.NET Advanced

ASP MVC

AP HOGESCHOOL ANTWERPEN

AP.BE

Topics

- MVC ?
  - Controller
  - View
  - Model
- Aanmaken van een nieuw project
  - Aanmaken van een “Controller”
  - Aanmaken van een “View”
  - Aanmaken van een “Model”
- Razor views

.NET Advanced

MVC

AP HOGESCHOOL ANTWERPEN

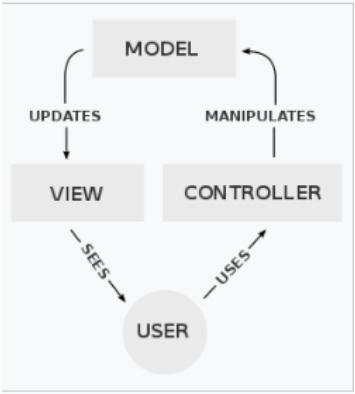
AP.BE

MVC

- Server-side framework
  - Standaard geen client side code nodig
    - Merk op: is echter wel mogelijk en/of soms nodig
  - Schermen worden aangemaakt op de server en in zijn geheel teruggestuurd naar de client
    - Merk op: Het is ook mogelijk om slechts een deel van een scherm op te vragen bij de server, bv. indien enkel een tabel moeten worden geüpdatet.

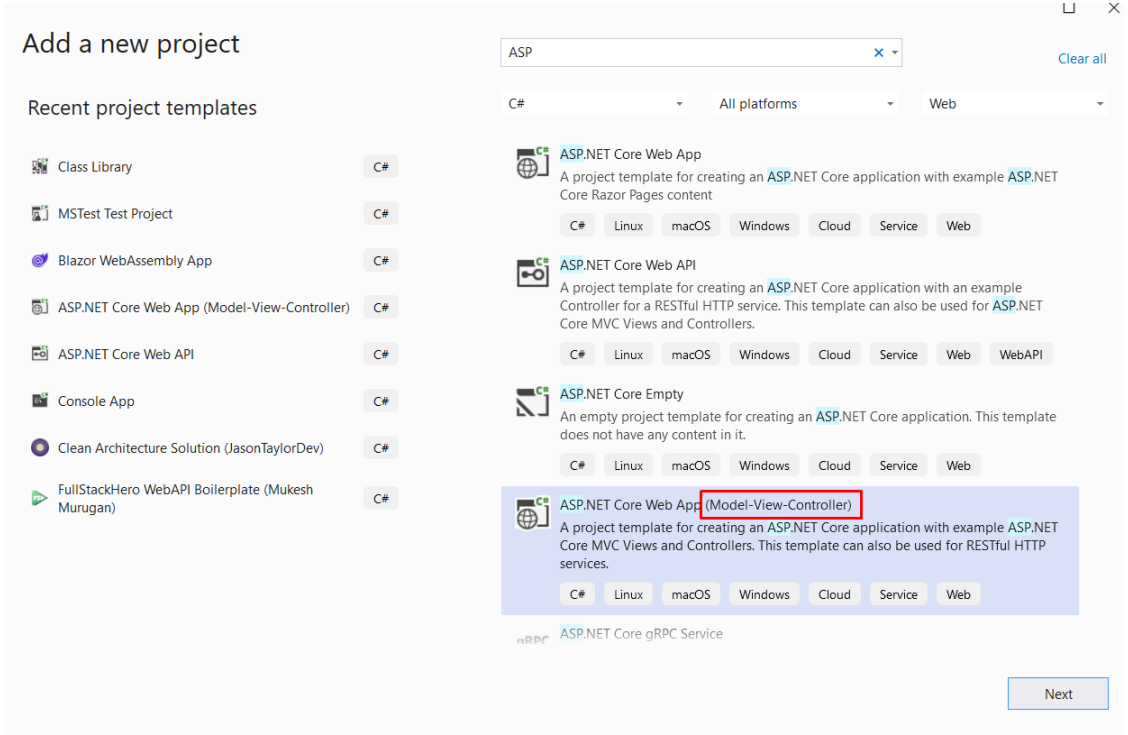
MVC

- Model-View-Controller
  - **Model**: De “data” (een store, boek, persoon,..)
  - **View**: Het eigenlijke scherm (HTML/CSS)
  - **Controller**: de tussenliggende partij
    - Vraagt data op en geeft door aan de view
    - Stuurt de view als response naar de client
    - Ontvangt een request van de client en verwerkt deze (data opslaan, nieuwe view aanmaken,…)
    - Communiceert met een achterliggende API

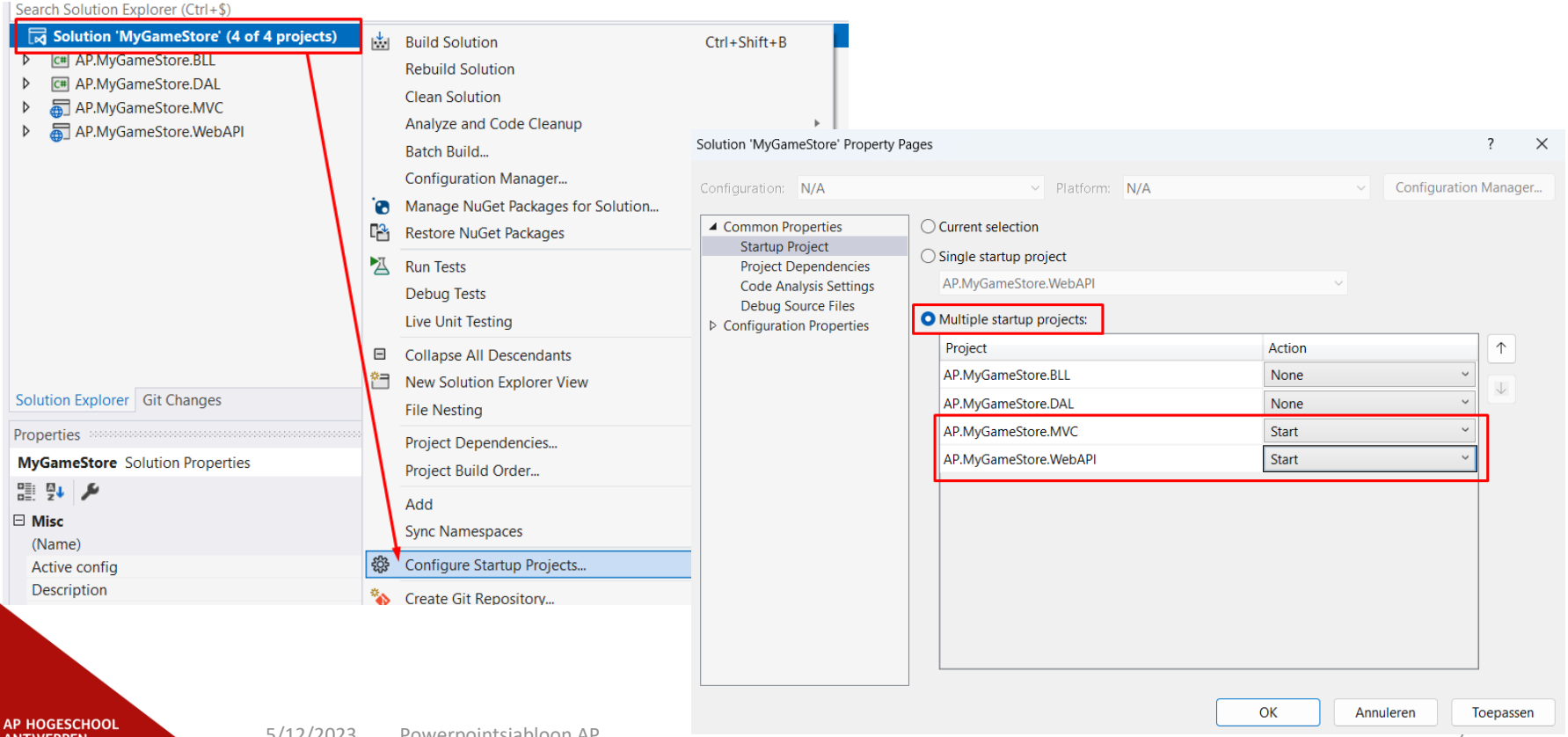


Bron: Wikipedia

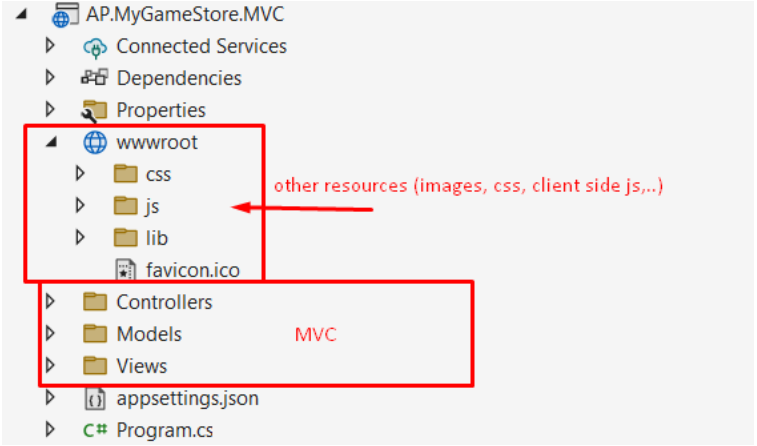
Aanmaken van een nieuw MVC project



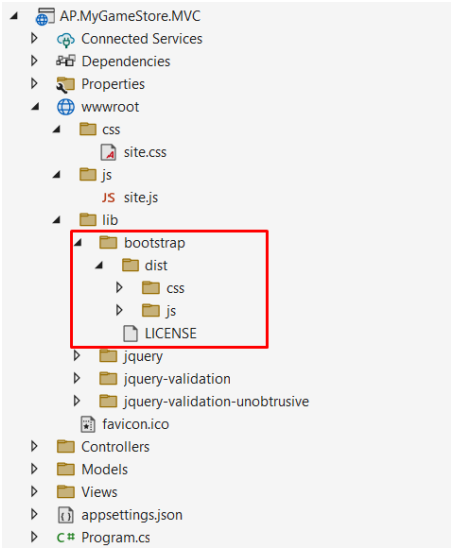
Meerdere startup projects (MVC + API)



Styling framework inbegrepen

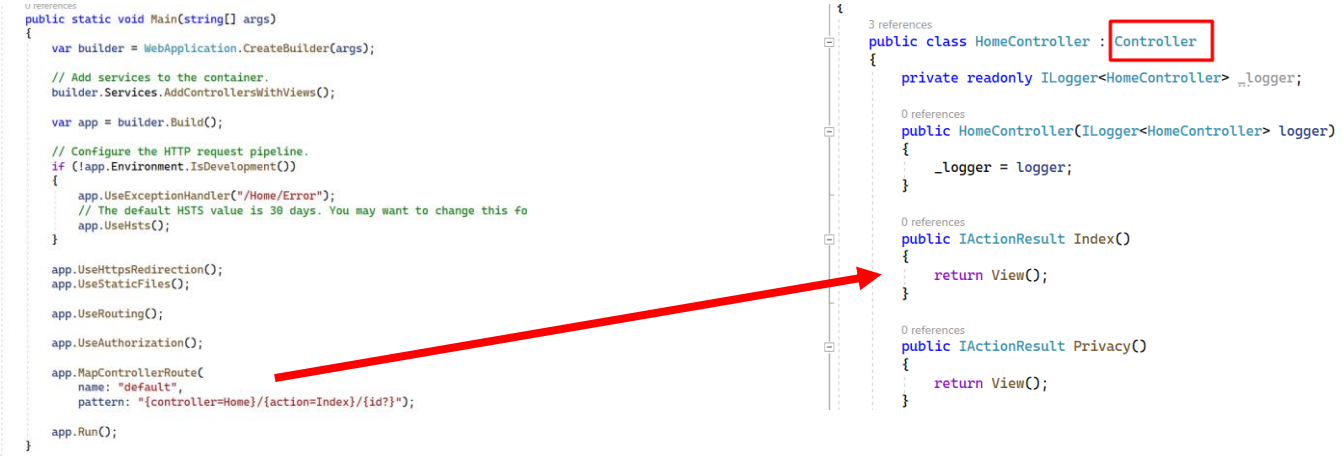


- Standaard wordt bootstrap voorzien als styling framework



Middleware en controllers

- Middleware pipeline en aanmaken van de controllers is vrij identiek aan een API project type
- MVC controllers erven echter van de **Controller** klasse
- Verschil met API is dat MVC controllers HTML teruggeven in de response (tegenover JSON bij de API)

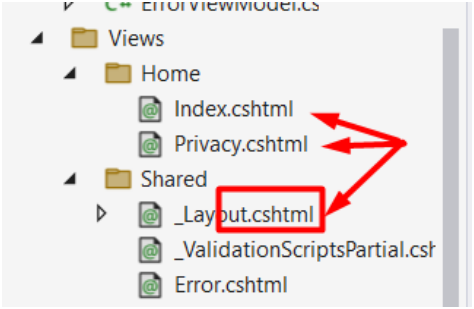


```
0 references
public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    // Add services to the container.
    builder.Services.AddControllersWithViews();

    var app = builder.Build();

    // Configure the HTTP request pipeline.
    if (!app.Environment.IsDevelopment())
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this fo
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

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

    app.Run();
}
```

```
{
3 references
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;

    0 references
    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

    0 references
    public IActionResult Index()
    {
        return View();
    }

    0 references
    public IActionResult Privacy()
    {
        return View();
    }
```

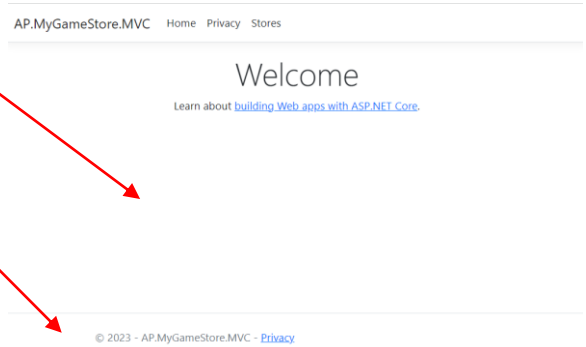
Views

- Deze map bevat alle **Razor** Views.
- Dit zijn **.cshtml** bestanden
- Een Razor view kan namelijk naast HTML ook nog c# code bevatten.
- Deze c# code zal op de server worden uitgevoerd vooraleer de view wordt teruggestuurd.
- Dit gebeurt aan de hand van de Razor rendering engine.



Shared Views

- Deze views worden hergebruikt doorheen de volledige applicatie
- **_Layout.cshtml** bevat de structuur van de applicatie
  - Header : menu, logo,..
  - Body: het gedeelte dat per pagina verschillend is
  - Footer: contact info,..

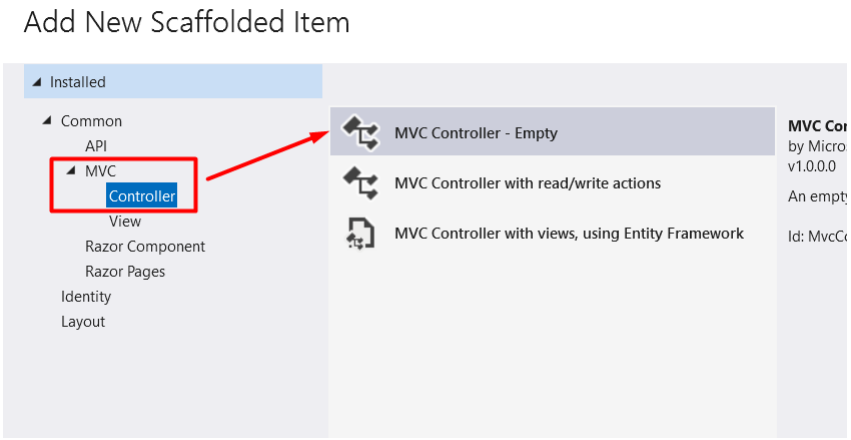


Weergeven van de Gamestores

- Om de gamestores te kunnen weergeven:
  1. Een HTTP request vanuit de browser naar onze MVC applicatie (route: ??)
  2. Een HTTP request vanuit MVC naar onze API om alle gamestores op te halen (route: /api/v1/stores)
  3. De JSON lijst van gamestores omzetten naar gamestore objecten
  4. De lijst van objecten omzetten naar een HTML tabel adhv. een Razor View
  5. Het resultaat (HTML) terugsturen naar de browser.

Weergeven van de gamestores

- Om een request te kunnen doen naar onze MVC app hebben we een controller nodig.
- We gebruiken dezelfde conventie als bij de API
- [naam]Controller.cs



Weergeven van de gamestores

- De MVC controller heeft als route steeds de naam van de controller (zonder het controller keyword)
- De MVC controller bevat steeds een Index Action
- Deze action staat ingesteld als de ‘default’ action
- Maw. deze action is bereikbaar via volgende routes:
  - /Stores/Index
  - /Stores

```
1 reference
public class StoresController : Controller
{
    0 references
    public StoresController()
    {
    }

    0 references
    public IActionResult Index()
    {
        return View();
    }
}
```

Weergeven van de gamestores

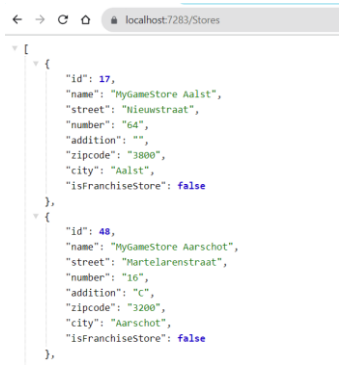
- Vanuit deze Index action kunnen we de data gaan opvragen aan onze API.
- Dit is een nieuwe HTTP (GET) request die we vanuit MVC naar de API gaan uitvoeren. We kunnen hiervoor gebruik maken van de **HTTPClient**
- Uiteraard gebeuren deze calls ook asynchroon

```
1 reference
public class StoresController : Controller
{
    0 references
    public StoresController()
    {
    }

    private static HttpClient sharedClient = new()
    {
        BaseAddress = new Uri("https://localhost:44352"),
    };

    0 references
    public async Task<IActionResult> Index()
    {
        var list = await sharedClient.GetStringAsync("/api/v1/stores");
        return Ok(list);
    }
}
```

adres van onze API

route om stores op te vragen

is dit wel een lijst ?



Weergeven van de gamestores

- Uiteraard is het niet de bedoeling dat de MVC controller een JSON string teruggeeft.
- We moeten deze nog omzetten naar c# objecten, die we vervolgens gaan omzetten naar een HTML tabel aan de hand van een “Razor View”
- Deze view bestaat echter nog niet….

```
1 reference
public class StoresController : Controller
{
    0 references
    public StoresController()
    {
    }

    private static HttpClient sharedClient = new()
    {
        BaseAddress = new Uri("https://localhost:44352"),
    };

    0 references
    public async Task<IActionResult> Index()
    {
        var stores = await sharedClient.GetFromJsonAsync<List<Store>>("/api/v1/stores");
        return View(stores);

    }
}
```

deserialisatie van JSON naar objecten

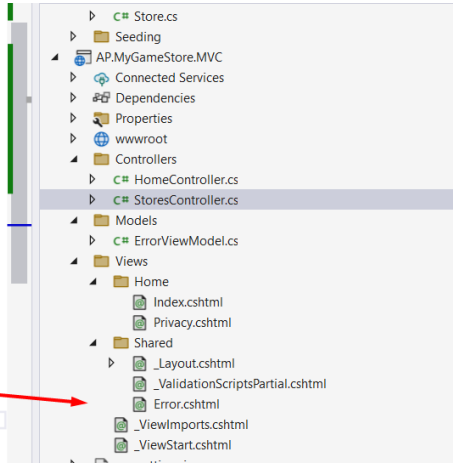
doorgeven van de data (lijst van Store objecten) naar de View

Weergeven van de gamestores

- Razor views, opdeling en naamgeving
- Er wordt een mapje voorzien per controller
- Voor elke action kan er dan een view worden voorzien

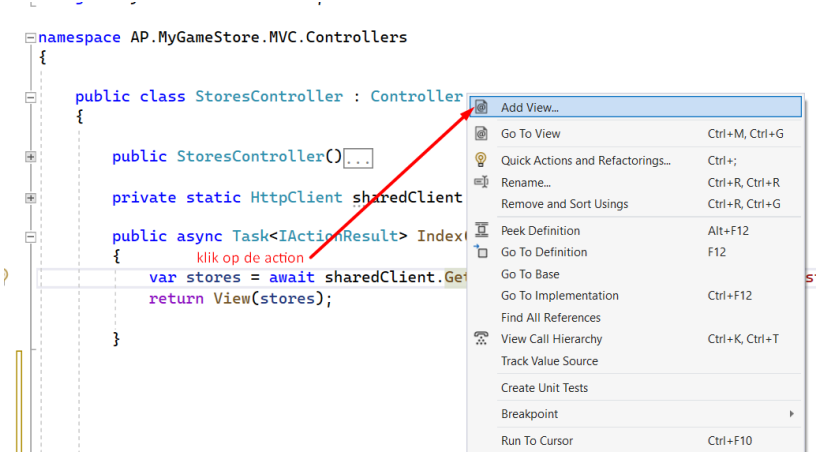
```
1 reference
public class StoresController : Controller
{
    0 references
    public StoresController()
    {
    }

    private static HttpClient sharedClient = new()
    {
        BaseAddress = new Uri("https://localhost:44352"),
    };

    0 references
    public async Task<IActionResult> Index()
    {
        var stores = await sharedClient.GetFromJsonAsync<List<Store>>("/api/v1/stores");
        return View(stores);
    }
}
```

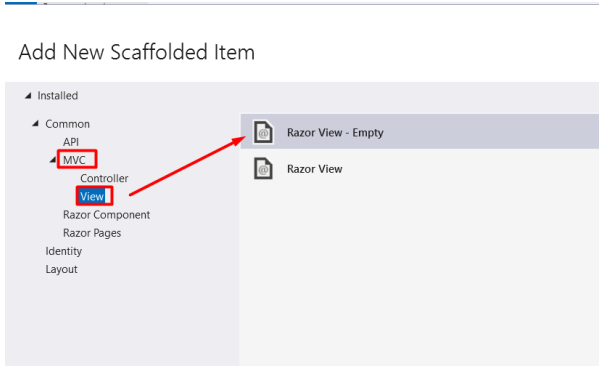
MVC zal op zoek gaan naar een view in het mapjes Stores
met als naam : Index .cshtml

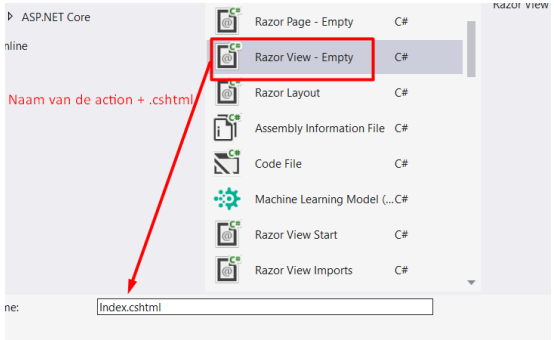


Weergeven van de gamestores

- Aanmaken van een view kan eenvoudig via de controller action





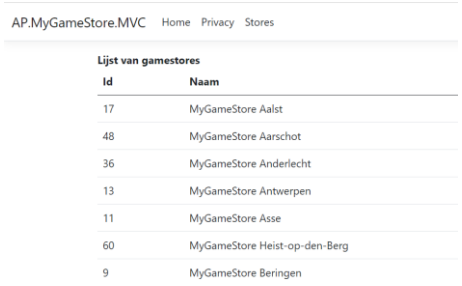


Weergeven van de gamestores

- Het model dat wordt doorgegeven aan de view kunnen we een type toekennen
- Vervolgens kunnen we de gegevens uit het model via c# code verwerken in de HTML van de razor view
- C# statements laten we vooraf gaan door een @

Index.cshtml

```
1  @using AP.MyGameStore.DAL.Model
2  @model List<Store>                          <- Aangeven wat er in het model zit
3
4  <strong>Lijst van gamestores</strong>
5
6  @if (Model.Count() > 0)                     <- controle of er stores aanwezig zijn in de lijst
7  {
8      <table class="table">
9          <thead>
10             <tr>
11                 <th>Id</th>
12                 <th>Naam</th>
13             </tr>
14         </thead>
15         <tbody>
16         @foreach (var s in Model)           <- Aan de hand van de lijst genereren we rijen in de tabel
17         {
18             <tr>
19                 <td>@s.Id</td>              <- We geven van elke store de Id en de naam weer
20                 <td>@s.Name</td>
21             </tr>
22         }
23         </tbody>
24     </table>
25 }
26 else
27 {
28     <div>Er zijn nog geen stores aanwezig...</div>
29 }
30
```



Toevoegen van GameStores

- Om een store te kunnen toevoegen:
  1. Een HTTP request vanuit de browser naar onze MVC applicatie (route: ??)
  2. Deze zal (via een view) een HTML **form** terugsturen waarin de gegevens kunnen worden ingevuld
  3. Nadat de gegevens werden ingevoerd zal nu een POST gebeuren naar de MVC applicatie (route: ??)
  4. De MVC controller zal de ingevulde gegevens doorsturen naar de API adhv. een POST request (route: /api/v1/stores)
  5. Nadien kunnen we terug navigeren naar de lijst van stores

Toevoegen van een gamestore

```
1 reference
public class StoresController : Controller
{
    0 references
    public StoresController()...

    private static HttpClient sharedClient = new()...;
    0 references
    public async Task<IActionResult> Index()...

    0 references
    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    0 references
    public async Task<IActionResult> Create(Store store)
    {
        await sharedClient.PostAsJsonAsync<Store>("/api/v1/stores", store);
        return RedirectToAction("Index");
    }
}
```

default verb = GET

view naam = Stores/Create.cshtml

verb = POST

wegschrijven via de API

redirect naar het overzicht van stores (lijst)

Create.cshtml

```
1  @using AP.MyGameStore.DAL.Model
2  @model Store

3

4  <form method="post">
5      <input asp-for="Name" />
6      <input asp-for="Zipcode"/>
7      <input type="submit" value="Save" />
8      <a class="btn" asp-action="Index">Cancel</a>
9  </form>
```

model is noodzakelijk om de form te kunnen koppele aan een store object

invulformulier voor een store

invulvelden voor elke property

submit knop

annuleer knop