zijn voor de applicatie
        - wij willen dat onze **ApplicationDbContext** en onze **BeerhalldataInitializer** als een service beschikbaar worden in de applicatie
    - het framework roept de methode **Configure** aan
      - configuratie van de request pipeline
        - wij kunnen op dit punt onze **databank initialiseren**

3. De Datalaag

StartUp.cs
- Startup(…)
- ConfigureServices(…)
- Configure (…)

Stap 3 – Configuratie

StartUp.cs – de constructor

- public Startup(IConfiguration configuration)
  - het template voorziet in een constructor met 1 parameter die via DI wordt geïnjecteerd, deze parameter bevat de configuratie voor je applicatie

public interface **IConfiguration**
Member of Microsoft.Extensions.Configuration

Summary:
Represents a set of key/value application configuration properties.

  - de configuratie wordt beschikbaar via de Configuration property

```
public Startup(IConfiguration configuration)
{
     Configuration = configuration;
}

public IConfiguration Configuration { get; }
```

At its simplest, `Configuration` is just a collection of sources, which provide the ability to read and write name/value pairs. If a name/value pair is written to `Configuration`, it is not persisted. This means that the written value will be lost when the sources are read again.

3. D


IConfigurationProvider
Interface
Methods
GetChildKeys(IEnumerable<string> earlierKeys, string parentPath, string delimiter) : IEnumerable<string>
Load() : void
Set(string key, string value) : void
TryGet(string key, out string value) : bool
IConfigurationBuilder
Interface
Properties
Properties { get; } : Dictionary<string, object>
Providers { get; } : IEnumerable<IConfigurationProvider>
Methods
Add(IConfigurationProvider provider) : IConfigurationBuilder
Build() : IConfigurationRoot
ConfigurationProvider
Abstract Class
Properties
Data { get; set; } : IDictionary<string, string>
Methods
ConfigurationProvider()
GetChildKeys(IEnumerable<string> earlierKeys, string parentPath, string delimiter) : IEnumerable<string>
Load() : void
Set(string key, string value) : void
TryGet(string key, out string value) : bool
ConfigurationBuilder
Class
Properties
Properties { get; } : Dictionary<string, object>
Providers { get; } : IEnumerable<IConfigurationProvider>
Methods
Add(IConfigurationProvider provider) : IConfigurationBuilder
Add(IConfigurationProvider provider, bool load) : IConfigurationBuilder
Build() : IConfigurationRoot
ConfigurationBuilder()
JsonConfigurationProvider
Class
ConfigurationProvider
Properties
Optional { get; } : bool
Path { get; } : string
Methods
JsonConfigurationProvider(string path)
JsonConfigurationProvider(string path, bool optional)
Load() : void
CommandLineConfigurationProvider
Class
ConfigurationProvider
Properties
Args { get; set; } : IEnumerable<string>
Methods
CommandLineConfigurationProvider(IEnumerable<string> args, IDictionary<stri...
Load() : void
IEnumerable<KeyValuePair<string, string>>
IEnumerable
EnvironmentVariablesConfigurationProvider
Class
ConfigurationProvider
Methods
EnvironmentVariablesConfigurationProvider()
EnvironmentVariablesConfigurationProvider(string prefix)
Load() : void
MemoryConfigurationProvider
Class
ConfigurationProvider
Methods
Add(string key, string value) : void
GetEnumerator() : IEnumerator<KeyValuePair<string, string>>
MemoryConfigurationProvider()
MemoryConfigurationProvider(IEnumerable<KeyValuePair<string, string>> initialData)
IniConfigurationProvider
Class
ConfigurationProvider
Properties
Optional { get; } : bool
Path { get; } : string
Methods
IniConfigurationProvider(string path)
IniConfigurationProvider(string path, bool optional)
Load() : void
XmlConfigurationProvider
Class
ConfigurationProvider
Properties
Optional { get; } : bool
Path { get; } : string
Methods
Load() : void
XmlConfigurationProvider(string path)
XmlConfigurationProvider(string path, bool optional)

3. De Datalaag

StartUp.cs
- Startup(…)
- ConfigureServices(…)
- Configure (…)

Stap 3 – Configuratie

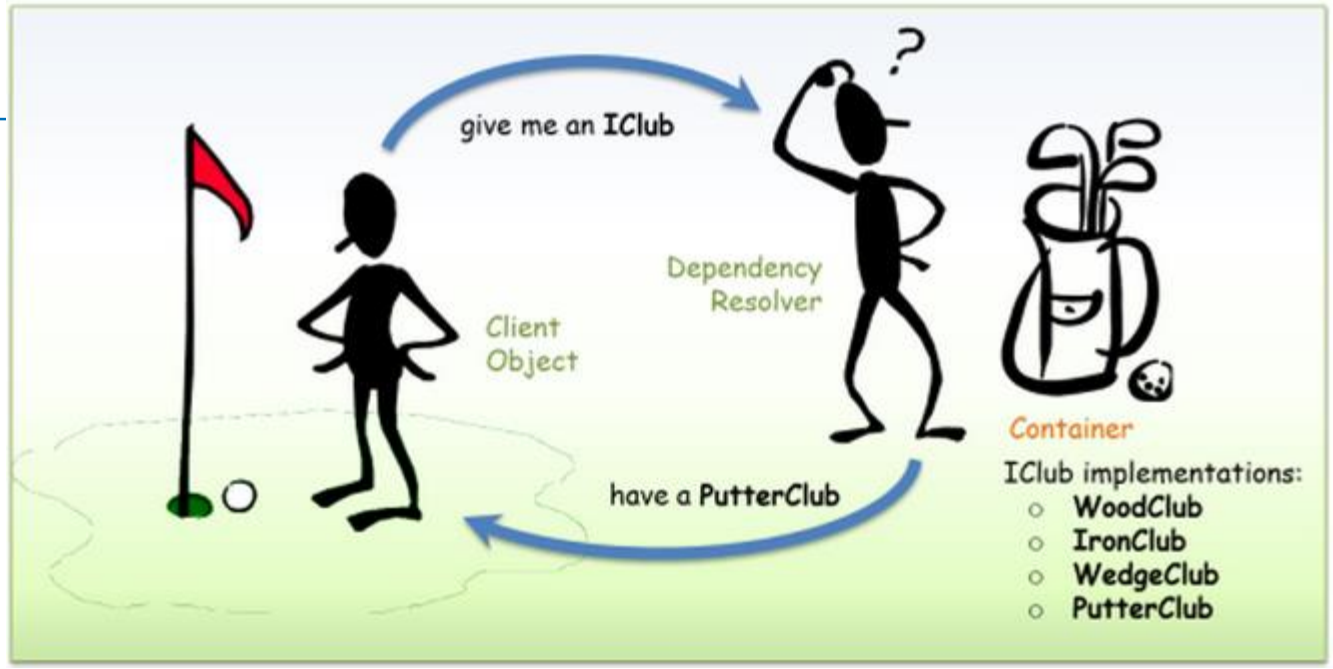
- **StartUp.cs - ConfigureServices(IServiceCollection services)**
  - services is ASP.NET's built in **IoC container**
  - de types die beheerd worden door deze IoC container noemen we services
  - standaard wordt reeds een service die door het framework wordt voorzien toegevoegd: AddControllersWithViews
  - je kan je eigen types toevoegen aan deze container om ze later te kunnen gebruiken bij bv. constructor injectie
    - er zijn 3 opties voor de lifetime van een service

Transient
Transient lifetime services are created each time they are requested. This lifetime works best for lightweight, stateless services.

Scoped
Scoped lifetime services are created once per request.

Singleton
Singleton lifetime services are created the first time they are requested (or when `ConfigureServices` is run if you specify an instance there) and then every subsequent request will use the same instance. If your application requires singleton behavior, allowing the services container to manage the service's lifetime is recommended instead of implementing the singleton design pattern and managing your object's lifetime in the class yourself.

https://docs.asp.net/en/latest/fundamentals/dependency-injection.html



Strategy Pattern

- At a high level, the goal of Dependency Injection is that a class (e.g. *the golfer*) needs something that satisfies an interface (e.g. *IClub*). It doesn't care what the concrete type is (e.g. *WoodClub, IronClub, WedgeClub* or *PutterClub*), it wants someone else to handle that (e.g. a good *caddy*). The Service container allow you to register your dependency logic.
- IServiceCollection bevat extension methods
  - Add*ServiceName* om services van een MVC applicatie te registeren, zoals bvb AddControllersWithViews
  - AddScoped : om een abstract type te mappen naar een concrete service, die dan per request geinstantieerd wordt als een object er om vraagt.
  - …

3. De Datalaag

StartUp.cs
- Startup(…)
- ConfigureServices(…)
- Configure (…)

Stap 3 – Configuratie

StartUp.cs - ConfigureServices(IServiceCollection services)

- invulling voor onze applicatie:
  - we registreren de **ApplicationDbContext** als een service
    - we gaan de context niet zelf instantiëren, we laten dit over aan de IoC container; als een klasse nood heeft aan de context kan deze via de constructor geïnjecteerd worden
    - EF voorziet hiervoor in de **AddDbContext** methode
      - via de options parameter kunnen we specifieren welke fysische DB zal gebruikt worden
        - UseSqlServer(connectionString)
        - de connectionstring hebben we gedefinieerd in appsettings.json

de default lifetime bij AddDbContext is scoped

```
services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetValue<string>("Data:DefaultConnection:ConnectionString")));
```

appsettings.json bevat configuratie informatie voor de applicatie, in de ctor StartUp werd deze json file toegevoegd aan Configuration

```
"Data": {
  "DefaultConnection": {
    "ConnectionString": "Server=localhost\\SQLEXPRESS;Database=BeerhallMVC;Trusted_Connection=True;"
  }
},
```

3. De Datalaag

StartUp.cs
- Startup(…)
- ConfigureServices(…)
- Configure (…)

Stap 3 – Configuratie

- **StartUp.cs - ConfigureServices(IServiceCollection services)**
  - invulling voor onze applicatie (vervolg):
    - we registreren de **BeerhallDataInitializer** als een service
      - in een volgende stap wordt duidelijk waarom we dit doen (injectie in Configure method)

```
services.AddScoped<BeerhallDataInitializer>();
```

services die gebruik maken van de EF dbContext service declareer je met een lifetime die dezelfde is als die van de dbContext, nl. scoped

3. De Datalaag

StartUp.cs
- Startup(…)
- ConfigureServices(…)
- Configure (…)

- **Stap 3 – Configuratie vervolg**
  - **StartUp.cs - Configure(…)**
    - wordt aangeroepen door de runtime, **ná de methode ConfigureServices(…)**
      - de services (zoals bv. onze BeerhallDataInitializer) zijn dus reeds geregistreerd en kunnen geinjecteerd worden in deze methode
      - standaard worden reeds IApplicationBuilder (verplicht) en IWebHostEnvironment geïnjecteerd

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
```

interface Microsoft.AspNetCore.Builder.IApplicationBuilder
Defines a class that provides the mechanisms to configure an application's request pipeline.

interface Microsoft.AspNetCore.Hosting.IWebHostEnvironment
Provides information about the web hosting environment an application is running in.

3. De Datalaag

StartUp.cs
- Startup(…)
- ConfigureServices(…)
- Configure (…)

- **Stap 3 – Configuratie vervolg**
  - **StartUp.cs - Configure(…) vervolg**
    - we gaan zorgen dat onze Db bij de start van de applicatie geinitialiseerd wordt:
      - injectie van de BeerhallDataInitializer
      - aanroep naar InitializeData(…)

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, BeerhallDataInitializer beerhallDataInitializer) {
            …

             app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Home}/{action=Index}/{id?}");
            });

            beerhallDataInitializer.InitializeData();
        }
```

3. De Datalaag

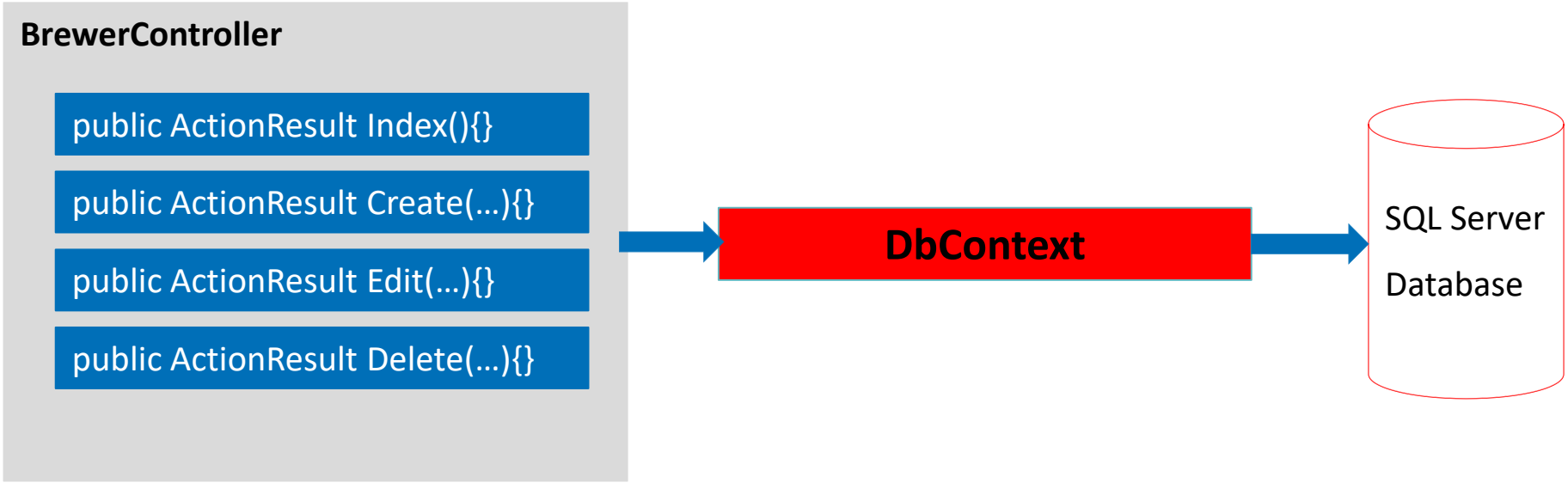
- Run de applicatie en inspecteer de gegenereerde databank
  - bekijk de inhoud van de tabellen
  - bekijk het ontwerp van de tabellen
  - je kan hiervoor gebruik maken van
    - SQL Server Management Studio, of werken via
    - Server Explorer in Visual Studio

```
commit Configure EF and add database initialization
```

HoGent

3. Data Access Layer

- Het is de **Controller** die met de **persistentielaag** zal communiceren voor de CRUD
  - Controller kan hiervoor rechtstreeks communiceren met een instantie van ApplicationDbContext…



3. Data Access Layer

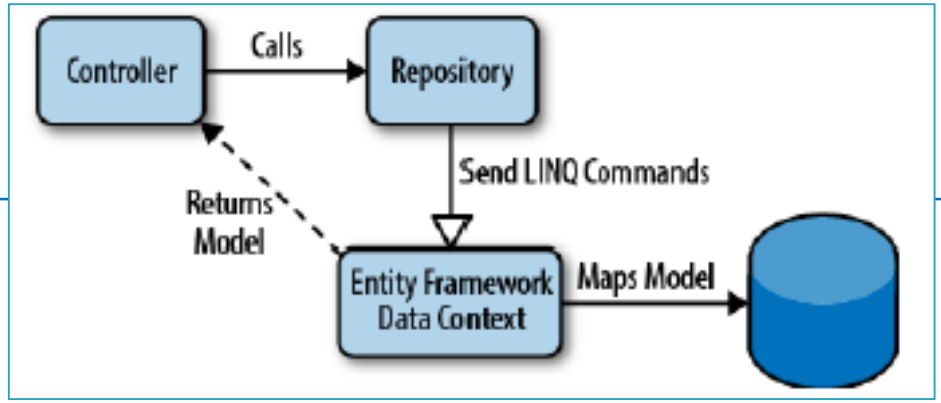
- Nadelen :
  - Unit testen van de controller.
    - Unit testen werken niet rechtstreeks met databases (te traag, niet geisoleerd, niet repeatable…)
  - Sterke koppeling Controller – DbContext
    - je hebt een dependency op EntityFramework in de controller

3. Data Access Layer

- Door het toepassen van het **repository pattern**, en **dependency injection** zullen we een **unit testable** controller maken die **geen dependency** heeft op Entity Framework …

3. DAL

- Repository pattern



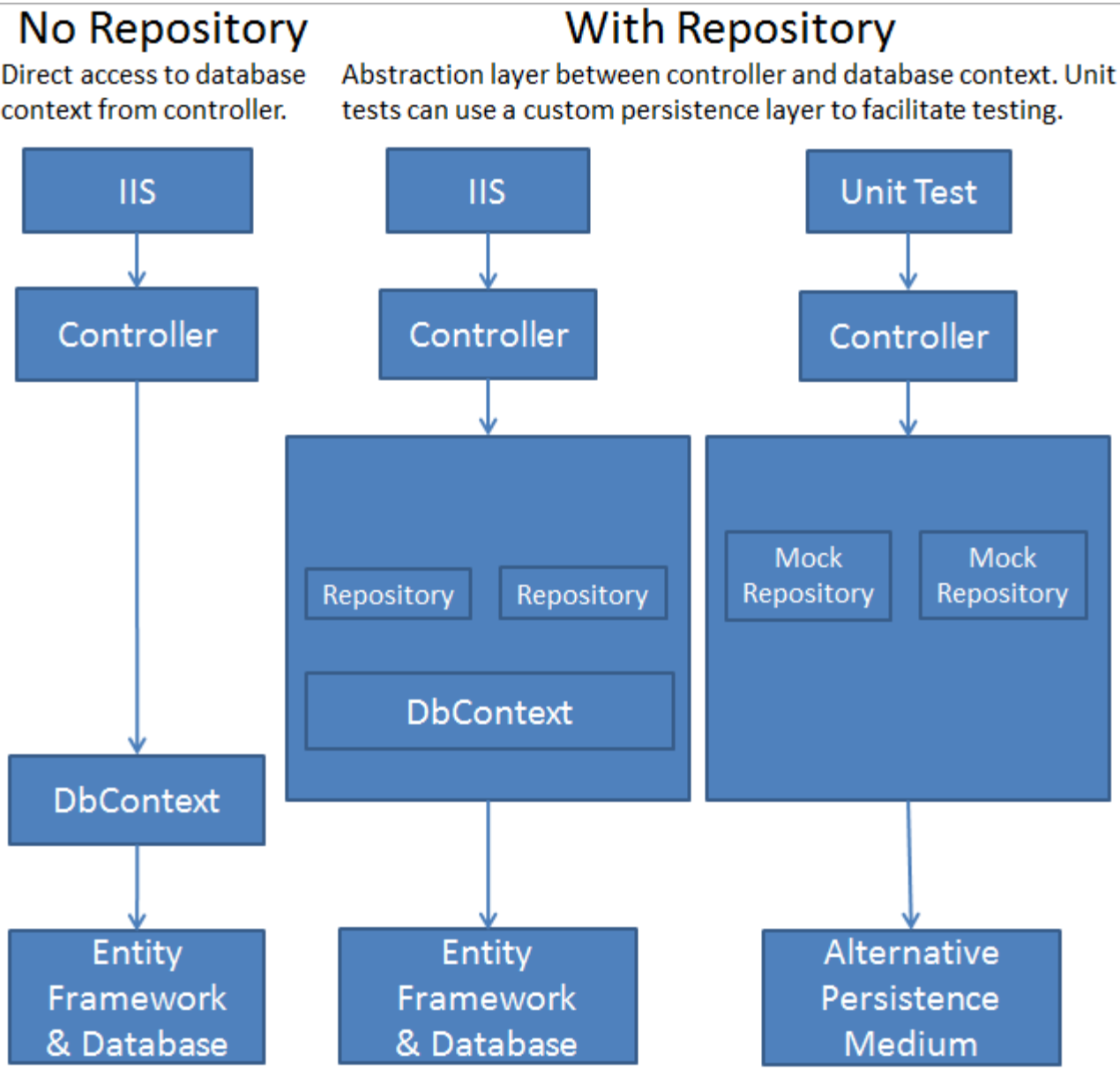
> Martin Fowler writes:
>
> *"A Repository mediates between the domain and data mapping layers, acting like an in-memory domain object collection. Client objects construct query specifications declaratively and submit them to Repository for satisfaction. Objects can be added to and removed from the Repository, as they can from a simple collection of objects, and the mapping code encapsulated by the Repository will carry out the appropriate operations behind the scenes. Conceptually, a Repository encapsulates the set of objects persisted in a data store and the operations performed over them, providing a more object-oriented view of the persistence layer. Repository also supports the objective of achieving a clean separation and one-way dependency between the domain and data mapping layers."*

P of EAA: Repository. (n.d.). Retrieved August 19, 2014, from http://martinfowler.com/eaaCatalog/repository.html

- Creëert een abstractie laag tussen de Controller en de Data laag.
- De repository gedraagt zich als een in-memory lijst van objecten waarin we objecten kunnen toevoegen, verwijderen en updaten.
- Voordelen:
  - Isoleert de toepassing van wijzigingen in de data opslag
  - Het maakt ook de toegang tot de gegevens beter testbaar. (TDD)

3. D

- Rep



Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application (9 of 10) | The ASP.NET Site. (n.d.). Retrieved August 19, 2014, from http://www.asp.net/mvc/tutorials/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

3. Data Access Layer

- Repository pattern
  - Maak een interface, dan kan je later de concrete implementatie injecteren.
  - Een typische **repository interface** bevat enkel wat nodig is voor de CRUD.
  - Meestal heb je minstens volgende 5 methodes :
    - **Geef alle**, eventueel met bijkomende zoekmogelijkheden (**GetAll**)
    - **Geef één** aggregate by id (primary key) (**GetBy**)
    - **Voeg** een nieuwe aggregate **toe** aan de repository (**Add**)
    - **Verwijder** een aggregate van de repository (**Delete**)
    - **Opslaan** van de wijzigingen (**Save**)
  - MERK OP: Je hebt geen **wijzig** methode
    - **Reden**: de repository moet bij de **Save** opdracht zelf in staat zijn om te detecteren welke objecten gewijzigd zijn en de nodige aanpassingen in de databank doorvoeren

3. Data Access Layer

- Repository pattern
  - In onze applicatie
    - Brewers kunnen opvragen, creëren, wijzigen, verwijderen
      - => **IBrewerRepository**
    - Alle locations kunnen opvragen, 1 location kunnen opvragen
      - => **ILocationRepository**
    - We moeten niet rechtstreeks Beers kunnen opvragen
      - we zullen in het domein via Brewer de Beers kunnen opvragen

3. Data Access Layer

- Repository pattern - IBrewerRepository
  - Een typische repository interface
    - conventie: interfaces starten met I
    - merk op, dit behoort tot de **Beerhall.Models.Domain** namespace

```
namespace Beerhall.Models.Domain {
    public interface IBrewerRepository {
        Brewer GetBy(int brewerId);
        IEnumerable<Brewer> GetAll();
        void Add(Brewer brewer);
        void Delete(Brewer brewer);
        void SaveChanges();
    }
}
```

Opm: void Delete (int brewerId) is ook mogelijk

3. Data Access Layer

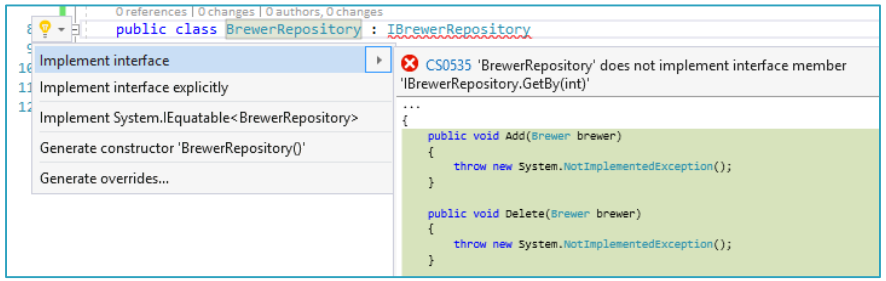
- Repository pattern - ILocationRepository
  - we zullen geen locaties moeten kunnen toevoegen of verwijderen
    - daar we enkel read-operaties hebben is er geen nood aan een SaveChanges() methode

```
namespace Beerhall.Models.Domain {
    public interface ILocationRepository {
        Location GetBy(string postalCode);
        IEnumerable<Location> GetAll();
    }
}
```

3. Data Access Layer

- Implementatie van de repository
  - De implementatie van de interface is onderdeel van de **Data laag**
  - In deze klassen hebben we een **dependency op EF**, data wordt opgehaald via **Linq queries**

- Maak een nieuwe klasse BrewerRepository aan in Data > Repositories folder
  - Erf van IBrewerRepository
  - Je kan automatisch de methodes van de interface implementeren via VS



3. Data Access Layer

- Implementatie van de repository
  - maak gebruik van constructor injectie om de DbContext te injecteren
  - schrijf LINQ queries om de data op te halen
    - we maken gebruik van **eager loading**
      - je moet expliciet aangeven welke data je wil ophalen
        - indien je bij GetAll() enkel de DbSet Brewers retourneert heb je geen toegang tot de Beers die bij een Brewer behoren
        - bv. op de index pagina willen we voor elke Brewer het aantal Beers dat er gebrouwen wordt tonen: **Include(b => b.Beers)**
        - bv. bij elke Brewer wordt ook de Location getoond: **Include(b => b.Location)**

*EF Core ondersteunt ook **lazy loading**. Bij lazy loading zou je wel toegang hebben tot de Beers die bij een Brewer behoren, ook al retourneert de repository enkel de DbSet<Brewer>. EF zou ze de Beers voor jou ophalen wanneer je ernaar refereert. Het is belangrijk dat je weet in welke loading strategieën je ORM tool voorziet en welke je wil gebruiken… Een groot nadeel van lazy loading is de zware belasting van de DB server die de performantie fel doet dalen…*

3. Data Access Layer

```
public class BrewerRepository : IBrewerRepository {
        private readonly ApplicationDbContext _dbContext;
        private readonly DbSet<Brewer> _brewers;

        public BrewerRepository(ApplicationDbContext dbContext) {
            _dbContext = dbContext;
            _brewers = dbContext.Brewers;
        }

        public Brewer GetBy(int brewerId) {
            return _brewers.SingleOrDefault(b => b.BrewerId == brewerId);
        }

        public IEnumerable<Brewer> GetAll() {
            return _brewers.Include(b => b.Location).ToList();
        }

        public void Add(Brewer brewer) {
            _brewers.Add(brewer);
        }

        public void Delete(Brewer brewer) {
            _brewers.Remove(brewer);
        }

        public void SaveChanges() {
            _dbContext.SaveChanges();
        }
    }
```

Indien er nood is aan andere methodes kan deze repository uitgebreid worden. Zo zal het, indien nodig, bijvoorbeeld efficiënter zijn een methode

public IEnumerable<Brewer> GetBrewersEstablishedBefore(DateTime established)

te voorzien in de repository. Deze zal enkel de gevraagde brewers uit de databank halen.

Indien je niet in deze methode voorziet moet de client van deze repository een GetAll() aanroepen. Deze retourneert dan alle brewers, en deze moeten nadien door de client gefilterd worden met een Where clause…

3. Data Access Layer

- Implementatie van de repository - vervolg
  - maak gebruik van info uit analyse/ontwerp om te bepalen welke includes noodzakelijk zijn
  - creëer indien nodig extra methodes volgens de behoeften…

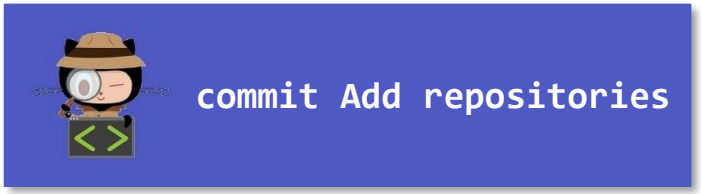
```
public class BrewerRepository : IBrewerRepository {
        …

        public Brewer GetBy(int brewerId) {
            return _brewers.SingleOrDefault(b => b.BrewerId == brewerId);
        }

        public IEnumerable<Brewer> GetAll() {
            return _brewers.Include(b => b.Location).ToList();
        }

        public Brewer GetByWithBeers(int brewerId) {
            return _brewers.Include(b => b.Beers).SingleOrDefault(b => b.BrewerId == brewerId);
        }

        public IEnumerable<Brewer> GetAllWithBeers() {
            return _brewers.Include(b => b.Location).Include(b => b.Beers).ToList();
        }
…
    }
```

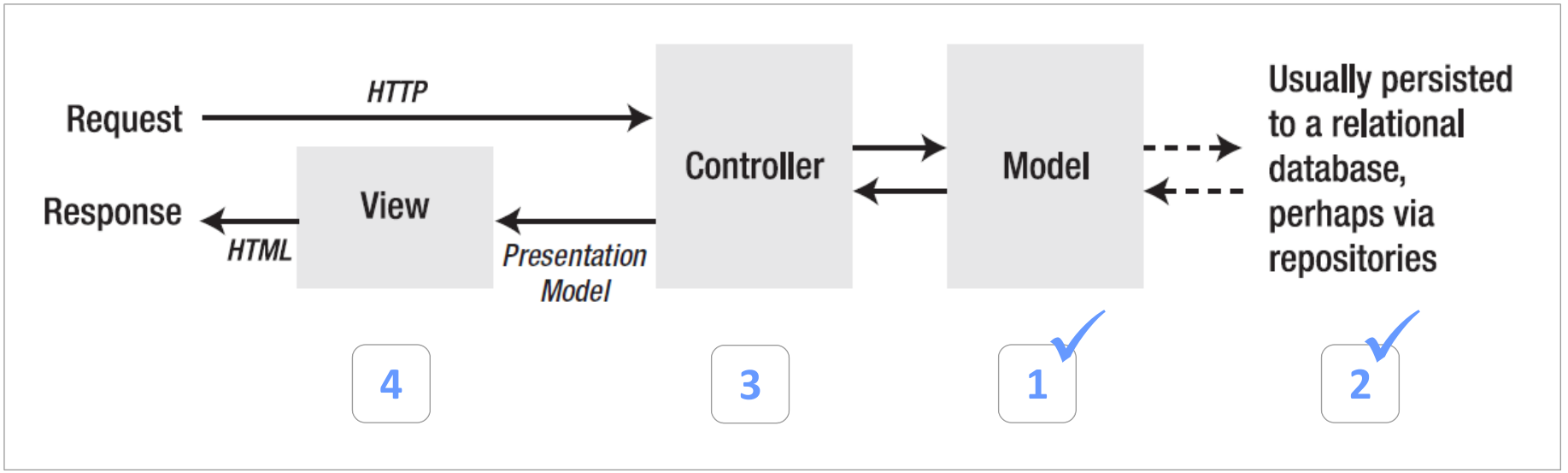
Voorbeeld extra methodes GetByWithBeers en GetAllWithBeers

3. Data Access Layer

```
namespace Beerhall.Models.Data.Repositories {
    public class LocationRepository : ILocationRepository {
        private readonly DbSet<Location> _locations;

        public LocationRepository(ApplicationDbContext dbContext) {
            _locations = dbContext.Locations;
        }
        public Location GetBy(string postalCode) {
            return _locations.SingleOrDefault(l => l.PostalCode == postalCode);
        }

        public IEnumerable<Location> GetAll() {
            return _locations.ToList();
        }
    }
}
```

3. Data Access Layer

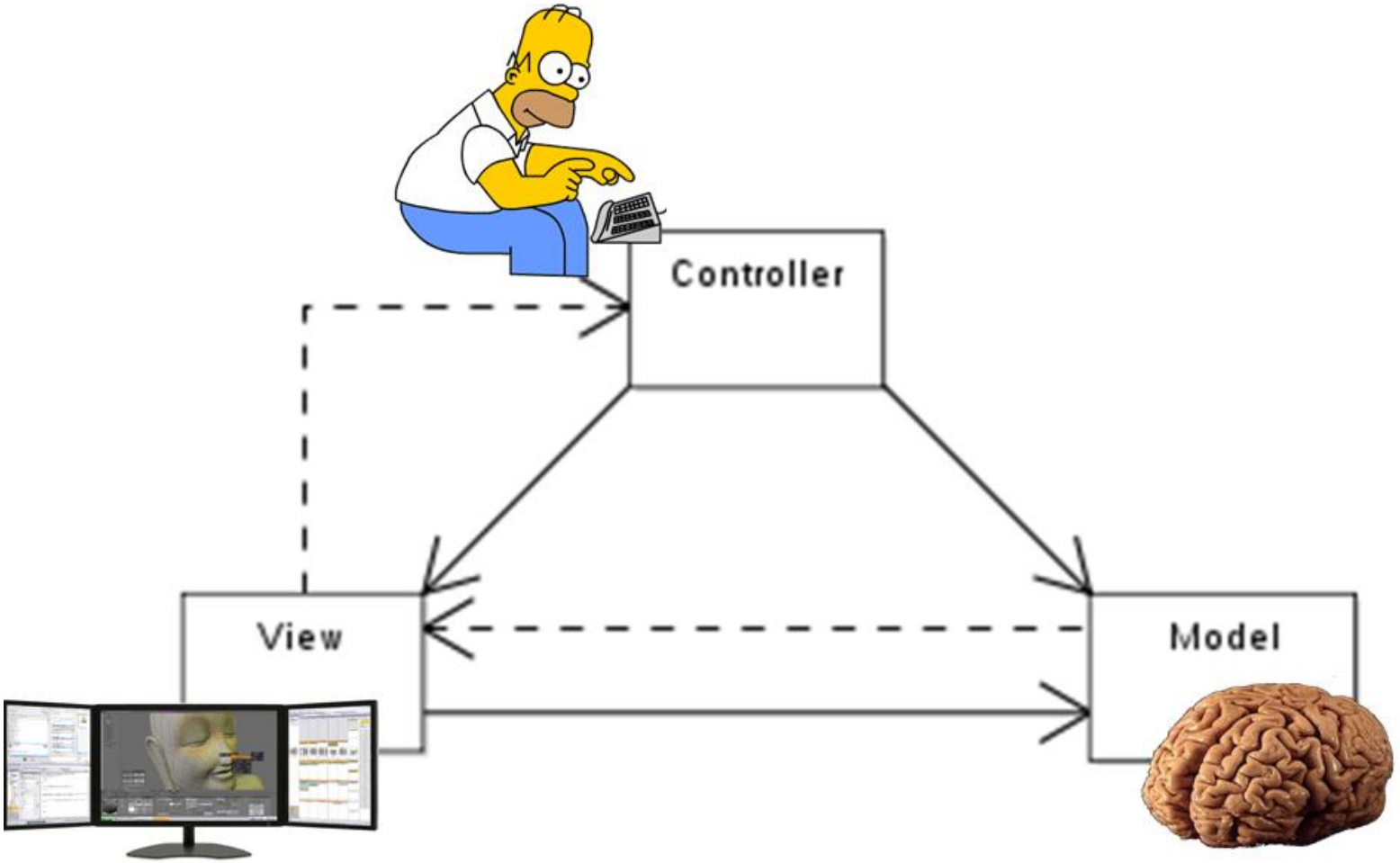




MVC - Controllers en Views

HoGent

MVC

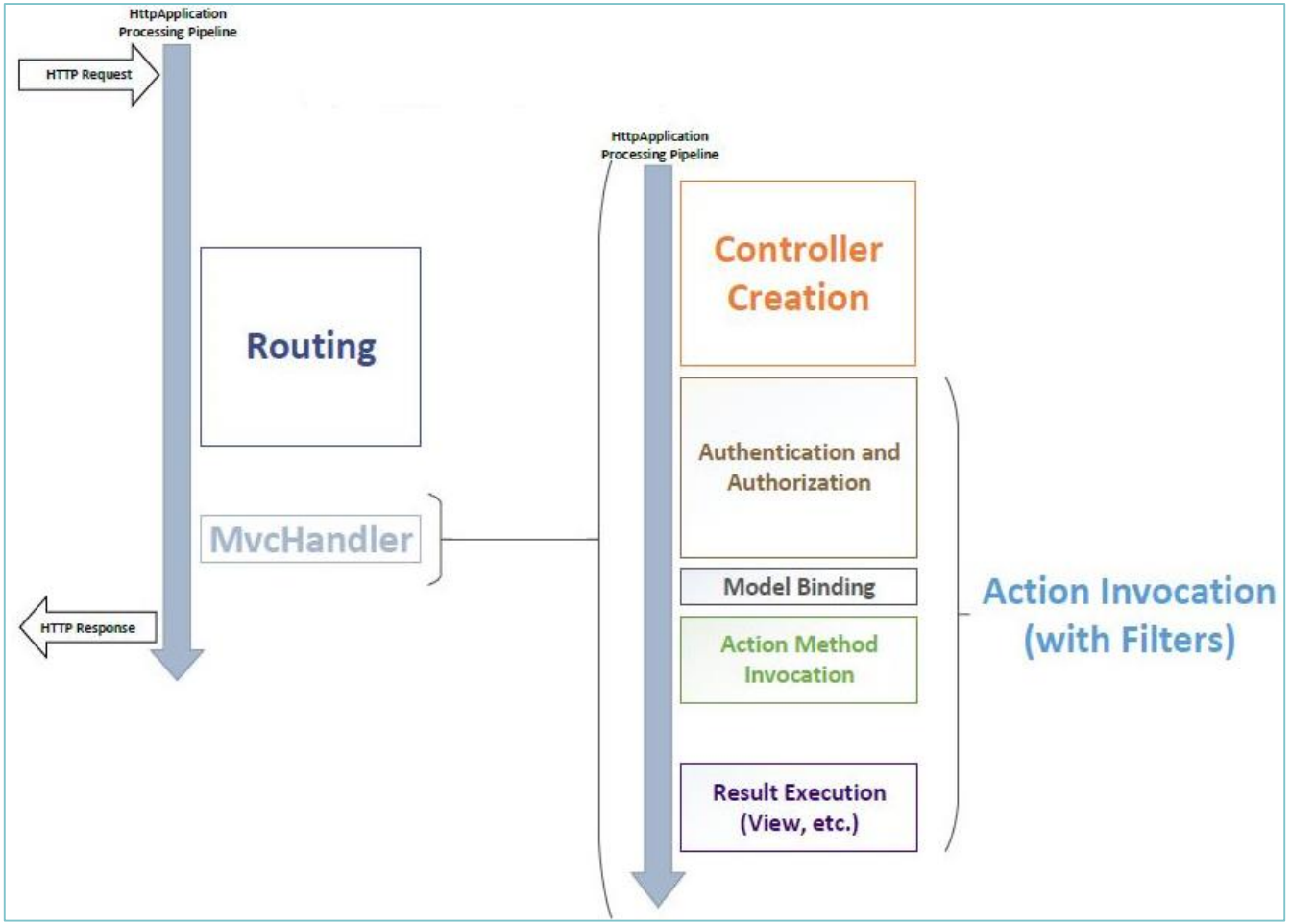


4. Controller

- Routing, Controller en Action Method
  - Url Routing System: handelt requests af
    - MVC infrastructuur dat binnenkomende HTTP requests analyseert. Het gebruikt segmenten van de URL om een specifieke controller te instantiëren en daarbinnen een specifieke action method aan te roepen.
  - Controller: is een klasse met methodes
    - publieke action methods handelen de logica van de request af
    - via model binding kan informatie van de request via parameters doorgegeven worden aan een action method
    - Best Practice : 1 Controller/Use case
  - Action Method: voert de door de client gevraagde request uit
    - communiceert met de domein laag
    - communiceert met de DAL
    - bepaalt de response
      - bv. vraagt aan View om een model(data) weer te geven

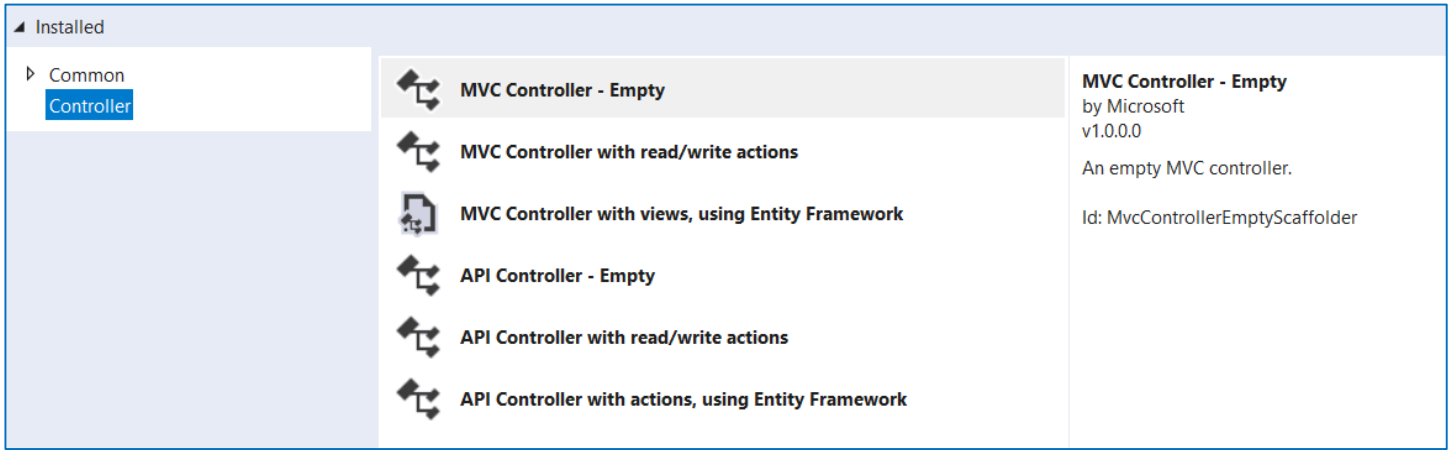
4. Controller

- Routing, Controller en Action Method



4. Controller

- Maak de BrewerController aan
  - Rechtsklik op de map “Controllers” > Add > Controller…
  - Kies MVC Controller Class, geef naam BrewerController > Add



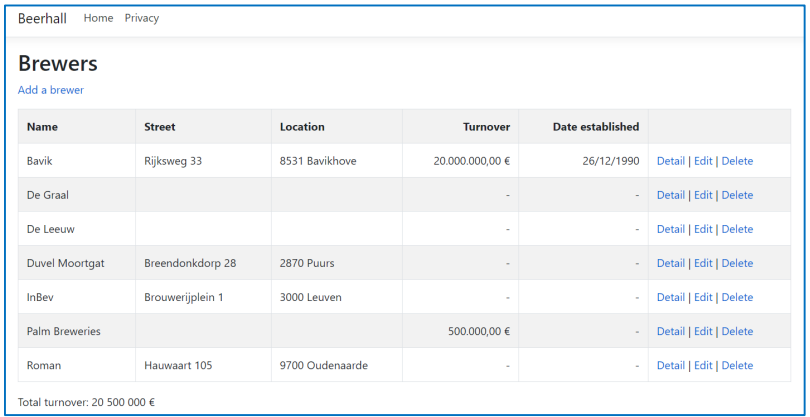
- Pas de routing aan zodat de Brewer/Index de Start pagina wordt

Index

HoGent

Index

- De startpagina van Beerhall geeft een overzicht van alle brouwers met mogelijkheid om nieuwe brouwers aan te maken of bestaande te editeren of te verwijderen.
  - **Controller** verantwoordelijkheden
    - alle nodige gegevens van alle brewers ophalen
    - de juiste View selecteren en deze de nodige gegevens aanbieden
  - **View** verantwoordelijkheden
    - de gegevens mooi presenteren



Index - Controller

- De BrewerController communiceert met de BrewerRepository in de DAL laag voor het ophalen van de brewers
  - MVC framework zal de BrewerController instantiëren
  - we maken gebruik van constructor injectie om de BrewerRepository te injecteren in de BrewerController
    - we moeten eerst de repository toevoegen aan de IoC container, zie StartUp.cs, methode ConfigureServices(…)

```
public void ConfigureServices(IServiceCollection services) {
    …
    services.AddScoped<IBrewerRepository, BrewerRepository>();
    …
}
```

services die gebruik maken van de EF dbContext service declareer je met een lifetime die dezelfde is als die van de dbContext, nl. scoped

Index - Controller

- nu kan de BrewerController de BrewerRepository gebruiken…

```
public class BrewerController : Controller {
        private readonly IBrewerRepository _brewerRepository;

        public BrewerController(IBrewerRepository brewerRepository) {
            _brewerRepository = brewerRepository;
        }
}
```

Index - Controller

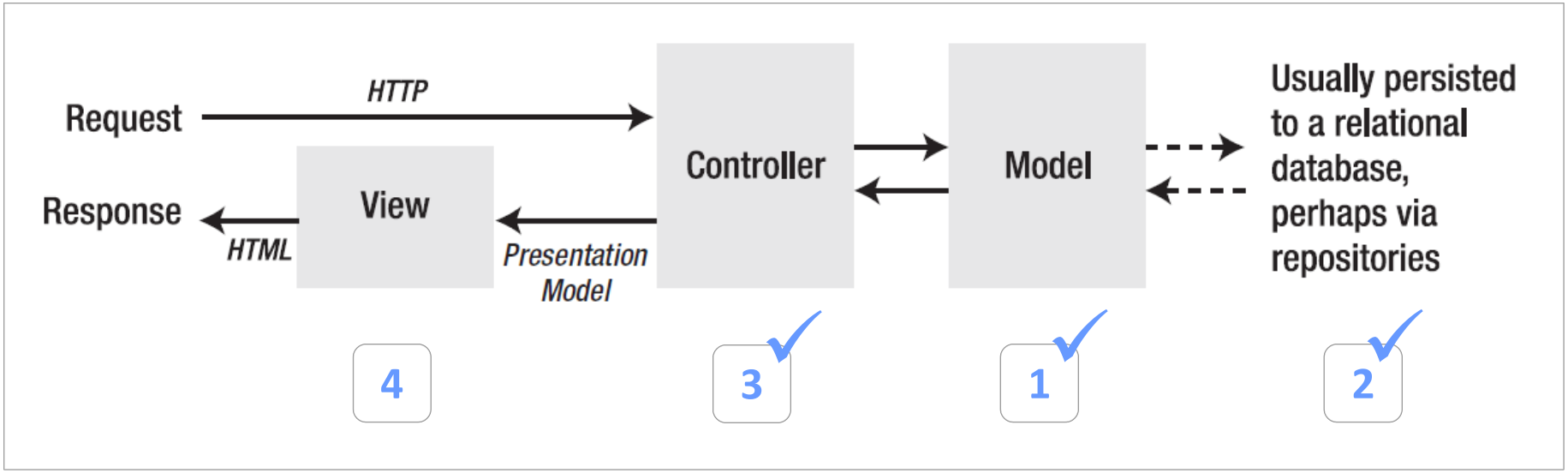
- … en kan de Index action methode geïmplementeerd worden

```
namespace Beerhall.Controllers {
    public class BrewerController : Controller {
        private readonly IBrewerRepository _brewerRepository;

        public BrewerController(IBrewerRepository brewerRepository) {
            _brewerRepository = brewerRepository;
        }

        public IActionResult Index() {
            IEnumerable<Brewer> brewers = _brewerRepository.GetAll();
            return View(brewers);
        }
    }
}
```

de action method retourneert een ViewResult

de gegevens die doorgegeven worden aan de view, deze komen in de Model property van de ViewResult

Index - Controller

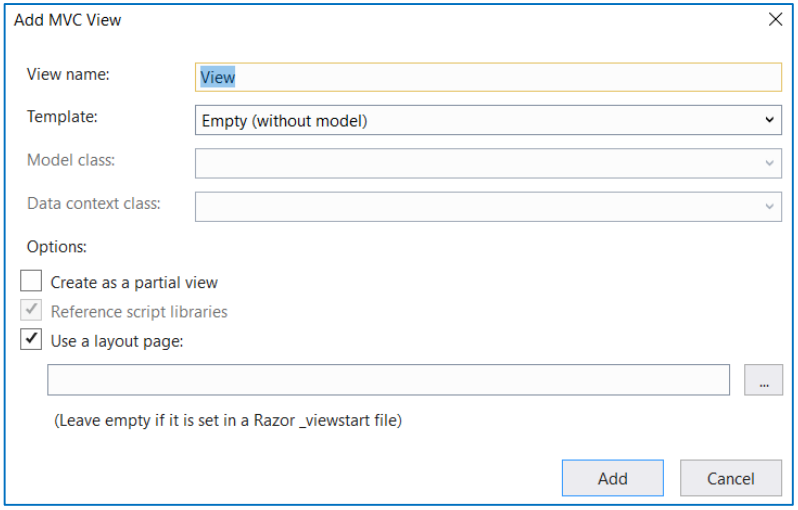


Index - View

- Onze View moet de data die de controller aanlevert weergeven in een **HTML** pagina.
- Het bevat **geen business logica**! Het plaatst enkel de aangeleverde data op de juiste plaats in de HTML pagina.
  - data aangeleverd via **ViewData**
  - data aangeleverd via **Model**
- De view bevat een mix van C# en HTML
  - **Razor**
  - **Taghelpers**

Index - View

- Maak een folder Brewer aan in de folder Views
- Maak een view genaamd Index aan in die folder
  - we volgen de conventies: alle views voor de BrewerController stoppen we in een submap van Views genaamd Brewer
  - het resultaat is een cshtml bestand Index
    - merk op: later zullen we gebruik maken van scaffolding en zal er reeds veel voor ons automatisch gegenereerd worden



Index - View

- Specificatie van het model voor de view: **@model**
  - op deze manier krijgen we voor de instantie die wordt doorgegeven strong type checking en intellisense in de view

```
@model IEnumerable<Beerhall.Models.Domain.Brewer>
```

```
public class BrewerController : Controller {
            …
        public IActionResult Index() {
            IEnumerable<Brewer> brewers =
_brewerRepository.GetAll();
            return View(brewers);
        }
    }
```

zorg dat het aangegeven type overeenkomt met het type dat doorgegeven wordt!

Index - View

- Inhoud: gebruik van **Model** voor het overzicht

```
<table class="table table-striped table-condensed table-bordered">
    <tr>
        <th>Name</th>
        <th>Street</th>
        <th>Location</th>
        <th class="text-right">Turnover</th>
        <th class="text-right">Date established</th>
        <th></th>
    </tr>

    @foreach (var item in Model) {
        <tr>
            <td>@item.Name</td>
            <td>@item.Street</td>
            <td>@item.Location?.PostalCode @item.Location?.Name</td>
            <td class="text-right">@(item.Turnover?.ToString("c") ?? "-")</td>
            <td class="text-right">@(item.DateEstablished?.Date.ToString("d") ?? "-")</td>
            <td>
                <a asp-controller="Brewer" asp-action="Detail" asp-route-id="@item.BrewerId">Detail</a> |
                <a asp-controller="Brewer" asp-action="Edit" asp-route-id="@item.BrewerId">Edit</a> |
                <a asp-controller="Brewer" asp-action="Delete" asp-route-id="@item.BrewerId">Delete</a>
            </td>
        </tr>
    }
</table>
```

gebruik van Model

anchor tag helpers, tag helpers worden verderop in dit hoofdstuk toegelicht

Index - View

- Inhoud: via ViewBag of ViewData de **Title** aanleveren voor de **_Layout** pagina
  - standaard wordt de _Layout pagina gebruikt, deze wordt verderop in dit hoofdstuk in detail toegelicht…

onze view moet dit aanleveren

```
@{
    ViewData["Title"] = "Brewers";
}
```

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - BeerhallMVC</title>

    …
</head>
```

stukje uit _Layout

Index - View

- Inhoud: en zo valt de puzzel in elkaar

Index.cshtml

```
@model IEnumerable<Beerhall.Models.Domain.Brewer>

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

<h2>@ViewData["Title"]</h2>

<p>
    <a asp-controller="Brewer" asp-action="Create">Voeg een nieuwe brouwer toe</a>
</p>

<table class="table table-striped table-condensed table-bordered">
    <tr>
        …
    </tr>

    @foreach (var item in Model) {
        <tr>
            …
        </tr>
    }
</table>
```

```
<div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; 2016 - BeerhallMVC</p>
        </footer>
    </div>
```

stukje uit _Layout

Index - View

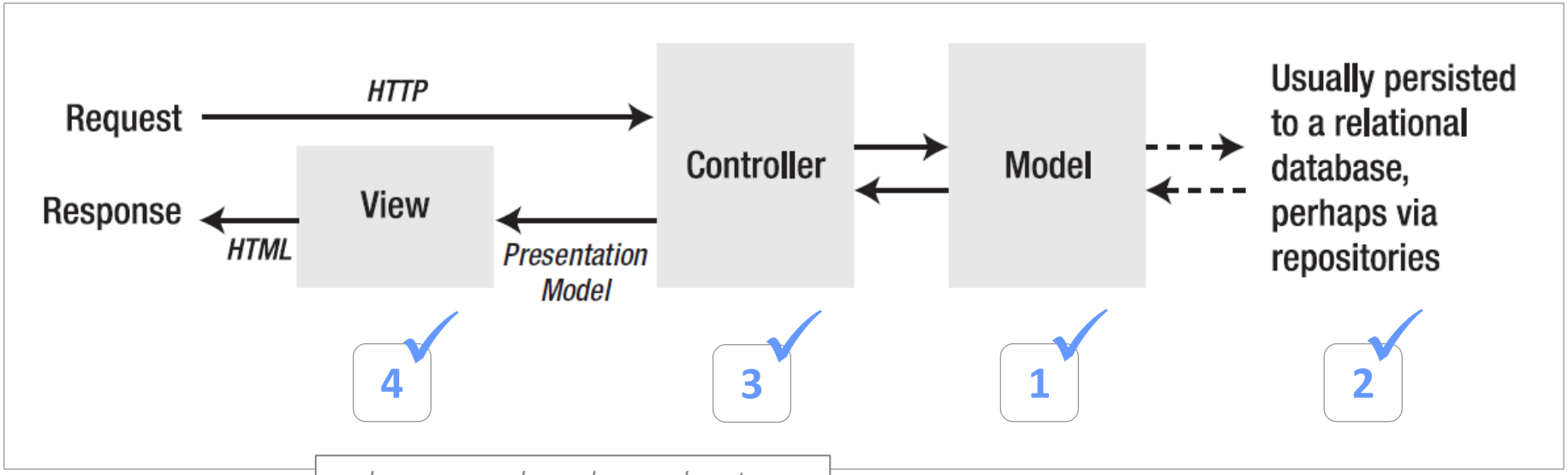
- Aanpassen view om de **totale omzet** van de brouwers samen te tonen
  - de view berekent de totale omzet **niet,** anders trek je business logica binnen in de view
  - de controller kan de totale omzet aanleveren via de ViewData
    - via het Model wordt de lijst van brewers aangeleverd
    - via ViewData wordt de totale omzet aangeleverd

```
public IActionResult Index() {
        IEnumerable<Brewer> brewers = _brewerRepository.GetAll().OrderBy(b=>b.Name).ToList();
        ViewData["TotalTurnover"] = brewers.Sum(b => b.Turnover);
        return View(brewers);
    }
```

BrewerController.cs

```
@model IEnumerable<Beerhall.Models.Domain.Brewer>

…

<table class="table table-striped table-condensed table-bordered">
…
</table>
<p>Total turnover: @($"{(int)ViewData["TotalTurnover"]:C}")</p>
```

Index - View



we kunnen nu verder werken aan de rest van de CRUD operaties…

```
commit Add functionality Brewer - Index
```

Edit

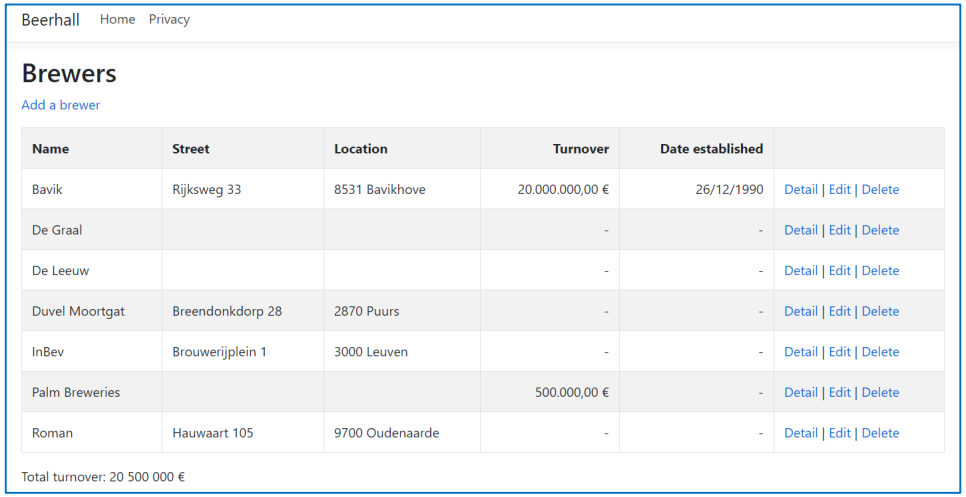
HoGent

Edit

- De pagina geeft een formulier met alle gegevens van de brewer die we wensen te editeren
- De gebruiker kan editeren en de gewijzigde gegevens bewaren, of het editeren annuleren

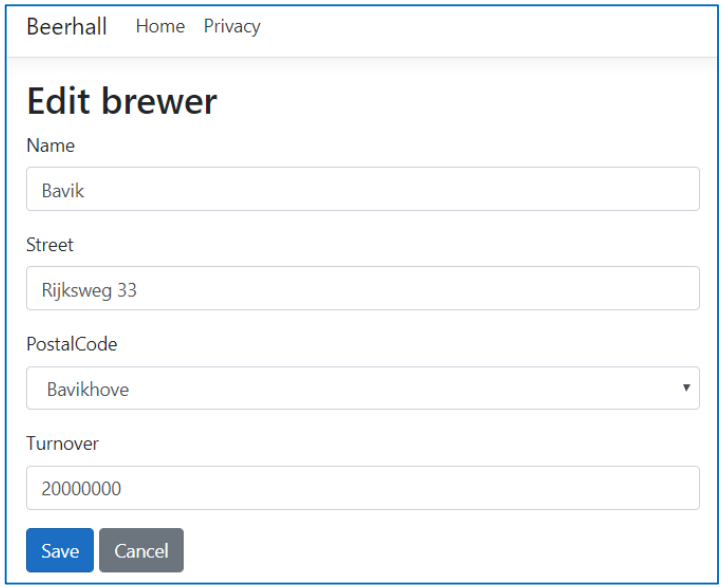
*gebeurt via **HTTPPOST***

*gebeurt via **HTTPGET***



Edit [GET]

- **Edit [GET]**
  - **Controller** verantwoordelijkheden
    - alle nodige gegevens van de gewenste brewer ophalen
    - de juiste view selecteren en deze de gegevens aanbieden
  - **View** verantwoordelijkheden
    - de gegevens presenteren zodat gebruiker kan editeren, bewaren of annuleren



Edit [GET] - Controller

- de controller krijgt via de parameter id de nodige informatie binnen van de view Index

```
<td>
    <a asp-controller="Brewer" asp-action="Detail" asp-route-id="@item.BrewerId">Detail</a> |
    <a asp-controller="Brewer" asp-action="Edit" asp-route-id="@item.BrewerId">Edit</a> |
    <a asp-controller="Brewer" asp-action="Delete" asp-route-id="@item.BrewerId">Delete</a>
</td>
```

Index.cshtml

```
app.UseMvc(routes => {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Brewer}/{action=Index}/{id?}");
            });
```

Routing configuratie in StartUp.cs

```
public IActionResult Edit(int id) {
            throw new NotImplementedException();
}
```

Edit action method in BrewerController

Edit [GET] - Controller

- … de controller kan zo de gewenste brewer uit de repository ophalen

```
public IActionResult Edit(int id) {
            Brewer brewer = _brewerRepository.GetBy(id);
            …
}
```

- deze brewer kan nu doorgegeven worden aan de view, maar we gaan gebruik maken van een **ViewModel**
- een viewmodel heeft als specifieke doel de **gewenste data aan te leveren voor een view**
  - het viewmodel zal **enkel en alleen die properties van onze domeinobjecten** bevatten, die nodig zijn in de view
    - als bv. DateEstablished geen deel uitmaakt van de edit view zal het geen deel uitmaken van ons viewmodel
    - overposting is nu niet meer mogelijk
  - het viewmodel kan **mogelijks een combinatie van properties van verschillende domeinklassen** bevatten

Edit [GET] - Controller

- de viewmodels stoppen we in een submap van Models genaamd ViewModels
- de naam van het viewmodel geeft aan voor welke view het zal dienen: **BrewerEditViewModel**

```
using Beerhall.Models.Domain;

namespace Beerhall.Models.ViewModels {
    public class BrewerEditViewModel {
        public string Name { get; set; }
        public string Street { get; set; }
        public string PostalCode { get; set; }
        public int? Turnover { get; set; }

        public BrewerEditViewModel(Brewer brewer)
        {
            Name = brewer.Name;
            Street = brewer.Street;
            PostalCode = brewer.Location?.PostalCode;
            Turnover = brewer.Turnover;
        }
    }
}
```

Merk op dat alle setters publiek zijn, via input velden in de view zal de waarde kunnen aangepast worden

We zullen een dropdownlist maken met alle gemeenten in, wanneer de gebruiker een gemeente selecteert zal de unieke postalcode naar de conrtoller gestuurd worden

Constructor ontvangt domeinobject(en) en bouwt het viewmodel

Edit [GET] - Controller

- de controller geeft nu het viewmodel door aan de view…

```
public IActionResult Edit(int id) {
            Brewer brewer = _brewerRepository.GetBy(id);
            return View(new BrewerEditViewModel(brewer));
}
```

Edit [GET] - Controller

- **let op:** om het viewmodel aan te maken hebben we nood aan de Postalcode die in Location zit, we moeten er voor zorgen dat als we de brewer ophalen, we ook zijn location ophalen, even checken in de BrewerRepository…

aanpassing in BrewerRepository

```
public class BrewerRepository : IBrewerRepository {
    …
    public Brewer GetBy(int brewerId) {
            return _brewers.Include(b => b.Location).SingleOrDefault(b => b.BrewerId == brewerId);
    }
    …
}
```

```
public class BrewerEditViewModel {
        …
        public string PostalCode { get; set; }
        …

        public BrewerEditViewModel(Brewer brewer)
        {
            …
            PostalCode = brewer.Location?.PostalCode;
            …
        }
}
```

EF – Eager Loading

Edit [GET] - View

- we kunnen nu de view **Edit.cshtml** aanmaken in de map Views > Brewer

```
@model Beerhall.Models.ViewModels.BrewerEditViewModel
@{
    ViewData["Title"] = "Edit brewer";
}
<h2>@ViewData["Title"]</h2>

<form asp-action="Edit" method="post">
    <div class="form-group">
        <label asp-for="Name"></label>
        <input asp-for="Name" class="form-control" />
    </div>
    <div class="form-group">
        <label asp-for="Street"></label>
        <input asp-for="Street" class="form-control" />
    </div>
    <div class="form-group">
        <label asp-for="PostalCode"></label>
        <input asp-for="PostalCode" class="form-control" />
    </div>
    <div class="form-group">
        <label asp-for="Turnover"></label>
        <input asp-for="Turnover" class="form-control" />
    </div>
    <div>
        <button class="btn btn-primary" type="submit">Save</button>
        <a asp-action="Index" class="btn btn-default">Cancel</a>
    </div>
</form>
```

Edit.cshtml

Er wordt standaard gebruik gemaakt van Bootstrap

Focus on TagHelper

- vereenvoudigen het werk om HTML pagina's op de server te genereren
  - onderdeel van de .cshtml views, Razor markup
  - very HTML-like, met IntelliSense
  - werken in op HTML elementen
    - ~element name
    - ~attribute name
    - ~parent tag
- MVC Core bevat een aantal voorgedefinieerde tag helpers
- Je kan eenvoudig je eigen tag helpers ontwikkelen in C#

Focus on TagHelper

- `@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers`
  - tag helpers moeten expliciet toegevoegd worden aan een view
  - standaard wordt dit gedaan in **_ViewImports** die in de map **Views** zit
    - de inhoud van _ViewImports heeft effect op alle views in de folder waarin het is geplaatst, alsook de subfolders daarvan
      - standaard kan je dus op elke view gebruik maken van TagHelpers, tenzij je op een view gebruik maakt van @removeTagHelper
    - een ander typisch gebruik van _ViewImports is de declaratie van @using
      - zo vermijd je het herhalen van @using in verschillende views

Focus on TagHelper

- voorbeeld **anchor** tag helper

```
<a asp-action="Index" asp-controller="Brewer" class="btn btn-
default">List brewers</a>
```

de taghelper zal de generatie van een <a> element bepalen

de naam van de action method die zal aangeroepen worden

de naam van de controller die zal gebruikt worden

klassiek attribuut, verwijzend naar Bootstrap klassen

```
<a class="btn btn-default" href="/">List brewers</a>
```

gegenereerde HTML

in de routing (zie StartUp – Configure) is Brewer als de default-controller en Index als de default-action gedefinieerd…

Focus on TagHelper

- voorbeeld **form** tag helper

```
<form asp-action="Edit" method="post">
```

  - genereert en stelt het action-attribuut in voor een action method in een controller of een benoemde route
    - gebruik **asp-controller** attribuut om controller te selecteren
      - hier niet gespecifieerd, neemt huidige controller: Brewer
    - gebruik **asp-action** attribuut om action method in te stellen
      - hier expliciet ingesteld op Edit
    - gebruik **asp-route-<parameter name>** attribuut om extra parameters toe te voegen aan de route
      - hier niet gespecifieerd, enkel de huidige route-value: id = 1
  - genereert een hidden request verification token om cross-site request forgery te kunnen voorkomen
    - meer hierover in een volgend hoofdstuk

```
<form method="post" action="/Brewer/Edit/1">
…
    <input name="__RequestVerificationToken" type="hidden" value="CfDJ8OIHYFYge0dPrEOE92DFCD-
E2HfdMSFjtD8taub8hS6Lr88XKs0aiE6iQfMS9Ds4KVgmvMPKrr9_g6IUjWeUb4B1IugDrvkDjOct_p0An0T7Uhpecvu3-Inqb71d6lO-
2W9pEvKkRdKfXMVn8jWTY0E" />
</form>
```

gegenereerde HTML

Focus on TagHelper

- voorbeeld **input** tag helper
  - in de form op de Brewer – Edit view vinden we

```
<input asp-for="Turnover" class="form-control" />
```

dit is de naam van een property van BrewerEditViewModel die doorgegeven werd aan deze view

  - asp-for zorgt voor de generatie van een **id** en **name** attribuut gebaseerd op de property “TurnOver”
  - asp-for stelt het **type** attribuut in gebaseerd op het type van de Property

.NET type	Input Type
Bool	type="checkbox"
String	type="text"
DateTime	type="datetime"
Byte	type="number"
Int	type="number"
Single, Double	type="number"

gegenereerde HTML

```
<input class="form-control"
type="number" id="Turnover"
name="Turnover" value="20000000" />
```

Focus on TagHelper

https://docs.asp.net/en/latest/mvc/views/tag-helpers/index.html

- in een volgend hoofdstuk gaan we data annotaties gebruiken en zal de kracht van tag helpers nog duidelijker worden…

Edit [GET] - View

- **aanpassen van de view:** we willen de gebruiker een **dropdownlist** aanbieden voor het editeren van de locatie
  - **de controller** kan alle locaties ophalen en aanleveren via ViewData, hiervoor heeft de controller nood aan de LocationRepository: constructor injectie

aanpassing in StartUp.cs

```
public void ConfigureServices(IServiceCollection services) {
    …
    services.AddScoped<ILocationRepository, LocationRepository>();
}
```

aanpassing in BrewerController.cs

```
public class BrewerController : Controller {
    private readonly IBrewerRepository _brewerRepository;
    private readonly ILocationRepository _locationRepository;

    public BrewerController(IBrewerRepository brewerRepository, ILocationRepository locationRepository) {
        _brewerRepository = brewerRepository;
        _locationRepository = locationRepository;
    }
…
}
```

Edit [GET] - View

- **aanpassen van de view: dropdownlist** vervolg
  - SelectList is het aangewezen type om de lijst van locaties aan te leveren aan de view

public class ***SelectList*** *:* ***Microsoft.AspNetCore.Mvc.Rendering.MultiSelectList***

zie API of VS Object browser

Member of ***Microsoft.AspNetCore.Mvc.Rendering***

Summary:

Represents a list that lets users select a single item. This class is typically rendered as an HTML <select> element with the specified collection of Microsoft.AspNetCore.Mvc.Rendering.SelectListItem objects.

aanpassing in BrewerController.cs

```
public IActionResult Edit(int id) {
        Brewer brewer = _brewerRepository.GetBy(id);
        ViewData["Locations"] = new SelectList(
            _locationRepository.GetAll().OrderBy(l => l.Name),
            nameof(Location.PostalCode),
            nameof(Location.Name));
        return View(new BrewerEditViewModel(brewer));
}
```

IEnumerable van items

bij selectie wordt de waarde van deze property geretourneerd

de waarde van deze property wordt getoond in de dd-list

Edit [GET] - View

- **aanpassen van de view: dropdownlist** vervolg
  - de view gebruikt de ViewData
  - de **select tag helper** helpt om de dropdownlist te genereren

aanpassing in Edit.cshtml

```
…
<div class="form-group">
    <label asp-for="PostalCode"></label>
    <select asp-for="PostalCode" asp-items="@(ViewData["Locations"] as SelectList)" class="form-
control">
        <option value="">-- select location --</option>
    </select>
</div>
…
```

Wanneer de location van een brewer niet gekend is wordt "-- select location --" getoond in de dropdownlist

de select tag helper

Edit

HoGent

Edit [POST] - Controller

- **Edit [POST]**
  - **Controller** verantwoordelijkheden
    - de gegevens van het formulier ontvangen
    - controleren of de gegevens geldig zijn
    - het domein en de repositories aansturen
      - de brewer moet gewijzigd worden volgens de formuliergegevens
      - de gewijzigde brewer moet gepersisteerd worden
    - indien alles goed verloopt moet de controller redirecten naar de Index pagina
    - indien er iets verkeerd loopt moet de controller het formulier terug aanbieden

Edit [POST] - Controller

- **Edit [POST]**
  - de formulier gegevens worden met een **HTTP Post request** meegestuurd
  - deze HTTP Post request wordt afgehandeld door **een nieuwe action method**
  - via **MVC model binding** zullen de formulier gegevens als parameter aan deze action method aangeleverd worden

via dit attribuut kunnen we aangeven dat deze action method de HTTP Post Edit request zal afhandelen

model binding zal zorgen dat de formuliergegevens in deze parameter terecht komen

```
[HttpPost]
public IActionResult Edit(BrewerEditViewModel brewerEditViewModel, int id) {
    throw new NotImplementedException();
}
```

Edit [POST] - Controller

- **Merk op**
  - we hebben **eenzelfde URL** (bv. …/Edit/1) maar **twee action methods** dewelke, afhankelijk van de HTTP verb zullen worden aangeroepen
    - **get** – aanbieden van een formulier met initiële gegevens
    - **post** - verwerken van de formuliergegevens die werden teruggezonden
  - indien je een aparte URL neemt voor de post (bv. …/Save/1) dan zal
    - deze URL gebruikt worden wanneer gebruikers formuliergegevens opnieuw moeten invullen
    - kunnen gebruikers deze URL bookmarken, met mogelijks nare gevolgen…

Focus on Post/Redirect/Get pattern

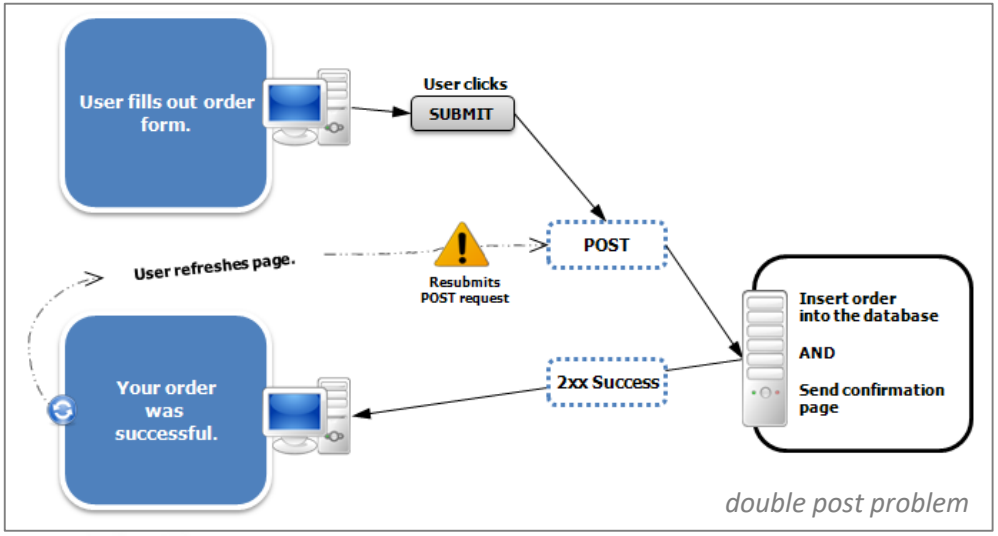
Post/Redirect/Get (**PRG**) is a web development design pattern that prevents some duplicate form submissions, creating a more intuitive interface for user agents (users). **PRG** supports bookmarks and the refresh button in a predictable way that does not create duplicate form submissions.

When a web form is submitted to a server through an HTTP POST request, a web user that attempts to refresh the server response in certain user agents can cause the contents of the original **HTTP POST** request to be resubmitted, possibly causing undesired results, such as a duplicate web purchase.[1]

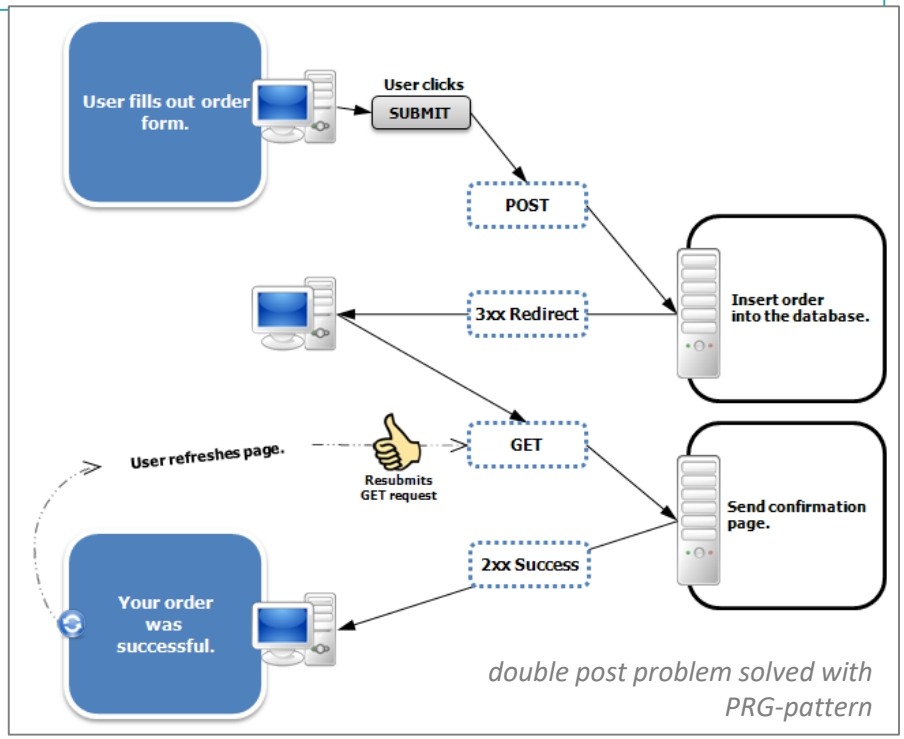
To avoid this problem, many web developers use the PRG pattern[2] — instead of returning a web page directly, the POST operation returns a redirection command. The HTTP 1.1 specification introduced the HTTP 303 ("See other") response code to ensure that in this situation, the web user's browser can safely refresh the server response without causing the initial HTTP POST request to be resubmitted. However most common commercial applications in use today (new and old alike) still continue to issue HTTP 302 ("Found") responses in these situations.

The PRG pattern cannot address every scenario of duplicate form submission. Some known duplicate form submissions that PRG cannot solve are:

- If a web user refreshes before the initial submission has completed because of server lag, resulting in a duplicate HTTP POST request in certain user agents.



double post problem

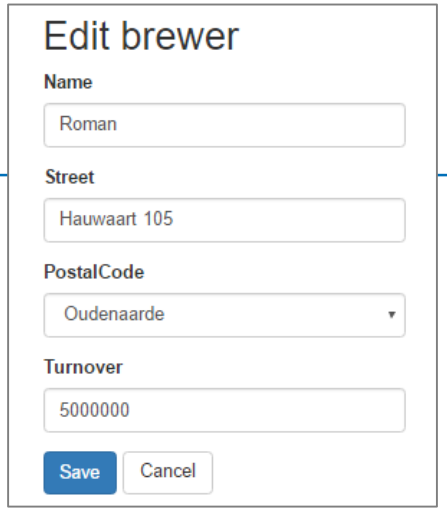


double post problem solved with PRG-pattern

HoGent

Edit [POST] - Controller

- **Model binding**
  - Laat ons kijken hoe de gegevens van het formulier de controller bereiken



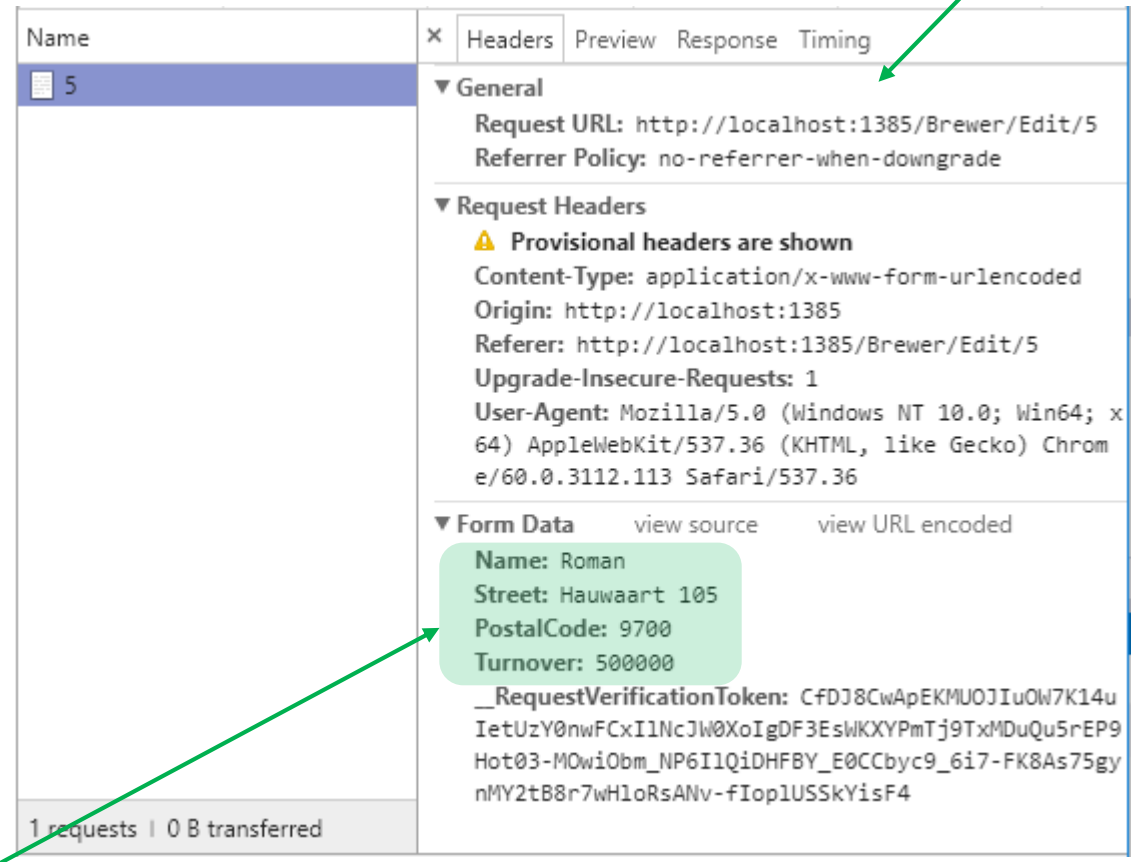

```
<h2>Edit brewer</h2>

<form method="post" action="/Brewer/Edit/5">
    <div class="form-group">
        <label for="Name">Name</label>
        <input class="form-control" type="text" id="Name" name="Name" value="Roman" />
    </div>
    <div class="form-group">
        <label for="Street">Street</label>
        <input class="form-control" type="text" id="Street" name="Street" value="Hauwaart 105" />
    </div>
    <div class="form-group">
        <label for="PostalCode">PostalCode</label>
        <select class="form-control" id="PostalCode" name="PostalCode">
            <option value="">-- select location --</option>
        <option value="1790">Affligem</option>
<option value="8531">Bavikhove</option>
<option value="3000">Leuven</option>
<option selected="selected" value="9700">Oudenaarde</option>
<option value="2870">Puurs</option>
<option value="8800">Roeselare</option>
</select>
    </div>
    <div class="form-group">
        <label for="Turnover">Turnover</label>
        <input class="form-control" type="number" id="Turnover" name="Turnover" value="" />
    </div>
    <div>
        <button class="btn btn-primary" type="submit">Save</button>
        <a class="btn btn-default" href="/">Cancel</a>
    </div>
<input name="__RequestVerificationToken" type="hidden"
value="CfDJ8CwApEKMUOJIuOW7K14uIeswm49U0nVlJy61sz_TN832ZTHM_SLRiLO1ZS_sF9FbCpzWfOD00yQuN8jUDva1vD7dMerxzLR61wQb
        <hr />
        <footer>
            <p>&copy; 2017 - Beerhall</p>
        </footer>
    </div>
```

de broncode van …/Brewer/Edit/5

*de waarden van de **name attributen** van de input velden zijn belangrijk voor model binding…*

Edit [POST] - Controller

- **Model binding**
  - … de gebruiker klikt op 



In de request body vinden we de Form Data: een lijst van key/value pairs,

key: naam attribuut van het input field

value: de waarde die werd ingevoerd

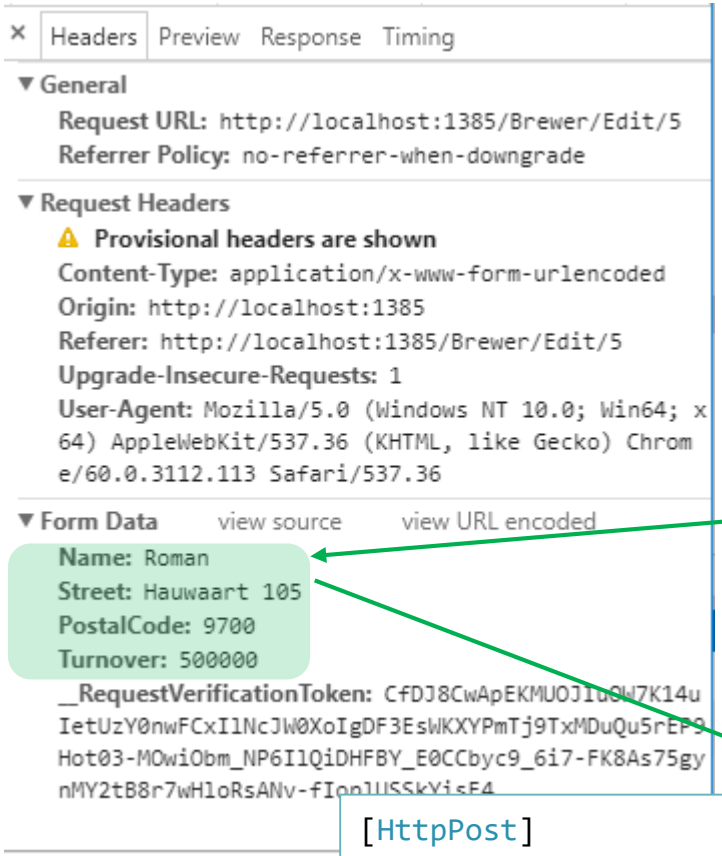
Edit [POST] - Controller

- **Model binding in actie**

MVC gaat een BrewerEditViewModel instantiëren,

keys uit de form data matchen met de properties van de instantie,

en de values gebruiken om waarden aan die properties toe te kennen




```
public class BrewerEditViewModel {
    public string Name { get; set; }
    public string Street { get; set; }
    public string PostalCode { get; set; }
    public int? Turnover { get; set; }
    …
}
```

```
[HttpPost]
public IActionResult Edit(BrewerEditViewModel brewerEditViewModel, int id) {
    throw new NotImplementedException();
}
```

Edit [POST] - Controller

- **Model binding in actie**
  - Om te kunnen instantiëren moet BrewerEditViewModel een **default/parameterloze constructor** hebben!
    - dit moeten we nog toevoegen, er wordt een run-time exception geworpen als deze niet aanwezig is…
  - Om waarden aan de properties te kunnen toekennen moeten de properties **publieke setters** hebben
    - dit is reeds ok

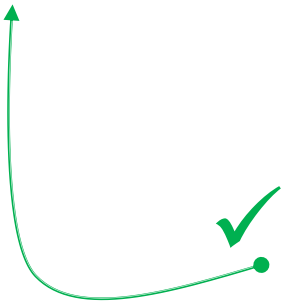
```
public class BrewerEditViewModel {
    public string Name { get; set; }
    public string Street { get; set; }
    public string PostalCode { get; set; }
    public int? Turnover { get; set; }

    public BrewerEditViewModel() {
    }

    public BrewerEditViewModel(Brewer brewer) : this() {
        Name = brewer.Name;
        Street = brewer.Street;
        PostalCode = brewer.Location?.PostalCode;
        Turnover = brewer.Turnover;
    }}
```

Edit [POST] - Controller

- **De Edit [POST] action method implementatie**
  - de controller kan nu met brewerEditViewModel aan de slag…

```
[HttpPost]
public IActionResult Edit(BrewerEditViewModel brewerEditViewModel, int id) {
    Brewer brewer = _brewerRepository.GetBy(id);
    brewer.Name = brewerEditViewModel.Name;
    brewer.Street = brewerEditViewModel.Street;
    brewer.Location = brewerEditViewModel.PostalCode == null ? null
:_locationRepository.GetBy(brewerEditViewModel.PostalCode);
    brewer.Turnover = brewerEditViewModel.Turnover;
    _brewerRepository.SaveChanges();
    return RedirectToAction(nameof(Index));
}
```

het is belangrijk de veranderingen te persisteren!



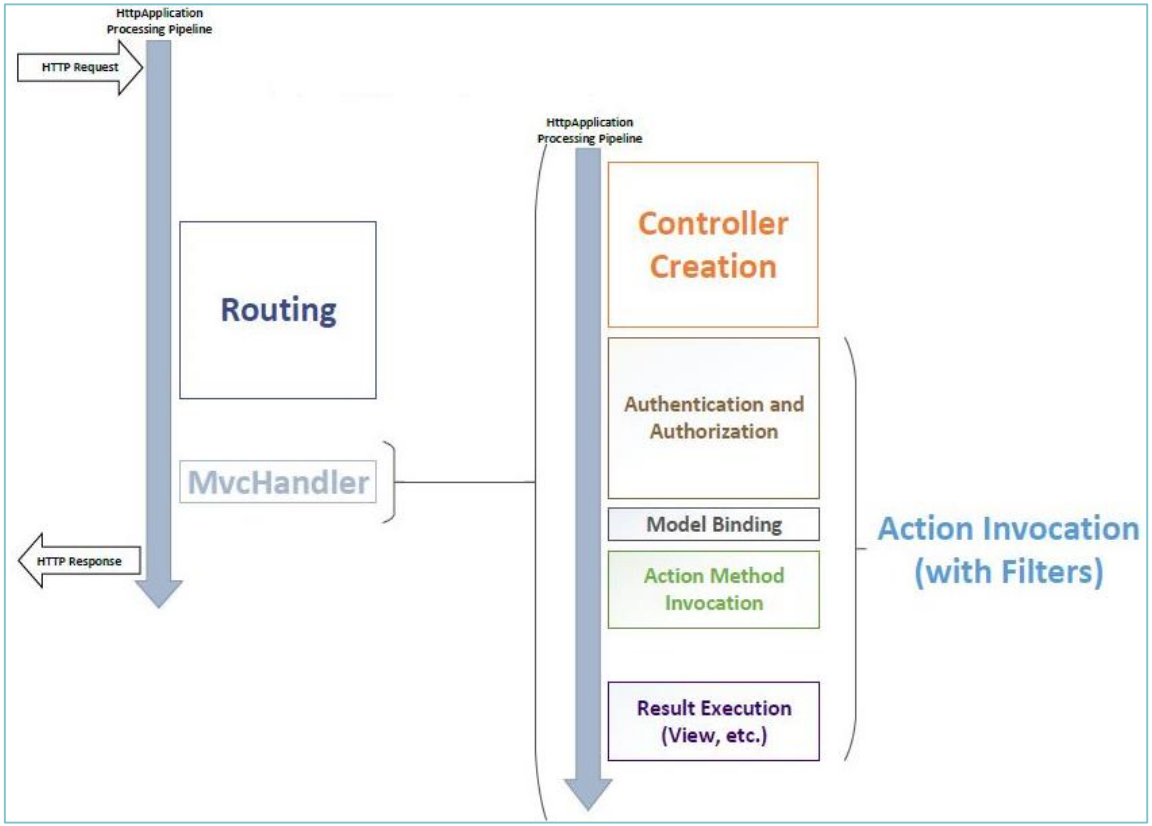
- Edit [POST]
  - Controller verantwoordelijkheden
    - de gegevens van het formulier ontvangen
    - controleren of de gegevens geldig zijn
    - het domein en de repositories aansturen
      - de brewer moet gewijzigd worden volgens de formuliergegevens
      - de gewijzigde brewer moet gepersisteerd worden
    - indien alles goed verloopt moet de controller redirecten naar de Index pagina
    - indien er iets verkeerd loopt moet de controller het formulier terug aanbieden

komt aan bod in een volgend hoofdstuk

Edit [POST] - Controller

Focus on Model Binding

https://docs.asp.net/en/latest/mvc/models/model-binding.html#how-model-binding-works



Model binding in ASP.NET Core MVC maps data from HTTP requests to action method parameters. The parameters may be simple types such as strings, integers, or floats, or they may be complex types. This is a great feature of MVC because mapping incoming data to a counterpart is an often repeated scenario, regardless of size or complexity of the data. MVC solves this problem by abstracting binding away so developers don't have to keep rewriting a slightly different version of that same code in every app. Writing your own text to type converter code is tedious, and error prone.

Focus on Model Binding

- er zijn verschillende manieren waarop gegevens van de client kunnen doorgegeven worden
  - **Form values**
  - **Route values**
  - **Query strings**

de model binder zal in deze volgorde aangereikte gegevens proberen te **binden**

Focus on Model Binding

- **Form values**
  - gegevens zitten in de **HTTP POST request**
  - dit kunnen **primitieve** types zijn
    - die op basis van parameter_name gebonden worden
    - voorbeeld: een alternatieve action method voor Brewer Edit [POST]

```
[HttpPost]
public IActionResult Edit(int id, string name, string street, string postalCode, int? turnover) {
    …
}
```

```
<form method="post" action="/Brewer/Edit/5">
    <input type="hidden" data-val="true" data-val-required="The BrewerId field is required." id="BrewerId" name="BrewerId" value="5" />
    <div class="form-group">
        <label for="Name">Name</label>
        <input class="form-control" type="text" id="Name" name="Name" value="Roman" />
    </div>
    <div class="form-group">
        <label for="Street">Street</label>
        <input class="form-control" type="text" id="Street" name="Street" value="Hauwaart 105" />
    </div>
```

Focus on Model Binding

- **Form values, vervolg**
  - dit kunnen **complexe** types (klassen zijn)
    - de Brewer Edit [POST] was hier een voorbeeld van
    - model binding kan ook met nog **meer complexe types**
      - wanneer een klasse properties heeft die op zich weer klassen zijn zal via reflectie, en op een recursieve manier, de structuur doorlopen worden en op basis van parameter_name.property_name binding gerealiseerd worden
    - model binding kan ook met **collections**
      - binding kan gebeuren op basis van parameter_name[index] (of kortweg [index]), of
      - op basis van parameter_name[key] (kortweg [key]) voor bv. dictionary types

Focus on Model Binding

- **Route values**
  - gegevens zitten in een benoemd **URL segment**
  - de definitie van de **routing** is hier belangrijk
    - namen van segmenten in MapRoute komen overeen met namen van parameters van de action method
    - we gebruikten dit in de Brewer Edit [GET]
      - zie anchor tag helper, asp-route-<parameter name> attribuut

```
<td>
    <a asp-controller="Brewer" asp-action="Detail" asp-route-id="@item.BrewerId">Detail</a> |
    <a asp-controller="Brewer" asp-action="Edit" asp-route-id="@item.BrewerId">Editeer</a> |
    <a asp-controller="Brewer" asp-action="Delete" asp-route-id="@item.BrewerId">Verwijder</a>
</td>
```

Index.cshtml

```
app.UseMvc(routes => {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Brewer}/{action=Index}/{id?}");
            });
```

Routing configuratie in StartUp.cs

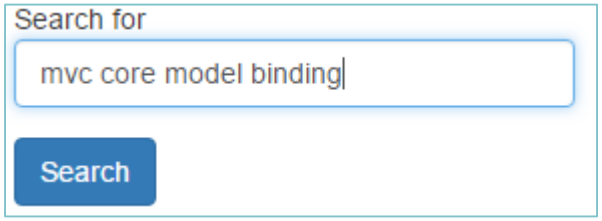
voorbeeld van een gegenereerde URL: **…/Brewer/Edit/5**

```
public IActionResult Edit(int id) {
            throw new NotImplementedException();
}
```

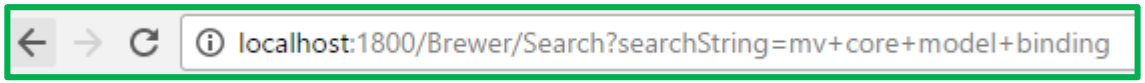
Edit action method in BrewerController

Focus on Model Binding

- **Query strings**
  - gegevens zitten in een **query string deel van de URL**
    - de query string bevat de naam=waarde paren die als basis voor de binding dienen
    - voorbeeld




```
<form asp-action="Search" method="get">
    <div class="form-group">
        Search for <input type="search" name="searchString"
class="form-control" />
    </div>
     <div>
        <button class="btn btn-primary" type="submit">Search</button>
    </div>
</form>
```



"mvc core model binding"

```
public IActionResult Search(string searchString) {

            …

}
```

Create

HoGent

Create

- De pagina geeft een formulier waarop alle gegevens van een brewer kunnen ingevuld worden
  - dit formulier is analoog aan het edit-formulier
- De gebruiker kan de gegevens bewaren (i.e. de brewer aanmaken), of kan annuleren
- Ook hier zullen we weerom
  - een GET en een POST hebben
  - gebruik maken van het PRG pattern

Create [GET] - Controller

- URL: …/Brewer/Create
- Action method

```
public IActionResult Create() {
    throw new NotImplementedException();
}
```

  - de controller zal de view gebruiken die we voor Edit hebben gemaakt
    - er moet een leeg brewerEditViewModel doorgegeven worden als model
    - de lijst met locations moet doorgegeven worden via de ViewData
      - dit deden we reeds voor Edit, we kunnen deze code hergebruiken
      - refactor time: we extraheren dit stukje code in een aparte methode
        - maak gebruik van de refactoring mogelijkheden van Visual Studio

Create [GET] - Controller

- Create [GET] action method

```
public IActionResult Create() {
    ViewData["Locations"] = GetLocationsAsSelectList();
    return View(nameof(Edit), new BrewerEditViewModel());
}
```

we moeten de naam van de view specifieren want dit is niet de default view voor deze action method

  - de methode die we via de refactoring verkregen:

```
private SelectList GetLocationsAsSelectList() {
    return new SelectList(
        _locationRepository.GetAll().OrderBy(l => l.Name),
        nameof(Location.PostalCode),
        nameof(Location.Name));
}
```

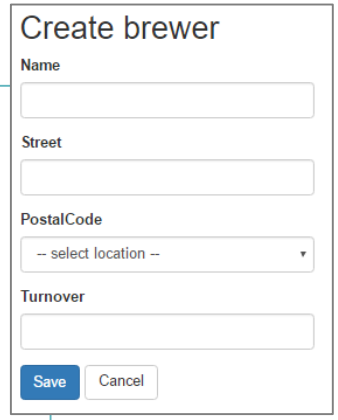
Create [GET] - View

- Via ViewData kunnen we doorgeven aan de view of het een edit of een create betreft? bv, in Create method

```
ViewData["IsEdit"] = false;
```

- ...en een kleine ingreep in de Edit view volstaat om dezelfde view te kunnen gebruiken voor Create en voor Edit

```
@model Beerhall.Models.ViewModels.BrewerEditViewModel

@{
    ViewData["Title"] = (bool)ViewData["IsEdit"] ? "Edit brewer" : "Create brewer";
}

<h2>@ViewData["Title"]</h2>

<form asp-action="Edit" method="post">
    …
</form>
```



Merk op dat als we geen expliciete controller/action opgeven bij een form-tag , automatisch bij httpPost de controller/action gebruikt wordt van de HttpGet

Create

HoGent

Create [POST] - Controller

- **Create [POST]**
  - **Controller** verantwoordelijkheden
    - de gegevens van het formulier ontvangen
    - controleren of de gegevens geldig zijn
    - het domein en de repositories aansturen
      - de brewer moet aangemaakt worden volgens de formuliergegevens
      - de brewer moet gepersisteerd worden
    - indien alles goed verloopt moet de controller redirecten naar de Index pagina
    - indien er iets verkeerd loopt moet de controller het formulier terug aanbieden

Create [POST] - Controller

- **Create [POST]**

```
[HttpPost]
public IActionResult Create(BrewerEditViewModel brewerEditViewModel) {
    throw new NotImplementedException();
}
```

  - de implementatie is vrij analoog aan Edit [Post] en ook hier kunnen we gebruik maken van refactoring
    - we extraheren een methode voor het mappen van een BrewerEditViewModel naar een Brewer…

```
private void MapBrewerEditViewModelToBrewer(BrewerEditViewModel brewerEditViewModel, Brewer brewer) {
        brewer.Name = brewerEditViewModel.Name;
        brewer.Street = brewerEditViewModel.Street;
        brewer.Location = brewerEditViewModel.PostalCode == null
            ? null
            : _locationRepository.GetBy(brewerEditViewModel.PostalCode);
        brewer.Turnover = brewerEditViewModel.Turnover;
    }
```

Create [POST] - Controller

- Resulterende implemenatatie voor Edit [POST] en Create [POST]

```
[HttpPost]
public IActionResult Create(BrewerEditViewModel brewerEditViewModel) {
    Brewer brewer = new Brewer(brewerEditViewModel.Name);
    MapBrewerEditViewModelToBrewer(brewerEditViewModel, brewer);
    _brewerRepository.Add(brewer);
    _brewerRepository.SaveChanges();
    return RedirectToAction(nameof(Index));
}
```

```
[HttpPost]
public IActionResult Edit(BrewerEditViewModel brewerEditViewModel) {
    Brewer brewer = _brewerRepository.GetBy(brewerEditViewModel.BrewerId);
    MapBrewerEditViewModelToBrewer(brewerEditViewModel, brewer);
    _brewerRepository.SaveChanges();
    return RedirectToAction(nameof(Index));
}
```

```
commit Add functionality Brewer - Create
```

Delete

HoGent

Delete

- De pagina geeft de naam van de brewer weer en vraagt om een bevestiging
- De gebruiker kan bevestigen, of kan annuleren
- Ook hier zullen we weerom een **GET** en een **POST** hebben
  - **Golden rule:** waneer gegevens worden aangepast, of verwijderd, op de server, maken we steeds gebruik van eenzelfde URL en voorzien we een GET en een POST versie voor de action method. Zo kan de aanpassing/verwijdering niet gebeuren door het volgen van een link…

Delete [GET] - Controller

- URL: …/Brewer/Delete/4
- Action method

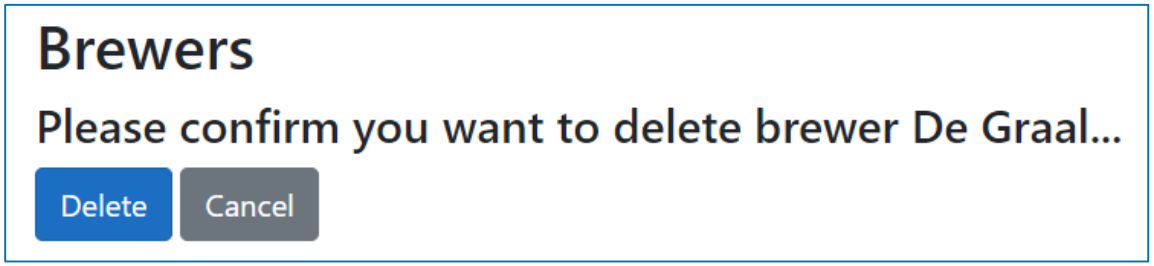
```
public IActionResult Delete(int id) {
    throw new NotImplementedException();
}
```

  - de controller dient enkel de naam van de brewer door te geven aan de view, dit kan via ViewData gebeuren

```
public IActionResult Delete(int id) {
    ViewData[nameof(Brewer.Name)] = _brewerRepository.GetBy(id).Name;
    return View();
}
```

Delete [GET] - View

- een eenvoudige view kan volstaan




```
@{
    ViewData["Title"] = "Brewers";
}

<h2>@ViewData["Title"]</h2>

<h4>Please confirm you want to delete brewer @ViewData["Name"]…</h4>

<form asp-action="Delete" method="post">
    <div>
        <button class="btn btn-primary" type="submit">Delete</button>
        <a asp-action="Index" class="btn btn-default">Cancel</a>
    </div>
</form>
```

Delete

HoGent

Delete [POST] - Controller

- **Delete [POST]**
  - **Controller** verantwoordelijkheden
    - de gegevens van het formulier ontvangen
    - controleren of de gegevens geldig zijn
    - het domein en de repositories aansturen
      - de brewer moet verwijderd worden
    - de controller redirect naar de Index pagina

Delete [POST] - Controller

- **Delete [POST]**

```
[HttpPost, ActionName("Delete")]
public IActionResult DeleteConfirmed(int id) {
    _brewerRepository.Delete(_brewerRepository.GetBy(id));
    _brewerRepository.SaveChanges();
    return RedirectToAction(nameof(Index));
}
```

binnen een klasse kunnen geen twee methodes met identieke signatuur bestaan, de actionmethod in deze klasse geven we een andere naam, DeleteConfirmed, maar voor MVC blijft dit de action method Delete

- **Delete [POST]**
  - **Controller** verantwoordelijkheden
    - de gegevens van het formulier ontvangen
    - controleren of de gegevens geldig zijn
    - het domein en de repositories aansturen
      - de brewer moet verwijderd worden
    - de controller redirect naar de Index pagina

controle en reactie als iets verkeerd loopt komt aan bod in een volgend hoofdstuk

`commit Add functionality Brewer - Delete`

Layout

HoGent

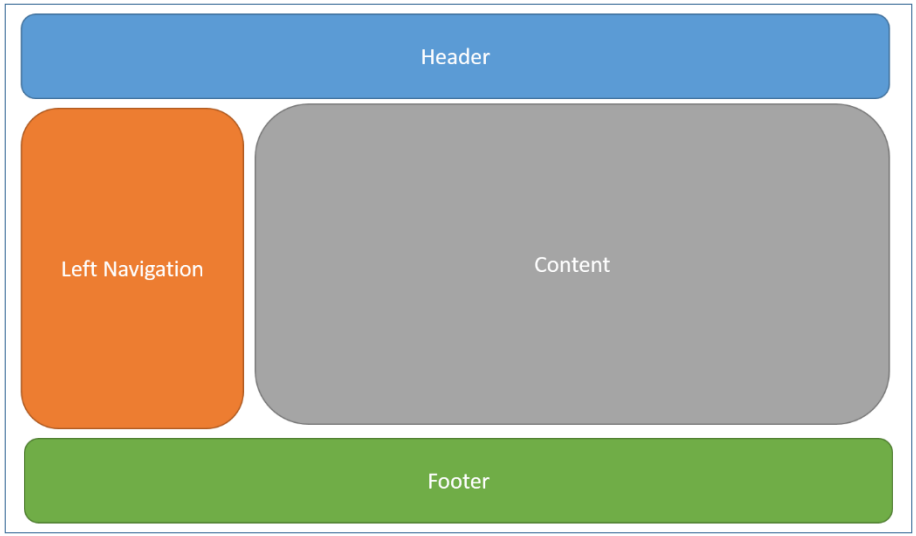
5. Layouts

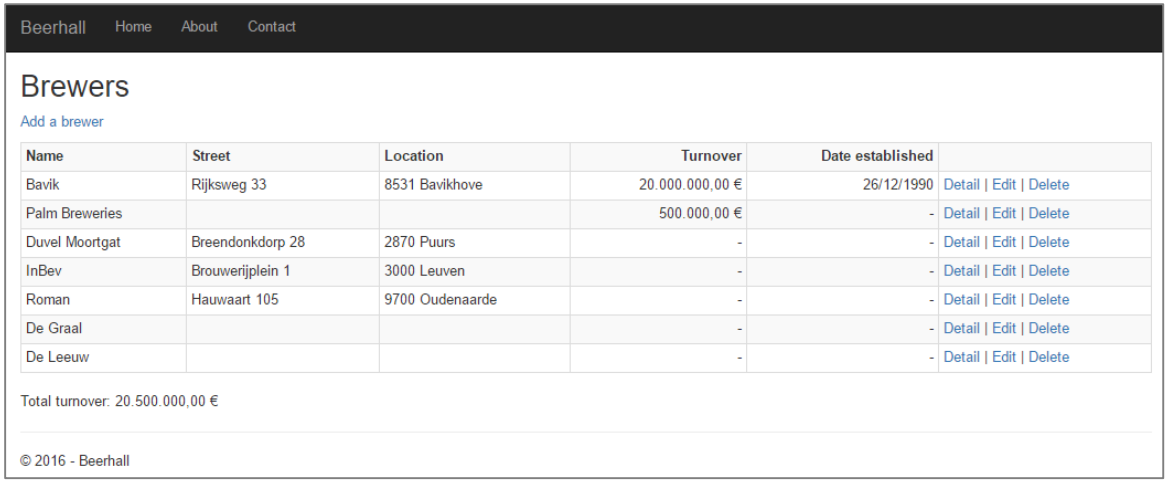
- De meeste websites bieden een **consistente lay-out** aan over de verschillende pagina's heen
  - typische elementen in zo'n lay-out zijn header, navigation of menu, footer
  - scripts en stylesheets, ook dikwijls gebruikt door meerdere pagina's van een site, kunnen opgenomen worden in een lay-out
- De lay-out voorziet in sections. Zo kunnen pagina's, die gebruik maken van de lay-out, op specifieke plaatsen inhoud plaatsen

Layouts reduce duplicate code in views, helping them follow the Don't Repeat Yourself (DRY) principle.

5. Layouts

- Typische layout





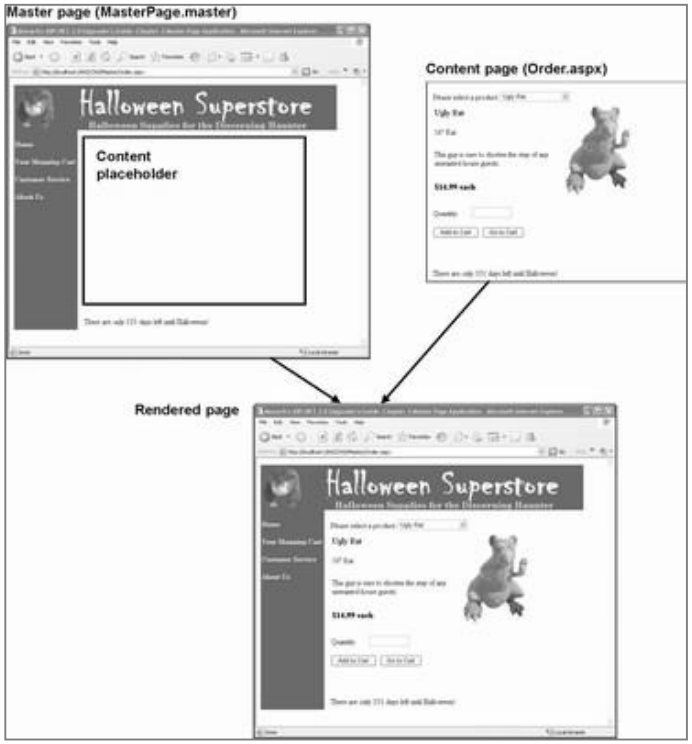
Header/Navigation

Content

Footer

5. Layouts

- Het concept Layout kent 2 onderdelen
  - Enerzijds zijn er de **paginasjablonen** waarin de layout website bepaald wordt (in core mvc is dit standaard _layout.cshtml in de Views > Shared map)
  - In de **pagina's gebaseerd op een layout** dien je enkel nog de inhoud te plaatsen **(= content page**).



De daadwerkelijke inhoud van een pagina wordt gecombineerd met het geselecteerd sjabloon.

Hierdoor bepaal je op 1 plaats hoe de indeling van de website is, en krijgen alle web pagina's dezelfde indeling.

5. Layouts

- **_layout.cshtml** in Views > Shared
  - bevat **HTML markup**
    - <html>, <head>, <body>, …
  - bevat de **gemeenschappelijke layout en inhoud**
    - header, footer, menu
  - bevat **placeholders**
    - @RenderBody: de plaats waar de inhoud van de webpage komt
    - @RenderScripts: de plaats waar de links voor de scripts van de webpage komen

5. Layouts

ViewData[“Title”]: inhoud van de title tag.

links naar de CSS

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Beerhall</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-b
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Be
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp
                        </li>
```

Navigation

5. Layouts

```
                        <a class="nav-link text-dark" asp-area="" asp-controller="Home'
                    </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">
    <div class="container">
        &copy; 2019 - Beerhall - <a asp-area="" asp-controller="Home" asp-action="Priva
    </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@RenderSection("Scripts", required: false)
</body>
</html>
```

De aangeleverde content wordt hier geplaatst

links naar de scripts

de aangeleverde scripts worden hier geplaatst

5. Layouts

- Hoe maken pagina's gebruik van _Layout?
  - **_ViewStart** in map Views
    - wordt uitgevoerd voor de rendering van **elke pagina**
    - bevat de instructie om _Layout te gebruiken

```
@{
    Layout = "_Layout";
}
```

  - **ViewData["title"]**
    - wordt gebruikt in _Layout
    - in onze pagina kunnen we de ViewData instellen

```
@{
    ViewData["Title"] = "Brewers";
}
```

  - De **inhoud** van onze pagina wordt geplaatst waar @RenderBody() staat

5. Layouts

- Je kan ook zelf sections toevoegen aan de layout via **@RenderSection("<section name>", required)**
  - let op: indien required true is zal er een exception geworpen worden indien de section niet aangeleverd wordt
- Voorbeeld
  - in de layout page:
  - in de view:

```
<div id="footer">@RenderSection("footer")</div>
```

```
@{
    ViewData["Title"] = "Contact";
}
<h2>@ViewData["Title"].</h2>

<address>
    …
</address>

@section footer {
    We provide a <strong>footer</strong> like this...
}
```

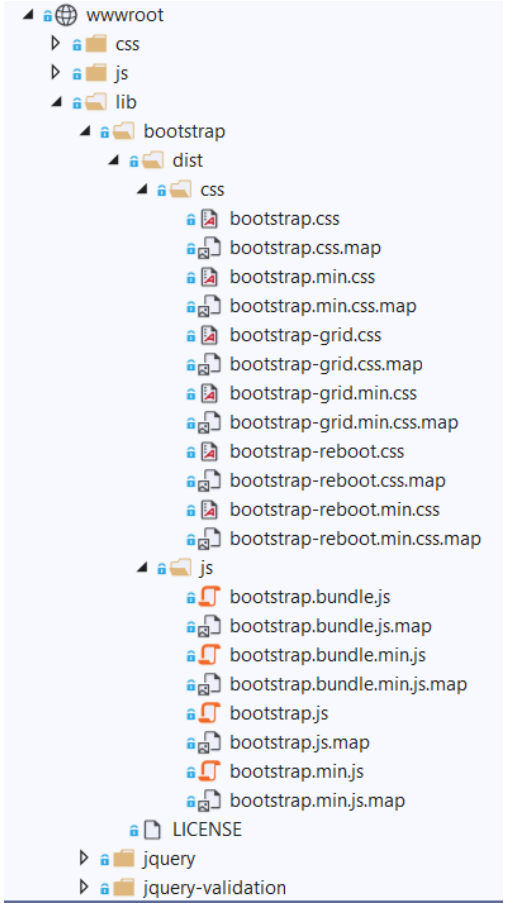
5. Layouts

- Je kan in de _Layout ook **default content** voorzien voor een section
- Voorbeeld _Layout

```
<div id="footer">
    @if (IsSectionDefined("footer"))
    {
        @RenderSection("footer")
    }
    else
    {
        <span>This is the default footer</span>
    }
</div>
```

5. Layouts

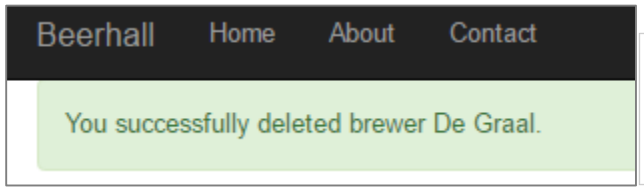
- **CSS** is standaard aanwezig in wwwroot > css map
  - bevat o.a. de opmaak van de site, er wordt naar verwezen in _Layout




```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Beerhall</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
</head>
```

5. Layouts

- _Layout kan ook gebruikt worden om op consistente manier **meldingen** weer te geven



Voorbeeld: als de brouwer werd verwijderd wordt geredirect naar de Index pagina en willen we de melding "You successfullly deleted brewer <brewer name>" tonen. Analoog voor Edit en Create...

  - merk op: dit kan niet via ViewData want deze is na elke request leeg en het prg pattern resulteert in een post en een get request
- **TempData**
  - via een cookie kan data bijgehouden worden **over verschillende HTTP requests**
    - TempData wordt geledigd zodra ze gelezen wordt in de View
    - Als je TempData in de View wil gebruiken zonder ze te ledigen kan je gebruik maken van Peek of Keep
      - meer info op http://www.c-sharpcorner.com/UploadFile/ansh06031982/using-tempdata-peek-and-keep-in-Asp-Net-mvc/

5. Layouts

- **TempData** in _Layout

```
<div class="container">
        <main role="main" class="pb-3">
    @if (TempData["message"] != null) {
        <div class="alert alert-success">@TempData["message"]</div>
    }
    @RenderBody()
    …
```

- **TempData** in Controller

```
[HttpPost, ActionName("Delete")]
 public IActionResult DeleteConfirmed(int id) {
    Brewer brewer = _brewerRepository.GetBy(id);
    _brewerRepository.Delete(brewer);
    _brewerRepository.SaveChanges();
    TempData["message"] = $"You successfully deleted brewer {brewer.Name}.";
    return RedirectToAction(nameof(Index));
}
```

dit doen we analoog voor Edit en Create

commit Add success messages using TempData

Exception handling

HoGent

6. Exceptionhandling in MVC

- Hoe kan de Controller **exceptions** op te vangen?
  - **TempData**
    - als een exception geworpen wordt bij het verwijderen/wijzigen/creëren van een brewer wordt geredirect naar de Index pagina en kunnen we een foutmelding meegeven in TempData
  - **ModelState**
    - **wanneer er iets fout loopt tijdens de model binding zal het framework geen exception werpen maar de ModelState aanpassen**. Het is de verantwoordelijkheid van de controller om deze te verifiëren en er gepast op te reageren
      - als gegevens onvolledig zijn ingevuld kan de controller zorgen dat het formulier opnieuw wordt aangeboden met de reeds ingevulde gegevens
      - deze vorm van validatie wordt in een later hoofdstuk behandeld

6. Exception handling

- Voorbeeld: afhandelen van exceptions via TempData

```
[HttpPost, ActionName("Delete")]
public IActionResult DeleteConfirmed(int id) {
    Brewer brewer = null;
    try {
        brewer = _brewerRepository.GetBy(id);
        _brewerRepository.Delete(brewer);
        _brewerRepository.SaveChanges();
        TempData["message"] = $"You successfully deleted brewer {brewer.Name}.";
    }
    catch {
        TempData["error"] = $"Sorry, something went wrong, brewer {brewer?.Name} was not deleted…";
    }
    return RedirectToAction(nameof(Index));
}
```

BrewerController

```
    @if (TempData["message"] != null) {
        <div class="alert alert-success">@TempData["message"]</div>
    }
    @if (TempData["error"] != null) {
        <div class="alert alert-warning">@TempData["error"]</div>
    }
    @RenderBody()
```

Catch exceptions and add error messages using TempData

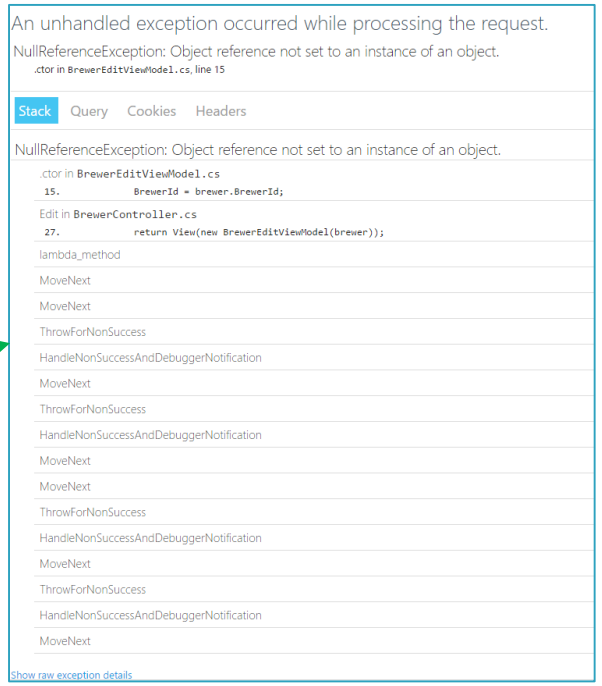
6. Exceptionhandling in MVC

- Wat als een exception **niet wordt opgevangen**?
  - **development:**
    - debug mode: exception wordt getoond op de lijn in broncode waar exception geworpen werd
    - without debugging: exception wordt getoond in de developer exception page in de browser

Environments, like “Development” and “Production”, are a first-class notion in ASP.NET Core and can be set using environment variables.

```
public void Configure(…) {
    …
    if (env.IsDevelopment()) {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
        }
    else {
        app.UseExceptionHandler("/Home/Error");
    }
    …
```

StartUp.cs



6. Exceptionhandling in MVC

- Wat als een exception **niet wordt opgevangen**?
  - **production:**

```
public void Configure(…) {
    …
    if (env.IsDevelopment()) {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
        }
    else {
        app.UseExceptionHandler("/Home/Error");
    }
    …
```

StartUp.cs

attribute based routing

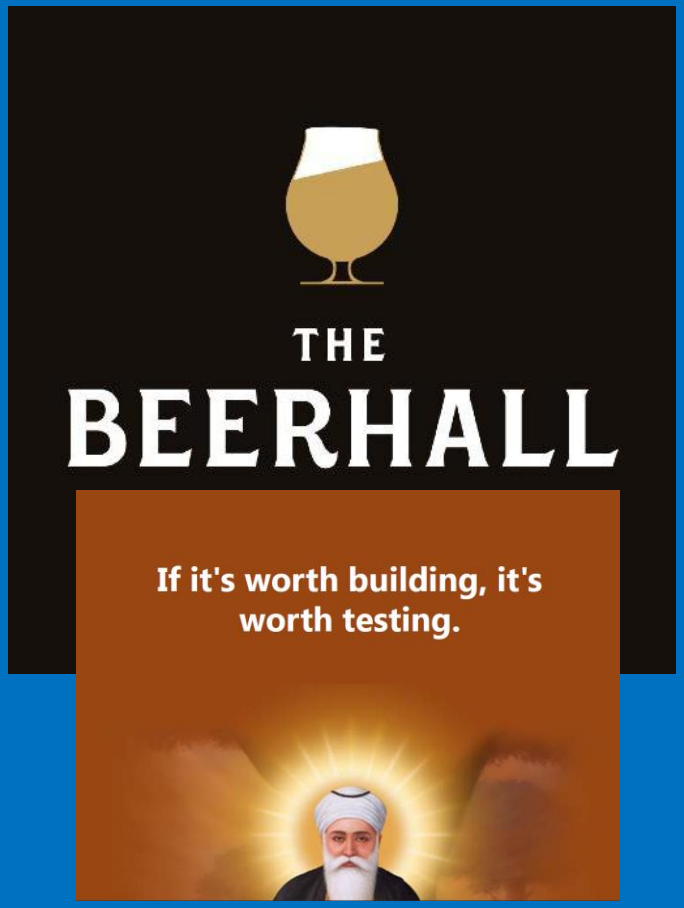
```
[Route("/Error")]
public IActionResult Error() {
    // Handle error here
}
```

HomeController.cs

6. Exceptionhandling in MVC

- **Exception Filters**
  - **Action Filters** zijn stukjes code die uitgevoerd worden net **na** of net **voor** de uitvoering van een action methode of ActionResult.
    - **Exception filters** kunnen gebruikt worden om exceptions die gebeuren tijdens controller instantiatie of model binding en niet worden opgevangen af te handelen
      - het uitvoeren van een filter kan worden ingesteld op niveau van actie methode, controller of op application niveau
      - maak bij voorkeur gebruik van de error handling middleware
        - gebruik exception filters enkel wanneer foutafhandeling specifiek voor een bepaalde action moet worden beschreven
      - zie hoofdstuk MVC In Depth voor meer details over filters

Unit testen van de Controller



HoGent

7. Unit testen voor Controller

- Unit testen maken **geen gebruik van de database**
  - unit testen zijn isolated, ze moeten ook runnen als de databank niet bereikbaar is…
  - unit testen zijn snel…
  - unit testen zijn repeatable…

7. Unit testen voor Controller

- Tijdens het unit testen kunnen we **dummy repositories** injecteren in de controller…
  - deze repositories kunnen gebruik maken van een **dummy DbContext**
  - de dummies bevatten een minimale implementatie
    - i.e. enkel wat nodig is om de testen te runnen

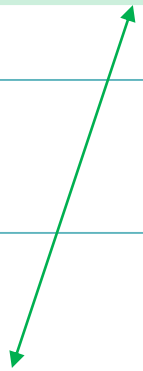
```
public class DummyBrewerRepository : IBrewerRepository {
…
    public DummyBrewerRepository(DummyApplicationDbContext dbContext) {
        this.dbContext = dbContext;
    }
…
```

```
public class DummyApplicationDbContext : DbContext {
    …
}
```

7. Unit testen voor Controller

- Unit testen met dummies…

```
public class BrewerControllerTest {
    private BrewerController _controller;

    public BrewerControllerTest() {
        DummyApplicationDbContext dbContext = new DummyApplicationDbContext();
        _controller = new BrewerController(new DummyBrewerRepository(dbContext),
                                           new DummyLocationRepository(dbContext));
    }
}
```



```
public class BrewerController : Controller {
    private readonly IBrewerRepository _brewerRepository;
    private readonly ILocationRepository _locationRepository;

    public BrewerController(IBrewerRepository brewerRepository, ILocationRepository locationRepository) {
         _brewerRepository = brewerRepository;
        _locationRepository = locationRepository;
    }
…
}
```

7. Unit testen voor Controller

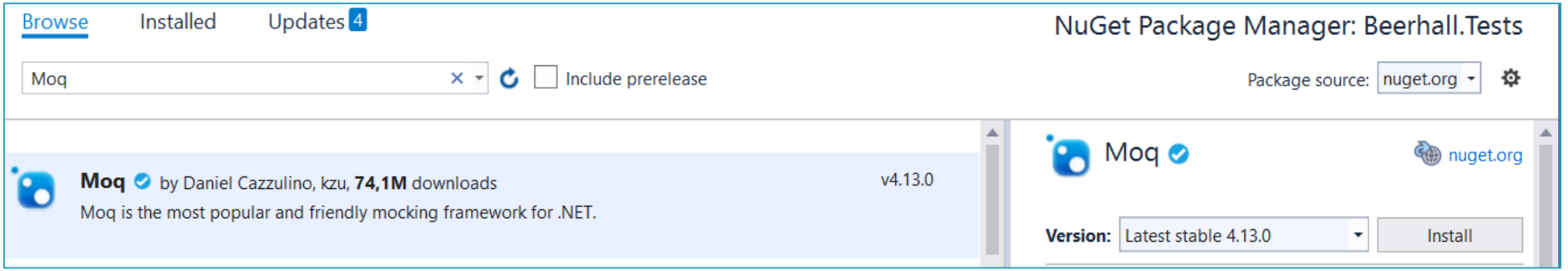
- Tijdens het unit testen kunnen we ook gebruik maken van **mocking**
- **Mock objecten** zijn krachtiger dan dummy objecten
  - Beschrijven hoe een object reageert op een call
    - we hoeven geen volledige implementatie van de klasse te voorzien
  - Maakt **verificatie** van de calls mogelijk
    - we kunnen vastleggen welke soort calls we kunnen verwachten, welke parameters die calls moeten bevatten, in welke volgorde de calls moeten gebeuren, hoeveel keer die calls moeten gebeuren…
      - bv. bij testen van de Index methode uit BrewerController moet de controller alle brewers ophalen: de methode GetAll() uit de _brewerRepository moet 1 keer aangeroepen worden

7. Unit testen voor Controller

- Mocking frameworks helpen ons om mock objecten aan te maken
  - Typemock, Rhino Mocks, Moq, …
  - Wij gaan gebruik maken van **Moq**: https://github.com/Moq

7. Unit testen voor Controller

- **Installatie** van Moq in het project Beerhall.Tests



7. Unit testen voor Controller

- **Declaratie en instantiatie** van de mocks

```
public class BrewerControllerTest {
        private BrewerController _controller;
        private Mock<IBrewerRepository> _brewerRepository;
        private Mock<ILocationRepository> _locationRepository;

        public BrewerControllerTest() {
            _brewerRepository = new Mock<IBrewerRepository>();
            _locationRepository = new Mock<ILocationRepository>();
            _controller = new BrewerController(_brewerRepository.Object,
_locationRepository.Object);
        }
```

Mock<Type> is een soort proxyklasse

Type is een interface, of een concrete klasse (enkel virtual methods van de klasse kunnen gemocked worden)

Creatie van een instantie van de Mock

.Object: een instantie van het type die gemocked wordt

7. Unit testen voor Controller

- De unit testen voor Index

Index

- De startpagina van Beerhall geeft een overzicht van alle brouwers met mogelijkheid om nieuwe brouwers aan te maken of bestaande te editeren of te verwijderen.
  - **Controller** verantwoordelijkheden
    - alle nodige gegevens van alle brewers ophalen
    - de juiste View selecteren en deze de nodige gegevens aanbieden

  - De Index action method moet
    - een geordende lijst van Brewers door geven aan de View
    - de totale turnover via ViewData doorgeven

```
public void Index_PassesOrderedListOfBrewersInViewResultModelAndStoresTotalTurnoverInViewData()
```

  - Elke test bevat
    - **A**rrange: trainen van de Mock
    - **A**ct: uitvoeren van de methode
    - **A**ssert: Controleren van het resultaat

7. Unit testen voor Controller

- **Trainen** van de mocks
  - we moeten aangeven hoe de mock objecten reageren op methode aanroepen
  - we moeten de mock enkel trainen voor de relevante methodes
    - **Setup(x).Returns(y)**
    - **Setup(x).Throws(z)**
      - x: de methode die je aanroept met zijn parameterwaarden
      - y: wat de methode zal retourneren
      - z: de exception die de methode dan zal retourneren
    - Het trainen kan je in de constructor doen of in de testmethodes zelf.

7. Unit testen voor Controller

- **Trainen** van de mocks
  - voorbeeld:
    - de action method Index moet alle brewers uit de repository halen, dit gebeurt via de methode GetAll()
    - in de testmethode kunnen we de mock als volgt trainen

```
//Arrange
Brewer bavik = new Brewer("Bavik") { BrewerId = 1 };
Brewer moortgat = new Brewer("Duvel Moortgat") { BrewerId = 2 };
_brewerRepository.Setup(m => m.GetAll()).Returns(new List<Brewer>() {bavik, moortgat});
```

    - wanneer de controller getAll() oproept zal deze call de lijst met de brewers bavik en moortgat retourneren
    - we kunnen expliciet nagaan of, en hoeveel keer, de methode GetAll() werd aangeroepen

7. Unit testen voor Controller

- **Act**
  - de action method Index moet worden uitgevoerd

```
//Act
IActionResult result = _controller.Index();
```

  - IActionResult is een interface met meerdere implementaties
    - return View()

View()

Creates a ViewResult object that renders a view to the response.

```
public virtual ViewResult View()
```

Returns

ViewResult

The created ViewResult object for the response.

    - return RedirectToAction()

RedirectToAction(String)

Redirects to the specified action using the *actionName*.

```
public virtual RedirectToActionResult RedirectToAction(string actionName)
```

Parameters

actionName System.String

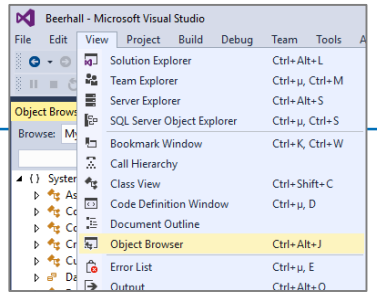
The name of the action.

Returns

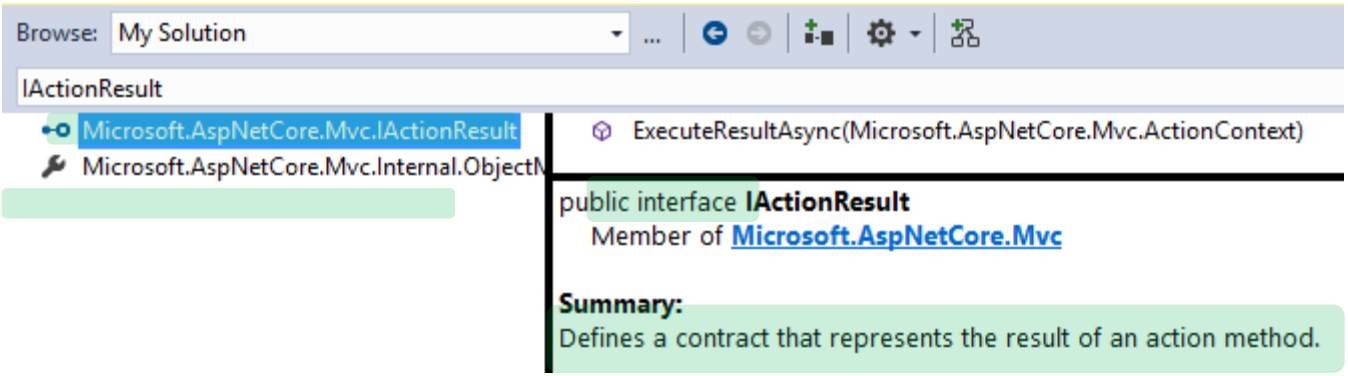
RedirectToActionResult

The created RedirectToActionResult for the response.

7. Unit testen voor Controller



- **IActionResult**
  - maak gebruik van de Object Browser (of de online help) om in detail te zien wat er voorzien is in klassen…



7. Unit testen voor Controller

- **ViewResult**
  - Is een implementatie van IActionResult.

Microsoft.AspNetCore.Mvc.ViewResult
Microsoft.AspNetCore.Mvc.ViewResult.View
Microsoft.AspNetCore.Mvc.ViewFeatures.Int
Microsoft.AspNetCore.Mvc.ViewFeatures.Int
Beerhall.Tests.Controllers.BrewerControllerT
Microsoft.AspNetCore.Mvc.PartialViewResu
Microsoft.AspNetCore.Mvc.PartialViewResu
Microsoft.AspNetCore.Mvc.ViewFeatures.Int
Microsoft.AspNetCore.Mvc.ViewFeatures.Int

ExecuteResultAsync(Microsoft.AspNetCore.Mvc.ActionContext)
ViewResult()
ContentType
Model
StatusCode
TempData
ViewData
ViewEngine
ViewName

Gets the View Data Model

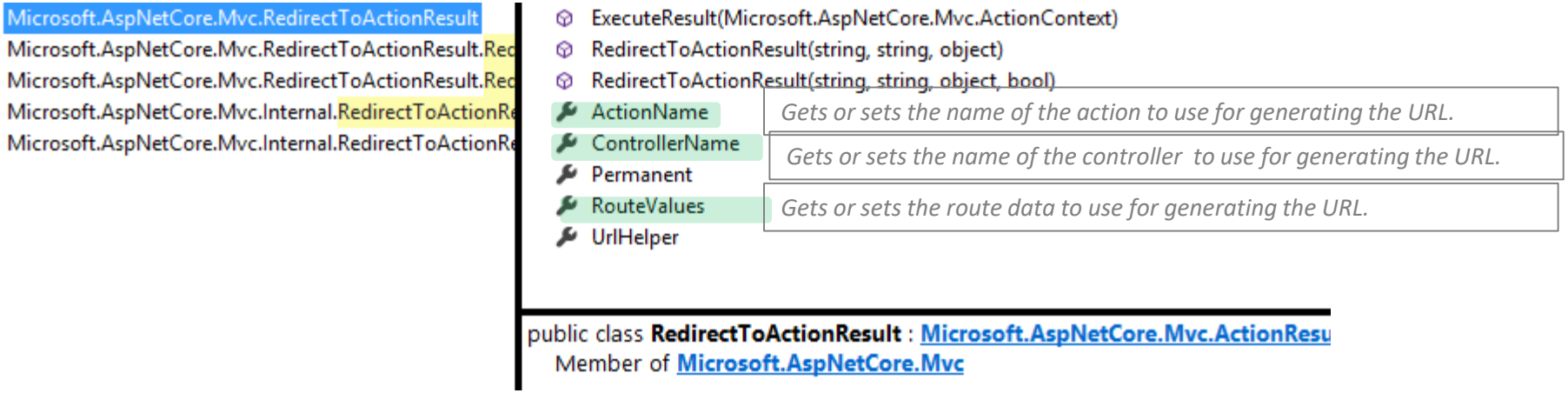
Gets or sets the Microsoft.AspNetCore.Mvc.ViewFeatures.ViewDataDictionary for this result.

public class **ViewResult** : Microsoft.AspNetCore.Mvc.ActionResult
Member of Microsoft.AspNetCore.Mvc

Summary:
Represents an Microsoft.AspNetCore.Mvc.ActionResult that renders a view to the response.

7. Unit testen voor Controller

- **RedirectToActionResult**
  - Is een implementatie van IActionResult.



7. Unit testen voor Controller

- **Act**
  - de action method Index retourneert dus een ViewResult

```
//Act
var result = Assert.IsType<ViewResult>(_controller.Index());
```

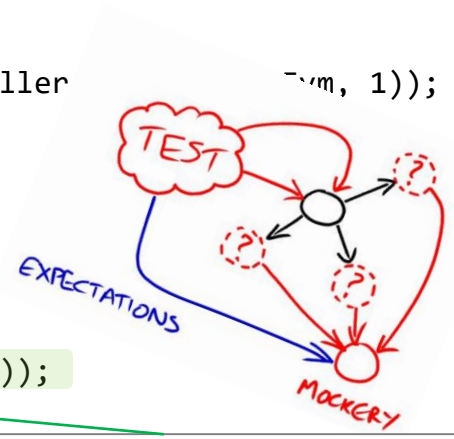
- **Assert** : controleert of property Model van ViewResult lijst van Brewers bevat, en Viewdata correct werd ingevuld

```
//Assert
var brewersInModel = Assert.IsType<List<Brewer>>(result.Model);
Assert.Equal(3, brewersInModel.Count);
Assert.Equal("Bavik", brewersInModel[0].Name);
Assert.Equal("De Leeuw", brewersInModel[1].Name);
Assert.Equal("Duvel Moortgat", brewersInModel[2].Name);
Assert.Equal(20050000, result.ViewData["TotalTurnover"]);
```

7. Unit testen voor Controller

- **Gebruik en verificatie** van de mocks
  - voorbeeld:
    - de testmethode Edit_ValidEdit_… bevat het trainen van een moq en de verificatie na de act.

```
public void Edit_ValidEdit_UpdatesAndPersistsBrewerAndRedirectsToActionIndex() {
        _brewerRepository.Setup(m => m.GetBy(1)).Returns(_dummyContext.Bavik);
        var brewerEvm = new BrewerEditViewModel(_dummyContext.Bavik)
        {
            Street = "nieuwe straat 1"
        };
        var result = Assert.IsType<RedirectToActionResult>(_controller
                                                            vm, 1));
        var bavik = _dummyContext.Bavik;
        Assert.Equal("Index", result?.ActionName);
        Assert.Equal("Bavik", bavik.Name);
        Assert.Equal("nieuwe straat 1", bavik.Street);
        _brewerRepository.Verify(m => m.SaveChanges(), Times.Once());
    }
```



hier gaan we na of de methode SaveChanges() exact 1 keer werd aangeroepen

7. Unit testen voor Controller

- Gebruik van een **DummyContext**
  - de concrete repositories maken gebruik van een ApplicationDbContext
  - we kunnen een **DummyApplicationDbContext** voorzien
    - deze stellen we zo op dat ze al onze nodige testgevallen bevat
    - we hoeven zo geen uitgebreide initialisatie code opnemen in de testmethodes of in de constructor van de testklasse
    - we denken aan het Don't Repeat Yourself principe…

```
public class BrewerControllerTest {
    private BrewerController _controller;
    private Mock<IBrewerRepository> _brewerRepository;
    private Mock<ILocationRepository> _locationRepository;
    private DummyApplicationDbContext _dummyContext;

    public BrewerControllerTest() {
        _dummyContext = new DummyApplicationDbContext();
        _brewerRepository = new Mock<IBrewerRepository>();
        _locationRepository = new Mock<ILocationRepository>();
        _controller = new BrewerController(_brewerRepository.Object, _locationRepository.Object);
}
```

7. Unit testen voor Controller

- Gebruik van een **DummyContext**
  - onze testmethode kan er nu als volgt uit zien

```
[Fact]
 public void Index_PassesOrderedListOfBrewersInViewResultModelAndStoresTotalTurnoverInViewData() {
            _brewerRepository.Setup(m => m.GetAll()).Returns(_dummyContext.Brewers);
            var result = Assert.IsType<ViewResult>(_controller.Index());
            var brewersInModel = Assert.IsType<List<Brewer>>(result.Model);
            Assert.Equal(3, brewersInModel.Count);
            Assert.Equal("Bavik", brewersInModel[0].Name);
            Assert.Equal("De Leeuw", brewersInModel[1].Name);
            Assert.Equal("Duvel Moortgat", brewersInModel[2].Name);
            Assert.Equal(20050000, result.ViewData["TotalTurnover"]);
        }
```

Focus on Mocking

- **Trainen** van een mock
  - Enkele voorbeelden
    - Alle brouwers retourneren

```
_brewerRepository.Setup(m => m.GetAll()).Returns(_dummyContext.Brewers);
```

    - Brouwer met id 1

```
_brewerRepository.Setup(m => m.GetBy(1)).Returns(_dummyContext.Bavik);
```

    - Onbestaande brouwer. (Mock retourneert null indien niet getraind voor een bepaald geval, maar je kan hier ook een regel voor maken). Als de methode null retourneert, moet je de null casten naar het juiste type

```
_brewerRepository.Setup(m => m.GetBy(10)).Returns((Brewer) null);
```

Focus on Mocking

- **Trainen** van een mock, vervolg
  - Als de methode void retourneert heb je geen Returns

```
_brewerRepository.Setup(m => m.Add(new Brewer("TestBrewer")));
```

  - Als de methode een exception throwt, gebruik Throws

```
_brewerRepository.Setup(m => m.Add(null)).Throws<ArgumentNullException>();
```

Focus on Mocking

- **Verificatie** van een mock
  - nagaan of de dependent klasse volgens een verwacht patroon gebruikt wordt

```
// Method should be called with specified parameter
mock.Verify(foo => foo.Execute("ping“))

 // Method should never be called
mock.Verify(foo => foo.Execute("ping"), Times.Never());

// Called at least once
mock.Verify(foo => foo.Execute("ping"), Times.AtLeastOnce());

// Verify setter invocation of property Name, regardless of value.
mock.VerifySet(foo => foo.Name);

// Verify setter called with specific value
mock.VerifySet(foo => foo.Name ="foo");

// Mocking for a range of values
mock.VerifySet(foo => foo.Value = It.IsInRange(1, 5, Range.Inclusive));
```

Focus on Mocking

- **Generischer** maken van **setup**
  - **It**: laat toe om matching conditie op te geven
  - Enkele mogelijkheden:

```
//IsAny<T> matches if parameter is any instance of type T
mock.Setup(foo => foo.Execute(It.IsAny<string>())).Returns(true);

// Is<T> matches based on a specified predicate
mock.Setup(foo => foo.Add(It.Is<int>(i => i % 2 == 0))).Returns(true);

// IsInRange<T> matches if parameter is between the defined values
mock.Setup(foo => foo.Add(It.IsInRange<int>(0, 10, Range.Inclusive))).Returns(true);

// IsRegex : matches a string parameter if it matches the specified regular expression
mock.Setup(x => x.Execute(It.IsRegex("[a-d]+", RegexOptions.IgnoreCase))).Returns("foo");
```

meer info? Zie: https://github.com/Moq/moq4/wiki/Quickstart

7. Unit testen voor Controller

- De action methods die gebruik maken van **TempData** verdienen speciale aandacht
  - de property TempData (van het type ITempDataDictionary) voor de controller moet ingesteld worden
  - we kunnen dit oplossen met een **mock** voor TempData

```
public class BrewerControllerTest {
    private BrewerController _controller;
    private Mock<IBrewerRepository> _brewerRepository;
    private Mock<ILocationRepository> _locationRepository;
    private DummyApplicationDbContext _dummyContext;
    public BrewerControllerTest() {
        _dummyContext = new DummyApplicationDbContext();
        _brewerRepository = new Mock<IBrewerRepository>();
        _locationRepository = new Mock<ILocationRepository>();
        _controller = new BrewerController(_brewerRepository.Object,
_locationRepository.Object){
        _controller.TempData = new Mock<ITempDataDictionary>().Object;
    }
}
```

7. Unit testen voor BrewerController

- Wat verwachten we van de **Edit-Get methode**?
- Wat verwachten we van de **Edit-Post methode**?
- Wat verwachten we van de **Create-Get methode**?
- Wat verwachten we van de **Create-Post methode**?
- Wat verwachten we van de **Delete-Get methode**?
- Wat verwachten we van de **Delete-Post methode**?

```
commit Add unit tests for BrewerController
```

Oefening

HoGent

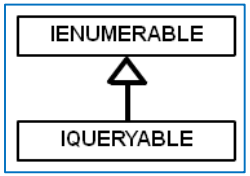
8. Oefening

- **Werk BeerHall verder uit** zodat
  - je voor een brouwer de lijst van bieren kunt tonen
    - ~detail pagina
  - je een nieuw bier aan een brouwer kunt toevoegen
  - je een bier van een brouwer kunt verwijderen

Enkele extra's

HoGent

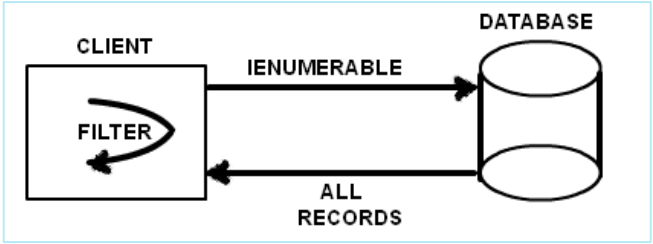
9. Extra's: Het type IQueryable



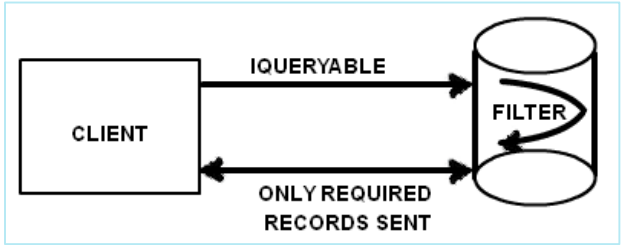
- In de namespace System.Linq vind je IQueryable<T>
  - dit kan ook gekozen worden als het returntype voor onze repository methodes die IEnumerable retourneren

```
IQueryable<Brewer> FindAll();
```

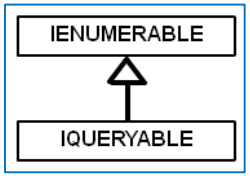
  - hoe verschillen deze twee collections?
    - **IEnumerable<T>**: aanvullingen (include, where, orderby,….) worden uitgevoerd in het geheugen



    - **IQueryable<T>**: aanvullingen (include, where, orderby,….) worden uitgevoerd in de database, i.e. ze breiden de query verder uit



9. Extra's: Het type IQueryable



- IQueryable

Repository

```
public IQueryable<T> FindAll()
{
    return _objectSet.Where(e => e.Id > 0);
}

public IEnumerable<T> FindAllAsEnumerable()
{
    return _objectSet.Where(e => e.Id > 0);
}
```

Controller

```
public ViewResult Index()
{
    var model = _repository.FindAll()
                           .OrderBy(e => e.HireDate);

    return View(model);
}
```

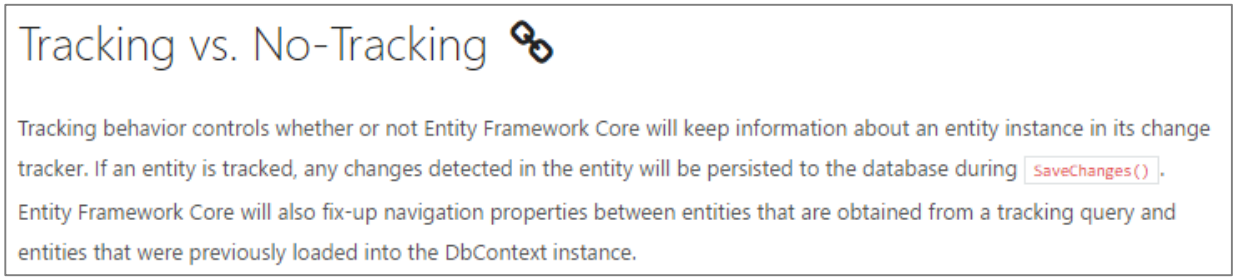
```
public ViewResult Index()
{
    var model = _repository.FindAllAsEnumerable()
                           .OrderBy(e => e.HireDate)
                           .Take(10);

    return View(model);
}
```

Generated queries

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name],
[Extent1].[HireDate] AS [HireDate]
FROM [dbo].[Employees] AS [Extent1]
WHERE [Extent1].[Id] > 0
ORDER BY [Extent1].[HireDate] ASC
```

```
SELECT
[Extent1].[Id] AS [Id],
[Extent1].[Name] AS [Name],
[Extent1].[HireDate] AS [HireDate]
FROM [dbo].[Employees] AS [Extent1]
WHERE [Extent1].[Id] > 0
```

9. Extra's: EF: AsNoTracking



Tracking vs. No-Tracking

Tracking behavior controls whether or not Entity Framework Core will keep information about an entity instance in its change tracker. If an entity is tracked, any changes detected in the entity will be persisted to the database during `SaveChanges()`. Entity Framework Core will also fix-up navigation properties between entities that are obtained from a tracking query and entities that were previously loaded into the DbContext instance.

- Per default worden alle queries die entities retourneren getracked
- Je kan de performantie verbeteren door voor read-only queries expliciet aan te geven dat de entities niet hoeven getracked te worden…

```
public IEnumerable<Brewer> GetAll() {
    return _brewers.Include(b => b.Location).Include(b => b.Beers).AsNoTracking().ToList();
}
```

voorbeeld van de GetAll met AsNoTracking in de BrewerRepository

9. Extra's: EF Logging

- Soms is het handig te zien welke queries EF genereert, zo kan je bv. de performantie in het oog houden
  - zie https://docs.microsoft.com/nl-nl/aspnet/core/fundamentals/logging?tabs=aspnetcore2x om logging te voorzien
  - of maak gebruik van SQL Server Profiler

9. Extra's: Good practice - Controller

- **Controllers bevatten geen business logica**
  - hiervoor **delegeert** de controller naar het **domein**
  - **voorbeeld**: in een action method moeten we alle bieren van een bepaald type voor een bepaalde brouwer ophalen
    - Slechte oplossing :

```
public ActionResult Bieren(int brouwerid, string type) {
  Brouwer b = brouwerRepository.GetBy(brouwerid);
  return View(b.Bieren.Where(b=>b.Type==type).ToList());
}
```

    - **Pas Law of Demeter toe in MVC**
      - Vermijd . . notatie (is in Linq heel eenvoudig)

9. Extra's: Good practice - Controller

- **Controllers bevatten geen business logica**
  - voorbeeld vervolg, een tweede slechte oplossing:

```
public ActionResult Bieren(int brouwerid, string type) {
  Ienumerable<Bier> bieren= bierenRepository.GetBy(brouwerid);
  return View(bieren.Where(b=>b.Type==type).ToList());}
```

- **Gebruik enkel repositories voor het ophalen van het root object!**
  - ophalen van geassocieerde objecten gebeurt via het domein!

9. Extra's: Good practice - Controller

- **Controllers bevatten geen business logica**
  - voorbeeld vervolg, een goede oplossing:

```
public ActionResult Bieren(int brouwerid, string type) {
  Brouwer b = brouwerRepository.GetBy(brouwerid);
  return View(b.GetBieren(type));}
```

- Merk op: **ook Views bevatten geen businesslogica**
  - presenteren enkel data
- Pas **design patterns** toe in je ontwerp

Referenties

HoGent

10. Referenties

- ASP.NET Core Fundamentals by Scott Allen – Pluralsight cursus - https://app.pluralsight.com/library/courses/aspdotnet-core-fundamentals/table-of-contents
- Building a Web App with ASP.NET CORE, MVC 6, EF Core, and Angular by Shawn Wildermuth – Pluralsight cursus - https://app.pluralsight.com/library/courses/aspnetcore-mvc-efcore-bootstrap-angular-web/table-of-contents
- Pro ASP.NET Core MVC: Seveth edition by Adam Freeman - Apress
  Softcover ISBN 978-1-4842-3149-4
  eBook ISBN 978-1-4842-3150-0

- Micrsoft documentatie
  - https://docs.microsoft.com/nl-nl/aspnet/index#pivot=core&panel=core_overview
  - https://www.asp.net/freecourses
  - https://docs.microsoft.com/en-us/ef/core/