Programming advanced .NET

Dependency injection

Configure

Startup.cs

```
public void Configure(IApplicationBuilder app,
                      IWebHostEnvironment env,
                      IGreeter greeter)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }


            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapGet("/", async context =>
                {
                    var greeting = greeter.GetMessageOfTheDay();
                    await context.Response.WriteAsync(greeting);
                });
            });
```

Het Tonen van een message op de pagina “/”. ZONDER CONFIGURESERVICES WERKT DIT NIET.

ConfigureServices

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddSingleton<IGreeter, Greeter>();
}
```

AddSingleton ⇒ 1 instantie is nodig in heel uw applicatie.

AddTransient ⇒ Creëer een nieuwe instantie elke keer de interface nodig is.

AddScoped ⇒ Elk HTTP request creëert een nieuwe service.

Adding Parameter to Constructor

Greeter.cs

```
private readonly IConfiguration _configuration;

public Greeting(IConfiguration configuration)
{
  _configuration = configuration;
}
```

Met behulp van WebHostBuilder kan ASP.NET core dit zelf echterhalen wat hij moet doen.

Async / Await

1. 1 await expressie
2. method moet async modifier zijn
3. Return type is Task of Task<type die je wil int, string,…>
4. Method naam moet eindigen met Async

```
public async Task<ActionResult> Details(int id)
{
  using (var client = new HttpClient())
  {
    client.BaseAddress = new Uri("http://localhost:33902/");
    client.DefaultRequestHeaders.Accept.Clear();
    client.DefaultRequestHeaders.Accept.Add(New MediaTypeWithHeaderValue("application/json"));

    string url = "api/Schools/" + id;
    HttpResponseMessage response = await client.getAsync(url);
    if (response.IsSuccessStatusCode)
    {
      Course course = await response.Content.ReadAsAsync<Course>();
      return View(Course);
    }
  }
  return View();
}
```

Setting up MVC Middleware

Attribute Routes (OdeToFood)

Startup.cs

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

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<OdeToFoodContext>(options =>
                options.UseSqlServer(
                    Configuration.GetConnectionString("DefaultConnection")));

            services.AddDefaultIdentity<User>(options => options.SignIn.RequireConfirmedAccount = false)
                .AddEntityFrameworkStores<OdeToFoodContext>();

            services.AddControllersWithViews();
            services.AddRazorPages();

            services.Configure<IdentityOptions>(options =>
            {
                // Password settings.
                options.Password.RequireDigit = true;
                options.Password.RequireLowercase = true;
                options.Password.RequireNonAlphanumeric = true;
                options.Password.RequireUppercase = true;
                options.Password.RequiredLength = 6;
                options.Password.RequiredUniqueChars = 1;

                // Lockout settings.
                options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
                options.Lockout.MaxFailedAccessAttempts = 5;
                options.Lockout.AllowedForNewUsers = true;
                // User settings.
                options.User.RequireUniqueEmail = false;
            });

            services.AddAuthentication()
                .AddJwtBearer(options =>
                {
                    var tokenSettings = new TokenSettings();
                    Configuration.Bind("Token", tokenSettings);
                    options.TokenValidationParameters = new TokenValidationParameters
                    {
                        ValidIssuer = tokenSettings.Issuer,
                        ValidAudience = tokenSettings.Audience,
                        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(tokenSettings.Key))
                    };
                });

            services.AddScoped<IRestaurantRepository, RestaurantDbRepository>();
            services.AddScoped<IReviewRepository, ReviewDbRepository>();
            services.AddSingleton<IConverter, Converter>();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseDatabaseErrorPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-h
                app.UseHsts();
            }
            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseAuthentication();
            app.UseAuthorization();

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

HomeController.cs

```
public class HomeController : Controller
    {
        private readonly IRestaurantRepository _restaurantRepository;
        private readonly IReviewRepository _reviewRepository;
        private readonly IConverter _converter;

        public HomeController(IRestaurantRepository restaurantRepository, IReviewRepository reviewRepository, IConverter converter)
        {
            _restaurantRepository = restaurantRepository;
            _reviewRepository = reviewRepository;
            _converter = converter;
        }

        public IActionResult Index()
        {
            var restaurants = _restaurantRepository.GetAll();
            return View(restaurants);
        }

        [HttpGet("Home/Details/{id}")]
        public async Task<ActionResult> Details(int id)
        {
            RestaurantReviewsViewModel viewModel = new RestaurantReviewsViewModel
            {
                Restaurant = _restaurantRepository.GetById(id),
                Reviews = await _reviewRepository.GetReviewsByRestaurantAsync(id)
            };
            return View(viewModel);
        }

        // GET:
        [HttpGet("Home/AddReview/{restaurantId}")]
        public IActionResult AddReview(int restaurantId)
        {
            return View(new EditReviewViewModel { RestaurantId = restaurantId });
        }

        [HttpPost("Home/AddReview/{restaurantId}")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> AddReview(int restaurantId, EditReviewViewModel model)
        {
            if (model.RestaurantId != restaurantId)
            {
                ModelState.AddModelError("InvalidRestaurantId", "Restaurant id's in model and url don't match.");
            }

            if (ModelState.IsValid)
            {
                Review review = _converter.ConvertEditReviewViewModelToReview(model);

                await _reviewRepository.AddAsync(review);

                return RedirectToAction(nameof(Details), new { id = restaurantId });
            }

            return View(model);
        }

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
```

RestaurantsController.cs

```
[Route("api/[controller]")]
    [ApiController]
    //[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    public class RestaurantsController : ControllerBase
    {
        private readonly IRestaurantRepository _restaurantRepository;

        public RestaurantsController(IRestaurantRepository restaurantRepository)
        {
            _restaurantRepository = restaurantRepository;
        }

        // GET: api/Restaurants
        [HttpGet]
        public IActionResult GetAll()
        {
            return Ok(_restaurantRepository.GetAll());
        }

        // GET: api/Restaurants/5
        [HttpGet("{id}")]
        public IActionResult GetRestaurant(int id)
        {
            var restaurant = _restaurantRepository.GetById(id);
            if (restaurant == null) return NotFound();

            return Ok(restaurant);
        }

        // POST: api/Restaurants
        [HttpPost]
        public IActionResult Post([FromBody] Restaurant newRestaurant)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            var createdRestaurant = _restaurantRepository.Add(newRestaurant);

            return CreatedAtAction(nameof(GetRestaurant), new { id = createdRestaurant.Id }, createdRestaurant);
        }

        // PUT: api/Restaurants/5
        [HttpPut("{id}")]
        public IActionResult Put(int id, [FromBody] Restaurant restaurant)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            if (id != restaurant.Id)
            {
                return BadRequest();
            }

            if (_restaurantRepository.GetById(id) == null)
            {
                return NotFound();
            }

            _restaurantRepository.Update(restaurant);

            return Ok();
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public IActionResult Delete(int id)
        {
            if (_restaurantRepository.GetById(id) == null)
            {
                return NotFound();
            }

            _restaurantRepository.Delete(id);

            return Ok();
        }
    }
```

ReviewersController.cs

```
[Route("api/[controller]")]
    [ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    public class ReviewsController : ControllerBase
    {
        private readonly IReviewRepository _reviewRepository;

        public ReviewsController(IReviewRepository reviewRepository)
        {
            _reviewRepository = reviewRepository;
        }

        // GET: api/Reviews
        [HttpGet]
        public async Task<IActionResult> Get()
        {
            return Ok(await _reviewRepository.GetAllAsync());
        }
        // GET: api/Reviews/5
        [HttpGet("{id}")]
        public async Task<IActionResult> Get(int id)
        {
            var review = await _reviewRepository.GetByIdAsync(id);

            if (review == null) return NotFound();

            return Ok(review);
        }

        // POST: api/Reviews
        [HttpPost]
        public async Task<IActionResult> Post([FromBody] Review newReview)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            var createdReview = await _reviewRepository.AddAsync(newReview);

            return CreatedAtAction(nameof(Get), routeValues: new {id = createdReview.Id}, value: createdReview);
        }

        // PUT: api/Reviews/5
        [HttpPut("{id}")]
        public async Task<IActionResult> Put(int id, [FromBody] Review review)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest();
            }

            if (id != review.Id)
            {
                return BadRequest();
            }

            if (await _reviewRepository.GetByIdAsync(id) == null)
            {
                return NotFound();
            }

            await _reviewRepository.UpdateAsync(review);

            return Ok();
        }

        // DELETE: api/ApiWithActions/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> Delete(int id)
        {
            if (await _reviewRepository.GetByIdAsync(id) == null)
            {
                return NotFound();
            }

            await _reviewRepository.DeleteAsync(id);

            return Ok();
        }
    }
```

AuthenticationController.cs

```
[Route("api/[controller]")]
    [ApiController]
    public class AuthenticationController : ControllerBase
    {
        private readonly UserManager<User> _userManager;
        private readonly RoleManager<Role> _roleManager;
        private readonly IPasswordHasher<User> _passwordHasher;
        private readonly IOptions<TokenSettings> _tokenSettings;

        public AuthenticationController(UserManager<User> userManager, RoleManager<Role> roleManager,
            IPasswordHasher<User> passwordHasher, IOptions<TokenSettings> tokenSettings)
        {
            _userManager = userManager;
            _roleManager = roleManager;
            _passwordHasher = passwordHasher;
            _tokenSettings = tokenSettings;
        }

        [AllowAnonymous]
        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody] RegisterModel model)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);

            var user = new User
            {
                UserName = model.Email,
                Email = model.Email
            };

            var result = await _userManager.CreateAsync(user, model.Password);
            if (result.Succeeded)
            {
                if (!await _roleManager.RoleExistsAsync("User"))
                {
                    await _roleManager.CreateAsync(new Role
                    {
                        Name = "User"
                    });
                }

                await _userManager.AddToRoleAsync(user, "User");

                return Ok();
            }

            foreach (var error in result.Errors)
            {
                ModelState.AddModelError(error.Code, error.Description);
            }

            return BadRequest(ModelState);
        }

        [AllowAnonymous]
        [HttpPost("Token")]
        public async Task<IActionResult> CreateToken([FromBody] LoginModel model)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);

            var user = await _userManager.FindByNameAsync(model.Email);
            if (user == null)
            {
                return Unauthorized();
            }

            if (_passwordHasher.VerifyHashedPassword(user, user.PasswordHash, model.Password) != PasswordVerificationResult.Success)
            {
                return Unauthorized();
            }
            var token = await CreateJwtToken(user);

            return Ok(token);
        }

        private async Task<string> CreateJwtToken(User user)
        {
            var userClaims = await _userManager.GetClaimsAsync(user);
            var allClaims = new[]
            {
                new Claim(JwtRegisteredClaimNames.NameId, user.Id.ToString()),
                new Claim(JwtRegisteredClaimNames.Sub, user.UserName),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
                new Claim(JwtRegisteredClaimNames.Email, user.Email),
            }.Union(userClaims).ToList();

            var userRoles = await _userManager.GetRolesAsync(user);
            foreach (var role in userRoles)
            {
                var roleClaim = new Claim(ClaimTypes.Role, role);
                allClaims.Add(roleClaim);
            }
            var keyBytes = Encoding.UTF8.GetBytes(_tokenSettings.Value.Key);
            var symmetricSecurityKey = new SymmetricSecurityKey(keyBytes);
            var signingCredentials = new SigningCredentials(symmetricSecurityKey, SecurityAlgorithms.HmacSha256);

            var jwtSecurityToken = new JwtSecurityToken(
                issuer: _tokenSettings.Value.Issuer,
                audience: _tokenSettings.Value.Audience,
                claims: allClaims,
                expires: DateTime.UtcNow.AddMinutes(_tokenSettings.Value.ExpirationTimeInMinutes),
                signingCredentials: signingCredentials);

            string encryptedToken = new JwtSecurityTokenHandler().WriteToken(jwtSecurityToken);

            return encryptedToken;
        }
    }
```

RestaurantDbRepository.cs

```
internal class RestaurantDbRepository : IRestaurantRepository
    {
        private readonly OdeToFoodContext _context;

        public RestaurantDbRepository(OdeToFoodContext context)
        {
            _context = context;
        }

        public Restaurant Add(Restaurant restaurant)
        {
            _context.Restaurants.Add(restaurant);

            _context.SaveChanges();

            return restaurant;
        }

        public IEnumerable<Restaurant> GetAll()
        {
            return _context.Restaurants.ToList();
        }

        public Restaurant GetById(int id)
        {
            return _context.Restaurants.FirstOrDefault(r => r.Id == id);
        }

        public void Update(Restaurant restaurant)
        {
            //Restaurant might not be tracked (attached) by the entity framework -> get original from DB and copy values
            var original = _context.Restaurants.Find(restaurant.Id);
            var entry = _context.Entry(original);
            entry.CurrentValues.SetValues(restaurant);

            _context.SaveChanges();
        }

        public void Delete(int id)
        {
            var entityToDelete = _context.Restaurants.Find(id);
            _context.Restaurants.Remove(entityToDelete);

            _context.SaveChanges();
        }
    }
```

ReviewerDbRepository.cs

```
internal class ReviewDbRepository : IReviewRepository
    {
        private readonly OdeToFoodContext _context;

        public ReviewDbRepository(OdeToFoodContext context)
        {
            _context = context;
        }

        public async Task<IReadOnlyList<Review>> GetAllAsync()
        {
            return await _context.Reviews.ToListAsync();
        }

        public async Task<Review> GetByIdAsync(int id)
        {
            return await _context.Reviews.FirstOrDefaultAsync(r => r.Id == id);
        }

        public async Task<IReadOnlyList<Review>> GetReviewsByRestaurantAsync(int id)
        {
            return await _context.Reviews.Where(r => r.RestaurantId == id).ToListAsync();
        }

        public async Task<Review> AddAsync(Review review)
        {
            _context.Reviews.Add(review);

            await _context.SaveChangesAsync();

            return review;
        }

        public async Task UpdateAsync(Review review)
        {
            //Review might not be tracked (attached) by the entity framework -> get original from DB and copy values
            var original = _context.Reviews.Find(review.Id);
            var entry = _context.Entry(original);
            entry.CurrentValues.SetValues(review);

            await _context.SaveChangesAsync();
        }

        public async Task DeleteAsync(int id)
        {
            var entityToDelete = _context.Reviews.Find(id);
            _context.Reviews.Remove(entityToDelete);

            await _context.SaveChangesAsync();
        }
    }
```

OdeToFoodContext.cs

```
internal class OdeToFoodContext : IdentityDbContext<User, Role, int>
    {
        public DbSet<Restaurant> Restaurants { get; set; }
        public DbSet<Review> Reviews { get; set; }

        public OdeToFoodContext(DbContextOptions options): base(options)
        {

        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);

            builder.Entity<Restaurant>().HasData(new List<Restaurant>
            {
                new Restaurant
                {
                    Id = 1,
                    Name = "Wok palace",
                    City = "Hasselt",
                    Country = "Belgium"
                }
            });
        }
    }
```

Conventional Routes (MusicStore)

Startup.cs

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

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews();
            services.AddSingleton<IFileProvider, HostFileProvider>();
            services.AddSingleton<IGenreRepository, GenreDummyRepository>();
            services.AddSingleton<IAlbumRepository, AlbumDummyRepository>();
            services.AddSingleton<IAlbumViewModelFactory, AlbumViewModelFactory>();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-h
                app.UseHsts();
            }
            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "Search",
                    pattern: "SearchMusic/{genre=Rock}",
                    defaults: new { controller = "Home", action = "Search" });
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
```

HomeController.cs

```
public class HomeController : Controller
    {
        public const string RockUrl = @"https://www.youtube.com/watch?v=v2AC41dglnM";

        private readonly IFileProvider _fileProvider;

        public HomeController(IFileProvider fileProvider)
        {
            _fileProvider = fileProvider;
        }

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

        public IActionResult About()
        {
            return CreateRouteInfoContent();
        }
        public IActionResult Details(int id)
        {
            return CreateRouteInfoContent();
        }

        public IActionResult Search(string genre)
        {
            switch (genre.ToLower())
            {
                case "rock":
                    return RedirectPermanent(RockUrl);
                case "jazz":
                    return RedirectToAction("Index");
                case "metal":
                    return RedirectToAction("Details", new { id = new Random().Next() });
                case "classic":
                    return File(_fileProvider.GetFileBytes(@"wwwroot\css\site.css"), "text/css", "site.css");
                default:
                    return CreateRouteInfoContent();
            }
        }

        private ContentResult CreateRouteInfoContent()
        {
            var controllerName = RouteData.Values["controller"];
            var actionName = RouteData.Values["action"];
            var message = $"{controllerName}:{actionName}";

            var id = RouteData.Values["id"];
            if (id != null)
            {
                message += $":{id}";
            }

            var genreParam = RouteData.Values["genre"];
            if (genreParam != null)
            {
                message += $":{genreParam}";
            }

            return Content(message);
        }
    }
```

Views

_Layout.cshtml

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>OdeToFood - @ViewBag.Title</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
    <footer>
        @await Component.InvokeAsync("Greeting")
        @RenderSection("footer", false)
    </footer>
</body>
</html>
```

@RenderBody() komt al onze content van onze views.

@RenderSection("footer", false) als een specifieke view een footer wilt tonen.

Als je een section wil tonen. JUISTE SECTION NAAM GEVEN.

```
@section footer
{
  <div>@Model.CurrentGreeting</div>
}
```

_ViewStart.cshtml

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

1 ⇒ _ViewStart daarna alle individuele views.

Alle individuele views gaan de layout van “_Layout.cshtml” krijgen.

_ViewImports.cshtml

```
@using OdeToFood.Web
@using OdeToFood.Web.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Het globaal maken van directives voor “Razor Pages”.

Directives:

- **@addTagHelper**
- **@tagHelperPrefix**
- **@removeTagHelper**
- **@namespace**
- **@inject**
- **@model**
- **@using**

@TagHelpers

```
<form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="City" class="control-label"></label>
                <input asp-for="City" class="form-control" />
                <span asp-validation-for="City" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Country" class="control-label"></label>
                <input asp-for="Country" class="form-control" />
                <span asp-validation-for="Country" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
```

Adding update method Or any other method

Applogic

```
public interface IRestaurantRepository
    {
        IEnumerable<Restaurant> GetAll();
        Restaurant GetById(int id);
        Restaurant Add(Restaurant restaurant);
        void Update(Restaurant restaurant);
        void Delete(int id);
    }
```

```
internal class RestaurantDbRepository : IRestaurantRepository
    {
        private readonly OdeToFoodContext _context;

        public RestaurantDbRepository(OdeToFoodContext context)
        {
            _context = context;
        }

        public Restaurant Add(Restaurant restaurant)
        {
            _context.Restaurants.Add(restaurant);

            _context.SaveChanges();

            return restaurant;
        }

        public IEnumerable<Restaurant> GetAll()
        {
            return _context.Restaurants.ToList();
        }

        public Restaurant GetById(int id)
        {
            return _context.Restaurants.FirstOrDefault(r => r.Id == id);
        }

        public void Update(Restaurant restaurant)
        {
            //Restaurant might not be tracked (attached) by the entity framework -> get original from DB and copy values
            var original = _context.Restaurants.Find(restaurant.Id);
            var entry = _context.Entry(original);
            entry.CurrentValues.SetValues(restaurant);

            _context.SaveChanges();
        }

        public void Delete(int id)
        {
            var entityToDelete = _context.Restaurants.Find(id);
            _context.Restaurants.Remove(entityToDelete);

            _context.SaveChanges();
        }
```

Authentication en Authorization

Domain layer

<int> betekent dat de primary key de type int heeft

```
public class Role : IdentityRole<int>
{
    public static class Constants
    {
        public const string Lector = "Lector";
        public const string Student = "Student";
    }
}

public class User : IdentityUser<int>
{
    // De klasse IdentityUser heeft zelf al veel default props zoals username, email, password... Hier voegen we DateOfBirth extra toe.
    public DateTime DateOfBirth { get; set; }
}

public class ExamScore
{
    public int Id { get; set; }
    public string Course { get; set; }
    public double Score { get; set; }
    public User User { get; set; }
    public int UserId { get; set; }
}
```

Business Logic layer

```
public interface IExamScoreRepository
{
    void AddScoreForUser(string course, double score, int userId);
    IList<ExamScore> GetAllScores();
    double GetAverageScore();
}
```

Infrastructure layer

```
// Vergeet AssemblyInfo.cs niet
using System.Runtime.CompilerServices;

[assembly: InternalsVisibleTo("SecureDemo.Api")]

// DBContext (internal)
internal class DemoContext : IdentityDbContext<User, Role, int>
{
    public DbSet<ExamScore> ExamScores { get; set; }

    public DemoContext(DbContextOptions options) : base(options) { }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        builder.Entity<ExamScore>().Property(es => es.Course).IsRequired();
        base.OnModelCreating(builder);
    }
}

// Repository
internal class ExamScoreDbRepository : IExamScoreRepository
{
    private readonly DemoContext _context;

    public ExamScoreDbRepository(DemoContext context)
    {
        _context = context;
    }

    public void AddScoreForUser(string course, double score, int userId)
    {
        var examScore = new ExamScore {Course = course, Score = score, UserId = userId};
        _context.ExamScores.Add(examScore);
        _context.SaveChanges();
    }

    public IList<ExamScore> GetAllScores()
    {
        return _context.ExamScores.ToList();
    }
    public double GetAverageScore()
    {
        return _context.ExamScores.Average(es => es.Score);
    }
}
```

API layer

```
// Eerst Dbcontext en Repository injecteren in Startup.cs
services.AddDbContext<DemoContext>(options =>
{
    string connectionString = @"Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=SecureWebApiDb;Integrated Security=True";
    options.UseSqlServer(connectionString);
});

services.AddScoped<IExamScoreRepository, ExamScoreDbRepository>();

// Vervolgens de Identity service injecteren met verschillende options voor bv wachtwoorden, emails
// Daarna de database linken en tot slot de service om token te creeren injecteren.
services.AddIdentity<User, Role>(options =>
{
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(15);
    options.Lockout.MaxFailedAccessAttempts = 8;
    options.Lockout.AllowedForNewUsers = true;

    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequiredLength = 6;

    options.SignIn.RequireConfirmedEmail = false;
    options.SignIn.RequireConfirmedPhoneNumber = false;
}).AddEntityFrameworkStores<DemoContext>()
  .AddDefaultTokenProviders();

// Database en migration aanmaken
ADD-MIGRATION Initial -verbose
UPDATE-DATABASE -verbose
```

AuthenticationController

```
// We hebben eerst een register model nodig
public class RegisterModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [MinLength(6)]
    public string Password { get; set; }

    public DateTime DateOfBirth { get; set; }
}

[Route("api/[controller]")]
[ApiController]
public class AuthenticationController : ControllerBase
{
    private readonly UserManager<User> _userManager;
    private readonly RoleManager<Role> _roleManager;
    private readonly IPasswordHasher<User> _passwordHasher;
    private readonly IOptions<TokenSettings> _tokenSettings;

    public AuthenticationController(
        UserManager<User> userManager,
        RoleManager<Role> roleManager,
        IPasswordHasher<User> passwordHasher,
        IOptions<TokenSettings> tokenSettings)
    {
        _userManager = userManager;
        _roleManager = roleManager;
        _passwordHasher = passwordHasher;
        _tokenSettings = tokenSettings;
    }
    [HttpPost("register")]
    [AllowAnonymous]
    public async Task<IActionResult> Register([FromBody] RegisterModel model)
    {
        //The next line can be commented out because of the [ApiController] attribute
        //if (!ModelState.IsValid) return BadRequest(ModelState);
        //TODO (someday): add captcha validation to prevent registration by bots

        var user = new User
        {
            UserName = model.Email,
            Email = model.Email,
            DateOfBirth = model.DateOfBirth
        };

        IdentityResult result = await _userManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            string role = user.Email.ToLower().EndsWith("@pxl.be") ? Role.Constants.Lector : Role.Constants.Student;
            await EnsureRoleExists(role);
            await _userManager.AddToRoleAsync(user, role);
            //TODO (someday): make the user confirm his email address

            return Ok();
        }

        //put the errors that Identity reported into the response
        foreach (IdentityError error in result.Errors)
        {
            ModelState.AddModelError(error.Code, error.Description);
        }
        return BadRequest(ModelState);
    }
}

// Vervolgens moeten we onze bearer token genereren, hiervoor hebben we ook een secret key nodig die we gaan gebruiken om onze token te enc
// De instellingen over onze token gaan we in appsettings.json toevoegen maar de secret key gaan we in onze secrets.json file toevoegen.
// appsettings.json wordt vaak meegepushed naar bv source control (git), rmk op project -> kies Manage user Secrets en voeg daar je secret

// appsettings.json
"Token": {
    "Issuer": "Hogeschool PXL",
    "Audience": "Pxl audience",
    "ExpirationTimeInMinutes": 60
  }

// secrets.json
{
  "Token": {
    "Key":  "SuperSecretKey!!!" // Must be at least 16 chars long.
  }
}

// Vervolgens gaan we een klasse TokenSettings aanmaken in onze Settings folder.
public class TokenSettings
{
    public string Key { get; set; }
    public string Issuer { get; set; }
    public string Audience { get; set; }
    public int ExpirationTimeInMinutes { get; set; }
}

// Vervolgens gaan we onze TokenSettings klasse injecteren in onze Startup.cs file
services.Configure<TokenSettings>(Configuration.GetSection("Token"));

// Om een CreateToken action te maken hebben we eerst een LoginModel nodig.
public class LoginModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [MinLength(6)]
    public string Password { get; set; }
}

[HttpPost("token")]
[AllowAnonymous]
public async Task<IActionResult> CreateToken([FromBody] LoginModel model)
{
    User user = await _userManager.FindByNameAsync(model.Email);
    if (user == null)
    {
        return Unauthorized();
    }

    if (_passwordHasher.VerifyHashedPassword(user, user.PasswordHash, model.Password) !=
        PasswordVerificationResult.Success)
    {
        return Unauthorized();
    }

    string token = await CreateJwtToken(user);
    return Ok(token);
}

// Create jwt token
private async Task<string> CreateJwtToken(User user)
{
    var userClaims = await _userManager.GetClaimsAsync(user);
    var allClaims = new[]
    {
        new Claim(JwtRegisteredClaimNames.NameId, user.Id.ToString()),
        new Claim(JwtRegisteredClaimNames.Sub, user.UserName),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        new Claim(JwtRegisteredClaimNames.Email, user.Email)
    }.Union(userClaims).ToList();

    var userRoles = await _userManager.GetRolesAsync(user);
    foreach (var role in userRoles)
    {
        var roleClaim = new Claim(ClaimTypes.Role, role);
        allClaims.Add(roleClaim);
    }

    var keyBytes = Encoding.UTF8.GetBytes((string)_tokenSettings.Value.Key);
    var symmetricSecurityKey = new SymmetricSecurityKey(keyBytes);
    var signingCredentials = new SigningCredentials(symmetricSecurityKey, SecurityAlgorithms.HmacSha256);

    var jwtSecurityToken = new JwtSecurityToken(
        issuer: _tokenSettings.Value.Issuer,
        audience: _tokenSettings.Value.Audience,
        claims: allClaims,
        expires: DateTime.UtcNow.AddMinutes(_tokenSettings.Value.ExpirationTimeInMinutes),
        signingCredentials: signingCredentials);

    string encryptedToken = new JwtSecurityTokenHandler().WriteToken(jwtSecurityToken);
    return encryptedToken;
}
```

SetUP Authenticatie en Authorization

```
//Setup bearer token authentication
services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options =>
{
    var tokenSettings = new TokenSettings();
    Configuration.Bind("Token", tokenSettings);
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidIssuer = tokenSettings.Issuer,
        ValidAudience = tokenSettings.Audience,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(tokenSettings.Key))
    };
});

//Make sure bearer token authorization is used by default by the [Authorize] attribute
services.AddAuthorization(options =>
{
    options.DefaultPolicy = new AuthorizationPolicyBuilder(JwtBearerDefaults.AuthenticationScheme)
        .RequireAuthenticatedUser().Build();
});
```

Controller die nu gebruik maakt van authorizatie

```
[Route("api/[controller]")]
[ApiController]
// [Authorize] kunnen we ook hier toevoegen, dan moeten we boven de routes nietmeer de authorize annotatie toevoegen
public class ExamScoresController : ControllerBase
{
    private readonly IExamScoreRepository _examScoreRepository;

    public ExamScoresController(IExamScoreRepository examScoreRepository)
    {
        _examScoreRepository = examScoreRepository;
    }

    [HttpPost]
    [Authorize] // we maken onze route protected
    public IActionResult AddExamScore(ExamScoreModel model)
    {
        _examScoreRepository.AddScoreForUser(model.Course, model.Score, model.UserId);
        return Ok();
    }
}

[HttpGet("average")]
[AllowAnonymous] // route is publiek toegankelijk
public IActionResult GetAverageScore()
{
    double average = _examScoreRepository.GetAverageScore();
    return Ok(average);
}

[HttpGet("my-claims")]
public IActionResult GetClaimsOfUser()
{
    var userInfoBuilder = new StringBuilder();
    foreach (var userClaim in User.Claims)
    {
        userInfoBuilder.Append($"{userClaim.Type}: {userClaim.Value} * ");
    }
    return Ok(userInfoBuilder.ToString());
}
```

Globale filter om te authenticeren bij alle inkomende requesten

```
// Bovenaan in Startup.cs

services.AddControllers(options =>
{
    var policy = new AuthorizationPolicyBuilder(JwtBearerDefaults.AuthenticationScheme)
        .RequireAuthenticatedUser().Build();
    options.Filters.Add(new AuthorizeFilter(policy));
});
```

Test Driven Development (TDD)

NUNIT

NUnit basics

```
[TestFixture]
public class GameTests
{
    [OneTimeSetUp]
    public void Init()
    {
        // Code die 1 keer runt voor de test klasse
    }

    [SetUp]
    public void SetUp()
    {
        // Code die voor elke test runt, meestal private fields initialiseren
        // Die in de verschillende testen worden gebruikt.
    }

    [Test]
    public void ScoreIsZeroAfterGutterGame()
    {
        // Arrange - Only setup code needed by the act step
        Game game = new Game(); // ... Verdere initialisatie

        // Act - Only the action(s) under test
        var sut = game.CalculateScore(); // SUT -> System Under Test

        // Assert - Verification of the excepted behavior
        Assert.That(sut, Is.EqualTo(0));
    }

    [TearDown]
    public void TestCleanup()
    {
        // Runs na elke test
    }

    [OneTimeTearDown]
    public void Cleanup()
    {
        // Runs nadat alle testen gerunt zijn
    }
}
```

Vergeet de 3A’s niet te volgen, [TextFixture] gebruiken we om een klasse als testklasse aan te duiden, [Test] annotatie gebruiken we boven alle testen.

NUnit Assertions

```
// Test Exception
Assert.Throws(Is.TypeOf<InvalidOperationException>()
                .And.Message.EqualTo("Cannot read temperature before initializing."),
              () => sut.ReadCurrentTemperature());

// Tests whether the specified values are equal.
Assert.That(28, Is.EqualTo(_actualFuel));

// Tests whether the specified values are unequal. Same as AreEqual for numeric values.
Assert.That(28, Is.Not.EqualTo(_actualFuel));

// Tests whether the specified objects both refer to the same object
Assert.That(_expectedRocket, Is.SameAs(_actualRocket));

// Tests whether the specified objects refer to different objects
Assert.That(_expectedRocket, Is.Not.SameAs(_actualRocket));

// Tests whether the specified condition is true
Assert.That(_isThereEnoughFuel, Is.True);

// Tests whether the specified condition is false
Assert.That(_isThereEnoughFuel, Is.False);

// Tests whether the specified object is null
Assert.That(_actualRocket, Is.Null);

// Tests whether the specified object is non-null
Assert.That(_actualRocket, Is.Not.Null);

// Tests whether the specified object is an instance of the expected type
Assert.That(_actualRocket, Is.InstanceOf<Falcon9Rocket>());

// Tests whether the specified object is not an instance of type
```

```
Assert.That(_actualRocket, Is.Not.InstanceOf<Falcon9Rocket>());

// Tests whether the specified object greater than the specified value
Assert.That(_actualFuel, Is.GreaterThan(20));

// Tests whether the specified values are nearly equal within the specified tolerance.
Assert.That(28, Is.EqualTo(_actualFuel).Within(0.50));

// Tests whether the specified values are nearly equal within the specified % tolerance.
Assert.That(28, Is.EqualTo(_actualFuel).Within(2).Percent);

// Tests whether the specified collection has exactly the stated number of items in it.
Assert.That(_actualRocketParts, Has.Exactly(10).Items);

// Tests whether the items in the specified collections are unique.
Assert.That(_actualRocketParts, Is.Unique);

// Tests whether a given items is present in the specified list of items.
Assert.That(_actualRocketParts, Does.Contain(_expectedRocketPart));

// Tests whether the specified collection has exactly the stated item in it.
Assert.That(_actualRocketParts, Has.Exactly(1).Matches<RocketPart>(part => part.Name == "Door" && part.Height == "200"));
```

MOQ

Nuget Package = Moq

Moq - AAA

```
[Test]
public void TheRepositorySaveShouldBeCalled()
{
    // Arrange -> Meestal ga je wss de repository mocken in de constructor
    // van de testklasse
    var mockRepository = new Mock<ICustomerRepository();

    /* Setup methode zet expectations, bv we verwachten dat de methode Save
       Opgeroepen zal worden met een der welke Customer parameter.
    */
    mockRepository.Setup(x => x.Save(It.IsAny<Customer>()));

    /*
      We kunnen ook een return object meegeven die de sut dan zal returnen. Standaard
        returned bv de mock een default object van de gegeven type in de Setup functie.
      We kunnen als volgt een eigen return object meegeven.
    */
    // _producten kan bv een dummy lijst van producten zijn die bovenaan
    // geinitialiseerd wordt
    mockRepository.Setup(x => x.GetAll()).Returns(_producten);
    mockRepository.Setup(x => x.From(It.IsAny<CustomerToCreateDto>()))
                  .Returns(() => new Address());

    // Act
    customerService.Create(new CustomerToCreateDto());

    // Assert
    /* Werden alle verwachte, in de setup gedefinieerde
      (en als "verifieerbaar" aangemerkte) methoden aangeroepen?
    */
    mockRepository.Verify();

    /* Werd de methode Save 3 keer opgeroepen met een Customer paramter? */
    mockRepository.Verify(x => x.Save(It.IsAny<Customer>()), Times.Exactly(3));

    /* Werden alle verwachte, in de setup gedefinieerde methodes aangeroepen
      (zelfs als ze niet als verifieerbaar zijn gemarkeerd)?
    */
    mockRepository.VerifyAll();
}
```

Examples - Voorbeelden MOQ

Controller test MET Views

```
[TestFixture]
public class HomeControllerTests
{
    private Mock<IRestaurantRepository> _restaurantsRepositoryMock;
    private Mock<IReviewRepository> _reviewsRepositoryMock;
    private Mock<IConverter> _converterMock;
    private HomeController _controller;
    [SetUp]
    public void Setup()
    {
        _restaurantsRepositoryMock = new Mock<IRestaurantRepository>();
        _reviewsRepositoryMock = new Mock<IReviewRepository>();
        _converterMock = new Mock<IConverter>();
        _controller = new HomeController(_restaurantsRepositoryMock.Object, _reviewsRepositoryMock.Object, _converterMock.Object);
    }

    [Test]
    public void Index_ShouldReturnAListOfRestaurants()
    {
        var restaurants = new List<Restaurant>
        {
            new RestaurantBuilder().Build()
        };

        _restaurantsRepositoryMock.Setup(r => r.GetAll()).Returns(restaurants);

        var viewResult = _controller.Index() as ViewResult;

        Assert.That(viewResult, Is.Not.Null);
        Assert.That(viewResult.Model, Is.SameAs(restaurants));
        _restaurantsRepositoryMock.Verify(r => r.GetAll(), Times.Once);
    }

    [Test]
    public void Details_ShouldReturnDetailsOfRestaurantAndReviews()
    {
        var restaurant = new RestaurantBuilder().WithId().Build();
        var reviews = new List<Review>
        {
            new ReviewBuilder().WithId().Build()
        };

        _restaurantsRepositoryMock.Setup(r => r.GetById(It.IsAny<int>())).Returns(restaurant);
        _reviewsRepositoryMock.Setup(review => review.GetReviewsByRestaurantAsync(It.IsAny<int>())).ReturnsAsync(reviews);

        var viewResult = _controller.Details(restaurant.Id).Result as ViewResult;

        Assert.That(viewResult, Is.Not.Null);
        RestaurantReviewsViewModel model = viewResult.Model as RestaurantReviewsViewModel;
        Assert.That(model, Is.Not.Null);
        Assert.That(model.Restaurant, Is.SameAs(restaurant));
        Assert.That(model.Reviews, Is.SameAs(reviews));

        _restaurantsRepositoryMock.Verify(r => r.GetById(restaurant.Id), Times.Once);
        _reviewsRepositoryMock.Verify(r => r.GetReviewsByRestaurantAsync(restaurant.Id), Times.Once);
    }

    [Test]
    public void AddReview_ShouldReturnAViewWithAnEditModelContainingTheRestaurantId()
    {
        int restaurantId = new Random().Next(1, int.MaxValue);

        var viewResult = _controller.AddReview(restaurantId) as ViewResult;

        Assert.That(viewResult, Is.Not.Null);
        var viewModel = viewResult.Model as EditReviewViewModel;
        Assert.That(viewModel, Is.Not.Null);
        Assert.That(viewModel.RestaurantId, Is.EqualTo(restaurantId));
    }

    [Test]
    public void AddReview_ValidReviewPosted_ShouldPersistReviewAndRedirectToRestaurantDetails()
    {
        EditReviewViewModel viewModel = new EditReviewViewModelBuilder().Build();
        Review review = new ReviewBuilder().WithRestaurantId(viewModel.RestaurantId).Build();

        _converterMock.Setup(c => c.ConvertEditReviewViewModelToReview(It.IsAny<EditReviewViewModel>()))
            .Returns(review);

        _reviewsRepositoryMock.Setup(r => r.AddAsync(It.IsAny<Review>())).ReturnsAsync(review);

        var redirectResult = _controller.AddReview(review.RestaurantId, viewModel).Result as RedirectToActionResult;

        Assert.That(redirectResult, Is.Not.Null);
        Assert.That(redirectResult.Permanent, Is.False);
        Assert.That(redirectResult.ActionName, Is.EqualTo(nameof(HomeController.Details)));
        Assert.That(redirectResult.RouteValues["id"], Is.EqualTo(review.RestaurantId));

        _converterMock.Verify(c => c.ConvertEditReviewViewModelToReview(viewModel), Times.Once);
        _reviewsRepositoryMock.Verify(r => r.AddAsync(review), Times.Once);
    }

    [Test]
    public void AddReview_InvalidModel_ShouldReturnTheSameView()
    {
        EditReviewViewModel viewModel = new EditReviewViewModelBuilder().Build();

        _controller.ModelState.AddModelError("someError", "Message");

        var viewResult = _controller.AddReview(viewModel.RestaurantId, viewModel).Result as ViewResult;

        Assert.That(viewResult, Is.Not.Null);
        Assert.That(viewResult.Model, Is.SameAs(viewModel));
        _reviewsRepositoryMock.Verify(r => r.AddAsync(It.IsAny<Review>()), Times.Never);
    }

    [Test]
    public void AddReview_RestaurantIdsDoNotMatch_ShouldReturnTheSameView()
    {
        EditReviewViewModel viewModel = new EditReviewViewModelBuilder().Build();

        var viewResult = _controller.AddReview(viewModel.RestaurantId + 1, viewModel).Result as ViewResult;

        Assert.That(viewResult, Is.Not.Null);
        Assert.That(viewResult.Model, Is.SameAs(viewModel));
        _reviewsRepositoryMock.Verify(r => r.AddAsync(It.IsAny<Review>()), Times.Never);
    }
}
```

Rest Api Controller test zonder async en zonder views

```
private RestaurantsController _controller;
private Mock<IRestaurantRepository> _restaurantRepositoryMock;
private readonly Random _random = new Random();

[SetUp]
public void Setup()
{
    _restaurantRepositoryMock = new Mock<IRestaurantRepository>();
    _controller = new RestaurantsController(_restaurantRepositoryMock.Object);
}

/* Test een GET request met een OK response met een lijst */
[Test]
public void GetAll_ReturnsAllRestaurantsFromRepository()
{
    //Arrange
    var restaurants = new List<Restaurant>();
    _restaurantRepositoryMock.Setup(r => r.GetAll()).Returns(restaurants);

    //Act
    var okResult = _controller.GetAll() as OkObjectResult;

    //Assert
    Assert.That(okResult, Is.Not.Null);
    Assert.That(okResult.Value, Is.SameAs(restaurants));
    _restaurantRepositoryMock.Verify(r => r.GetAll(), Times.Once);
}
```

```
[Test]
public void GetRestaurant_ReturnsRestaurantIfItExists()
{
    //Arrange
    var restaurant = new RestaurantBuilder().WithId().Build();
    _restaurantRepositoryMock.Setup(r => r.GetById(It.IsAny<int>())).Returns(restaurant);

    //Act
    var okResult = _controller.GetRestaurant(restaurant.Id) as OkObjectResult;

    //Assert
    Assert.That(okResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(r => r.GetById(restaurant.Id), Times.Once);
    Assert.That(okResult.Value, Is.SameAs(restaurant));
}

/* Test of GetById een NotFound returned als de ID niet bestaat */
[Test]
public void GetRestaurant_ReturnsNotFoundIfItDoesNotExists()
{
    //Arrange
    _restaurantRepositoryMock.Setup(r => r.GetById(It.IsAny<int>())).Returns(() => null);

    //Act
    int someId = _random.Next();
    var notFoundResult = _controller.GetRestaurant(someId) as NotFoundResult;

    //Assert
    Assert.That(notFoundResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(r => r.GetById(someId), Times.Once);
}

/* Test een POST request die in entiteit opslaat in de db */
[Test]
public void Post_ValidRestaurantIsSavedInRepository()
{
    //Arrange
    var newRestaurant = new RestaurantBuilder().Build();

    _restaurantRepositoryMock.Setup(repo => repo.Add(It.IsAny<Restaurant>())).Returns(() =>
    {
        newRestaurant.Id = _random.Next();
        return newRestaurant;
    });

    //Act
    var createdResult = _controller.Post(newRestaurant) as CreatedAtActionResult;

    //Assert
    Assert.That(createdResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(r => r.Add(newRestaurant), Times.Once);
    Assert.That(createdResult.Value, Is.SameAs(newRestaurant));
    Assert.That(createdResult.ActionName, Is.EqualTo(nameof(RestaurantsController.GetRestaurant)));
    Assert.That(createdResult.RouteValues["id"], Is.EqualTo(((Restaurant)createdResult.Value).Id));
}

/* Test een POST request met een ModelError */
[Test]
public void Post_InValidRestaurantCausesBadRequest()
{
    //Arrange
    _controller.ModelState.AddModelError("Name", "Name is required");

    //Act
    var badRequestResult = _controller.Post(new RestaurantBuilder().WithEmptyName().Build()) as BadRequestResult;

    //Assert
    Assert.That(badRequestResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(repo => repo.Add(It.IsAny<Restaurant>()), Times.Never);
}


/* Test een PUT request die in entiteit updated in de db */
[Test]
public void Put_ExistingRestaurantIsSavedInRepository()
{
    //Arrange
    var aRestaurant = new RestaurantBuilder().WithId().Build();

    _restaurantRepositoryMock.Setup(r => r.GetById(It.IsAny<int>())).Returns(aRestaurant);
    //Act
    var okResult = _controller.Put(aRestaurant.Id, aRestaurant) as OkResult;

    //Assert
    Assert.That(okResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(r => r.GetById(aRestaurant.Id), Times.Once);
    _restaurantRepositoryMock.Verify(r => r.Update(aRestaurant), Times.Once);
}

/* Test een PUT request met een id en entiteit die niet bestaat */
[Test]
public void Put_NonExistingRestaurantReturnsNotFound()
{
    //Arrange
    _restaurantRepositoryMock.Setup(r => r.GetById(It.IsAny<int>())).Returns(() => null);

    var aRestaurant = new RestaurantBuilder().WithId().Build();

    //Act
    var notFoundResult = _controller.Put(aRestaurant.Id, aRestaurant) as NotFoundResult;

    //Assert
    Assert.That(notFoundResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(r => r.GetById(aRestaurant.Id), Times.Once);
    _restaurantRepositoryMock.Verify(r => r.Update(It.IsAny<Restaurant>()), Times.Never);
}

/* Test een PUT request met ModelError */
[Test]
public void Put_InValidRestaurantModelStateCausesBadRequest()
{
    //Arrange
    _controller.ModelState.AddModelError("Name", "Name is required");

    var aRestaurant = new RestaurantBuilder().WithEmptyName().Build();

    //Act
    var badRequestResult = _controller.Put(aRestaurant.Id, aRestaurant) as BadRequestResult;

    //Assert
    Assert.That(badRequestResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(r => r.Update(It.IsAny<Restaurant>()), Times.Never);
}

/* Test een PUT request met een foute id maar een juiste entiteit */
[Test]
public void Put_MismatchBetweenUrlIdAndRestaurantIdCausesBadRequest()
{
    //Arrange
    var aRestaurant = new RestaurantBuilder().WithId().Build();

    //Act
    var badRequestResult = _controller.Put(aRestaurant.Id + 1, aRestaurant) as BadRequestResult;

    //Assert
    Assert.That(badRequestResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(r => r.Update(It.IsAny<Restaurant>()), Times.Never);
}

/* Test een DELETE request die een entiteit verwijdert */
[Test]
public void Delete_ExistingRestaurantIsDeletedFromRepository()
{
    //Arrange
    var aRestaurant = new RestaurantBuilder().WithId().Build();

    _restaurantRepositoryMock.Setup(r => r.GetById(It.IsAny<int>())).Returns(aRestaurant);

    //Act
    var okResult = _controller.Delete(aRestaurant.Id) as OkResult;

    //Assert
    Assert.That(okResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(r => r.GetById(aRestaurant.Id), Times.Once);
    _restaurantRepositoryMock.Verify(r => r.Delete(aRestaurant.Id), Times.Once);
}

/* Test een DELETE request met een foute id die NotFound returned */
[Test]
public void Delete_NonExistingRestaurantReturnsNotFound()
{
    //Arrange
    _restaurantRepositoryMock.Setup(r => r.GetById(It.IsAny<int>())).Returns(() => null);
    var someId = _random.Next();

    //Act
    var notFoundResult = _controller.Delete(someId) as NotFoundResult;

    //Assert
    Assert.That(notFoundResult, Is.Not.Null);
    _restaurantRepositoryMock.Verify(r => r.GetById(someId), Times.Once);
    _restaurantRepositoryMock.Verify(r => r.Delete(It.IsAny<int>()), Times.Never);
}
```

Rest Api Controller test MET Async en zonder views

```
[TestFixture]
public class ReviewControllerTests
{
    private ReviewsController _controller;
    private Mock<IReviewRepository> _reviewRepositoryMock;
    private readonly Random _random = new Random();

    [SetUp]
    public void Setup()
    {
        _reviewRepositoryMock = new Mock<IReviewRepository>();
        _controller = new ReviewsController(_reviewRepositoryMock.Object);
    }

    [Test]
    public void Get_ReturnsAllReviewsFromRepository()
    {
        //Arrange
        var reviews = new List<Review>();
        _reviewRepositoryMock.Setup(r => r.GetAllAsync()).ReturnsAsync(reviews);

        //Act
        var okResult = _controller.Get().Result as OkObjectResult;

        //Assert
        Assert.That(okResult, Is.Not.Null);
        _reviewRepositoryMock.Verify(r => r.GetAllAsync(), Times.Once());
        Assert.That(okResult.Value, Is.SameAs(reviews));
    }

    [Test]
    public void Get_ReturnsReviewIfItExists()
    {
        //Arrange
        var review = new ReviewBuilder().WithId().Build();
        //_reviewRepositoryMock.Setup(r => r.GetByIdAsync(review.Id)).Returns(() => Task.FromResult(review));
        _reviewRepositoryMock.Setup(r => r.GetByIdAsync(It.IsAny<int>())).ReturnsAsync(review);

        //Act
        var okResult = _controller.Get(review.Id).Result as OkObjectResult;

        //Assert
        Assert.That(okResult, Is.Not.Null);
        _reviewRepositoryMock.Verify(r => r.GetByIdAsync(review.Id), Times.Once);
        Assert.That(okResult.Value, Is.SameAs(review));
    }

    [Test]
    public void Get_ReturnsNotFoundIfReviewDoesNotExists()
    {
        //Arrange
        //_reviewRepositoryMock.Setup(r => r.GetByIdAsync(It.IsAny<int>())).Returns(() => Task.FromResult<Review>(null));
        _reviewRepositoryMock.Setup(r => r.GetByIdAsync(It.IsAny<int>())).ReturnsAsync(() => null);
        var someReviewId = _random.Next();

        //Act
        var notFoundResult = _controller.Get(someReviewId).Result as NotFoundResult;

        //Assert
        Assert.That(notFoundResult, Is.Not.Null);
        _reviewRepositoryMock.Verify(r => r.GetByIdAsync(someReviewId), Times.Once);
    }

    [Test]
    public void Post_ValidReviewIsSavedInRepository()
    {
        //Arrange
        var newReview = new ReviewBuilder().Build();
        _reviewRepositoryMock.Setup(repo => repo.AddAsync(It.IsAny<Review>())).Returns(() =>
        {
            newReview.Id = _random.Next(1, int.MaxValue);
            return Task.FromResult(newReview);
        });

        //Act
        var createdResult = _controller.Post(newReview).Result as CreatedAtActionResult;

        //Assert
        Assert.That(createdResult, Is.Not.Null);
        _reviewRepositoryMock.Verify(r => r.AddAsync(newReview), Times.Once);

        Assert.That(createdResult.Value, Is.SameAs(newReview));
        Assert.That(createdResult.ActionName, Is.EqualTo(nameof(ReviewsController.Get)));
        Assert.That(createdResult.RouteValues["id"], Is.EqualTo(newReview.Id));
    }

    [Test]
    public void Post_InValidReviewModelStateCausesBadRequest()
    {
        //Arrange
        _controller.ModelState.AddModelError("ReviewerName", "ReviewerName is required");

        var newReview = new ReviewBuilder().WithReviewerName(null).Build();

        //Act
        var badRequestResult = _controller.Post(newReview).Result as BadRequestResult;

        //Assert
        Assert.That(badRequestResult, Is.Not.Null);
        _reviewRepositoryMock.Verify(repo => repo.AddAsync(It.IsAny<Review>()), Times.Never);
    }

    [Test]
    public void Put_ExistingReviewIsSavedInRepository()
    {
        //Arrange
        var aReview = new ReviewBuilder().WithId().Build();
        //_reviewRepositoryMock.Setup(r => r.GetByIdAsync(aReview.Id)).Returns(() => Task.FromResult<Review>(aReview));
        _reviewRepositoryMock.Setup(r => r.GetByIdAsync(It.IsAny<int>())).ReturnsAsync(aReview);

        //Act
        var okResult = _controller.Put(aReview.Id, aReview).Result as OkResult;

        //Assert
        Assert.That(okResult, Is.Not.Null);
        _reviewRepositoryMock.Verify(r => r.GetByIdAsync(aReview.Id), Times.Once);
        _reviewRepositoryMock.Verify(r => r.UpdateAsync(aReview), Times.Once);
    }

    [Test]
    public void Put_NonExistingReviewReturnsNotFound()
    {
        //Arrange
        //_reviewRepositoryMock.Setup(r => r.GetByIdAsync(It.IsAny<int>())).Returns(() => Task.FromResult<Review>(null));
        _reviewRepositoryMock.Setup(r => r.GetByIdAsync(It.IsAny<int>())).ReturnsAsync(() => null);

        var aReview = new ReviewBuilder().WithId().Build();

        //Act
        var notFoundResult = _controller.Put(aReview.Id, aReview).Result as NotFoundResult;

        //Assert
        Assert.That(notFoundResult, Is.Not.Null);
        _reviewRepositoryMock.Verify(r => r.GetByIdAsync(aReview.Id), Times.Once);
        _reviewRepositoryMock.Verify(r => r.UpdateAsync(It.IsAny<Review>()), Times.Never);
    }

    [Test]
    public void Put_InValidReviewModelStateCausesBadRequest()
    {
        //Arrange
        _controller.ModelState.AddModelError("ReviewerName", "ReviewerName is required");

        var aReview = new ReviewBuilder().WithReviewerName(null).Build();

        //Act
        var badRequestResult = _controller.Put(aReview.Id, aReview).Result as BadRequestResult;

        //Assert
        Assert.That(badRequestResult, Is.Not.Null);
    }

    [Test]
    public void Put_MismatchBetweenUrlIdAndReviewIdCausesBadRequest()
    {
        //Arrange
        var aReview = new ReviewBuilder().WithId().Build();

        //Act
        var badRequestResult = _controller.Put(aReview.Id + 1, aReview).Result as BadRequestResult;

        //Assert
        Assert.That(badRequestResult, Is.Not.Null);
    }

    [Test]
    public void Delete_ExistingReviewIsDeletedFromRepository()
    {
        //Arrange
        var aReview = new ReviewBuilder().WithId().Build();
        //_reviewRepositoryMock.Setup(r => r.GetByIdAsync(aReview.Id)).Returns(() => Task.FromResult(aReview));
        _reviewRepositoryMock.Setup(r => r.GetByIdAsync(It.IsAny<int>())).ReturnsAsync(aReview);

        //Act
        var okResult = _controller.Delete(aReview.Id).Result as OkResult;

        //Assert
        Assert.That(okResult, Is.Not.Null);
        _reviewRepositoryMock.Verify(r => r.GetByIdAsync(aReview.Id), Times.Once);
        _reviewRepositoryMock.Verify(r => r.DeleteAsync(aReview.Id), Times.Once);
    }

    [Test]
    public void Delete_NonExistingReviewReturnsNotFound()
    {
        //Arrange
        //_reviewRepositoryMock.Setup(r => r.GetByIdAsync(It.IsAny<int>())).Returns(() => Task.FromResult<Review>(null));
        _reviewRepositoryMock.Setup(r => r.GetByIdAsync(It.IsAny<int>())).ReturnsAsync(() => null);

        var aReview = new ReviewBuilder().WithId().Build();

        //Act
        var notFoundResult = _controller.Delete(aReview.Id).Result as NotFoundResult;

        //Assert
        Assert.That(notFoundResult, Is.Not.Null);
        _reviewRepositoryMock.Verify(r => r.GetByIdAsync(aReview.Id), Times.Once);
        _reviewRepositoryMock.Verify(r => r.DeleteAsync(It.IsAny<int>()), Times.Never);
    }
}
```

Builders

```
internal class RestaurantBuilder
{
    private readonly Restaurant _restaurant;
    private static readonly Random Random = new Random();

    public RestaurantBuilder()
    {
        _restaurant = new Restaurant
        {
            Name = Guid.NewGuid().ToString(),
        };
    }

    public RestaurantBuilder WithId()
    {
        _restaurant.Id = Random.Next();
        return this;
    }

    public RestaurantBuilder WithEmptyName()
    {
        _restaurant.Name = null;
        return this;
    }

    public Restaurant Build()
    {
        return _restaurant;
    }
}

internal class ReviewBuilder
{
    private readonly Review _review;
    private readonly Random _random;

    public ReviewBuilder()
    {
        _random = new Random();

        _review = new Review
        {
            ReviewerName = Guid.NewGuid().ToString(),
            Rating = _random.Next(1, 6)
        };
    }

    public ReviewBuilder WithId()
    {
        _review.Id = _random.Next();
        return this;
    }

    public ReviewBuilder WithRestaurantId(int restaurantId)
    {
        _review.RestaurantId = restaurantId;
        return this;
    }

    public ReviewBuilder WithReviewerName(string name)
    {
        _review.ReviewerName = name;
        return this;
    }

    public Review Build()
    {
        return _review;
    }


}

public class EditReviewViewModelBuilder
{
    private readonly EditReviewViewModel _editReviewViewModel;

    public EditReviewViewModelBuilder()
    {
        var random = new Random();

        _editReviewViewModel = new EditReviewViewModel
        {
            Rating = random.Next(),
            Body = Guid.NewGuid().ToString(),
            RestaurantId = random.Next(),
            ReviewerName = Guid.NewGuid().ToString()
        };
    }
    public EditReviewViewModel Build()
    {
        return _editReviewViewModel;
    }
}
```