Visitor Registration App

Realisatie

Bachelor in de Toegepaste Informatica
Afstudeerrichting: Application Development

Matthias Reynders

Academiejaar 2020-2021

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

INHOUDSTAFEL

1 INLEIDING

De Visitor Registration App is mijn eerste opdracht tijdens mijn stage bij AllPhi. Dit project omvat het bouwen van een webapplicatie voor de receptie van het bedrijfsgebouw waar AllPhi is gevestigd. De applicatie maakt het mogelijk voor bezoekers om zichzelf te registreren, en biedt de receptionist de mogelijkheid om hier een overzicht van op te vragen.

In dit document worden de stappen beschreven in de ontwikkeling van de applicatie. Eerst een analyse van de opdracht, hieruit de benodigde functionaliteiten en het domeinmodel afleiden. Vervolgens werk ik mijn 3-lagen structuur van onder naar boven af, eerst de data access layer, daarna de business en presentation layer.

2 ANALYSE

In dit hoofdstuk beschrijf ik mijn analyse van de opdracht. Vooraleer ik begin met programmeren zijn er technische vereisten waar ik rekening mee moet houden, functionaliteiten waar de applicatie aan moet voldoen en een domeinmodel dat representeerd welke entiteiten nodig zijn hiervoor.

2.1 Technische vereisten

De technische vereisten van de opdracht bepalen in welke programmeertaal de oplossing dient geschreven te worden, m.b.v. welk framework en welke design patterns geïmplementeerd moeten worden. Omdat AllPhi gespecialiseerd is in software ontwikkeling met een duidelijke focus op Microsoft .NET, zijn de vereisten voor deze opdracht ook .NET-gericht.

De technische vereisten zijn:

- MsSql server based
- N-Tier setup
- Using .net Core
- Using Entity framework Core
- Implement a repository pattern
- Front-end MVC Core
- Make use of DTO’s
- Create mappers where necessary
- Implement and configure Dependency Injection

2.2 Functionaliteiten

De functionaliteiten kunnen we op basis van rol onderverdelen in twee groepen. Enerzijds wat een bezoeker moet kunnen en anderzijds de receptionist die de nodige administratie moet kunnen uitvoeren.

Functionaliteiten bezoeker:

- Zich aanmelden aan de front desk bij het binnen gaan.
- Uit een lijst het juiste bedrijf en medewerker kiezen.
- Zich afmelden bij het verlaten van het gebouw.

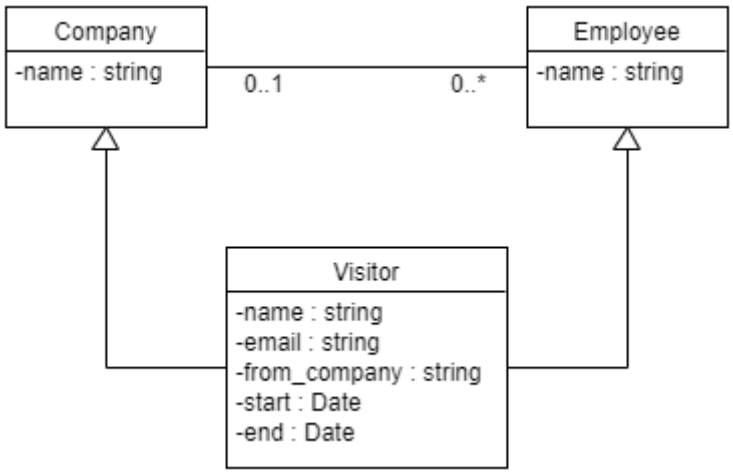
Functionaliteiten receptionist:

- De bedrijven en medewerkers binnen het gebouw kunnen aanpassen.
- Zien welke bezoekers er momenteel in het gebouw zijn.
- In de lijst van bezoekers zoeken op basis van verschillende criteria.

2.3 Domeinmodel

Voor de realisatie van deze applicatie zijn er 3 entiteiten nodig: Visitor, Company en Employee.

Het domeinmodel voor de Visitor Registration App ziet er als volgt uit.



Figuur 1: domeinmodel

Visitor heeft een unidirectionele relatie met Company en Employee omdat een bezoeker bij het binnen komen moet kunnen aangeven met welke werknemer hij/zij een afspraak heeft.

3 Data Access Layer (DAL)

In dit hoofdstuk beschrijf ik hoe ik de data access layer op zet. Deze laag zal een MsSql-database aanspreken en gebruik maken van Entity Framework Core als object-relational mapper. Tenslotte werk ik met een generic repository zodat ik niet voor elke repository in de data access layer de standaard CRUD-operaties hoef te schrijven.

3.1 Database

De technische vereisten van deze opdracht leggen voor dat voor de realisatie een MsSql-database wordt gebruikt.

Om een MsSql-server lokaal te draaien maak ik gebruik van SQL Server 2019 Developer Edition en SQL Server Management Studio.

SQL Server 2019 Developer is een volledig functionele gratis editie, die wordt gelicentieerd voor gebruik als ontwikkel- en testdatabase in een niet-productieomgeving.

SQL Server Management Studio (SSMS) is een geïntegreerde omgeving voor het beheren van elke SQL-infrastructuur, van SQL Server tot Azure SQL Database. SSMS biedt tools voor het configureren, bewaken en beheren van SQL Server-instances en databases.

3.2 Entity Framework Core

Entity Framework Core is een open-source object-relational mapping framework. Binnen het project maak ik gebruik van EF Core 5.0 om de tabellen te genereren (ook wel bekend als code-first). Entity Framework Core wordt ook gebruikt om de database aan te spreken en het resultaat te mappen naar mijn C# klassen.

In de DbContext definiëer ik de DbSets (tabellen) en de configurations (constraints en seed data). Hieronder ziet u de DbContext.

```
10 references
public class VisitorRegistrationDbContext : DbContext
{
    0 references
    public VisitorRegistrationDbContext(DbContextOptions<VisitorRegistrationDbContext> options) : base(options)
    {

    }

    2 references
    public DbSet<Company> Companies { get; set; }
    3 references
    public DbSet<Employee> Employees { get; set; }
    2 references
    public DbSet<Visitor> Visitors { get; set; }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfiguration(new CompanyConfiguration());
        modelBuilder.ApplyConfiguration(new EmployeeConfiguration());
        modelBuilder.ApplyConfiguration(new VisitorConfiguration());
    }

}
```

Figuur 2: VisitorRegistrationDbContext

Hieronder ziet u een voorbeeld van een configuration. Een aparte configuration-klasse laat toe om de constraints hier te definiëeren en seed data toe te voegen.

```
1 reference
public class CompanyConfiguration : IEntityTypeConfiguration<Company>
{
    0 references
    public void Configure(EntityTypeBuilder<Company> builder)
    {
        builder.ToTable("Companies");
        builder.HasKey(x => x.Id);
        builder.HasMany(c => c.Employees).WithOne(e => e.Company);
        builder.Property(c => c.Name).HasMaxLength(30);
        builder.HasData
            (
            new Company { Id = 1, Name = "Allphi" },
            new Company { Id = 2, Name = "Codefined" },
            new Company { Id = 3, Name = "Corvus" },
            new Company { Id = 4, Name = "CLT-S"},
            new Company { Id = 5, Name = "GMI Group" },
            new Company { Id = 6, Name = "LDS Advisory" },
            new Company { Id = 7, Name = "Produsoft" }
            );
    }
}
```

Figuur 3: CompanyConfiguration

3.3 Generic Repository

De generic repository is ontikkeld om de standaard CRUD-operaties die gelden voor alle entiteiten één keer te schrijven en deze dan te implementeren. Mijn generic repository is een abstract class die de interface IAsyncRepository implementeert.

```
7 references
public abstract class BaseRepository<T> : IAsyncRepository<T> where T : class
{
    protected readonly VisitorRegistrationDbContext _dbContext;

    3 references
    public BaseRepository(VisitorRegistrationDbContext dbContext)
    {
        _dbContext = dbContext;
    }
    4 references
    public async Task<T> AddAsync(T entity)
    {
        await _dbContext.Set<T>().AddAsync(entity);
        await _dbContext.SaveChangesAsync();
        return entity;
    }

    4 references
    public async Task DeleteAsync(T entity)
    {
        _dbContext.Set<T>().Remove(entity);
        await _dbContext.SaveChangesAsync();
    }

    6 references
    public async Task<T> GetByIdAsync(int id)
    {
        return await _dbContext.Set<T>().FindAsync(id);
    }

    2 references
    public async Task<IEnumerable<T>> ListAllAsync()
    {
        return await _dbContext.Set<T>().ToListAsync();
    }

    5 references
    public async Task UpdateAsync(T entity)
    {
        _dbContext.Entry(entity).State = EntityState.Modified;
        await _dbContext.SaveChangesAsync();
    }
}
```

Figuur 4: BaseRepository

```
4 references
public interface IAsyncRepository<T> where T : class
{
    6 references
    Task<T> GetByIdAsync(int id);
    2 references
    Task<IEnumerable<T>> ListAllAsync();
    4 references
    Task<T> AddAsync(T entity);
    5 references
    Task UpdateAsync(T entity);
    4 references
    Task DeleteAsync(T entity);
}
```

Figuur 5: IAsyncRepository

4 BUSINESS LOGIC LAYER (BLL)

In dit hoofdstuk beschrijf ik hoe de business logic layer dient als intermediair voor gegevensuitwisseling tussen de presentation layer en de data access layer. De business logic layer behandelt de bedrijfsregels, berekeningen en logica binnen een applicatie die dicteren hoe deze zich gedraagt. Dit wil zeggen dat de BLL bepaalt hoe gegevens uit de database worden gebruikt en wat ze wel en niet kunnen doen binnen de applicatie zelf. Mijn business logic layer maakt gebruik van mappings, services, specifications en validators voor de uitwisseling van gegevens.

4.1 Mappings

Mapping kan op vele plaatsen voorkomen binnen een project, maar meestal tussen twee lagen bijvoorbeeld van UI (informatie van buiten af) naar Domain/BL. De data die ik opsla in de databank heeft een aantal property's meer als de data die binnenkomt. Daarom maak ik bij deze opdracht gebruik van de package AutoMapper. In de afbeelding hieronder vertel ik AutoMapper hoe het de DTO's (data transfer object) moet mappen naar mijn entiteiten.

```
2 references
public class MappingProfile : Profile
{
    0 references
    public MappingProfile()
    {
        CreateMap<NewCompanyDTO, Company>().ForMember(c => c.Id, act => act.Ignore())
            .ForMember(c => c.Employees, act => act.Ignore());
        CreateMap<Company, NewCompanyDTO>();

        CreateMap<Employee, NewEmployeeDTO>();
        CreateMap<NewEmployeeDTO, Employee>().ForMember(e => e.Id, act => act.Ignore())
            .ForMember(e => e.Company, act => act.Ignore());

        CreateMap<NewVisitorDTO, Visitor>()
            .ForMember(v => v.Id, act => act.Ignore())
            .ForMember(v => v.VisitingCompany, act => act.Ignore())
            .ForMember(v => v.AppointmentWith, act => act.Ignore())
            .ForMember(v => v.Start, act => act.Ignore())
            .ForMember(v => v.End, act => act.Ignore());

        CreateMap<Company, EditCompanyDTO>().ReverseMap();
        CreateMap<EditEmployeeDTO, Employee>().ReverseMap();
    }
}
```

Figuur 6: MappingProfile

4.2 Services

De services zijn een doorgeefluik. Informatie voor de UI zal worden opgevraagd in de controller (dit komt in het volgende hoofdstuk aan bod). Een service wordt opgeroepen en zal de nodige informatie ophalen m.b.v. de Data Access Layer. Wanneer iets wordt toegevoegd of aangepast, zal een service eerst valideren of de meegegeven velden niet ingaan tegen de business rules aan de hand van een validator en specification pattern.

4.3 Specifications

Om specifications te gebruiken dient eerst de NuGet package DomainValidation van Eduardo Pires geïnstalleerd te worden. Vervolgens heb ik twee specifications. De IsNotNull (figuur 7) controleert of een veplicht veld niet leeg is en IsExpressionValid (figuur 8) controleert of een veld voldoet aan een bepaalde expressie (zoals de lengte van een string mag niet langer zijn als 30 tekens).

```
7 references
public class IsNotNull<T> : ISpecification<T>
{
    private readonly Expression<Func<T, object>> _expression;

    6 references
    public IsNotNull(Expression<Func<T, object>> expression)
    {
        _expression = expression;
    }

    0 references
    public bool IsSatisfiedBy(T entity)
    {
        var value = _expression.Compile()(entity);
        var type = typeof(T);

        if (type.Equals(typeof(string)))
            return !string.IsNullOrEmpty(value as string);
        else if (type.Equals(typeof(int)))
            return ((int)value) > 0;
        else if (type.Equals(typeof(DateTime)))
            return !((DateTime)value).Equals(DateTime.MinValue);

        return value != null;
    }
}
```

Figuur 7: IsNotNull

```
8 references
public class IsExpressionValid<T> : ISpecification<T>
{
    private readonly Expression<Func<T, bool>> _expression;

    7 references
    public IsExpressionValid(Expression<Func<T, bool>> expression)
    {
        _expression = expression;
    }

    0 references
    public bool IsSatisfiedBy(T entity) => _expression.Compile()(entity);
}
```

Figuur 8: IsExpressionValid

4.4 Validators

Nieuwe informatie die binnenkomt om te persisteren wordt door een service eerst gevalideerd op basis van een specifieke validator. Pas wanneer de validator aangeeft dat de velden correct, wordt de informatie verder doorgegeven naar de Data Access Layer.

Een voorbeeld hiervan is wanneer een bezoeker zich registreerd (figuur 9). De service moet eerst controleren of de bezoeker nog niet bestaat in het systeem. Vervolgens gaat de IsNewVisitorDTOValid validator (figuur 10) controleren of alle velden van de DTO correct zijn ingevuld. Als alles correct is wordt de VisitorDTO gemapt naar de Visitor entiteit die mijn Data Access Layer nodig heeft.

```
2 references
public async Task<Visitor> Create(NewVisitorDTO newVisitorDTO)
{
    var signedIn = _visitorRepository.Find().Where(v => v.Email == newVisitorDTO.Email && v.End == null).Any();
    if (signedIn) throw new Exception("signedIn");
    var validation = new IsNewVisitorDTOValid().Validate(newVisitorDTO);
    if (!validation.IsValid)
    {
        var msg = validation.Erros;
        throw new Exception(validation.Erros.First().Message);
    }
    var visitor = _mapper.Map<Visitor>(newVisitorDTO);
    visitor.VisitingCompany = await _companyRepository.GetCompanyIncludeEmployeesAsync(newVisitorDTO.SelectedCompanyId);
    visitor.AppointmentWith = visitor.VisitingCompany.Employees.Where(e => e.Id == newVisitorDTO.SelectedEmployeeId).SingleOrDefault();
    return await _visitorRepository.AddAsync(visitor);
}
```

Figuur 9: Create methode van VisitorService

```
2 references
public class IsNewVisitorDTOValid : Validator<NewVisitorDTO>
{

    private static string pattern = @"^(?!\.)(""([^""\r\\]|\\[""\r\\])*""|"
+ @"([-a-z0-9!#$%&'*+/=?^_`{|}~]|(?<!\.)\.)*)(?<!\.)"
+ @"@[a-z0-9][\w\.-]*[a-z0-9]\.[a-z][a-z\.]*[a-z]$";
    private Regex regex = new Regex(pattern, RegexOptions.IgnoreCase);

    1 reference
    public IsNewVisitorDTOValid()
    {
        // name
        Add("NameLength",
            new Rule<NewVisitorDTO>(new IsExpressionValid<NewVisitorDTO>(v => v.Name.Length >= 3 && v.Name.Length <= 50),
            "Name must be between 3 and 50 long"));

        // email
        Add("ValidEmail",
            new Rule<NewVisitorDTO>(new IsExpressionValid<NewVisitorDTO>(v => regex.IsMatch(v.Email)),
            "Email is not valid"));

        // from c
        Add("FromCompanyLength",
            new Rule<NewVisitorDTO>(new IsExpressionValid<NewVisitorDTO>(v => v.FromCompany.Length >= 3 && v.FromCompany.Length <= 30),
            "Name must be between 3 and 30 long"));

        // companyId
        Add("CompanyIdIsNotNullOrNegative",
            new Rule<NewVisitorDTO>(new IsNotNull<NewVisitorDTO>(v => v.SelectedCompanyId),
            "SelectedCompanyId is incorrect"));

        // employeeId
        Add("EmployeeIdIsNotNullOrNegative",
            new Rule<NewVisitorDTO>(new IsNotNull<NewVisitorDTO>(v => v.SelectedEmployeeId),
            "SelectedEmployeeId is incorrect"));

    }
}
```

Figuur 10: IsNewVisitorDTOValid

5 PRESENTATION LAYER

De presentatie laag is een standaard MVC-implementatie en maakt gebruik van Asp .net core 3.1. Binnen deze laag worden alle dependency's geregistreerd in de Startup klasse. Verder zijn er de controllers die de endpoints bepalen en requests afhandelen, views die het uiterlijk van de pagina beschrijven en viewmodels die nodig zijn voor informatie op bepaalde pagina's zoals dropdownlijsten.

5.1 Dependency Injection

Vanuit data access layer zijn de DbContext en de Repositories dependency's die moeten worden geinjecteerd. Vervolgens zijn er nog de Services van de business logic layer en tot slot AutoMapper voor de presentatie laag.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<VisitorRegistrationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"),
        options => options.MigrationsAssembly("VisitorRegistrationApplication.DAL")));
    services.AddMvc();

    services.AddScoped<ICompanyRepository, CompanyRepository>();
    services.AddScoped<IEmployeeRepository, EmployeeRepository>();
    services.AddScoped<IVisitorRepository, VisitorRepository>();

    services.AddScoped<ICompanyService, CompanyService>();
    services.AddScoped<IEmployeeService, EmployeeService>();
    services.AddScoped<IVisitorService, VisitorService>();

    services.AddControllersWithViews()
        .AddRazorRuntimeCompilation();
    services.AddAutoMapper(Assembly.Load("VisitorRegistrationApplication.BL"));
}
```

Figuur 11: ConfigureServices methode van Startup

5.2 Controllers

Voor Company, Employee en Visitor is er telkens een controller die de standaard CRUD operaties kan doorgeven. Bij Employee- en VisitorController is de index methode uitgebreider (figuur 12) omdat de eindgebruiker kan sorteren, filteren en de lijst wordt verdeeld in pagina's dankzij de helperklasse PaginatedList (figuur 13).

```
5 references
public async Task<ActionResult> Index(
    string sortOrder,
    string searchString,
    string currentFilter,
    bool showPast,
    int? pageNumber)
{
    ViewData["ShowPast"] = showPast;
    ViewData["ShowPastParm"] = showPast == false ? true : false;
    ViewData["CurrentSort"] = sortOrder;
    ViewData["NameSortParm"] = sortOrder == "Name" ? "name_desc" : "Name";
    ViewData["EmailSortParm"] = sortOrder == "Email" ? "email_desc" : "Email";
    ViewData["FromCompanySortParm"] = sortOrder == "FromCompany" ? "from_desc" : "FromCompany";
    ViewData["VisitingCompanySortParm"] = sortOrder == "VisitingCompany" ? "visiting_desc" : "VisitingCompany";
    ViewData["EmployeeSortParm"] = sortOrder == "Employee" ? "employee_desc" : "Employee";
    ViewData["StartSortParm"] = sortOrder == "Start" ? "start_desc" : "Start";

    if (searchString != null)
    {
        pageNumber = 1;
    }
    else
    {
        searchString = currentFilter;
    }

    ViewData["CurrentFilter"] = searchString;

    var visitors = _visitorService.Find();

    if (!String.IsNullOrEmpty(searchString))
    {
        visitors = visitors
            .Where(v => v.Name.Contains(searchString)
            || v.Email.Contains(searchString)
            || v.VisitingCompany.Name.Contains(searchString)
            || v.AppointmentWith.Name.Contains(searchString));
    }

    if (!showPast)
    {
        visitors = visitors.Where(v => v.End == null);
    }

    switch (sortOrder)
    {
        case "Name":
            visitors = visitors.OrderBy(v => v.Name);
            break;
        case "name_desc":
            visitors = visitors.OrderByDescending(v => v.Name);
            break;
        case "Email":
            visitors = visitors.OrderBy(v => v.Email);
            break;
        case "email_desc":
            visitors = visitors.OrderByDescending(v => v.Email);
            break;
        case "FromCompany":
            visitors = visitors.OrderBy(v => v.FromCompany);
            break;
        case "from_desc":
            visitors = visitors.OrderByDescending(v => v.FromCompany);
            break;
        case "VisitingCompany":
            visitors = visitors.OrderBy(v => v.VisitingCompany.Name);
            break;
        case "visiting_desc":
            visitors = visitors.OrderByDescending(v => v.VisitingCompany.Name);
            break;
        case "Employee":
            visitors = visitors.OrderBy(v => v.AppointmentWith.Name);
            break;
        case "employee_desc":
            visitors = visitors.OrderByDescending(v => v.AppointmentWith.Name);
            break;
        case "Start":
            visitors = visitors.OrderBy(v => v.Start);
            break;
        case "start_desc":
            visitors = visitors.OrderByDescending(v => v.Start);
            break;
        default:
            visitors = visitors.OrderByDescending(v => v.Start);
            break;
    }

    int pageSize = 10;
    var paginatedList = await PaginatedList<Visitor>.CreateAsync(visitors, pageNumber ?? 1, pageSize);
    return View(paginatedList);
}
```

Figuur 12: Index methode van VisitorController

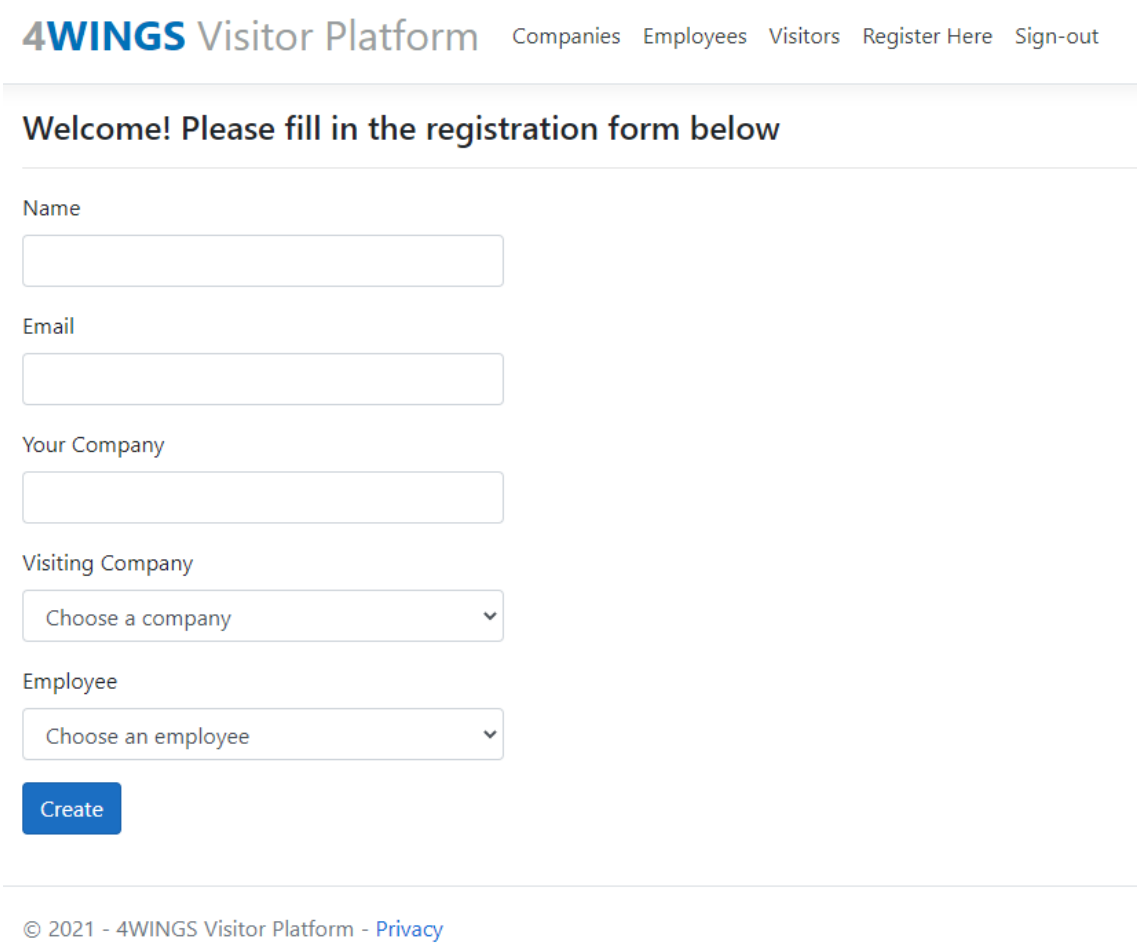
```
8 references
public class PaginatedList<T> : List<T>
{
    5 references
    public int PageIndex { get; set; }
    2 references
    public int TotalPages { get; set; }

    1 reference
    public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)
    {
        PageIndex = pageIndex;
        TotalPages = (int)Math.Ceiling(count / (double)pageSize);
        this.AddRange(items);
    }

    1 reference
    public bool HasPreviousPage
    {
        get
        {
            return (PageIndex > 1);
        }
    }

    1 reference
    public bool HasNextPage
    {
        get
        {
            return (PageIndex < TotalPages);
        }
    }

    2 references
    public static async Task<PaginatedList<T>> CreateAsync(IQueryable<T> source, int pageIndex, int pageSize)
    {
        var count = await source.CountAsync();
        var items = await source.Skip((pageIndex - 1) * pageSize).Take(pageSize).ToListAsync();
        return new PaginatedList<T>(items, count, pageIndex, pageSize);
    }
}
```

Figuur 13: PaginatedList

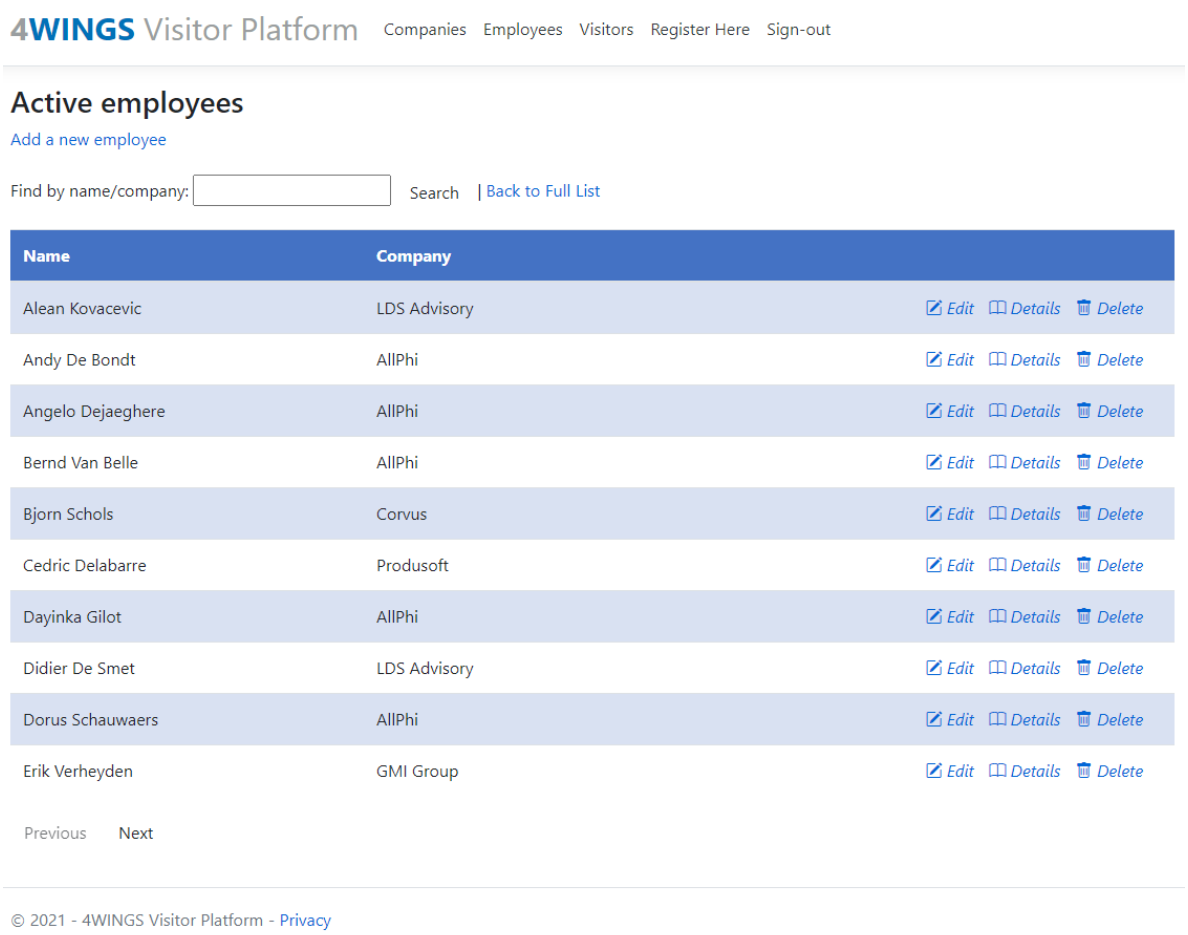
5.3 Views

De basis voor de views wordt dankzij scaffolding al gedaan door asp .net core (voor alles wat met Company en Employee te maken heeft). Hierdoor worden views voor alle CRUD operaties al gegenereerd. Vervolgens heb ik deze aangepast naar de vereisten van de eindgebruiker en specifieke views geschreven voor een bezoeker.

Hieronder ziet u de aanmeldpagina voor bezoekers als voorbeeld (figuur 14) en een overzichtspagina voor de receptionist van alle medewerkers (figuur 15).



Figuur 14: Views/Visitor/Create



Figuur 15: Views/Employees/Index

5.4 ViewModels

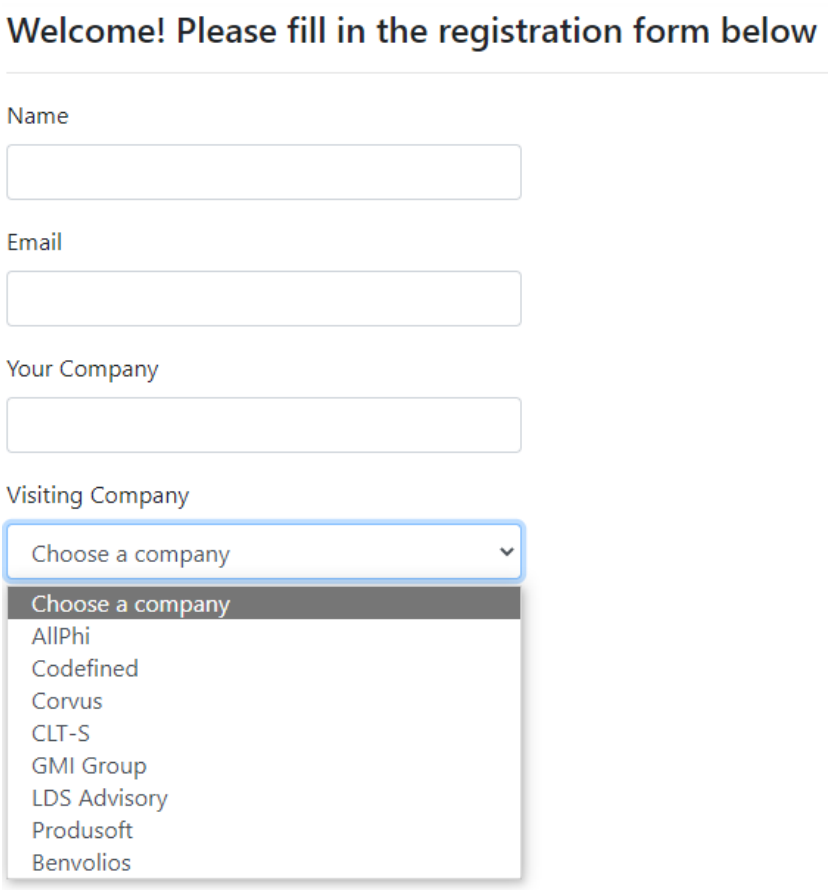
Een viewmodel representeerd data die nodig is uitsluitend voor het renderen van de pagina. Views voor een Employee aan te passen, toe te voegen en een nieuwe Visitor toe te voegen hebben alledrie een SelectList nodig zodat de eindgebruiker in een dropdown een gekend bedrijf/medewerker kan aanduiden.

Hieronder ziet u het viewmodel voor de aanmeldpagina. Hierbij is een dropdownlijst van zowel Companies als Employees nodig.

```
3 references
public class NewVisitorViewModel
{
    2 references
    public SelectList Companies { get; set; }
    0 references
    public SelectList Employees { get; set; }

    [StringLength(50, MinimumLength = 3, ErrorMessage = "Minimum length of 3 and maximum lenth of 50")]
    [Required]
    1 reference
    public string Name { get; set; }

    [StringLength(50, MinimumLength = 3, ErrorMessage = "Minimum length of 3 and maximum lenth of 50")]
    [Required]
    [EmailAddress(ErrorMessage = "Enter a valid email address")]
    1 reference
    public string Email { get; set; }

    [StringLength(30, MinimumLength = 3, ErrorMessage = "Minimum length of 3 and maximum lenth of 30")]
    [Required]
    1 reference
    public string FromCompany { get; set; }

    [Required(ErrorMessage = "Choose a company please")]
    1 reference
    public int SelectedCompanyId { get; set; }

    [Required(ErrorMessage = "Choose an employee please")]
    1 reference
    public int SelectedEmployeeId { get; set; }
}
```

Figuur 16: NewVisitorViewModel

Tot slot nog een voorbeeld van een viewmodel waarbij gegevens van zo'n dropdownlijst nodig zijn.



Figuur 17: Views/Visitor/Create