Avans Maaltijdreservering

Opleverdocument

Diagrammen, API Documentatie & Testing

Stijn Robben
Avans Hogeschool
Server-side webprogramming individuele opdracht

15 oktober 2025

Inhoudsopgave

1 Projectoverzicht

1.1 Context

Het Avans Maaltijdreservering systeem is een “Too Good To Go” alternatief specifiek ontwikkeld voor Avans studenten. Het doel is om voedselverspilling tegen te gaan door overschotten aan het eind van de dag tegen gereduceerde prijs aan te bieden.

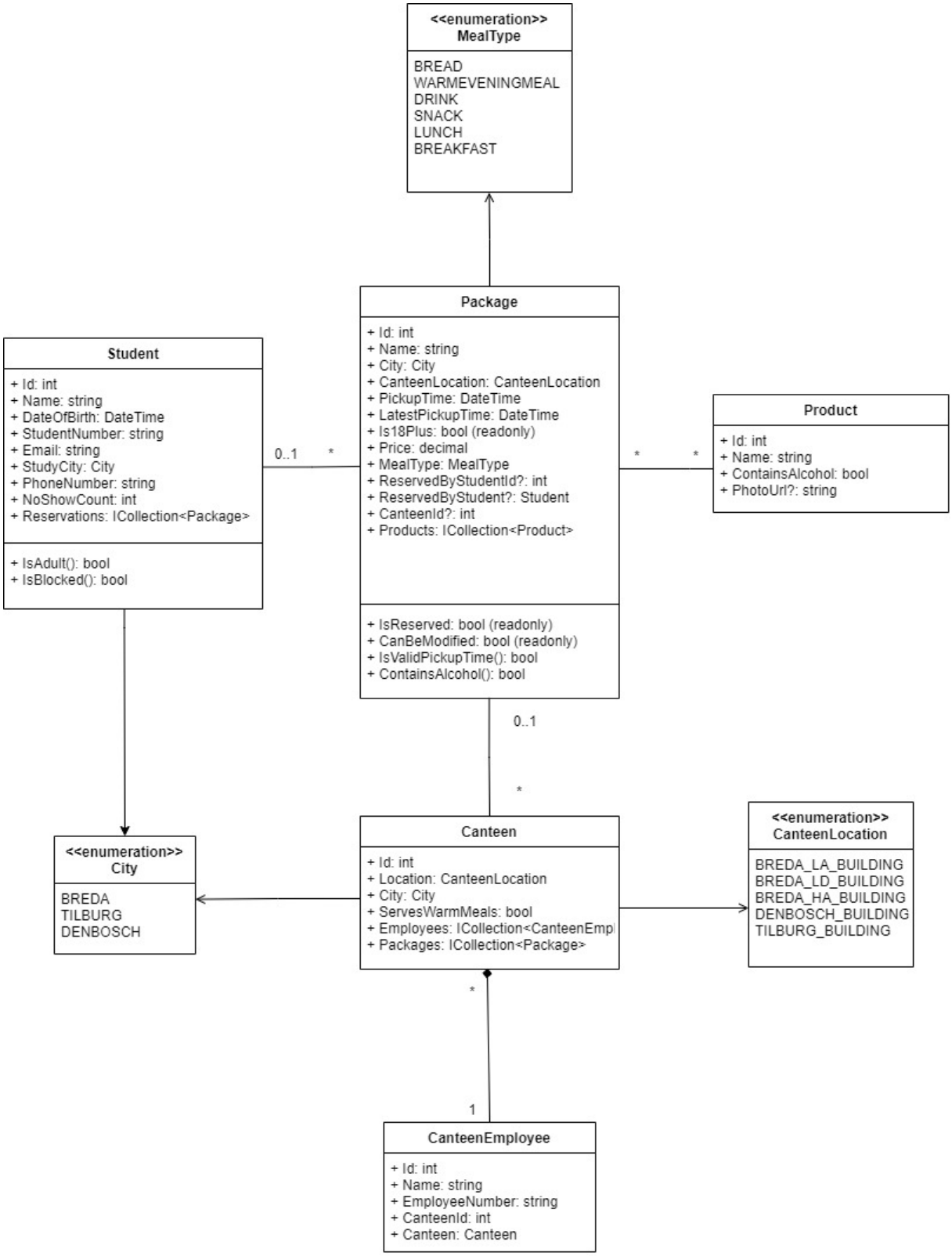
1.2 Technische keuzes

Het systeem moest gebouwd worden met:

- **ASP.NET Core 9.0** - Framework
- **Entity Framework Core** - ORM voor data access
- **Onion Architecture** - Layered architecture pattern
- **Microsoft Identity** - Authenticatie en autorisatie
- **REST API + GraphQL** - API endpoints
- **Azure SQL Database** - Twee databases (Business + Identity)

2 Diagrammen

2.1 Class Diagram



Figuur 1: Klasse diagram

2.1.1 Beschrijving

Het class diagram (figuur 1) toont het domeinmodel van het Avans Maaltijdreservering systeem. De kernentiteiten zijn Student, Package, Product, Canteen en CanteenEmployee, aangevuld met enumeraties voor City, MealType en CanteenLocation.

2.1.2 UML Relaties

1. Student — Package (0..1 naar *)

- **Type:** Bidirectionele associatie (geen pijl)
- **Multipliciteit:** Een Student kan 0 of 1 Package reserveren, een Package kan door 0 of 1 Student gereserveerd worden
- **Reden:** Package kan optioneel gereserveerd worden door Student. Beide klassen kennen elkaar (navigation properties), maar Package is niet “deel van” Student. Packages blijven bestaan als Student wordt verwijderd.

2. Package — Product (* naar *)

- **Type:** Many-to-many associatie (geen pijl)
- **Multipliciteit:** Een Package bevat meerdere Products, een Product kan in meerdere Packages zitten
- **Reden:** Producten worden hergebruikt in meerdere packages. Geen eigendomsrelatie - gewoon een koppeling tussen onafhankelijke entiteiten. Dit is de verplichte veel-op-veel relatie in het systeem.

3. Package — Canteen (* naar 1)

- **Type:** Bidirectionele associatie (geen pijl)
- **Multipliciteit:** Een Package hoort bij 1 Canteen, een Canteen heeft meerdere Packages
- **Reden:** Package wordt aangeboden via Canteen, maar is niet “deel van” de Canteen. Packages kunnen onafhankelijk bestaan (cascade delete is niet vereist).

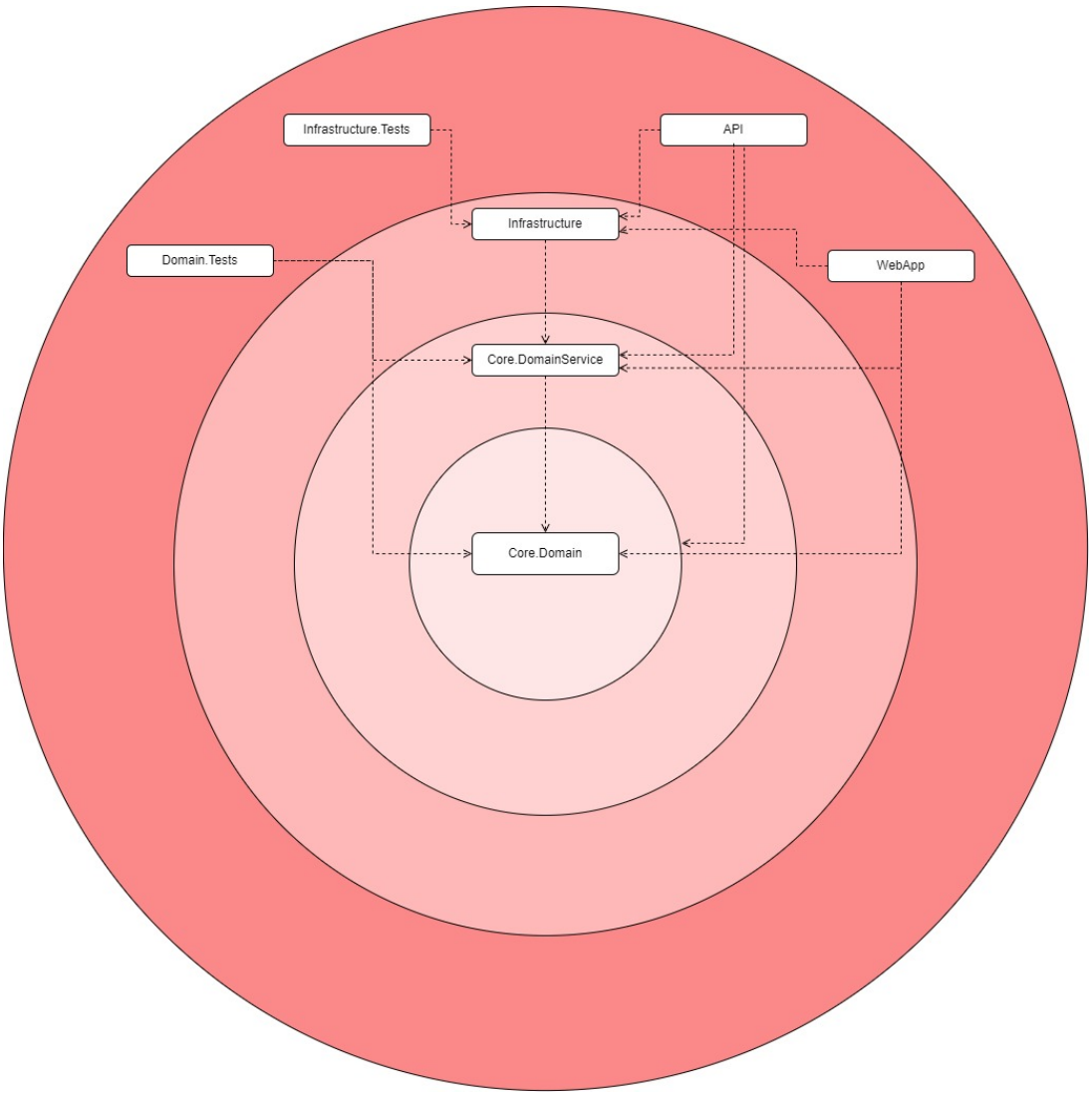
4. CanteenEmployee $\rightarrow$ Canteen (1 naar *)

- **Type:** Compositie (filled diamond)
- **Multipliciteit:** Een Employee hoort bij exact 1 Canteen, een Canteen heeft meerdere Employees
- **Reden:** Employee MOET bij een Canteen horen (CanteenId niet nullable). Als Canteen wordt verwijderd, heeft Employee geen bestaansrecht meer. Sterke eigendomsrelatie - Employee is “deel van” Canteen.

5. Dependencies naar Enums →

- **Type:** Dependency (open pijl)
- **Relaties:**
  - Package → MealType
  - Student → City (StudyCity property)
  - Canteen → City
  - Canteen → CanteenLocation
- **Reden:** Klasse gebruikt de enum als value type. Unidirectioneel - enum “kent” de klasse niet. Open pijl toont afhankelijkheidsrichting.

2.2 Component Diagram



Figuur 2: Component Diagram - Onion Architecture

2.2.1 Beschrijving

Het component diagram (figuur 2) visualiseert de Onion Architecture van het systeem. De concentrische cirkels tonen de verschillende architectuurlagen, waarbij dependencies altijd naar binnen wijzen (naar de Core.Domain laag).

2.2.2 Architectuur Lagen

Laag 1 - Core.Domain (Centrum)

- **Inhoud:** Entities, Enums, Repository Interfaces, Validation Attributes
- **Dependencies:** Geen - dit is de kern van het systeem
- **Doel:** Bevat de pure business logic zonder externe afhankelijkheden

Laag 2 - Core.DomainService

- **Inhoud:** Business logic services (StudentService, PackageService, ReservationService)
- **Dependencies:** → Core.Domain
- **Doel:** Orkestreert business logic en gebruikt repositories

Laag 3 - Infrastructure

- **Inhoud:** DbContexts, Repository implementaties, Identity, Migrations
- **Dependencies:** → Core.Domain, Core.DomainService
- **Doel:** Data access en externe infrastructuur concerns

Laag 4 - Presentation (API & WebApp)

- **Inhoud:** Controllers, Views (WebApp), REST endpoints, GraphQL (API)
- **Dependencies:** → Core.Domain, Core.DomainService, Infrastructure
- **Doel:** User interface en externe API's

Test Projecten

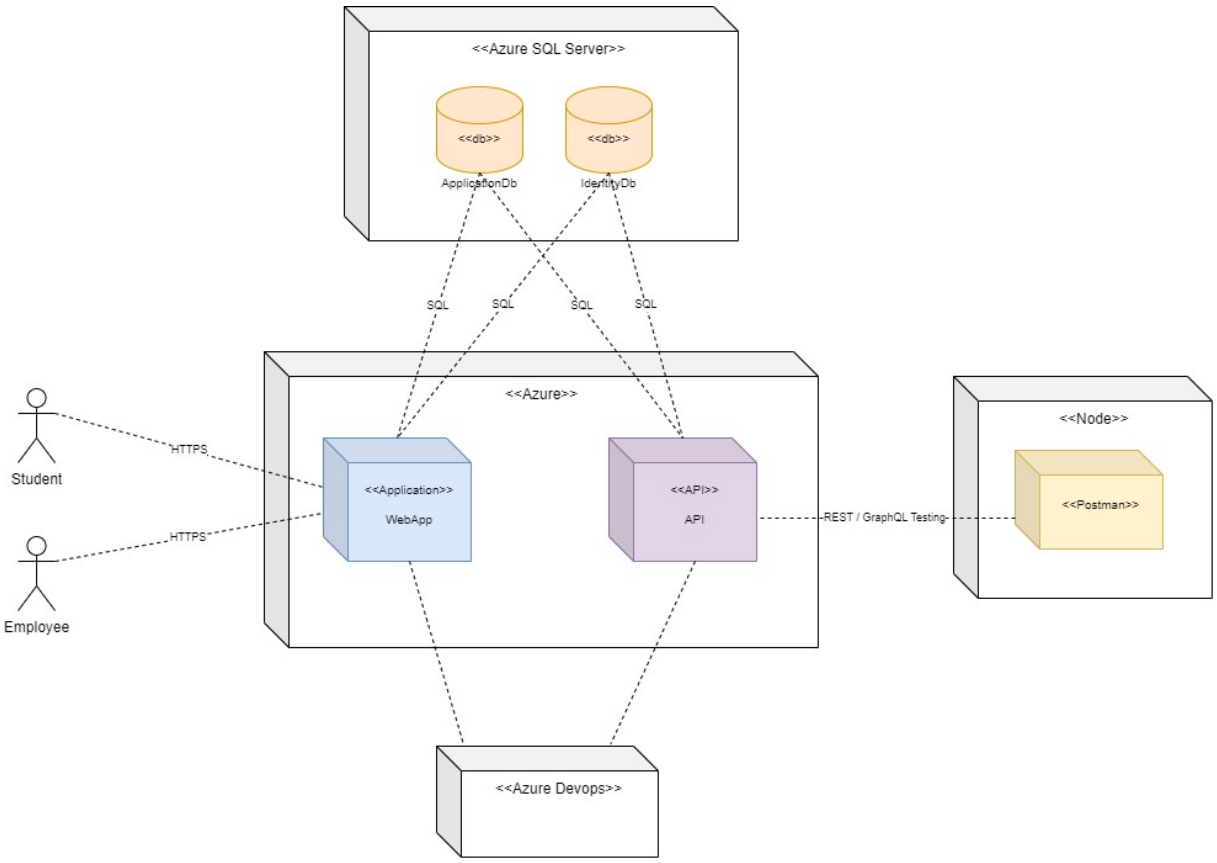
- **Core.Domain.Tests:** Unit tests voor domain logic en services
- **Infrastructure.Tests:** Integration tests voor repositories

2.2.3 Dependency Principe

Alle dependencies wijzen naar binnen, naar Core.Domain. Dit garandeert:

- Business logic is onafhankelijk van infrastructuur
- Domain kan zonder UI of database bestaan
- Flexibiliteit om implementaties te wisselen

2.3 Deployment Diagram



Figuur 3: Deployment Diagram - Azure Infrastructure

2.3.1 Beschrijving

Het deployment diagram (figuur 3) toont de fysieke infrastructuur op Microsoft Azure. Het visualiseert hoe de applicatiecomponenten gedeployed zijn en hoe ze met elkaar communiceren.

2.3.2 Azure Resources

Azure App Service

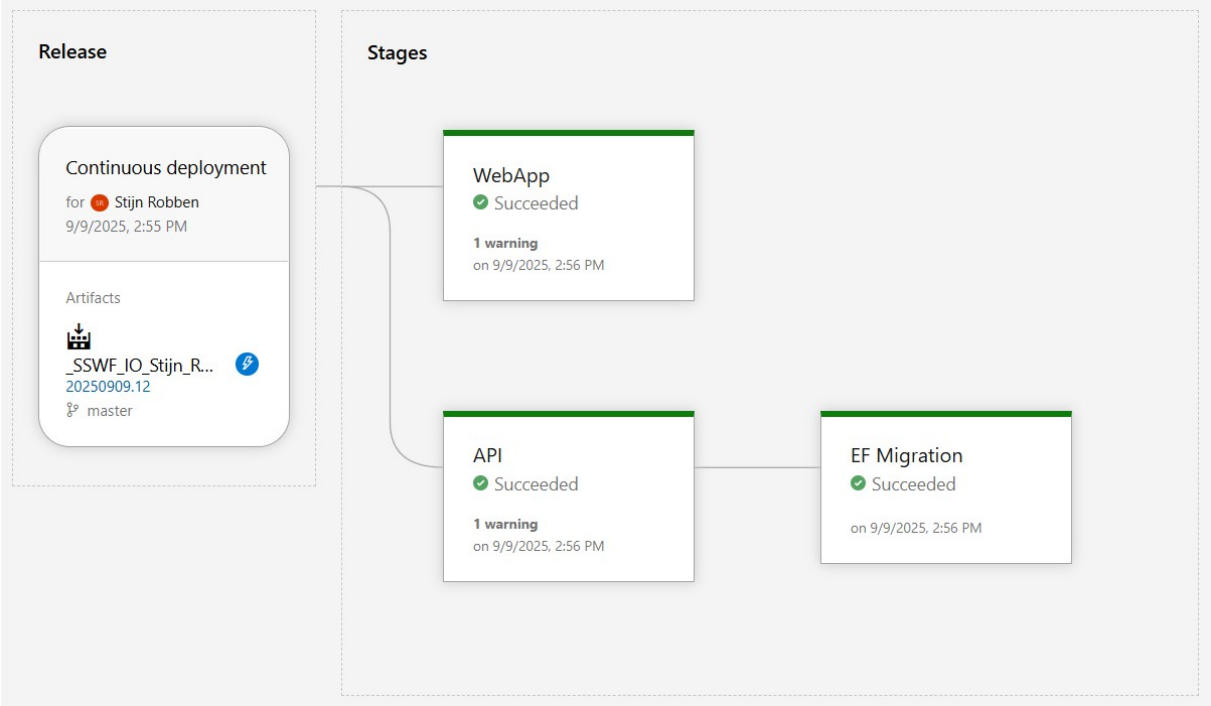
- **WebApp:** ASP.NET Core MVC applicatie
- **API:** REST API + GraphQL endpoints
- **Connecties:** HTTPS van gebruikers, SQL naar databases

Azure SQL Server

- **ApplicationDb:** Business data (Packages, Students, Products, Canteens)
- **IdentityDb:** Authentication data (Users, Roles, Claims)
- **Connecties:** SQL connecties naar beide applicaties

Azure DevOps (figuur 4)

- CI/CD pipeline voor automatische deployment
- Connecties naar WebApp, API en databases voor migrations
- Automatische builds bij code push naar master branch



Figuur 4: DevOps release pipeline

External Clients

- **Students & Employees:** HTTPS → WebApp
- **Postman:** REST API / GraphQL Testing → API
- **Mobile Apps:** Toekomstige AvansOne integratie via API

2.3.3 Database Architectuur

Beide applicaties (WebApp en API) hebben directe toegang tot beide databases:

- **ApplicationDbContext:** Voor business entities (Student, Package, Product, etc.)
- **ApplicationIdentityDbContext:** Voor ASP.NET Identity (ApplicationUser, Roles)

Dit is belangrijk omdat beide applicaties onafhankelijk functioneren:

- WebApp gebruikt GEEN API - directe database toegang
- API is voor externe clients (Postman, mobile apps)
- Beide delen dezelfde business logic layer (Services)

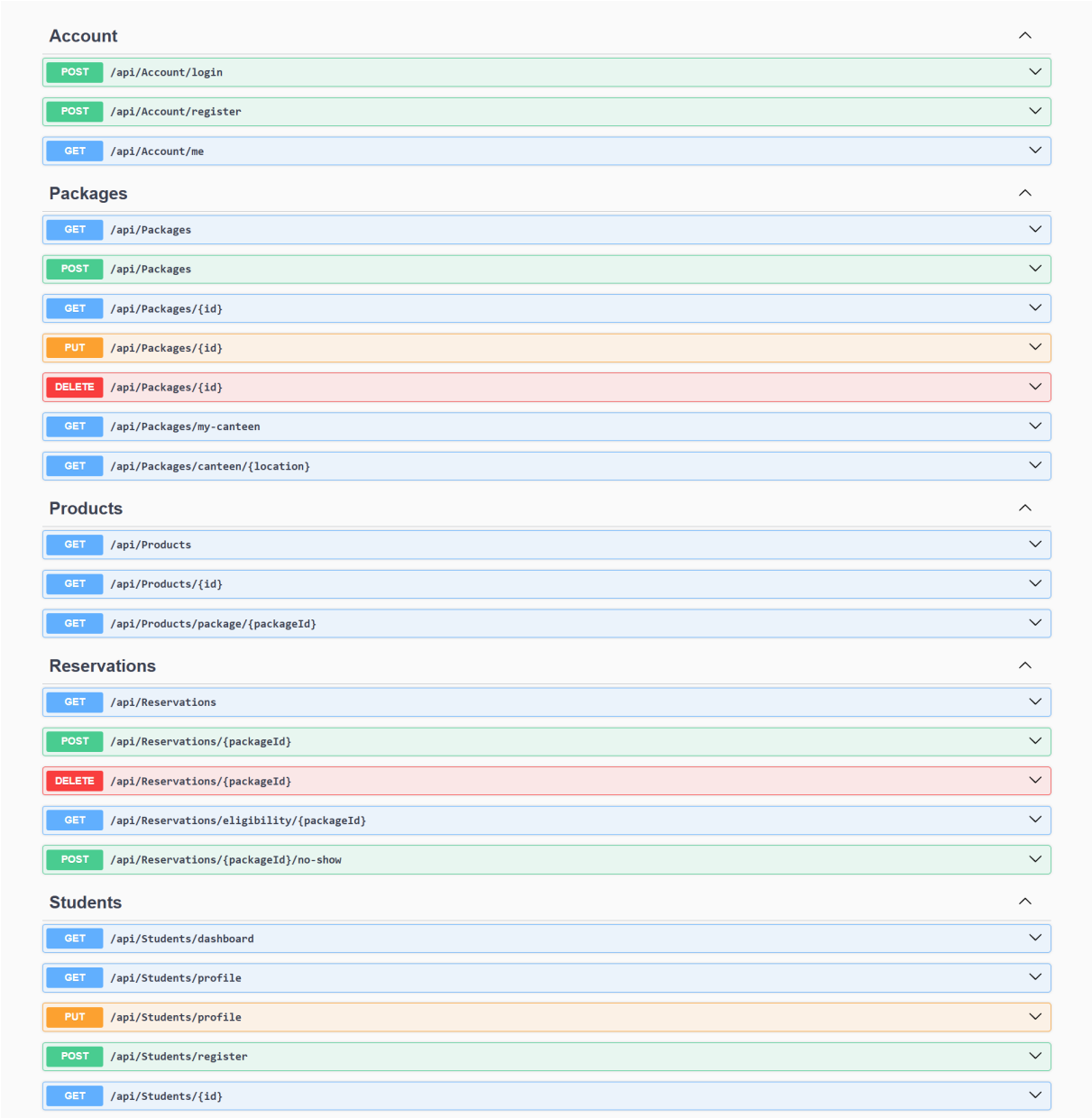
3 Swagger API Documentatie

3.1 Overzicht

De API is gedocumenteerd met Swagger/OpenAPI specificatie. Swagger is beschikbaar op zowel development als production omgevingen.

3.2 Endpoints

In figuur 5 zie je de verschillende endpoints. In swagger zelf heb ik voor elke endpoint een beschrijving/voorbeeld data object geschreven.



Figuur 5: Swagger, API endpoints

3.3 GraphQL Endpoint

```
POST /graphql
```

GraphQL is beschikbaar voor flexible querying, voornamelijk bedoeld voor mobile app integratie.

Voorbeeld Query:

```
query {
  packages {
    id
    name
    price
    pickupDateTime
    products {
      name
      containsAlcohol
    }
  }
}
```

4 Testing

4.1 Test Strategie

Het project implementeert een comprehensive testing strategie met unit tests en integration tests. De tests kun je runnen in de Test Explorer in Visual Studio (figuur 6).

4.2 Unit Tests

4.2.1 Test Framework

- **xUnit:** Test framework
- **Moq:** Mocking framework voor repositories
- **FluentAssertions:** Readable assertions

4.2.2 Core.Domain.Tests

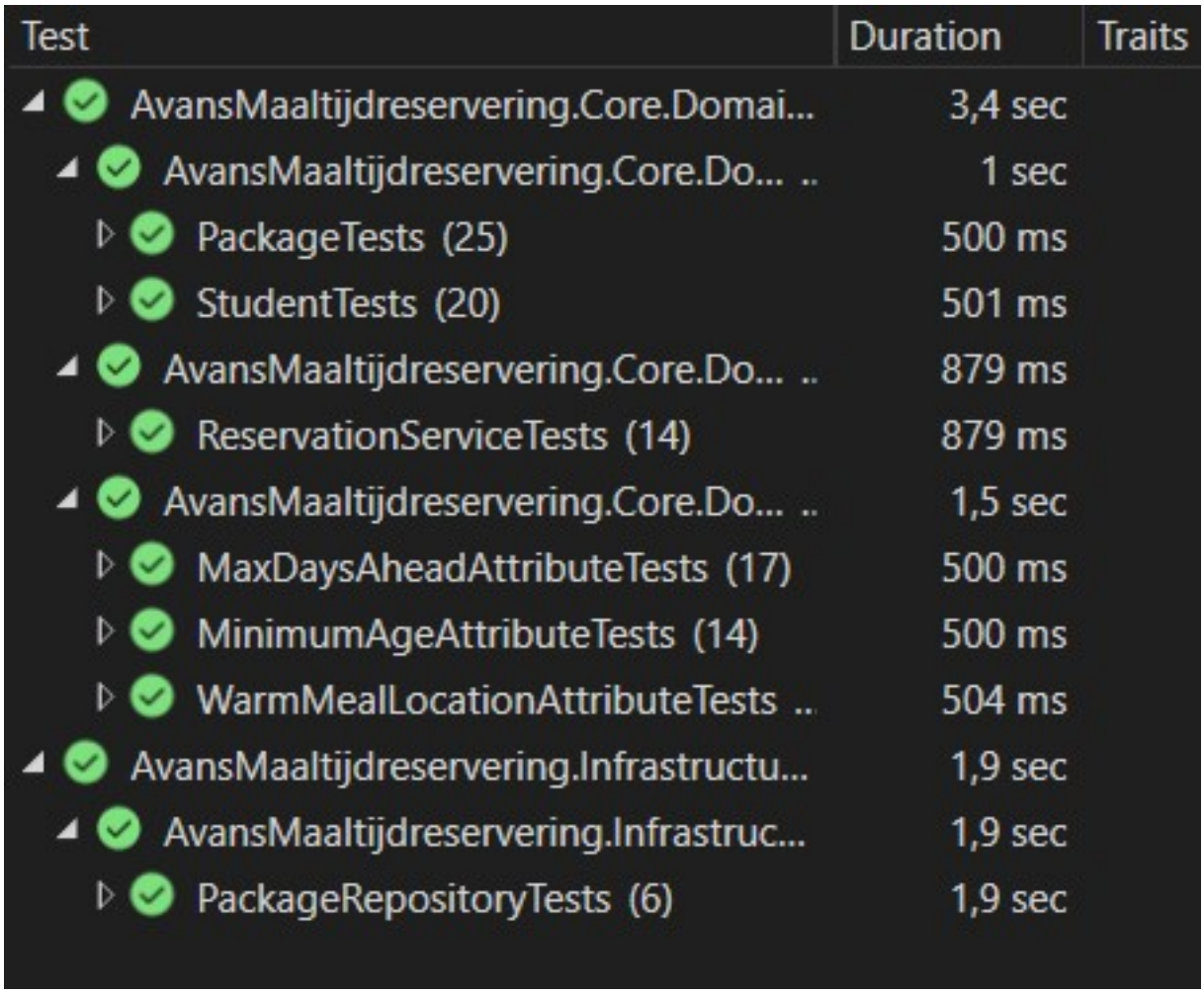
Test coverage voor:

- **Entity validatie:** Student, Package business rules
- **Service logic:** ReservationService, PackageService, StudentService
- **Validation attributes:** MinimumAge, MaxDaysAhead, WarmMealLocation
- **Test builders:** StudentTestDataBuilder, PackageTestDataBuilder voor clean test setup

Voorbeeld Test:

```
[Fact]
public async Task ReservePackage_StudentUnder18_
    ThrowsException_When18PlusPackage()
{
    // Arrange
    var student = new StudentTestDataBuilder()
        .WithAge(17)
        .Build();
    var package = new PackageTestDataBuilder()
        .With18PlusProduct()
        .Build();

    // Act & Assert
    await Assert.ThrowsAsync<BusinessRuleException>(
        () => _reservationService.ReservePackageAsync(
            student.Id, package.Id));
}
```



Figuur 6: Tests in Test Explorer

4.2.3 Infrastructure.Tests

Integration tests voor:

- **Repository implementaties:** PackageRepository, StudentRepository
- **Database operations:** CRUD operations
- **Entity Framework:** Relationships, cascade deletes, queries

4.3 End-to-End Tests

4.3.1 Postman Collections

- Complete tests voor alle API endpoints
- Automatisch uitvoerbaar zonder manual intervention
- JWT token management via collection variables
- Tests voor happy flow en error scenarios

Test Scenarios:

1. User registration en login
2. Package CRUD operaties
3. Reservation creation en validation
4. Business rule enforcement (18+, max 1 package per day)
5. Authorization tests (role-based access)

5 Deployment Informatie

5.1 Azure URLs

- **WebApp:** https://stijnrobben-sswpi-bneqbxejhfbmfdeb.germanywestcentral-01.azurewebsites.net/
- **API:** https://stijnrobben-sswpi-api-hxhvddefhsapgkgv.germanywestcentral-01.azurewebsites.net/api
- **Swagger:** https://stijnrobben-sswpi-api-hxhvddefhsapgkgv.germanywestcentral-01.azurewebsites.net/swagger
- **GraphQL:** https://stijnrobben-sswpi-api-hxhvddefhsapgkgv.germanywestcentral-01.azurewebsites.net/graphql

5.2 Test Accounts

Deze accounts werken wanneer je de volgende dingen uitvoert: ClearAllUsers.sql, daarna DatabaseSeed.sql, daarna endpoint-tests-azure.postman-collection.json. In de scripts zijn instructies gecomment.

5.2.1 Studenten

- Email: emma.janssen@student.avans.nl
- Password: Student123!

5.2.2 Kantinemedewerkers

- Email: maria.vandenbosch@avans.nl
- Password: Employee123!

5.3 Database Connections

Connection strings worden beheerd via:

- **Development:** appsettings.Development.json (local SQL Server)
- **Production:** Azure App Service Configuration (Environment Variables)

Belangrijk: Geen credentials in source control - alleen via Azure Configuration.