**Project Fotolab**

**Software Architecture Document**

**Versie 1.0**

**Revisie Historie**

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | **Versie** | **Beschrijving** | **Auteur** |
| 15-5-2014 | 0.5 | Eerste versie document | Ilja fiers |
| 28-6-2014 | 0.9 | Alle hoofdstukken ingevuld | Ilja Fiers |
| 29-6-2014 | 1.0 | Eerste versie voor publicatie | Ilja Fiers |
|  |  |  |  |

Inhoud

[1.                  Introductie 3](#_Toc391819880)

[1.1               Doel 3](#_Toc391819881)

[1.2               Scope 3](#_Toc391819882)

[1.3               Definities, Acroniemen en afkortingen 3](#_Toc391819883)

[1.4               Overzicht 3](#_Toc391819884)

[2.                  Architectural Representation 4](#_Toc391819885)

[3.                  Architectural Goals and Constraints 4](#_Toc391819886)

[4.                  Use-Case View 4](#_Toc391819887)

[4.1               Use-Case Realizations 4](#_Toc391819888)

[5.                  Logical View 6](#_Toc391819889)

[5.1               Overview 6](#_Toc391819890)

[5.2               Architecturally Significant Design Packages 6](#_Toc391819891)

[6.                  Process View 8](#_Toc391819892)

[7.                  Deployment View 8](#_Toc391819893)

[8.                  Data View 9](#_Toc391819894)

[9.             Size and Performance 11](#_Toc391819895)

[10.             Kwaliteit 11](#_Toc391819896)

**Software Architecture Document**

# 1.                  Introductie

Dit document beschrijft de implementatie van de casus “Fotolab” zoals ons aangeleverd. Deze uitwerking is het derde deel van het semester Enterprise Software van de NOH-i ICT opleiding.

In de casus “Fotolab” wordt een werkwijze beschreven van een foto reproductie laboratorium, dat foto’s en bijbehorende producten (denk aan frames, posters) produceert en deze door klanten kan laten bestellen.

De casus beschrijft een huidige werkwijze, die ervan uit gaat dat er veel handmatig wordt gewerkt. Daarna worden er verbetervoorstellen gedaan, waarvan wij de architectuur gaan uitwerken en delen van deze architectuur gaan implementeren.

## 1.1               Doel

Dit document is een onderdeel van de op te leveren producten van deze casus. Het bevat een architectuuroverzicht van het te bouwen systeem, en beschrijft diverse aspecten van de casus. Hierin worden ontwerpbeslissingen vastgelegd en overgebracht.

Het is geschreven als product, en dus bedoeld als aanwijzing voor de werkelijke implementatie van de hier beschreven casus. De doelgroep is allereerst wijzelf, zodat we de beschreven onderdelen ook kunnen bouwen. Daarnaast is dit document bedoeld als eindproduct, een van de op te leveren producten ter beoordeling van de casus.

## 1.2               Scope

In deze casus gaan wij niet alles zoals beschreven implementeren. Het gaat vooral om een goede beschrijving van de onderliggende architectuur, en niet zozeer de implementatie van het geheel. De architectuur gaan wij volledig uitwerken, met een beschreven databasemodel inclusief DDL en Insert-script. Verder maken we een high-level beschrijving van de omgeving waarin deze database draait.

En deel van de beschreven systemen gaan wij ook werkelijk bouwen. Deze systemen worden specifiek benoemd, waarbij de lezer er van uit kan gaan dat delen die niet specifiek genoemd zijn ook niet gebouwd gaan worden.

Dit document gaat kort in op zaken als performance en beveiliging. Deze zaken zijn in ieder geval geen onderdeel van de bouw.

## 1.3               Definities, Acroniemen en afkortingen

Fotoproducent: een persoon of firma die foto’s maakt en deze aanlevert aan het fotolab ter reproductie.

Fotolab: de uitvoerder van bestellingen van foto’s, tevens exploitant van dit software product.

## 1.4               Overzicht

In de rest van dit document worden de verschillende componenten van de oplossing die wij gebouwd hebben beschreven, vanuit diverse perspectieven. Hoofdstuk 4 kijkt vanuit een usecase-perspectief naar de opgaven. Hoofdstuk 5 beschrijft de verschillende componenten vanuit een package- en klassen perspectief.

# 2.                  Architectural Representation

Niet alle representaties uit een compleet SAD document zijn volledig van toepassing op de oplossing zoals wij die gebouwd hebben, zonder in herhaling te vallen. Hoofdstuk 4, Use-Case view, en hoofdstuk 5, logical view, zijn beter uitgewerkt dan hoofdstuk 7, (deployment view) en hoofdstuk 8 (implementation view). Dit omdat onze oplossing gebaseerd is op vele lagen bestaande techniek, en we ons gezien de tijdsdruk beperkt hebben tot onze eigen code bovenop de gegeven webtechnieken.

Aangezien het Fotolab vooral een database applicatie is vanuit ons perspectief hebben we in hoofdstuk 9 het optionele hoofdstuk ‘Data view’ uitgewerkt.

# 3.                  Architectural Goals and Constraints

Het doel van deze casus is vooral om de opgedane kennis uit het semester Enterprise Software toe te passen in een (soort van) ‘real-world’ applicatie. We hebben daarom besloten gebruik te maken van een WebAPI 2.0, geschreven in .NET, die via SQL een onderliggende database aanspreekt.

# 4.                  Use-Case View

## 4.1               Use-Case Realizations

Use Case: Fotoproducent upload fotoserie

|  |  |
| --- | --- |
| **Primary actor**:  Foto producent | |
| **Preconditions:**  -Producent heeft de appicatie ‘Fotoproducent’ opgestart.  -Er is een werkende verbinding met de WebAPI die de foto’s gaat ontvangen. | |
| **Postconditions (Success Guarantee):**  Er is een fotoserie toegevoegd aan de database. Deze fotoserie heeft een geldige klantvermelding en een naam. De bijbehorende foto’s staan op de server die de WebAPI draait. Er is een geldige en consistente vermelding in de database opgenomen van iedere foto. | |
| **Main Success Scenario (Basic Flow):** | |
| Actor Action | System Responsibility |
| 1. Fotoproducent plaatst de op te sturen foto’s op zijn computer. 2. Fotoproducent vult een geldige klantkey in bij “Klant Key” en drukt op “ophalen” 3. Fotoproducent vult een naam voor de serie in, bij vakje “naam van fotoserie” 4. Fotoproducent Klikt op selecteer.   7> Fotolijst wordt ingevuld in de listbox. Gebruiker controleert de gegevens en drukt op “Upload” | 1. Systeem haalt de klant op via de WebAPI en toont enkele gegevens. 2. Systeem toont een browsedialoog en stelt gebruiker in staat meerdere foto’s te selecteren. 3. Systeem maakt een fotoserie aan via de WebAPI, met de juiste naam en klantverwijzing. De ID wordt opgevraagd. 4. Systeem upload een voor een alle fotos en plaatst ze onder het fotoserie ID. 5. Een melding wordt getoond aan de gebruiker van het succesvol uploaden. |
| **Extensions (Alternative Flow):** | |
| 8a> Er komt geen verbinding tot stand met de WebAPI. Systeem toont een foutmelding en slaat de upload van foto’s over. |  |

# 5.                  Logical View

## 5.1               Overview

Het fotolab bestaat uit de volgende onderdelen, die aan elkaar verbonden zijn.

*Database*

Een database met alle gegevens van de onderliggende objecten. Deze database draait op een lokale PC en wordt in ons geval op een SQLEXPRESS installatie gedraaid. Deze database wordt mee-geïnstalleerd met alle versies van Visual Studio en lijkt goed te voldoen; er zijn geen relevante beperkingen gevonden tot nu toe.

*WebAPI*

Toegang tot de database wordt verzorgd door een WebAPI, deze is de enige die toegang heft tot de onderliggende database. Deze WebAPI is opgezet in ASP.NET. en maakt gebruik van HttpRoutes. Dit is een heel eenvoudig te gebruiken methode om een API te implementeren, zo is gebleken.

*Website*

Er wordt een website geproduceerd in ASP.NET, hierin worden enkele gegevens uit de database getoont zoals bijvoorbeeld een klantenlijst.

*Client applicatie*

We hebben een client applicatie gebouwd op basis van C# en Winforms. Deze applicatie is in staat om een serie foto’s te verzamelen, hier een klant aan te koppelen, en deze foto’s via de WebAPI te uploaden naar de database server. Het resultaat van deze component is een standaard windows applicatie, gebruik makend van .NET.

De foto’s worden lokaal opgeslagen bij de PC die de WebAPI draait. We hebben er niet voor gekozen om deze in de database zelf op te slaan, om ze makkelijker toegankelijk te maken voor een productieapplicatie. Het nadeel van deze keuze is dat er inconsistentie kan ontstaan tussen de database, die vermeldingen heeft van foto’s, en de bestandsstructuur op de WebAPI PC.

## 5.2               Architecturally Significant Design Packages

*WebAPI*

De WebAPI bevat voor iedere tabel in de database een zogenaamde controller en een repository. Iedere Repository heeft een MSSqlDataProvider als member.

De controllers worden aangeroepen door de WebAPI zelf. Het enige wat hiervoor nodig is, is een public functie in de controller, met de markup [HttpGet] of [HttpPost], en een [Route(“url”)]. De WebAPI zal dan bij een aanroep voor de genoemde URL deze functie aanroepen.

De controller heeft een instance van zijn bijbehorende repository. Op deze instance worden de benodigde functies aangeroepen, zoals FindIDForKey.

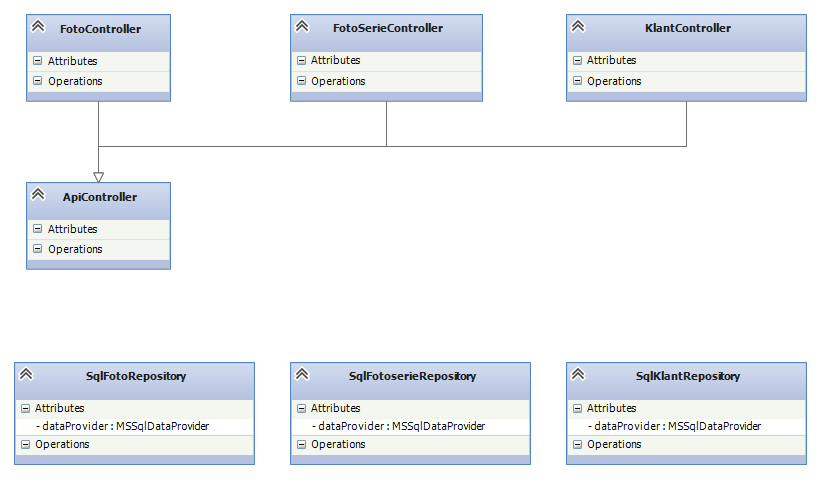
De repository vertaald de functieaanroep naar een SQL statement en maakt parameters aan. Daarna wordt het SQL statement uitgevoerd en de resultaten geretourneerd naar de controller. De controller kan in principe iedere object retourneren. Het object wordt door de WebAPI omgezet naar een http response, in JSON formaat.

Het is daarna aan de client om de http response te parsen en er verder iets mee te doen.

Patterns:

De implementatie van de WebAPI is een typisch voorbeeld van een MVC pattern (Model-view-controller). De controllers (TotoController, FotoSerieController, KlantController, hebben ieder een bijbehorend repository die het ‘model’ zijn binnen dit pattern. De view is echter niet direct zichtbaar, deze is extern geimplementeerd

Binnen de Repositories zelf zien we een Data Gateway pattern. De repository is de enige plek in de code waar ‘kennis’ is van de SQL code waarmee de database wordt aangesproken. De repositories exporteren zelf functies waarmee een client de data kan aanspreken.



Client application

In de client application wordt een actie naar de WebAPI uitgevoerd via de klasse HttpWebRequest. Deze klasse krijgt als parameter een url mee, waarin het verzoek is gecodeerd. De aanroep GetResponse communiceert met de WebAPI, en krijgt een returnwaarde in de vorm van een JSON string.

Deze JSON string wordt via een object van type JavaScriptSerializer naar een lokale klasse van bijvoorbeeld ‘Klant’ omgezet. Dit verloopt via introspectie, de JavaScriptSerializer kan direct de velden uit een JSON string naar members van de klasse omzetten, dit wordt gematched op naam.

*Patterns:*

Binnen de Client application zien we een Action Script. De knop ‘Upload’ triggert een lange reeks code die een stuk business logic implementeerd. Deze business logic bestaat uit het uiteindelijke proces van aanmaken van een fotoserie tot het werkelijk versturen van de foto’s.

# 6.                  Process View

Dit hoofdstuk beschrijft de gebouwde opzet vanuit het gezichtspunt van processen, waarbij een SAD onderscheid maakt tussen lightweight en ‘heavyweight’ processen, waarmee groepen van processen worden bedoeld. Aangezien onze componenten draaien op diverse onderliggende lagen van infrastructuur, zijn het binnen deze context allemaal heavyweight processen. We kunnen daarbij de volgende componenten onderscheiden:

WebAPI.

Draait als een ASP.Net applicatie binnen de context van een IIS webserver. De WebAPI zelf is een enkelvoudig proces, IIS zal iedere WebAPI een eigen procesomgeving aanbieden.

Website

De website draait binnen dezelfde IIS webserver. Deze webserver zal een proces opstarten per binnenkomende http request, waarbij processen hergebruikt worden nadat ze afgehandeld zijn.

Database

De onderliggende SQL database is een eigen proces dat als enige client de WebAPI heeft.

Winforms client

De Winforms client is een enkelvoudig proces, dat aangestuurd wordt door de Windows message pump.

Browser

Een klant die inlogt op de website maakt daarvor gebruik van een browser op zijn eigen computer of tablet. Ook dit is een proces binnen de context van deze applicatie.

# 7.                  Deployment View

Voor de deployment view hebben we de volgende componenten op te zetten. Hierbij de opmerking dat we deze opzet niet hebben gebouwd zoals hier beschreven. Dit hoofdstuk gaat uit van een drietal subsystemen die via een netwerk met elkaar verbonden zijn. Wijzelf hebben alle genoemde componenten locaal op onze eigen PCs gedraaid.

Database:

De database is gebaseerd op een MSSQL engine. Wij gebruiken SQLEXPRESS, die geleverd wordt bij Visual Studio. Daarbij is gebleken dat als er meerdere versies van Visual studio geïnstalleerd zijn, deze niet wordt vervangen en dus de oudste versie is. Dit geeft enkele beperkingen in het gebruik van SQL syntax. Het DDL en insert script zijn daar op aangepast. De SQL server wordt aangesproken via een netwerkverbinding.

*WebAPI.*

De WebAPI is een ASP.NET applicatie en draait daarmee op een Microsoft IIS webserver. Wederom gebruiken wij de standaardversie van Visual Studio. Bij een echte uitrol van deze applicatie zal deze naar een volwaardige Windows Server installatie gaan.

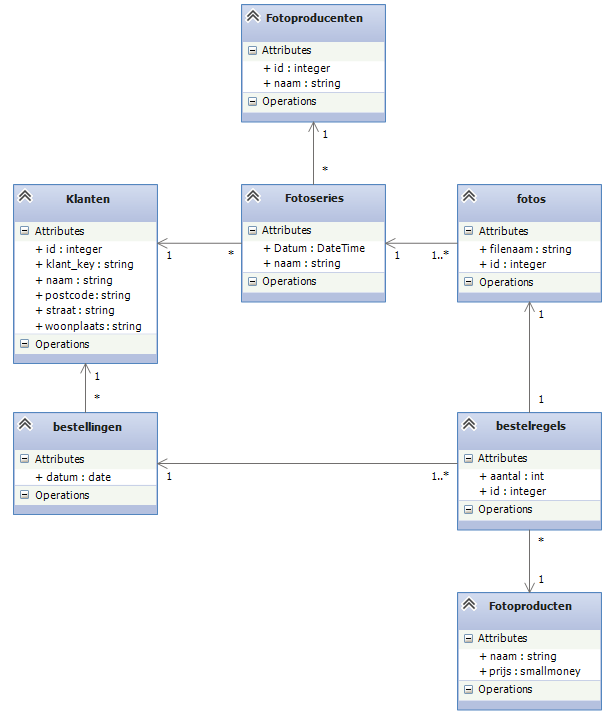
*Website*

De website is ook gebouwd op een ASP.NET webapplicatie. Deze kan op dezelfde IIS webserver gedraaid worden als de WebAPI. Indien de performance onvoldoende is kan deze op een aparte IIS webserver gaan draaien, dit is een keuze die makkelijk kan worden veranderd.

# 8.                  Data View

De casus bestaat uit een WebAPI die tegen een redelijk ingewikkelde database aan praat. We hebben deze database gemodelleerd aan de hand van een domeinmodel, wat we opgebouwd hebben aan de hand van de casus zelf. Waar nodig geven we aan waar we nog toevoegingen of optionele veranderingen zouden willen voorstellen. Na het domeinmodel hebben we een DDL en Insert script gemaakt, het DDL script is hier ter info aan toegevoegd.

Het domeinmodel ziet er als volgt uit:



*Fotoseries:*

Een fotoserie heeft een specifieke naam, en een of meerdere foto’s verwijzen er naar. Een fotoserie heeft een klant, maar die is optioneel en mag leeg zijn. Volgens de casus is een groepsfoto in principe te bestellen door iedereen, alle andere foto’s alleen door de klant waarvoor ze zijn gemaakt. We hebben dit gemodelleerd door de klant vermelding mogelijk null te laten zijn, wat dus betekent dat de fotoserie openbaar is.

*Bestellingen en bestelregel*

Een typische 1 op veel relatie: een bestelling bevat een of meer producten, wat we bestelregels hebben genoemd. Een bestelregel is een bepaalde foto, afgedrukt op een product (poster, foto, mok, etc) met een aantal erbij.

We hebben er voor gekozen om geen afgeleide gegevens in de database op te nemen. We zouden bijvoorbeeld de totaalprijs van een bestelling op kunnen slaan. Dit wordt typisch alleen gedaan als de berekening van deze prijs een duurdere operatie zou zijn dan het opslaan en synchroon houden van dit gegeven. Het is onze inschatting dat dit niet zo is en we de totaalprijs met een SQL query kunnen ophalen indien nodig.

# 9.             Size and Performance

Er is in de opdracht geen sprake van constraints voor grootte danwel performance. Wel stellen we vast dat, door te kiezen voor een aanpak van gescheiden componenten (database, webapi, website) dat we door dit ontwerp deze componenten kunnen scheiden en op gescheiden hardware kunnen laten draaien. In de huidige opzet is dit vanwege praktische redenen niet gedaan.

Een verdere verbetering die mogelijk is op het gebied van performance is het inzetten van een broker tussen de website en client application enerzijds, en de WebAPI anderzijds. Hiermee zou een vorm van load balancing bereikt kunnen worden met nog verdere performance verbetering tot gevolg.

# 10.             Kwaliteit

Drie zaken rond kwaliteit die we hier willen bespreken: uitbreidbaarheid, betrouwbaarheid, en portability.

Zoals eerder besproken is het system door zijn separate componenten goed uit te breiden. Het is mogelijk om separate componenten te vervangen door andere implementaties en/of technieken mocht dit opportuun zijn. De communicatie tussen de componenten wordt vooral uitgevoerd door de WebAPI, deze kan zowel voor zijn clients as voor de database verbinding redelijk makkelijk worden veranderd. De WebAPI is opgezet volgens een model-view-controller opzet, waarbij de view gevormd wordt door de api.

De betrouwbaarheid is wat lastiger te beschrijven: er is geen fallback mechanisme geïmplementeerd aangezien dit geen onderdeel uitmaakt van de casus. De gebruikte technieken zijn industrie standaarden die algemeen bekend zijn. Het is zeker mogelijk om de database engine (MSSQL) te repliceren, ook daar zijn we niet verder op in gegaan.

Portability: de WebAPI is gebouwd met C#, wat een bepaalde vorm van vendor-lockin met zich mee brengt. Dit is een bewuste keuze binnen de casus, maar in theorie is het zonder meer mogelijk de WebAPI te herschrijven naar een andere webtechniek naar keuze (PHP, Node.JS, etc). Hier speelt weer een rol dat de implementatie bestaat uit diverse onafhankeijke en afgeschermde delen.