ARTESIS PLANTIJN HOGESCHOOL ANTWERPEN

Team: LightGate

Klas: 3EA – 1

DOCUMENTATIE ARCHITECTUUR & API REQUESTS

Inhoud

Algemeen 2

Architectuur 2

Data Layer 2

Business Layer 3

Service Layer ..... **Fout! Bladwijzer niet gedefinieerd.**

Presentation Layer 5

Aanmaken van een nieuw project binnenin de architectuur 7

Aanmaken van een nieuwe facade binnenin de architectuur 9

Debugging 10

Sequentiediagram Architectuur 10

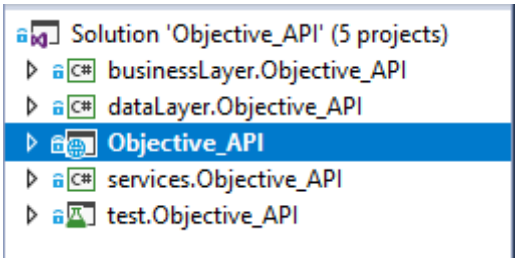
API requests 11

Algemeen

Dit document gaat u meer informatie geven over het toevoegen van bepaalde bestanden aan de architectuur, alsook het correct gebruiken van de API requests betreffende de **Objective_API**

Architectuur

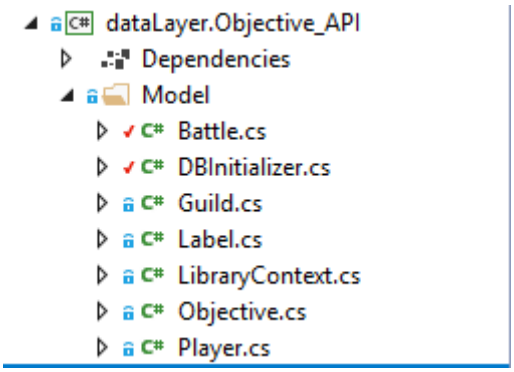
Onze architectuur ziet er momenteel als volgt uit:



We maken dus gebruik van een **business layer**, **data layer** en **service layer**, bovenop de **presentation layer**. Wat deze layers allemaal betekenen en bevatten zullen we nu eens even bekijken. We gaan van boven naar onder werken, zodat de principes al iets duidelijker worden.

Data Layer

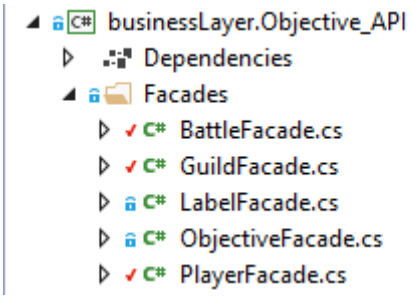
Voor de Objective_API omvat de data layer alle models. Het initialiseren van de database, de library context, … horen hier ook allemaal bij.



Deze models zijn nog altijd dezelfde models vanuit Entity Framework. Buiten dat we nu de models aanspreken vanuit de gewenste namespace, is er niets aan de code gewijzigd.

Business Layer

Voor de Objective_API omvat de business layer alle **facades**.



Een **facade** omvat op zijn beurt eigenlijk alle business logica van het project. In dit geval is dat dus alle logica omtrent de verscheidene controllers.

```
namespace businessLayer.Objective_API.Facades
{
    4 references
    public class ObjectiveFacade
    {
        private readonly LibraryContext context;

        0 references | 0 exceptions
        public ObjectiveFacade(LibraryContext context)
        {
            this.context = context;
        }

        // Get full library

        1 reference | 0 exceptions
        public List<Objective> GetObjectivesLibrary()
        {
            return context.Objectives.Include(d => d.Labels).ToList();
        }
```

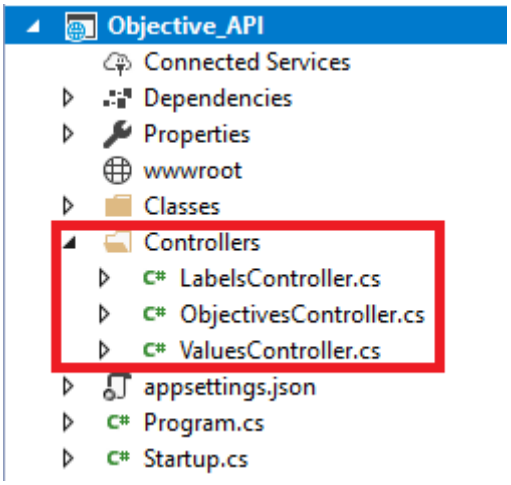
We maken eerst en vooral een readonly object aan van de context. Dit hebben we ook altijd bij Entity Framework moeten doen.

We injecteren nu de context in de facade, in plaats van in de controller zelf.

In plaats van dat de logica meteen in de controller zit, gaan we nu de logica schrijven in deze facade. Als we dus alle objectives willen opvragen, gaan we de query hier noteren, in plaats van in de controller zelf.

Controllers

Voor de Objective_API staan de controllers onder Objective_API -> Controllers.



Het enige wat de controllers mogen doen is een bepaalde **facade** aanspreken. Hieronder kan u een voorbeeld vinden vanuit de code.

```
1 reference
public class ObjectivesController : Controller
{
    private readonly ObjectiveFacade facade;

    0 references | 0 exceptions
    public ObjectivesController(ObjectiveFacade facade)
    {
        this.facade = facade;
    }

    // Get full library

    [HttpGet]

    0 references | 0 requests | 0 exceptions
    public List<Objective> GetObjectivesLibrary()
    {
        return facade.GetObjectivesLibrary();
    }
```

We maken eerst en vooral een readonly object aan van de desbetreffende facade.

We injecteren nu de facade in de controller, in plaats van de context.

In plaats van dat de logica meteen in de controller zit, gaan we nu een methode (met diezelfde logica) aanroepen vanuit de desbetreffende facade.

Service Layer

Voor de Objective_API omvat de service layer alle Interfaces. De facades van op de business layer zullen deze interfaces implementeren. In de controllers zullen we ook het interface injecteren, in plaats van de klasse zelf.


```
2 references | Denny Matthijs, 22 days ago | 1 author, 1 chang
public class BattleFacade : IBattleFacade
{
```

```
1 reference | Denny Matthijs, 22 days ago | 1 author, 1 change
public class BattlesController : Controller
{
    private readonly IBattleFacade facade;

    0 references | Denny Matthijs, 22 days ago | 1 author, 1 change | 0 exceptions
    public BattlesController(IBattleFacade facade)
    {
        this.facade = facade;
    }
```

Het enige wat de controllers mogen doen is een bepaalde **facade** aanspreken. Hieronder kan u een voorbeeld vinden vanuit de code.

Presentation Layer

Voor de Objective_API omvat de presentation layer de front-end projecten.

Projectnaam: OCT (Objective_Creation_Tool)

Projectnaam: LightGate

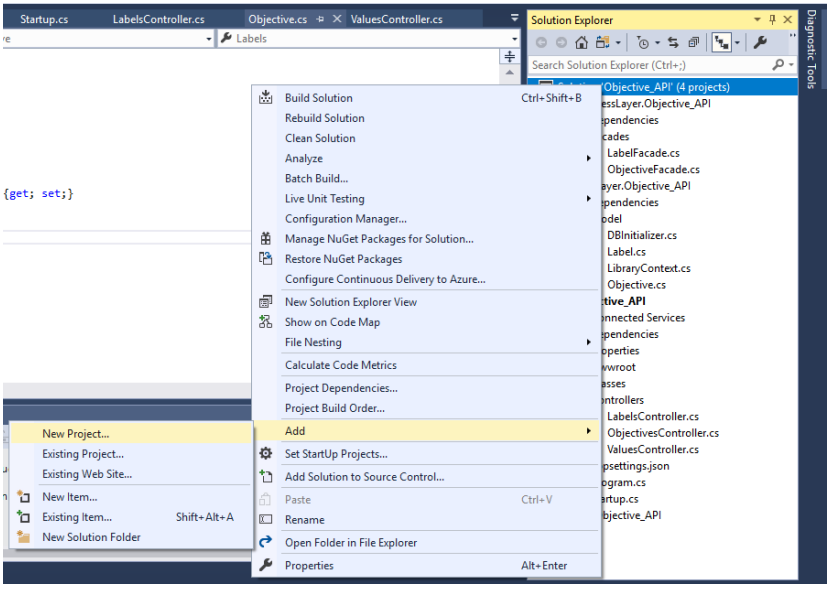
Alles betreffende de UI van ons project valt onder de presentation layer.

Aanmaken van een nieuw project binnenin de architectuur

Wanneer we voor een bepaalde reden nog een nieuw project willen toevoegen of aanmaken, dan zijn er hier een aantal stappen voor. Stel dat we bijvoorbeeld een tweede unit test project zouden willen aanmaken, dan volgen we deze stappen:

Stap 1: Aanmaken van UT project via de top-solution

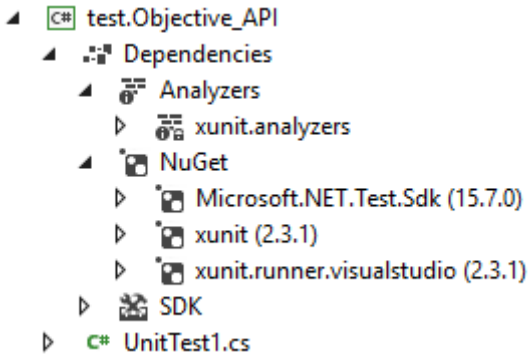
Het is aangeraden om de top-solution te gebruiken, zodat het project als een apart project wordt aangemaakt en niet als een onderdeel van een ander project.



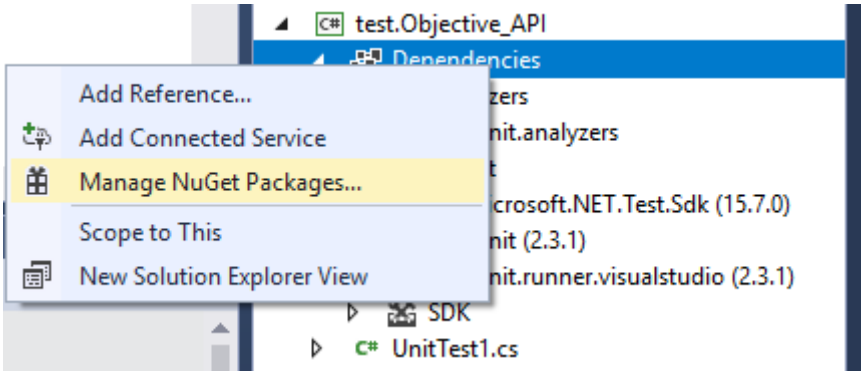
Vervolgens kiezen we dan natuurlijk voor een Unit Test project en geven we het een naam.

Stap 2: Nuget Packages, Analyzers, …

In het geval van een unit test project, zijn alle nodige packages al meteen klaargezet.

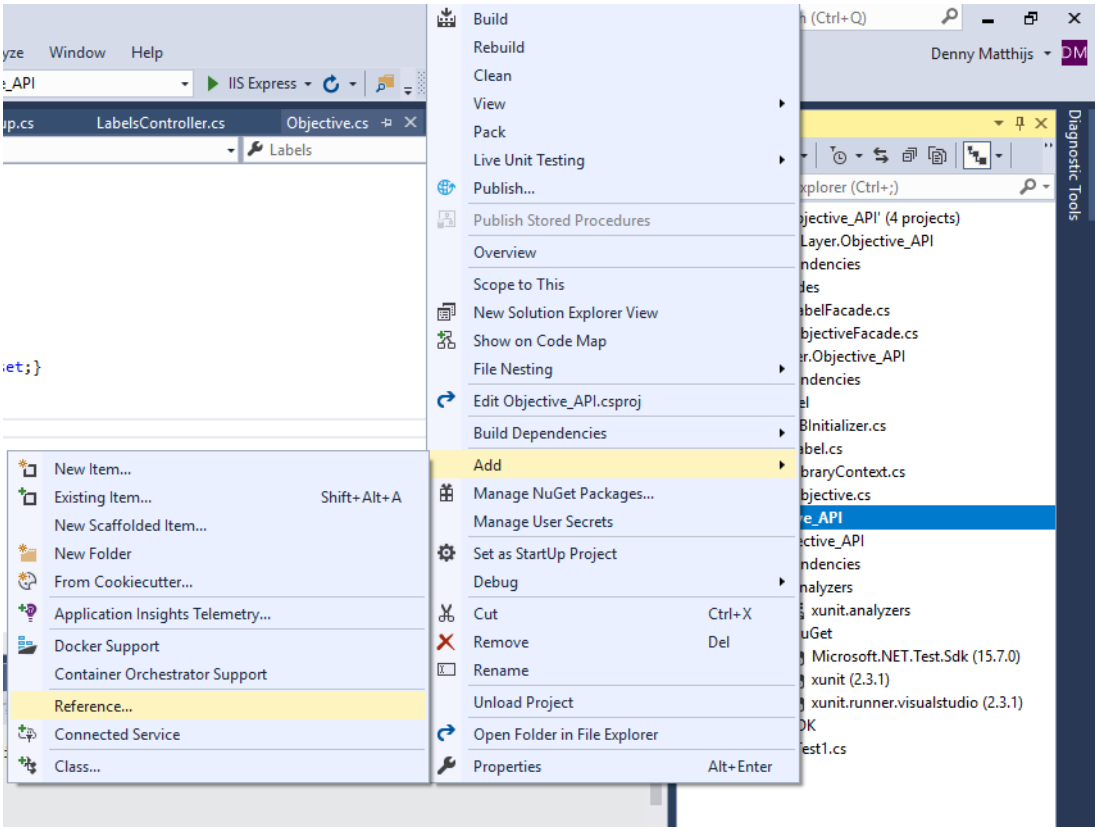


Moesten er nog bepaalde nuget packages zijn die we missen, dan kunnen we die toevoegen met behulp van **Manage NuGet Packages** (foto op volgende pagina).



Stap 3: Referenties toevoegen

Nog een **heel belangrijk** gegeven is het toevoegen van referenties tussen de verschillende projecten. De meeste projecten, net zoals dit unit test project, moeten alleen maar toegevoegd worden aan de **Objective_API**. De business layer heeft echter wel ook een referentie naar de data layer. Het hangt er dus soms een beetje van af. Als er hierover nog onduidelijkheden moesten zijn, dan kan u mij altijd contacteren.

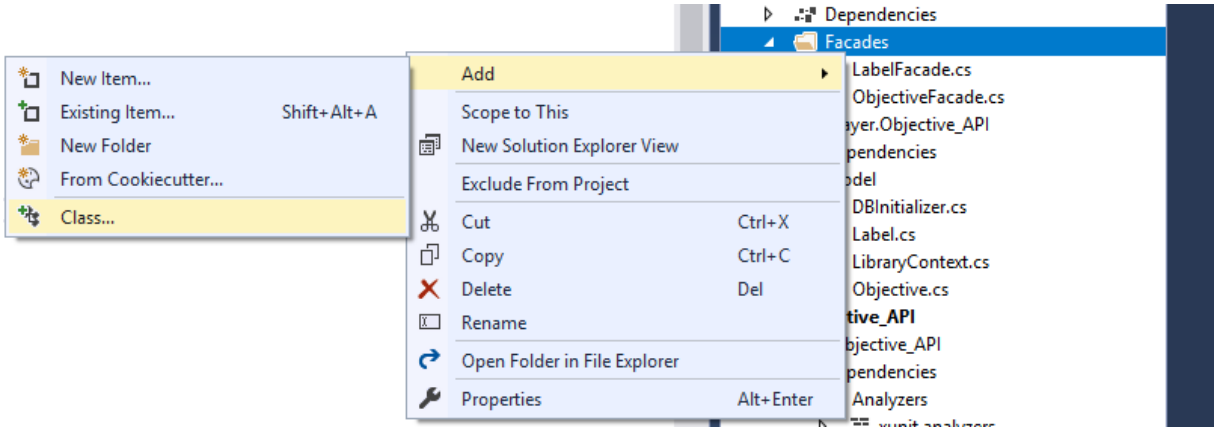


Aanmaken van een nieuwe facade binnenin de architectuur

Voor het aanmaken van een nieuwe facade zijn er ook nog twee stappen.

Stap 1: Aanmaken van een klasse

Deze eerste stap is denk ik vrij voor de hand liggend. We maken gewoon een nieuwe klasse aan doormiddel van te rechterklikken op de **Facades** folder, dan op add te drukken en dan “class” aan te duiden.



Stap 2: Toevoegen van de facade aan Startup.cs

Dit is ook een **heel belangrijke** stap. In **Startup.cs**, staat er een methode **ConfigureServices**. Hier heb ik in comment aangeduidt waar de facades moeten komen. Deze facades moeten altijd volgens dit principe worden toegevoegd:

```
// This method gets called by the runtime. Use this method to add services to the container.
0 references | 0 exceptions
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<LibraryContext>(
        options => options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")
    ));
    //Add facades here
    services.AddScoped<ObjectiveFacade>();
    services.AddScoped<LabelFacade>();

    services.AddCors();
    services.AddMvc().AddJsonOptions(options => {
        options.SerializerSettings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore;
    });
}
```

Debugging

Voor het debuggen van het programma, kan dit op **twee** manieren. Of terwijl debuggen we op de smartphone, of terwijl debuggen we met behulp van de localhost.

In de OCT heb ik bovenaan **objectives.service.ts** twee opties voorzien voor met te debuggen. We gaan later sowieso meer op mobile moeten debuggen, maar dat duurt wel wat langer om het altijd up te loaden. Als er dus iets is dat we gewoon op localhost kunnen testen, is deze optie er ook.

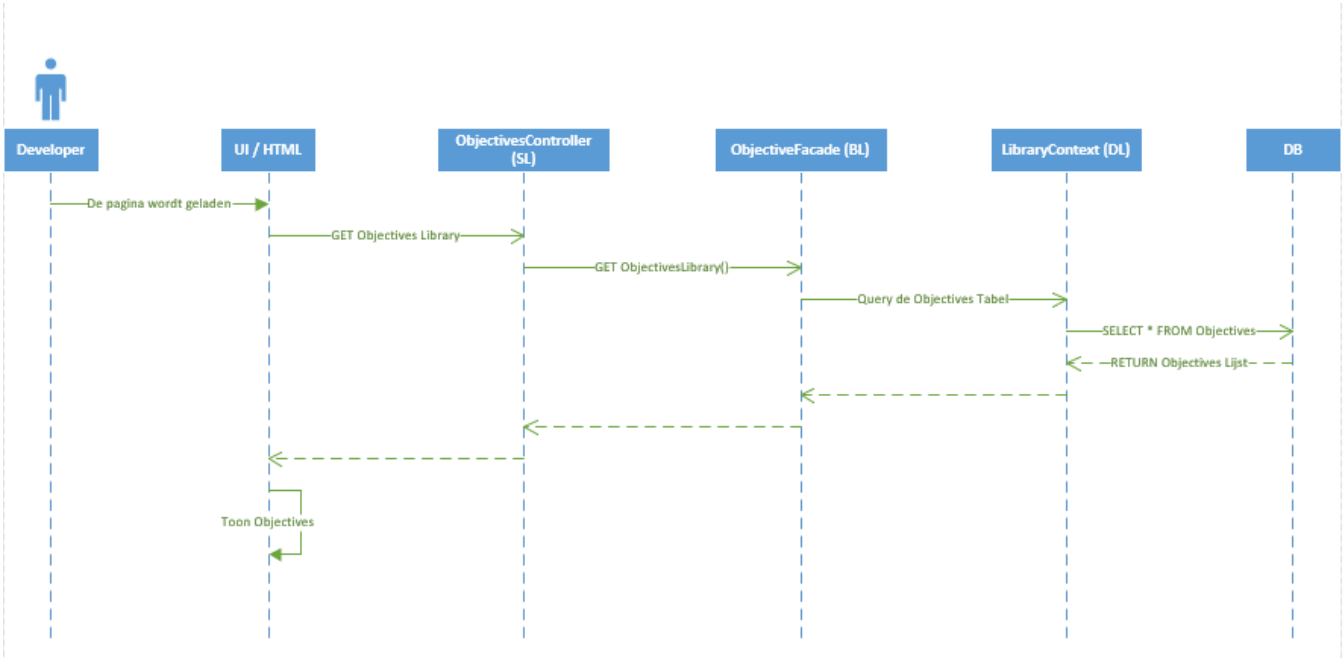
```
// Uncomment for mobile debug
// private url = "http://objective-creation-tool.azurewebsites.net/api/v1/objectives";

// Uncomment for localhost debug
private url = "http://localhost:2052/api/v1/objectives";
```

PS: Vergeet natuurlijk niet de Objective_API te runnen op localhost, wanneer je dit doet.

Sequentiediagram Architectuur

Hieronder kan u een voorbeeld zien van hoe alle objectives getoond zullen worden door middel van deze architectuur.



API requests

De API requests zijn natuurlijk ook heel belangrijk. Momenteel zijn dit de geïntegreerde requests in de nieuwe architectuur:

GET /api/v1/objectives/ → **GET alle objectives**

GET /api/v1/objectives/{id}/ → **GET een specifieke objective**

GET /api/v1/labels/ → **GET alle labels**

POST /api/v1/objectives/ → **POST een nieuwe objective**

GET /api/v1/players/ → **GET alle players**

GET /api/v1/players/{id}/ → **GET een specifieke player**

PUT /api/v1/players/{id}/ → **Update een specifieke player**

POST /api/v1/players/ → **Maak een nieuwe player aan**

GET /api/v1/guilds/ → **GET alle guilds**

GET /api/v1/guilds/{id}/ → **GET een specifieke guild**

POST /api/v1/guilds/ → **Maak een nieuwe guild aan**

GET /api/v1/battles/ → **GET alle battles**

GET /api/v1/battles/{id}/ → **GET een specifieke battle**