**Project Fotolab**

**Software Architecture Document**

**Versie 1.0**

**Revisie Historie**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 15-5-2014 | 0.5 | Eerste versie document | Ilja fiers |
| 28-6-2014 | 1.0 | Alle hoofdstukken ingevuld | Ilja Fiers |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[1.       Introduction](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#1.                  Introduction)

[1.1     Purpose](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#1.1               Purpose)

[1.2     Scope](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#1.2               Scope)

[1.3     Definitions, Acronyms and Abbreviations](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#1.3               Definitions, Acronyms and Abbreviations)

[1.4     References](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#1.4               References)

[1.5     Overview](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#1.5               Overview)

[2.       Architectural Representation](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#2.                  Architectural Representation)

[3.       Architectural Goals and Constraints](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#3.                  Architectural Goals and Constraints)

[4.       Use-Case View](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#4.                  Use-Case View)

[4.1     Use-Case Realizations](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#4.1               Use-Case Realizations)

[5.       Logical View](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#5.                  Logical View)

[5.1     Overview](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#5.1               Overview)

[5.2     Architecturally Significant Design Packages](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#5.2               Architecturally Significant Design Packages)

[6.       Process View](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#6.                  Process View)

[7.       Deployment View](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#7.                  Deployment View)

[8.       Implementation View](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#8.                  Implementation View)

[8.1     Overview](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#8.1               Overview)

[8.2     Layers](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#8.2               Layers)

[9.       Data View (optional)](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#9.                  Data View (optional))

[10.     Size and Performance](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#10.             Size and Performance)

[11.      Quality](file:///D:\Development\Rational\RationalUnifiedProcess\webtmpl\templates\a_and_d\rup_sad.htm#11.             Quality)

**Software Architecture Document**

**1.**                  **Introduction**

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**               **Purpose**

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**               **Definitions, Acronyms and Abbreviations**

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

Fotolab: de uitvoerder van bestellingen van fotos, tevens exploitant van dit software product.

**1.4**               **Overview**

In de rest van dit document worde 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**

*[This section describes what software architecture is for the current system, and how it is represented. Of the****Use-Case****,****Logical****,****Process****,****Deployment****, and****Implementation Views****, it enumerates the views that are necessary, and for each view, explains what types of model elements it contains.]*

**3.**                  **Architectural Goals and Constraints**

*[This section describes the software requirements and objectives that have some significant impact on the architecture, for example, safety, security, privacy, use of an off-the-shelf product, portability, distribution, and reuse. It also captures the special constraints that may apply: design and implementation strategy, development tools, team structure, schedule, legacy code, and so on.]*

**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 kant aan te koppelen, en deze fotos 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*

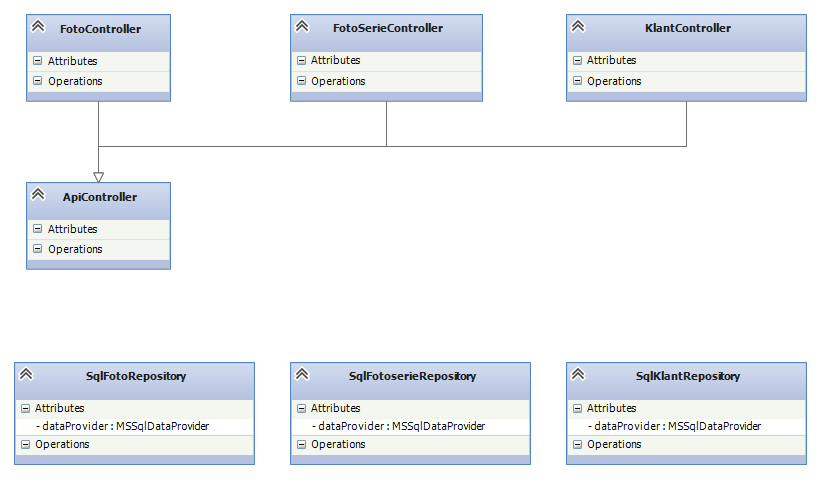
De WebAPI bevat voor iedere tabel in de database een zogenaamde controller en een repository. Iedere Repository heeft een MSSqlDataProvider als member.

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 geretourneert 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.



**6.**                  **Process View**

*[This section describes the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). Organize the section by groups of processes that communicate or interact. Describe the main modes of communication between processes, such as message passing, interrupts, and rendezvous.]*

**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.**                  **Implementation View**

*[This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.]*

**8.1**               **Overview**

*[This subsection names and defines the various layers and their contents, the rules that govern the inclusion to a given layer, and the boundaries between layers. Include a component diagram that shows the relations between layers. ]*

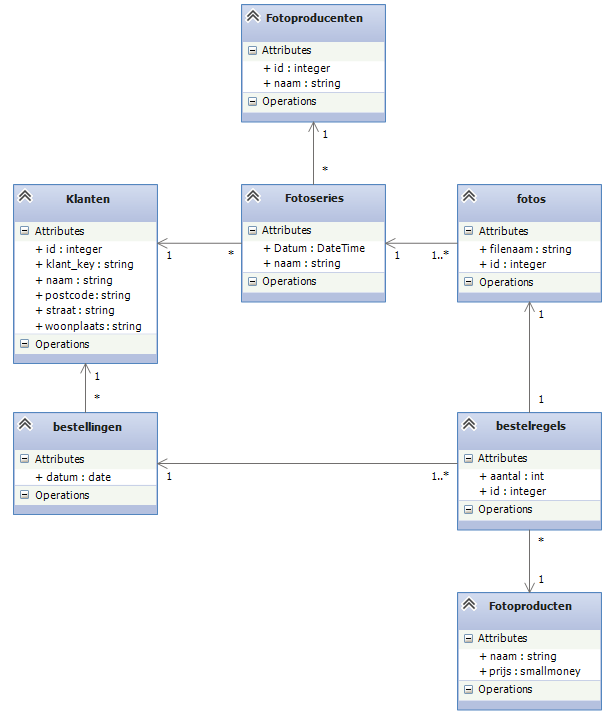
**8.2**               **Layers**

*[For each layer, include a subsection with its name, an enumeration of the subsystems located in the layer, and a component diagram.]*

**9.**                  **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.

**10.**             **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 separate 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.

**11.**             **Quality**

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.