Silicon 5 Codegeneratie
BEP Eindrapport

Mirko Dunnewind
Bachelor Student TU Delft
m.dunnewind@student.tudelft.nl

Ewoud van der Heide
Bachelor Student TU Delft
e.vanderheide@student.tudelft.nl

19 juni 2015

Eindrapport voor het Bachelor Eind Project van de TU Delft. Het bouwen van een codegenerator voor een internetapplicatie framework. Uitgevoerd als stage bij het bedrijf Fenêtre.

Begeleider TU Delft:
Jan Hidders (a.j.h.hidders@tudelft.nl)

Begeleider Fenêtre:
Roger Hendriks (info@fenetre.nl)

Bachelor Project Coördinatoren:
Martha Larson & Felienne Hermans

fenêtre
ONLINE SOLUTIONS

TU Delft
Delft University of Technology

Voorwoord

Dit is het eindrapport voor het Bachelor Eind Project van de TU Delft. Het bouwen van een codegenerator voor een internetapplicatie framework. Uitgevoerd als stage bij het bedrijf Fenêtre. De opdracht is speciaal samengesteld voor de TU Delft en deze hebben wij met veel plezier mogen doen.

Bij deze willen wij graag de opdrachtgever en de medewerkers van Fenêtre bedanken voor de goede samenwerking en de openheid bij het oplossen van problemen. Mede door hen is dit project tot een goed einde gekomen. Daarnaast willen wij de begeleider van de TU Delft bedanken voor het controleren van onze bezigheden en voortgang.

Ewoud van der Heide
Mirko Dunnewind

Inhoudsopgave

1 Samenvatting

Het doel van het project was de ontwikkeling van een codegenerator om snel nieuwe webprojecten op te kunnen starten. Het bedrijf Fenêtre ontwikkelt complexe internetapplicaties met behulp van een eigen framework. De projecten die met dit framework, Silicon 5, worden gemaakt bevatten vaak project-specifieke entiteiten. Voor deze entiteiten zijn bijna altijd onderhoudsschermen nodig. Elk project is uniek, daarom zullen deze schermen voor elk project op een andere manier gebouwd moeten worden. Dit is een tijdrovend proces. Omdat alle schermen volgens dezelfde structuur zijn gemaakt kan het bouwen van de schermen significant versneld worden met een codegenerator.

De uitdaging zat in het zo goed mogelijk benutten van de beschikbare informatie over de tabellen in het framework. Door een reeds bestaande generator is van elke databasetabel een model gemaakt. De combinatie van de metadata uit de database en de beschikbare modellen, biedt veel mogelijkheden. Voornamelijk het analyseren en genereren van de elementen voor relaties tussen verschillende tabellen en views (en dus tussen schermen) is een uitdaging. Samen met het bedrijf zijn op dit gebied veel complexe problemen bediscussieerd en aangepakt.

Het eindproduct is een volledig werkende applicatie die aan bijna alle gestelde eisen voldoet. De applicatie biedt de mogelijkheid om met enkele klikken een serie schermen te genereren op basis van een databasetabel. Deze nieuwe schermen zijn daarna direct te gebruiken in de webapplicatie, inclusief bijkomende functionaliteiten zoals het bewerken van data. Wensen en mogelijkheden die tijdens het project zijn ontdekt, zijn geformuleerd in een lijst van aanbevelingen.

De ontwikkelaars bij Fenêtre zullen deze generator gaan gebruiken bij het opstarten van projecten. Een aantal schermen die standaard in het framework zitten, zijn zelfs al gemaakt met de generator. Daarnaast zal de generator ook met zekerheid nog verder uitgebreid worden met extra functionaliteiten.

2 Inleiding

Om snel betrouwbare internetapplicaties aan haar klanten te kunnen leveren heeft het bedrijf Fenêtre een framework opgezet dat veelgebruikte functies bevat. Voor delen van het framework bestaan codegeneratoren om het ontwikkelproces te versnellen. Door wijzigingen aan het framework zijn de bestaande generatoren ontoereikend en moet er een nieuwe generator ontwikkeld worden.

Het doel van dit Bachelor Eind Project is het ontwikkelen van een codegenerator voor het Silicon framework. Het project is begonnen met een twee weken durende onderzoeksfase die gebruikt is om het probleem in kaart te brengen en een aanbeveling over de te gebruiken technieken te doen. Daarna is er een week gewerkt aan het ontwerp van de applicatie, gevolgd door zeven weken ontwikkelen.

Dit verslag beschrijft het ontwerp en ontwikkelproces van de applicatie. Als eerste wordt het probleem onderzocht en beschreven. Vervolgens worden de betrokken technieken en het framework uitgelegd. Daarna wordt de werkwijze kort toegelicht. Dan volgen het ontwerp en de implementatie waarin respectievelijk het ontwerp van de applicatie en de uitdagingen die in de implementatiefase ontstonden worden beschreven. Na de implementatie volgt de evaluatie met daarin een reactie op de kwaliteitscontrole en een terugblik op de werkwijze. In het hoofdstuk over de analyse wordt gekeken of de gestelde doelen behaald zijn. Tenslotte wordt in de conclusie het verloop van het project bediscussieerd en aanbevelingen gedaan aan de opdrachtgever.

Het onderzoeksrapport, dat het resultaat van de eerste twee weken bevat, is zoveel mogelijk in dit rapport verwerkt, in enkele gevallen zijn verkorte versies in de tekst verwerkt en wordt verwezen naar

de uitgebreide versie uit het onderzoeksrapport dat zich in de appendix bevindt.

3 Probleembeschrijving

De opdrachtbeschrijving en het probleem dat daaraan vast zit, zijn het belangrijkste uitgangspunt voor het hele project. Aan de hand van deze duidelijk beschreven onderdelen, kan aan het einde van het rapport worden geanalyseerd of het probleem is opgelost en dus aan de opdracht is voldaan. Dit is een belangrijke terugkoppeling die onder andere aan de hand van tussenproducten (bijvoorbeeld het programma van eisen) gegeven kan worden. Om context te geven aan de opdracht, zal eerst naar het uitgangspunt van het bedrijf worden gekeken. Daarna zal de opdracht zelf worden beschreven. Als laatste zal het probleem worden uitgewerkt met een onderbouwing van de voordelen als het probleem opgelost wordt.

3.1 Uitgangspunt van het bedrijf

Één van de activiteiten van Fenêtre is het ontwikkelen van op maat gemaakte internetapplicaties. Omdat in deze applicaties veel basisfuncties hergebruikt kunnen worden, is besloten om deze in een framework vast te leggen. Dit framework, Silicon, is al enige tijd in ontwikkeling en kent meerdere versies die gebruik maken van Microsoft .NET technologie. Voor eerdere versies van Silicon zijn code generatie tools gemaakt om bij het opstarten van een nieuw project snel de standaard functionaliteit toe te voegen. De nieuwste versie, Silicon 5, wordt sinds 2014 gebruikt en kan, door het gebruik van nieuwe technologie, niet volledig door de bestaande software gegenereerd worden. De belangrijkste wijziging is het toevoegen van AngularJS. Het is de bedoeling dat er een nieuwe codegenerator gemaakt wordt om projecten op te zetten waarbij gebruik gemaakt wordt van Silicon. Deze nieuwe generator moet op de toekomst voorbereid zijn en zo eenvoudig mogelijk in het gebruik zijn. Aangezien het gaat om een product dat voor softwareontwikkelaars bedoelt is (beide partijen werken in hetzelfde domein), is het niet nodig om het domein verder te onderzoeken dan de uitgangspunten van het bedrijf.

3.2 Opdrachtbeschrijving

Maak een gebruiksvriendelijke codegenerator op basis van de bestaande Silicon 5 applicatie structuur. Het uitgangspunt is een bestaande database die gebruikt wordt om verschillende backend onderhoudsschermen te genereren. Deze schermen hebben een speciaal op maat gemaakte structuur die gebruik maakt van AngularJS. Deze structuur moet gegenereerd worden, wat inhoudt dat er AngularJS onderdelen en schermen worden gemaakt met als doel om direct een werkende basisapplicatie te hebben. Voor het genereren van de database-laag zal een bestaande generator gebruikt worden. Bij het genereren moet gekozen kunnen worden voor verschillende velden, opties, validatie, et cetera. Voor de templates moet een keuze worden gemaakt tussen verschillende voorgedragen en eventueel nieuwe technieken. Ook de soort applicatie moet gekozen en onderbouwd worden.

De originele opdrachtbeschrijving en de onderzoeksvragen voor het onderzoeksrapport zijn te vinden in Appendix A.

3.3 Uitwerking probleem

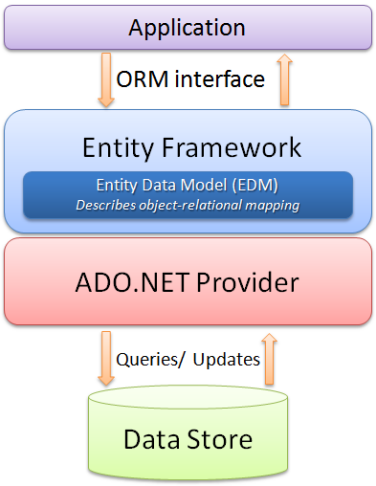
Veel schermen in het Silicon 5 framework bevatten herhalende code. Dit komt onder andere terug in de vele modellen en pagina's met formulieren. Er is door het bedrijf gekozen om deze code statisch te houden. Een alternatief is het dynamisch genereren tijdens runtime. Dit biedt echter veel minder mogelijkheden voor projectspecifieke aanpassingen omdat deze onderdelen dan in de generator aangepast moeten worden. Dit geeft een veel te complexe en niet generieke generator. Het maken van de code voor de onderhoudsschermen is tijdrovend en foutgevoelig. Tijdens het aanpassen van een kopie

kan makkelijk iets vergeten worden, wat voorkomen kan worden door te genereren vanaf een geteste template. Veel van de code heeft een vaste vorm met slechts enkele variabelen, waardoor statische codegeneratie goed mogelijk is. Het is duidelijk dat het maken van de generator in dit project een groot voordeel biedt bij het snel starten van nieuwe projecten.

4 Betrokken technieken

In het Silicon project wordt gebruik gemaakt van een aantal programmeertechnieken en frameworks. De meest relevante technieken worden hieronder beknopt verklaard. Daarnaast bevat Silicon een aantal specifieke uitbreidingen op de in dit hoofdstuk genoemde technieken en frameworks. Deze uitbreidingen zullen in het hoofdstuk over Silicon (hoofdstuk 4.2) toegelicht worden.

Er wordt aangenomen dat de gebruikte programmeertalen bekend zijn bij de lezer. Deze zullen daarom niet verder worden uitgelegd of beschreven. Een verklarende woordenlijst met beknopte uitleg over de gebruikte begrippen is aan het einde van dit rapport bijgevoegd.



Figuur 1: Entity Framework [1]

4.1 Basis Technieken

Binnen het Silicon 5 framework wordt gebruik gemaakt van verscheidene technieken. Voor de server-side kant wordt onder andere gebruik gemaakt van C# met .NET. Daarnaast wordt aan de client-side gebruik gemaakt van Javascript (JS) met AngularJS. In Appendix C worden deze technieken verder uitgewerkt.

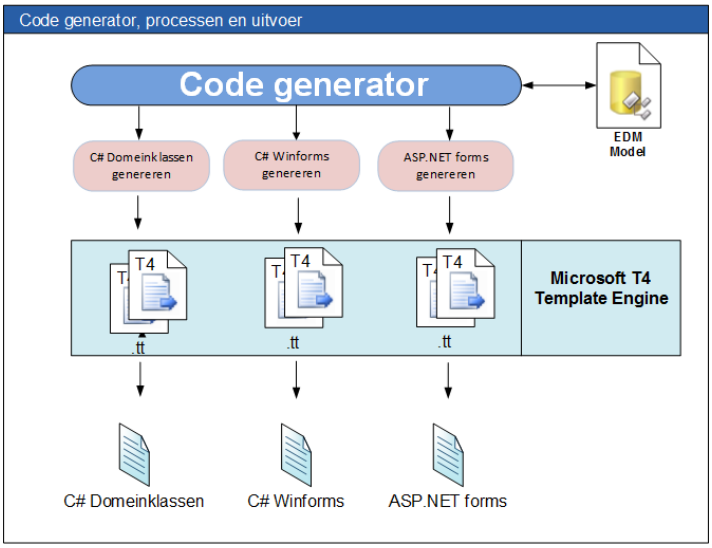
Het .NET framework wordt hier gebruikt om veel standaard operaties toe te voegen. Dit scheelt veel in de ontwikkeltijd. Daarnaast wordt het Entity Framework gebruikt om modellen te mappen naar database tabellen. In een nieuwe versie van het Entity Framework, zullen een aantal van deze opties komen te vervallen. Dit zal gevolgen hebben voor de oude generatoren. De nieuwe generator heeft dan ook als eis om met deze verandering mee te gaan, zodat het klaar is voor de toekomst. Tenslotte is de Silicon webapplicatie gemaakt volgens de MVC-structuur. MVC staat voor Model-View-Controller en is een veel gebruikte techniek. Het Silicon framework hanteert een aangepaste versie van MVC, wat in detail beschreven zal worden in sectie 4.2.2. De AngularJS techniek aan de client-side is een aangepaste variant van wat als standaard wordt beschouwd door de makers. Er wordt voornamelijk gebruik gemaakt van de data bindingen en vele standaard implementaties voor de controls en validatie in formulieren.

4.2 Silicon

Silicon is een framework dat door Fenêtre is ontwikkeld. Dit framework omvat functionaliteiten die in veel van de applicaties gebruikt worden. Een dergelijk framework is nodig omdat er anders elke keer weer dezelfde stukken code geschreven of gekopieerd moeten worden. Silicon is in de loop der jaren ontwikkeld en in een aantal iteraties steeds verder uitgebreid.

4.2.1 Silicon 4

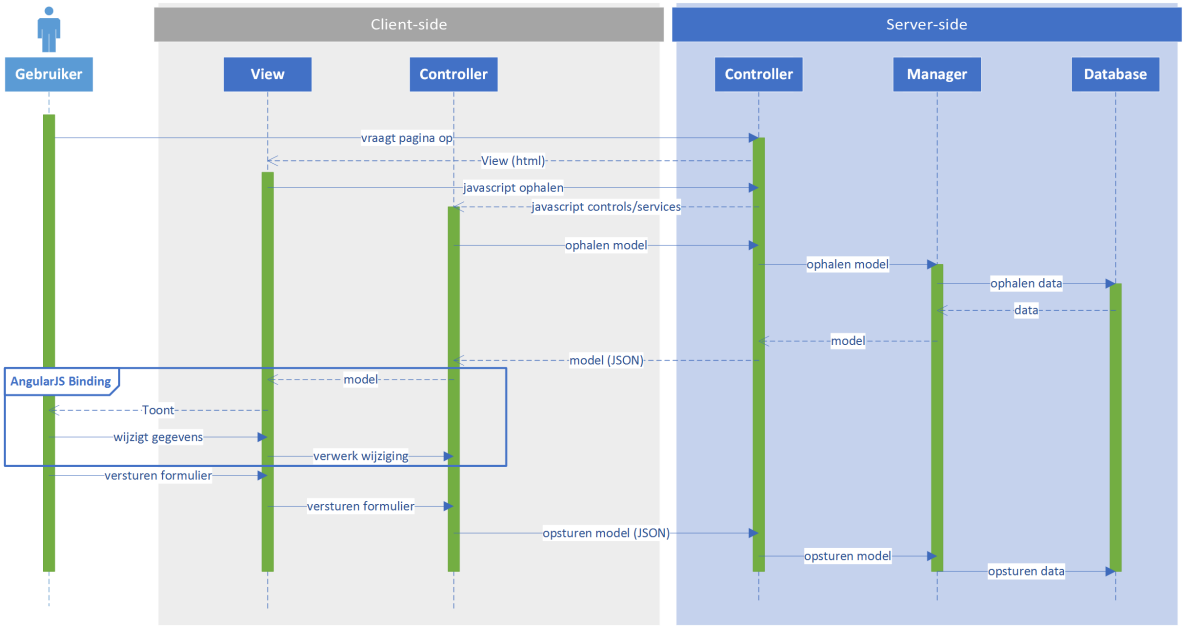
Versie 4 is al een zeer uitgebreide versie van het framework. Het bevat ondersteunende code voor alle aspecten van een applicatie. Enkele voorbeelden hiervan zijn datavalidatie, standaard operaties, userinterface controls en encryptie. Silicon 4 gebruikt ASP.NET Web Forms om de data dynamisch weer te geven. Een weergave van de generator voor Silicon 4 is in Figuur 2 te vinden.



Figuur 2: Silicon 4 generator

4.2.2 Silicon 5

Bij Silicon 5 is ASP.NET Web Forms vervangen door ASP.NET MVC 5, uitgebreid met AngularJS als framework aan de client-side. Omdat AngularJS ook gebruik maakt van een soort MVC model ontstaat hier een dubbele MVC structuur. Silicon 5 wijkt hierdoor af van het standaard MVC model. Hoe het model in Silicon 5 werkt is te zien in Figuur 3. Er bestaat in Silicon 5 een Model, View en Control laag aan de server-side, waarbij de View laag door middel van routing JSON modellen door geeft aan AngularJS aan de client-side. Silicon 5 is door het gebruik van deze web API structuur aan de client-side niet afhankelijk van het platform. De client-side kan naast een webpagina met AngularJS ook uit een losse applicatie of app bestaan. Binnen dit project zal alleen naar webpagina's met AngularJS gekeken worden. Als AngularJS het JSON model ontvangt, gebruikt het een eigen MVC structuur om dit JSON model goed weer te geven en eventuele bewerking toe te laten. Het aangepaste JSON model wordt uiteindelijk weer teruggestuurd naar de server. Het valideren van ingevulde velden gebeurt aan beide kanten, zowel bij de client als de server. Dit zorgt voor een snelle terugkoppeling naar de gebruiker maar biedt ook genoeg veiligheid, het JSON model zou namelijk aan de client-side gemanipuleerd kunnen worden. De extra validatie aan de server-side zorgt dat deze manipulatie geen kwaad kan.



Figuur 3: Sequence diagram dat de stroom in de Silicon 5 MVC structuur weergeeft.

4.2.3 Silicon codegeneratie

Voor Silicon bestaan er op het moment twee generatoren. De eerste generator is in 2009 ontwikkeld (zie Figuur 2) en is volledig gebouwd voor Silicon 4 waardoor hij nu alleen nog voor oude projecten bruikbaar is. Deze generator kan op basis van een bestaande database/entity model en invoer van de gebruiker, onder andere weergave (view), lijstweergave (list) en bewerk (edit) schermen genereren voor entiteiten. Daarnaast kan er ook SQL code gegenereerd worden om bepaalde gebruikers en rollen toegang te geven tot de gemaakte views. Deze generator is het uitgangspunt en de voorloper van de generator die in dit project ontwikkeld zal worden.

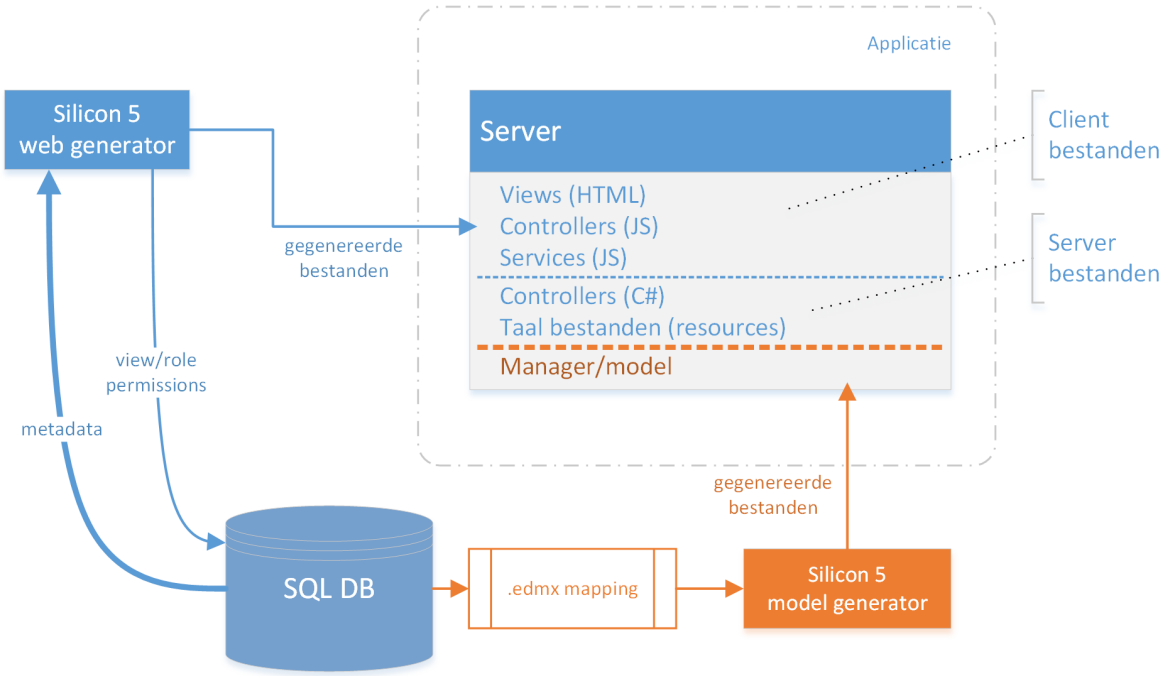
De tweede codegenerator is gebouwd in 2013 en was bedoeld voor Silicon 5 dat toen in ontwikkeling was. Nieuwe projecten worden met Silicon 5 gemaakt. Deze codegenerator maakt manager- en modelklassen aan; dit zijn alleen de C# domeinklassen zoals bij de Silicon 4 generator in Figuur 2 te zien is. Deze manager klassen zijn entiteit gebonden en gebruiken het Entity Framework voor de mapping naar de database. Deze generator hoeft van de opdrachtgever nog niet vervangen te worden en zal naast de te maken generator blijven bestaan. Hoe de twee generatoren naast elkaar gaan werken, is uitgewerkt in Figuur 4. Hierbij is ook aangegeven in welke programmeertaal de uiteindelijke bestanden zullen zijn tijdens het draaien van de applicatie.

Eerdere Silicon generatoren maken gebruik van het Entity Framework om schermen en basisklassen te genereren op basis van databasetabellen. Hiervoor werd gebruik gemaakt van het Entity Data Model zoals opgeslagen in een XML bestand, het .edmx bestand. Dit bestand bevat gestructureerde informatie over de relatie koppeling tussen een applicatie model en een database model.

> An .edmx file is an XML file that defines a conceptual model, a storage model, and the mapping between these models.[2]

In de volgende versie van Entity Framework, versie 7, zal dit .edmx bestand verdwijnen [3] en gaat er door Fenêtre gebruik worden gemaakt van een directe database connectie om op basis van metadata

uit de database de benodigde informatie te verkrijgen.



Figuur 4: Nieuwe generator positionering in Silicon 5

4.3 Generatie techniek

In het onderzoeksrapport is gekeken naar drie verschillende technieken voor het genereren van bestanden op basis van templates. Dit onderzoek heeft gekeken naar Razor, T4 en Roslyn, waarvan de uitwerkingen zijn toegevoegd in Appendix D. Uit dit onderzoek bleek Roslyn al direct af te vallen door de complexiteit en de focus op C#, waardoor het voor de generatie van HTML en JavaScript geen voordelen biedt. Razor en T4 zijn vergelijkbaar met elkaar, waarbij Razor nieuwer is. T4 is daarnaast door een complexere syntax minder overzichtelijk. Ook zal de support van T4 naar verwachting verminderen in de komende jaren. Op basis van deze bevindingen is een aanbeveling voor Razor gedaan. De volledige redeneringen en conclusie is te vinden in Appendix D.

De client heeft de aanbeveling voor Razor geaccepteerd en op basis daarvan is er gestart met de implementatie. Tijdens het gehele project is duidelijk geworden dat Razor een eenvoudige structuur heeft en weinig complicaties heeft. Er zijn echter een klein aantal uitzonderingen waar rekening mee gehouden moet worden bij het maken van templates. Deze zijn in een korte lijst van potentile valkuilen samen te vatten waardoor er makkelijk rekening mee te houden is. Een van de problemen was het escapen van Razorcode. Dit was nodig omdat een aantal van de te genereren bestanden zelf ook Razor bevat. Het is de bedoeling dat de generator het verschil weet tussen Razorcode die gebruikt moet worden om te genereren en code die ook na het genereren moet blijven bestaan. Uiteindelijk bleek Razor een goede aanbeveling te zijn geweest.

```
// Escapen van Razor in Razor
<div id="@("@")Resources.@(Name)">

// Na eerste keer draaien van Razor
```

```
<div id="@Resources.ProductName">
```

4.4 Soort applicatie

Ook voor de soort applicatie is er in het onderzoeksrapport naar drie verschillende vormen gekeken. Dit is de manier van presenteren van de interface waarbij elke manier andere voor en nadelen biedt. Er is gekeken naar een op zichzelf staande applicatie, een Visual Studio extensie en een webapplicatie. Hierbij is de webapplicatie vrijwel direct afgevallen omdat er geen significante voordelen waren ten opzichte van de andere twee applicatie soorten. Uiteindelijk is er advies gegeven voor de Visual Studio extensie omdat na een aantal tests bleek dat er hetzelfde mee bereikt kon worden als een losse applicatie. Daarnaast biedt een Visual Studio extensie extra mogelijkheden voor het uitlezen van de actieve projecten. De volledige redeneringen en conclusie is te vinden in Appendix E.

De client werd door deze aanbeveling aanbeveling verrast omdat eerdere ervaringen met Visual Studio extensies indiceerden dat het ontwikkelen complexer is dan bleek uit de tests die zijn gedaan voor het onderzoek. Toch is de client mee gegaan in de aanbeveling en is er een Visual Studio extensie gemaakt voor de interface van de generator. Dit bleek inderdaad niet moeilijker te zijn dan een losse applicatie en er zijn daarom ook nauwelijks problemen geweest die te wijten waren aan deze keuze. Ook van de voordelen van een extensie is gebruikgemaakt voor het maken van een gebruiksvriendelijke flow. De gegeven aanbeveling bleek zoals getest, inderdaad goed werkbaar te zijn.

5 Werkwijze

In dit hoofdstuk wordt uitgelegd hoe het project beheerd wordt, hoe de kwaliteit gewaarborgd is en hoe het implementeren georganiseerd is. De werkwijze die hier beschreven staat is gebruikt als richtlijn, de werkelijkheid is minder voorspelbaar en in enkele gevallen kan het verstandig zijn om van de geplande werkwijze af te wijken.

5.1 Agile projectmanagement

Er is bij dit project gebruik gemaakt van een op Scrum gebaseerde processtructuur. Omdat er in dit project maar twee ontwikkelaars waren is er op een groot aantal punten afgeweken van de officiële Scrum richtlijnen. De werkwijze die is gebruikt, komt in grote lijnen overeen met de manier van werken binnen het bedrijf. De volgende technieken zijn gebruikt:

- **Backlog, planning en breakdown:** Er is een lijst bijgehouden van alles wat gedaan moest worden. Er is voor gekozen om deze lijst uit te zetten tegen de beschikbare tijd om zo bij te kunnen houden of het proces op schema liep. Zo fungeerde deze lijst tegelijkertijd als backlog, breakdown en planning. Deze lijst was zeer dynamisch aangezien niet alle ontwikkelingen vooraf te voorspellen zijn en er geregeld nieuwe inzichten ontstaan. Ook schuiven de taken soms op doordat activiteiten uitlopen.
- **Sprints:** In de wekelijkse sprints werd een beperkte hoeveelheid taken uitgevoerd, gekozen uit de lijst die werd bijgehouden.
- **Dagelijkse Scrum:** Elke werkdag werd begonnen met een kort overleg van ongeveer vijftien minuten om de voortgang van de sprint te bespreken en te bepalen wat er die dag ging gebeuren.
- **Wekelijkse sprintreview en planning:** Elke week was er een gesprek met de opdrachtgever om de voortgang van het project te bespreken. Hierbij werd de laatste werkende versie van het product bekeken en bediscussieerd. Ook was dit een moment om vooruit te kijken naar de aankomende werkzaamheden.

- **Tijdige escalatie:** Er is afgesproken om bij urgente problemen of vragen eerder kort overleg te hebben dan de wekelijkse sprintreview.
- **Altijd een werkende versie:** Volgens dit principe staat er in het versiebeheersysteem altijd een werkende versie van het product. Hierdoor kan er altijd aan de opdrachtgever getoond worden wat de status is van het ontwikkelproces. Ook voorkomt dit dat er op een later moment veel tijd besteed moet worden aan het integreren van alle onderdelen.

5.2 Kwaliteit en testen

Om de kwaliteit van de applicatie te waarborgen zijn er vooraf afspraken gemaakt over de te hanteren codestijl. Hierbij kon uitgegaan worden van de binnen Fenêtre gehanteerde stijl. Ook is er tijdens het ontwikkelen gebruik gemaakt van een tool, Telerik JustCode, die extra aanwijzingen geeft over de kwaliteit van de geschreven code. Hierdoor wordt al tijdens het ontwikkelen gewaarschuwd voor stijl- en structuurfouten.

Er is bewust geen gebruik gemaakt van testdriven development, waarbij de tests geschreven worden voordat de applicatie gemaakt wordt. Deze keuze is zo gemaakt omdat binnen het project veel gebruik werd gemaakt van onbekende technieken die op experimentele wijze aan het project toegevoegd werden. Door deze manier van ontwikkelen is het van te voren vaak niet duidelijk wat de structuur van de applicatie zal worden met als gevolg dat het van te voren schrijven van tests vaak zinloos is. Er zijn wel unitttests gemaakt om achteraf te kunnen testen en bij toekomstige wijzigingen te kunnen controleren of de bestaande functionaliteit nog werkt (Regressie testen). Om onafhankelijk van complexe functies, database- en filesystemoperaties te kunnen testen is er gebruik gemaakt van zogenaamde shim functies en fake assemblies. Deze functies onderscheppen een aanroep van een bestaande functie en geven de mogelijkheid om specifiek voor de test de werking te overschrijven. Hierdoor kan bijvoorbeeld een exception geforceerd worden of kan voorkomen worden dat het programma daadwerkelijk een bestand probeert aan te maken. Met behulp van shims kan dus voorspelbaar gedrag geforceerd worden in een omgeving die anders van veel variabelen afhankelijk is.

Tijdens het ontwikkelproces zijn regelmatig kleine demonstraties van het programma om te controleren of de geïmplementeerde functionaliteit nog overeenkomt met de wensen van de opdrachtgever. Tegen het einde van het project zullen er enkele gebruikerstests plaatsvinden zodat bepaald kan worden of de gebruikers tevreden zijn met de werking van de applicatie. Deze gebruikerstests worden gedaan door de gebruiker een opdracht te geven, bijvoorbeeld het genereren van een bepaald scherm en vervolgens de handelingen van de gebruiker te analyseren, ook zal de gebruiker achteraf om zijn ervaring gevraagd worden.

5.3 Werkwijze implementeren

Tijdens het ontwikkelen is de applicatie opgedeeld in verschillende componenten. Deze componenten zijn vervolgens verdeeld onder de ontwikkelaars. Elke ontwikkelaar was verantwoordelijk voor een aantal componenten. Er is voor gekozen om geen componenten te wisselen, zodat elke ontwikkelaar expert is over een deel van de applicatie. Er is wel regelmatig overlegd over keuzes die gemaakt werden zodat beide ontwikkelaars op de hoogte waren van de werking van de componenten. Om ervoor te zorgen dat de componenten op elkaar aan bleven sluiten en het integreren soepel zou verlopen is er gebruik gemaakt van modellen. Meer hierover staat in de sectie over het technische ontwerp (hoofdstuk 6.3). Zolang beide componenten van hetzelfde model gebruik maken zal de integratie soepel verlopen. Om de losse componenten tijdens het ontwikkelen te kunnen testen zijn consoleapplicaties gebruikt, deze applicaties kunnen delen van de componenten onafhankelijk van de rest van de applicatie snel testen.

6 Ontwerp

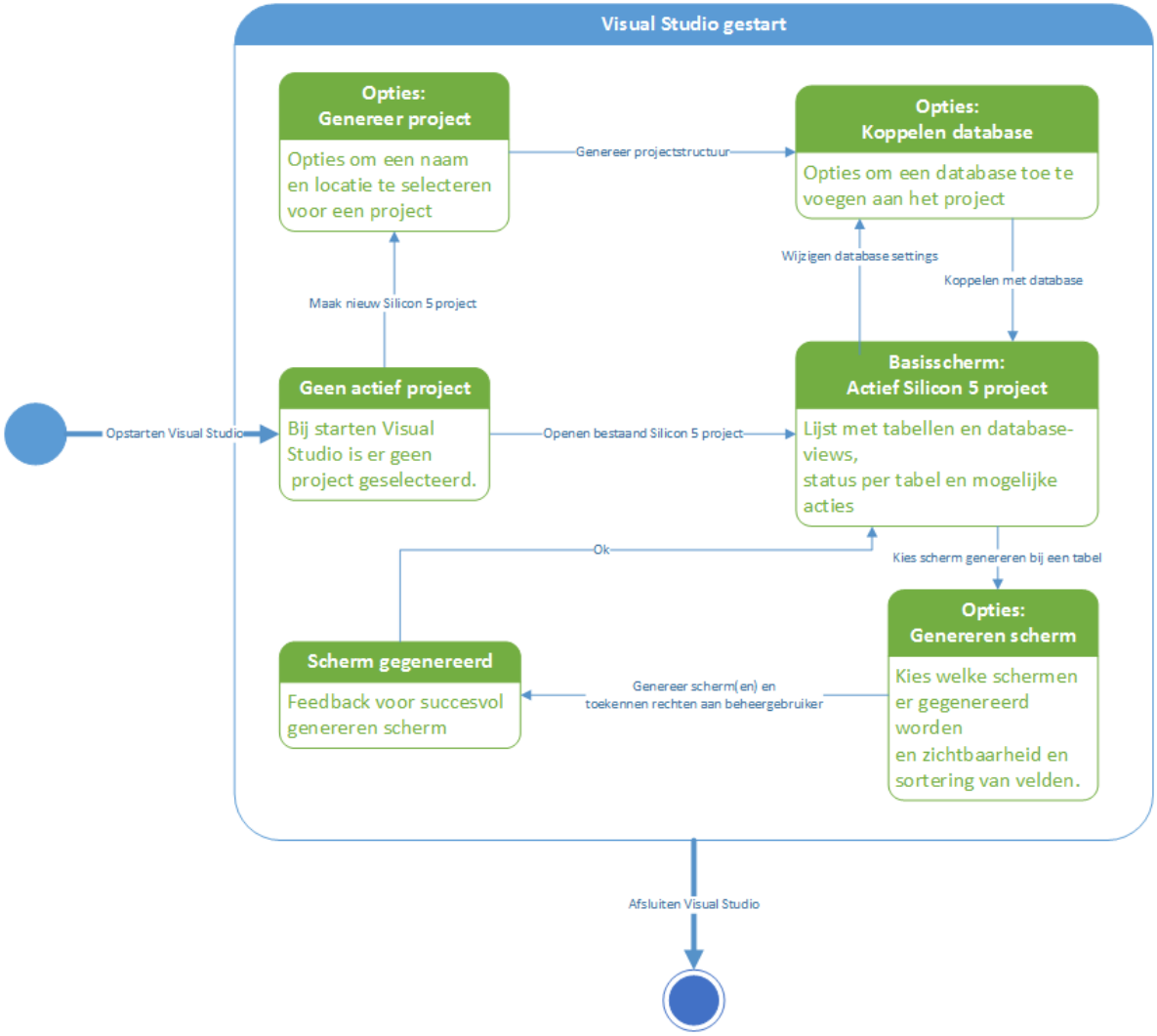
Op basis van het programma van eisen (zie Appendix G) en de keuze voor het type applicatie (zie Appendix E) die in de onderzoeksfase zijn opgesteld is een ontwerp gemaakt. Het ontwerp bestaat uit drie delen: Functioneel ontwerp, grafisch ontwerp en technisch ontwerp. In dit hoofdstuk worden deze afzonderlijke ontwerpen in detail toegelicht. Ook wordt er per deel van het ontwerp extra aandacht gegeven aan keuzes en wijzigingen tijdens het ontwerpproces.

6.1 Functioneel ontwerp

In het functioneel ontwerp worden de functionele eisen die in de onderzoeksfase zijn opgesteld omgezet in concrete werking van de applicatie. Hiervoor zijn een aantal scenario's opgesteld, zogenaamde use-cases. De uitgebreide use-cases zijn te vinden in Appendix H. De volgende scenario's zijn onderscheiden:

- **Projectskelet genereren:** het opzetten van een nieuw webproject dat gebruik maakt van Silicon 5. Het is de bedoeling dat dit project meteen te gebruiken is na het genereren. Dit project bevat standaard mogelijkheden om in te loggen en een (leeg) dashboard.
- **Koppelen database en project:** een (nieuw) webproject koppelen aan een database. Deze database kan vervolgens gebruikt worden door het gegenereerde project en door de generator om een overzicht van tabellen op te halen.
- **Genereren scherm bij databasetabel:** het genereren van schermen op basis van een databasetabel of databaseview. Het is mogelijk om meerdere schermen voor verschillende entiteiten te genereren. Hierbij kan per scherm, per databasekolom een bijbehorend Silicon type en gewenst controle element worden geselecteerd. Meer uitleg over deze types in de typemapping implementatie (hoofdstuk 7.3). Ook zijn er voor elke kolom instellingen te selecteren zoals zichtbaarheid en of een kolom gegenereerd moet worden. Tenslotte moet het mogelijk zijn om bij lijstschermen voor een aantal kolommen zoekvelden toe te voegen. Gekozen instellingen zijn per entiteit tussen de verschillende schermen te kopiëren
- **Toekennen rechten voor gegenereerd scherm:** Na het genereren van een scherm moeten de rechten goed ingesteld worden zodat in ieder geval de hoofdgebruiker de juiste rechten heeft om het nieuwe scherm te zien.

De laatste use-case, het toekennen van rechten, is tijdens het ontwerpen gewijzigd van een expliciete stap naar een impliciete actie. Oorspronkelijk zou de gebruiker de keuze krijgen of hij na het genereren ook rechten wil toekennen, maar de gebruiker zal dit altijd willen doen omdat het scherm anders niet bruikbaar is. In het nieuwe ontwerp worden de rechten automatisch ingesteld na het succesvol genereren van een scherm.

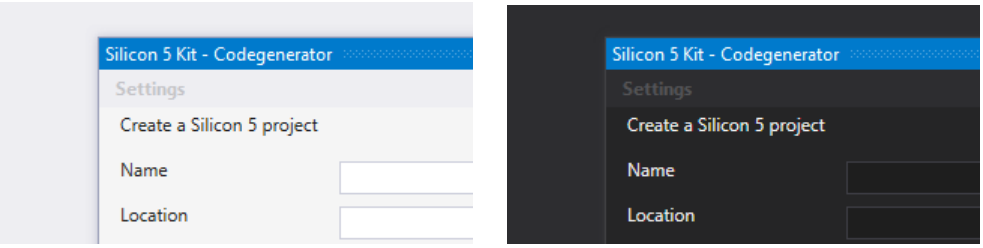


Figuur 5: Status diagram van de basis functionaliteiten van de applicatie

6.2 Grafisch ontwerp

Het gebruiksgemak van de applicatie had een hoge prioriteit voor de opdrachtgever, vanzelfsprekend speelt de grafische interface hier een grote rol bij. Tijdens de onderzoeksfase is besloten om de applicatie te ontwikkelen als Visual Studio extensie, dit heeft beperkte gevolgen voor de grafische interface. Over het algemeen gedraagt een Visual Studio extensie zich als een normale WPF applicatie. Om de gebruiker het gevoel te geven dat hij binnen Visual Studio blijft tijdens het genereren van projecten en schermen is besloten om de stijl van de extensie aan te passen aan het thema van Visual Studio. Voor alle schermen zijn mockups [1] gemaakt, dit zijn eenvoudige tekeningen die de indeling van de schermen tonen.

[1] Voor het tekenen van de mockups is het programma Balsamiq Mockups gebruikt.



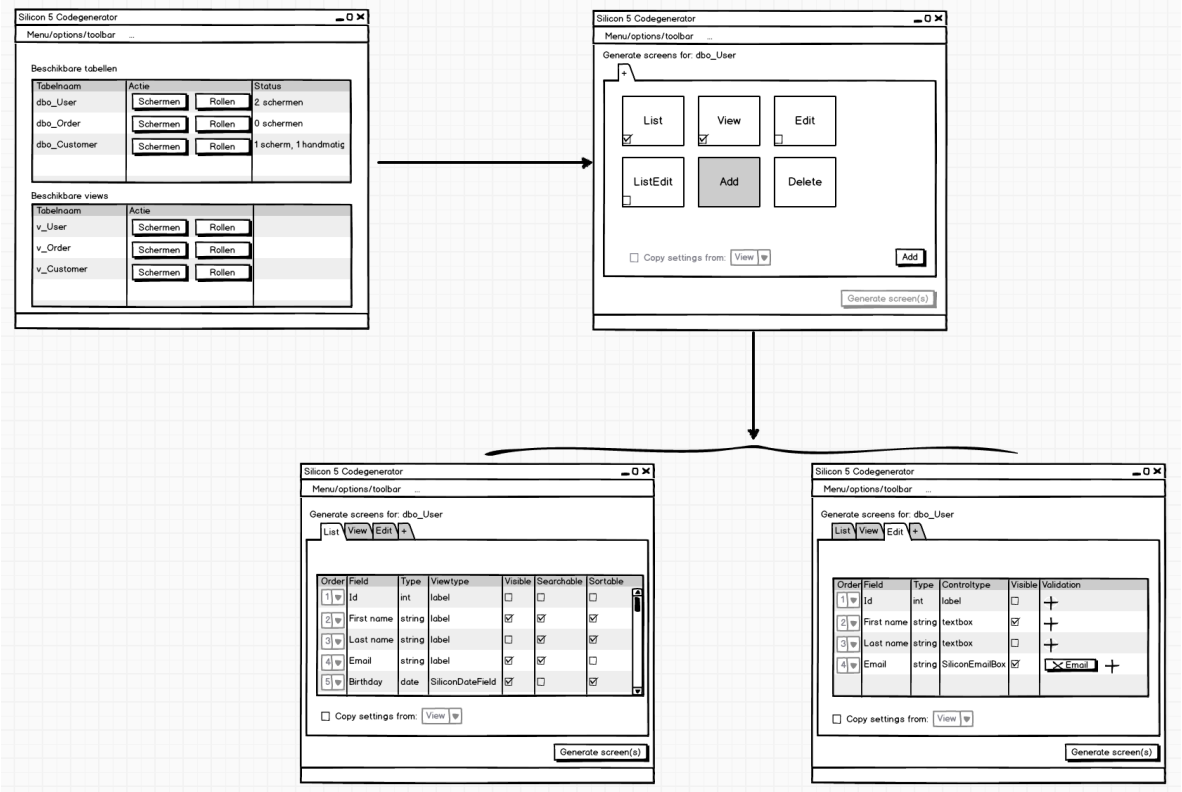
Figuur 6: Visual Studio thema's

De grafische interface is ontworpen na het functionele ontwerp en volgt dezelfde stroom. Als er nog geen project geopend is zal er een scherm getoond worden om een nieuw project te genereren. Bij een bestaand project met verkeerde instellingen, bijvoorbeeld geen werkende database verbinding, zal er een instellingenscherm getoond worden waar dit aangepast kan worden. Als er een project geopend is met correcte instellingen wordt een scherm getoond waarmee schermen bij entiteiten gegenereerd kunnen worden.

Dit laatste scherm heeft de grootste ontwikkeling doorgemaakt bij het ontwerpen. In het eerste ontwerp werd het aantal mogelijke gebruikersacties per scherm zo beperkt mogelijk te houden. Het idee was dat bij minder mogelijke acties het voor de gebruiker overzichtelijker is wat de mogelijke opties zijn. Hierdoor kunnen sneller keuzes gemaakt worden wat het gebruiksgemak vergroot. Een nadeel hiervan is dat minder opties per scherm automatisch meer schermen tot gevolg heeft bij dezelfde hoeveelheid mogelijkheden. Tijdens het wekelijkse overleg met de opdrachtgever bleek dat hij het niet eens was met deze benadering omdat het toevoegen van schermen juist tot meer complexiteit zou leiden, zowel voor de gebruiker als tijdens het ontwikkelen. In overleg werd besloten om een aantal schermen samen te voegen.

Het ging hierbij om de schermen voor entiteit selectie, scherm selectie en de opties per gekozen scherm. In het initiële ontwerp (figuur 7) kreeg de gebruiker eerst een tabel met mogelijke entiteiten waarvoor gegenereerd kan worden. Na de selectie van een entiteit werd een nieuw scherm getoond om schermen te selecteren die gegenereerd gaan worden. Als de selectie van schermen voltooid was werd een nieuw scherm getoond om de instellingen per scherm te wijzigen. Het was ook mogelijk om tussen de laatste twee schermen te wisselen om op een later moment nog nieuwe schermen toe te voegen. Per entiteit kunnen nu de gekozen schermen gegenereerd worden.

In het nieuwe ontwerp (figuur 8) zijn de schermen samengevoegd waardoor de gebruiker meer overzicht heeft in één scherm. Aan de linkerkant wordt een boom getoond met alle entiteiten en per entiteit de reeds gegenereerde schermen en de geselecteerde schermen die gemaakt gaan worden. Als één of meer entiteiten geselecteerd zijn toont de rechterkant welke schermen nog toegevoegd kunnen worden. De schermen die hier getoond worden zijn de doorsnede van de alle mogelijke schermen van de geselecteerde entiteiten, er worden dus alleen schermen getoond die voor alle geselecteerde entiteiten te genereren zijn. Als de selectie bevestigd wordt verschijnen de nieuwe schermen in de boom aan de linkerkant. Voor deze nieuwe schermen kunnen nu afzonderlijk de instellingen voor de kolommen aangepast worden. Dit kan maar bij één scherm tegelijk, de gekozen instellingen kunnen vervolgens wel naar andere schermen van dezelfde entiteit gekopieerd worden. Als de gebruiker klaar is met instellen kunnen alle schermen voor alle entiteiten tegelijkertijd gegenereerd worden.



Figuur 7: Het initiële grafische ontwerp van het genereren van schermen.

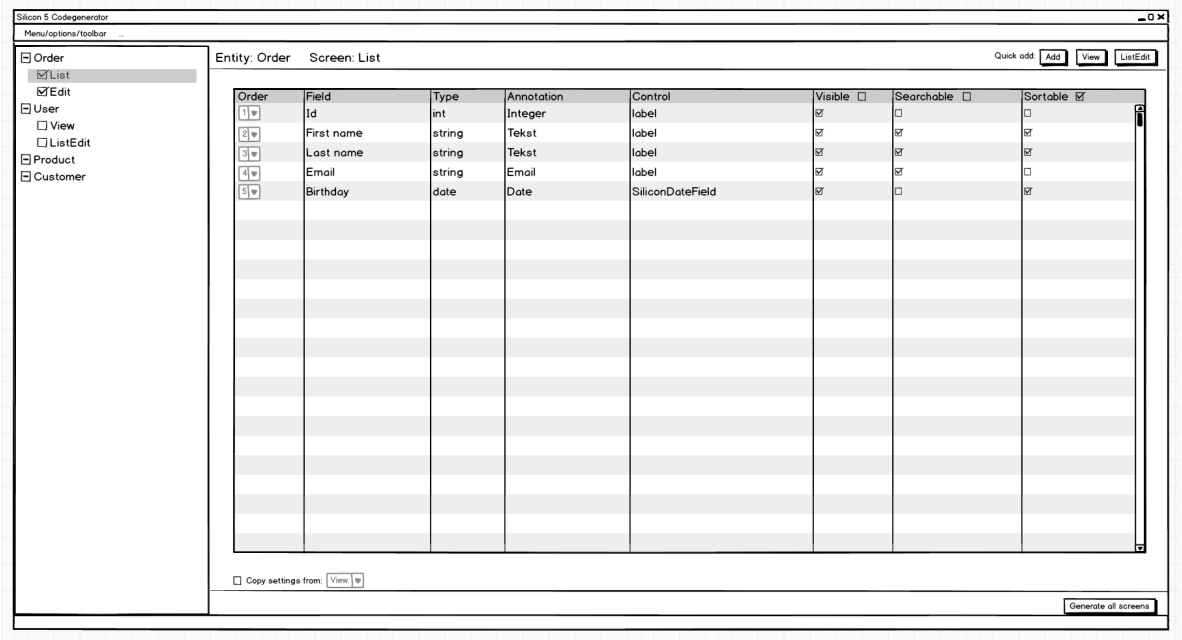
6.3 Technisch ontwerp

Het technische ontwerp is op een abstract niveau gehouden omdat vooraf al bekend was dat door voortschrijdend inzicht er veel wijzigingen in het ontwerp zouden zijn. Het technisch ontwerp bevat vooral de verschillende componenten van de applicatie, de specifieke implementatie werd tijdens het ontwikkelen bepaald. Hieronder staat de onderverdeling van componenten met daarin de belangrijkste onderdelen.

- **Views/GUI:** Een Visual Studio extensie maakt gebruik van WPF voor de weergave, elk scherm bestaat uit een .xaml bestand dat de weergave bepaald en een .cs bestand dat de logica van het scherm bevat. Dit deel van de applicatie is niet verder ontworpen dan de mockups uit het grafische ontwerp, de keuze voor het MVC patroon en het besluit om elk scherm als een los bestand te maken. De keuze om dit niet verder te ontwerpen is gemaakt omdat beide ontwikkelaars geen ervaring hadden met het maken van een WPF applicatie en de structuur en werking tijdens het project ontdekt moesten worden.

- **Helpers:** De helpers bevatten veel ondersteunende functies voor de applicatie. Dit zijn op zichzelf staande, statische methoden die op meerdere plaatsen binnen de applicatie gebruikt worden. De volgende helpers zijn in het ontworpen:

  - **ConfigHelper:** Om gegevens naar het configuratiebestand te schrijven, gegevens uit het configuratiebestand te lezen en als nodig een nieuw configuratiescherm aan te maken is de confighelper ontworpen. Deze helper kan overal in de applicatie eenvoudig gebruikt worden om instellingen aan het configuratie bestand toe te voegen.



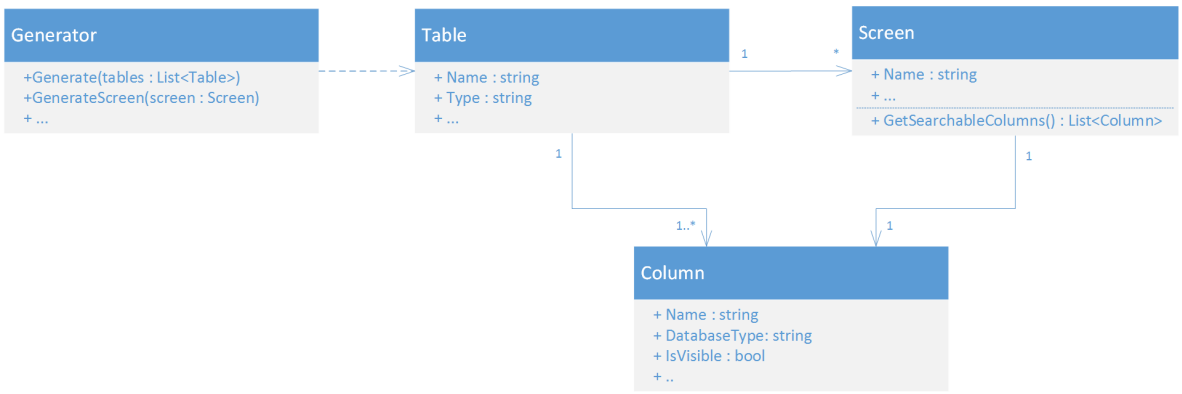
Figuur 8: Het aangepaste grafische ontwerp van het genereren van schermen.

  - **DatabaseHelper:** Deze helper bevat alles wat nodig is om een verbinding te maken met de database en waar nodig queries uit te voeren.
  - **MetaDataHelper:** In de onderzoeksfase werd besloten dat, om de applicatie op de toekomst voorbereid te laten zijn, er geen gebruik gemaakt kan worden van een .edmx bestand om gegevens over de databasetabellen uit de database te halen (zie hoofdstuk 4.2.3 en Appendix C). De MetaDataHelper is ontworpen om gegevens over de tabellen en views uit de database te halen. Deze metadata bevat gegevens over de kolommen zoals: naam, type en constraints (afhankelijkheden zoals relaties).
  - **TypeMappingHelper:** De kolominformatie zoals dit opgehaald wordt door de MetaDataHelper bevat een databasetype, op basis van dit type zijn er verschillende controller elementen mogelijk om de gegevens in de kolom weer te geven. Het koppelen van de juiste controller elementen aan een kolom wordt gedaan door de TypeMappingHelper. Hoe dit precies werkt wordt uitgelegd in sectie 7.3.

- **Classes:** De belangrijkste werking van de applicatie zit in de klassen. De klassen maken gebruik van de helpers en worden aangeroepen vanuit de interface views. De belangrijkste klassen zijn hieronder beschreven. De klassen die de opzet van de te genereren schermen bevatten, zijn een belangrijk onderdeel geweest van de ontwikkeling. Door een goed ontwerp van deze klassen, is. het mogelijk om los aan verschillende onderdelen te werken.

  - **Generator:** De generator bevat alle logica voor het genereren van schermen op basis van templates. De templates worden hier geladen en vervolgens gevuld op basis van het model van een scherm dat in de GUI door de gebruiker is ingesteld.
  - **ProjectCreator:** De ProjectCreator maakt een nieuw project aan op basis van een projecttemplate. Een nieuw project kan zowel in een bestaande solution als in een nieuwe solution toegevoegd worden. In beide gevallen is het mogelijk om aan te geven welk project de data modellen bevat.

- **Table:** Deze klasse representeert een tabel uit de database. Per tabel kunnen meerdere schermen gegenereerd worden. Ook bestaat een tabel uit meerdere kolommen.
- **TableScreen:** Een scherm bij een tabel bevat een set van de kolommen uit de tabel. Deze kolommen zijn kopieën van de originele kolommen die in de tabel zitten. Dit zorgt dat er voor verschillende schermen, verschillende instellingen per kolom kunnen gelden.
- **TableColumn:** Een kolom hoort bij een tabel en een scherm. Een TableColumn bevat een databasetype, mogelijke controller elementen en geselecteerde instellingen zoals zichtbaarheid, sorteerbaarheid en geselecteerde controller element. Deze geselecteerde instellingen bepaald hoe de kolom gegenereerd wordt voor het scherm waar de kolom aan gekoppeld is.



Figuur 9: Klassendiagram van de generator structuur.

7 Implementatie

Op basis van de ontwerpen die zijn gemaakt, is de implementatie gestart. Tijdens de implementatie zijn er een aantal lastige problemen/onderdelen langsgekomen waarbij beslissingen genomen moesten worden om verder te gaan. In dit hoofdstuk komen de belangrijkste moeilijkheden naar voren met een onderbouwing voor de uiteindelijke implementatie. Een overzicht van de uiteindelijke implementatie van alle klassen, helpers en views is te vinden in Appendix K.

7.1 Configuratie instellingen

Omdat er vaak meerdere ontwikkelaars tegelijkertijd aan een project werken moeten de instellingen van het project waarvoor gegenereerd wordt, gedeeld worden. Hiervoor is het "generator.config"bestand ontworpen. In dit configuratie bestand worden de gekozen instellingen zoals het bijbehorende model project, databaseconnectionstring opgeslagen. Het bestand maakt deel uit van de projectstructuur en wordt met het versiebeheer systeem gedeeld tussen de ontwikkelaars. Aanvankelijk was het de bedoeling om een lijst van reeds gegenereerde schermen ook in het configuratie bestand op te slaan. Tijdens het ontwikkelen bleek echter dat het eenvoudig is om de gegeneerde schermen direct uit de project structuur in te lezen. Dit heeft het voordeel dat ook handmatig aangemaakte schermen niet opnieuw gegenereerd kunnen worden zonder overtuigende handmatige acties.

7.2 View model, databinding

Aanvankelijk was het de bedoeling om het tonen van de data op het scherm volgens het MVC patroon (zie ook appendix C) te implementeren. De data wordt opgeslagen in het model, de view bepaald de

weergave en de controller zorgt voor de koppeling tussen beiden. Hierbij zouden vanuit een controller laag de waarden direct gezet worden in het scherm:

```
Label.Text = Model.Property;
```

En andersom bij een wijziging in de gui, op het moment dat er op een knop gedrukt wordt, moet de waarde weer terug gezet worden in het model:

```
Model.Property = Input.Text;
```

Het bleek echter complex om dit te implementeren omdat bij elke wijziging bewust de methodes aangeroepen moeten worden die de juiste waarden instellen. Daarom werd besloten om gebruik te gaan maken van databinding om de juiste waarden te tonen. Bij databinding wordt het model direct in de gui getoond, zonder tussenkomst van een controller. Wijzigingen in het model zijn dan direct zichtbaar op het scherm en wijzigingen op het scherm worden meteen, na eventuele validatie, in het model opgeslagen.

Een gevolg hiervan is dat er voor bepaalde gevallen extra properties aan het model toegevoegd moeten worden om een status op het scherm weer te geven, bijvoorbeeld of een element geselecteerd is. Omdat dergelijke properties niets te maken hebben met de data die door het model wordt gerepresenteerd is het verwarrend om dit in hetzelfde model weer te geven. Om dit op te lossen is er voor elk model dat op een scherm weergegeven moet worden een viewmodel toegevoegd. Een viewmodel bevat het originele model en breidt dit uit met methoden en properties die specifiek voor de weergave bedoeld zijn. Deze structuur staat bekend als het MVVM (Model-View-ViewModel) patroon. [4]

7.3 Typemapping

Het mappen van database types naar (Silicon) types en controle elementen is een belangrijk deel van de generator. Met behulp van deze mapping moet het koppelen van verschillende types een stuk makkelijker en dynamischer worden. Bij deze type mapping wordt met types een indicatie van wat er in het database veld staat bedoelt. Een database type met tekst kan bijvoorbeeld type email zijn, of gewoon een standaard type tekst. Per type is er dan een keuze uit controller elementen. Dit kan bijvoorbeeld label, tekstbox of email zijn. Van elk controller element is er ook een readonly variant. Deze variant genereert extra opties bij labels, bijvoorbeeld een preview geven van een URL.

In deze mapping is het belangrijk dat de structuur makkelijk, algemeen en herbruikbaar blijft. Dit geldt ook voor het gebruik van benamingen voor onder andere de controle elementen. Fenêtre heeft andere tools die op den duur van deze zelfde mapping gebruik moeten gaan maken. Op deze manier kan het bedrijf de integratie mogelijkheden groot houden. Een goed doordachte opzet is dan ook een vereiste. In figuur 10 is een visuele weergave te zien van de typemapping in Excel. Deze visuele weergave wordt geconverteerd naar een JSON bestand dat daarna ingelezen kan worden in de generator. Dit JSON bestand bevat ook extra mappings voor onder andere database types naar search operators. Daarnaast is de JSON structuur een vereiste aangezien dit goede integratie in andere software biedt.
Per database type en per type is het mogelijk om een standaard type of controller elementen in te stellen. Dit maakt het instellen van de generator een stuk sneller aangezien de meeste keuze mogelijkheden standaard al goed staan, wat de gebruiksvriendelijkheid bevorderd. Het correct instellen van een grote hoeveelheid schermen kost door goede standaardinstellingen veel minder tijd.

In de generator wordt alleen nog maar gebruik gemaakt van het uiteindelijk gekozen controller element. Per element bestaat er een bestand met een Razor template dat gebruikt wordt voor het genereren. Het toevoegen van extra handmatige en type afhankelijke validatie valt buiten de scope van het project, maar het is mogelijk dit vrij simpel te integreren door het uitbreiden van de typemapping modellen/JSON. Op die manier kunnen de Razor templates direct gebruik maken van deze extra validatie keuzes. Ook zijn template bestanden makkelijk toe te voegen, wat uitbreiding eenvoudig maakt.

		available	x	x
		##controls##	Textbox	Label
databaseType	Default	type		
nvarchar		PhoneNL		
		Email		
		Emails		
	x	Text		

Figuur 10: Deel van de typemapping visualisatie in Excel

7.4 Foreign Key relatie generatie

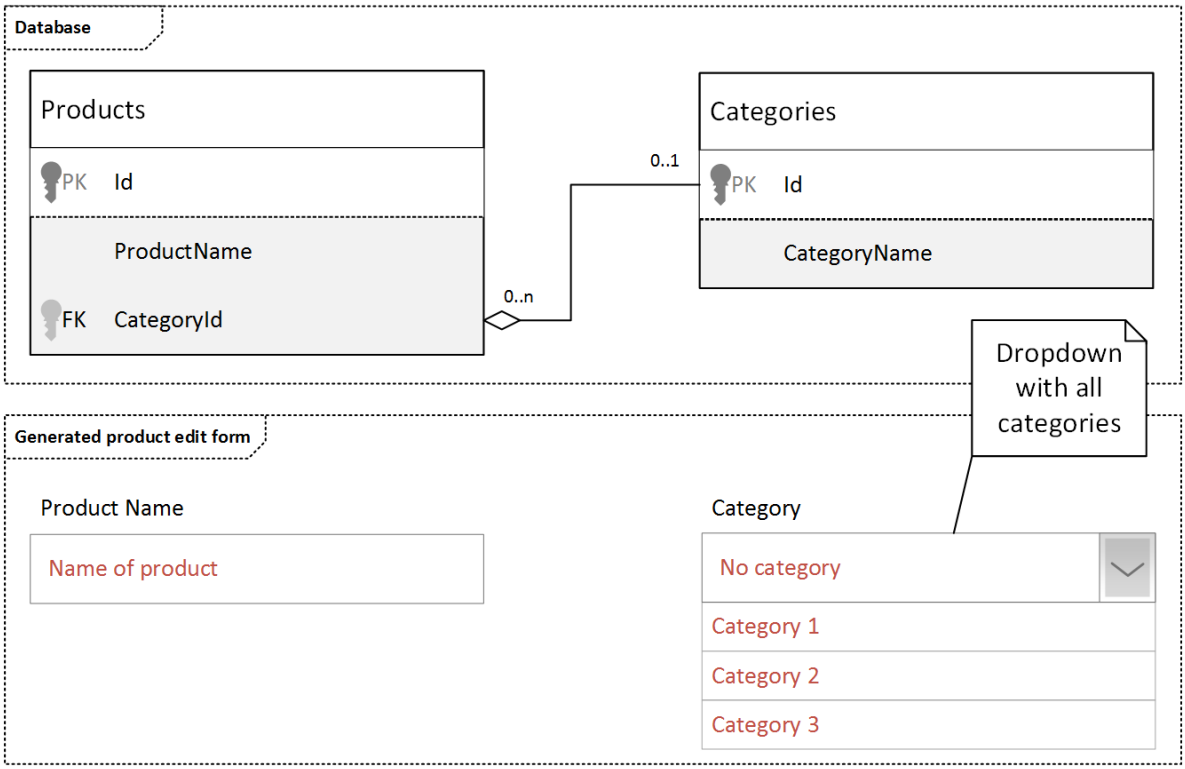
Nagenoeg alle database tabellen bevatten Foreign Keys (FK) of hebben een relatie naar een Foreign Key in een andere tabel. Foreign keys geven dan ook de relaties naar (Primary Keys van) andere tabellen weer. Deze keys zijn uitermate belangrijk voor de meeste type applicaties omdat ze structuur in de data bieden die nodig is voor de meeste applicaties. Ook binnen de generator is dit een belangrijk deel omdat voor deze relaties extra elementen gegenereerd moeten worden.

Samen met de ontwikkelaars van het Silicon framework is nagedacht over hoe deze relaties zo soepel en dynamisch mogelijk verwerkt kunnen worden in het framework. Hierbij is ook beveiliging een cruciaal onderdeel. Gebruikers die alleen rechten hebben om informatie te bekijken, mogen op de achtergrond geen toegang hebben tot methodes die de lijst met mogelijke relaties terug geeft. Neem als voorbeeld het kiezen van een klant voor een bestelling. Als de klant zijn eigen bestelling bekijkt, moet hij niet bij de hele lijst van alle klanten kunnen.

Het genereren van alle code voor Foreign Keys brengt een aantal moeilijkheden met zich mee. Het genereren van het controller element is hierbij geen probleem aangezien dit een stuk standaard Razor-code is. Voor de edit en create schermen is er een lijst nodig van beschikbare elementen in de tabel waar de Foreign Key naar verwijst (zie figuur 11 voor een visualisatie). Gebruikers moeten uit een leesbare lijst kunnen kiezen zonder dat zij de achterliggende identifier (id) zien. Om dit op te lossen worden er in de server-side controller een aantal speciale acties toegevoegd die deze lijsten kunnen ophalen. Echter zijn deze acties niet beschikbaar voor gebruikers zonder edit- of create-rechten. Zij zijn dan namelijk niet bevoegd om de gehele lijst op te halen.

Voor de readonly schermen is er een descriptor nodig voor de Foreign Key velden in een tabel. Een descriptor is een kolom of set van kolommen die de rij van een tabel beschrijft. In het geval van een Foreign Key is dit een rij in een andere tabel. In eerste instantie was dit nog niet aanwezig in het framework. Naar aanleiding van deze missende functionaliteit zijn verschillende implementaties geprobeerd en is uiteindelijk besloten de descriptor in het datamodel op te halen. Dit data model wordt gegenereerd door een andere generator en valt buiten de scope van dit project.

Het is nodig geweest om de Foreign Keys als constraints in het datamodel op te nemen. Tijdens het uitlezen van de database worden ook per kolom de constraints opgehaald. Zodra alle tabellen in het model zijn geladen, worden de referenties van alle Foreign Key constraints goed gezet. Er is dan een verwijzing naar een andere tabel gemaakt waardoor de generator alle informatie van die tabel en zijn kolommen kan ophalen. Doordat het model de relaties tussen de tabellen weet, veranderd het automatisch het databasetype van de kolommen die dienst doen als Foreign Key. Zij krijgen het virtuele type "relation". Dit type wordt virtueel genoemd omdat dit een niet bestaand databasetype is. Via dit type kan de eerder genoemde type mapping naar de verschillende control elementen eenvoudig worden



Figuur 11: Voorbeeld van de generatie van een edit scherm voor een product

toegepast op Foreign Key kolommen. Dit maakt het creëren van nieuwe controller elementen voor Foreign Keys eenvoudig en dynamisch wat goed is voor de onderhoudbaarheid.

Ook voor de grafische interface biedt dit model veel mogelijkheden voor uitbreiding. Als naar een andere tabel wordt verwezen, is het gebruikelijk om een zogenaamde "descriptor"van die andere tabel te bepalen. Door gebruik te maken van het al gemaakte model, kan eenvoudig een descriptor veld met kolom selectie worden toegevoegd in de interface. Aangezien de descriptor wordt opgeslagen in het model, valt dit nu buiten de scope van het project. Echter is dit wel relevant voor de toekomst want de opdrachtgever heeft aangegeven dat de beide generator op een gegeven moment samengevoegd zullen worden.

7.5 Performance RazorEngine

Tijdens het ontwikkelen van het gedeelte van de applicatie dat verantwoordelijk is voor het genereren van schermen op basis van Razor templates werd gebruik gemaakt van een losse console applicatie. Dit werd gedaan zodat niet elke keer de experimentele instantie van Visual Studio opgestart hoefde te worden (zie hoofdstuk 5.3 voor deze werkwijze keuze). De generator werkte uitstekend onder deze omstandigheden, toen de generator geïntegreerd werd met de rest van de applicatie bleek echter dat de snelheid van het genereren sterk terugliep. Bij de consoleapplicatie duurde het genereren minder dan een seconde per template, de Visual Studio extensie deed er dertig seconden over. Omdat een scherm uit een groot aantal templates bestaat liep de tijd voor het genereren van een scherm al snel op tot meer dan 5 minuten minuten. Hoewel dit nog steeds een aanzienlijke tijdswinst opleveren ten opzichte van de tijd die het kost voor een ontwikkelaar om de schermen zonder generator te maken, werden deze tijden toch als onacceptabel beschouwd. Daarnaast zou deze tijd nog verder oplopen bij

het maken van meerdere schermen.

Uit analyse van de generatiecode bleek dat het compileren van de templates voor de vertraging zorgde. Dit compileren gebeurt elke keer als een template voor het eerst gebruikt wordt tijdens het uitvoeren van de applicatie. De template wordt dan gecompileerd en opgeslagen in de cache zodat deze gebruikt kan worden als een model op de template wordt toegepast.

Er werd in de documentatie van RazorEngine gezocht naar verschillende mogelijke oplossingen van dit probleem. Geen van de gevonden oplossingen loste het probleem echter op. Uiteindelijk zijn de ontwikkelaars van RazorEngine om hulp gevraagd[2]. Uit de reacties bleek dat de RazorEngine bij het compileren van een template door alle assemblies die in het project gerefereerd worden itereert om te zorgen dat ze te gebruiken zijn bij het toepassen van de template. Door slechts een paar assemblies van te voren te selecteren die gebruikt gaan worden, was de tijd die het compileren duurt terug te brengen naar minder dan een seconde per template.

7.6 Generator

De generator is het onderdeel van de implementatie dat de uiteindelijke vertaalslag van het model van tabellen naar bruikbare code doet. Deze lastige stap is tijdens het hele project voorspoediger verlopen dan verwacht. Wel is de klasse die deze functionaliteit implementeert meerdere malen herschreven om de werking en kwaliteit te verbeteren. Door het goed doordachte model van tabellen in zowel de ontwerp- als ontwikkelfase, bleef de integratie en het testen op elk moment mogelijk.

Bij het daadwerkelijk genereren van een aantal tabellen en schermen, zal de generator deze per scherm bekijken. Voor elk scherm dat gegenereerd moet worden wordt onder andere gecontroleerd of: het scherm al bestaat, de modellen correct zijn, alle templatebestanden bestaan, alle templatebestanden gecompileerd kunnen worden. Wanneer aan één van deze condities niet voldaan wordt, zal het scherm niet gegenereerd worden. Achteraf zal dan een foutmelding zichtbaar zijn waarna de ontwikkelaar de templates en/of bestaande bestanden kan aanpassen en hetzelfde scherm opnieuw kan genereren.

Veel van de moeilijkheden in dit onderdeel zitten in het goed uitdenken van de meest logisch structuur. Er worden namelijk veel templates met elkaar gecombineerd waardoor de implementatie met Razor niet altijd voor de hand ligt. Daarom is er op sommige punten gebruik gemaakt van wat simpelere zoek en vervang opdrachten omdat een Razor implementatie de complexiteit te sterk zou verhogen. Het was hier voornamelijk belangrijk om alle stappen zorgvuldig te doorlopen zodat elke bewerking in het template en de generatie een bewuste keuze is geweest.

Een andere moeilijkheid en tevens eis was het aanpassen van al gegenereerde bestanden. In dit framework zijn er namelijk een aantal bestanden die gemaakt worden per entiteit en niet per scherm. Dit betekent dat, wanneer de schermen in meerdere keren worden gegenereerd, de generator al bestaande bestanden op de correcte manier moet bijwerken. Er is hier gekozen om zogenaamde "regions"te gebruiken. Regions zijn commentaar regels die de editor laten weten dat een stuk code bij elkaar hoort en inklapbaar is. Door de namen van de regions voor de verschillende schermen te standaardiseren, was het mogelijk om de generator de correcte region te laten vinden en herschrijven.

7.7 Visual Studio Extensie

Zoals in hoofdstuk 4.4 is beschreven, is de keuze om een Visual Studio extensie te maken vooral gebaseerd op de voordelen die het biedt voor het gebruiksgemak. Door de opdrachtgever werd echter al gewaarschuwd dat de complexe omgeving van Visual Studio voor extra moeilijkheden kan zorgen.

[2] `https://github.com/Antaris/RazorEngine/issues/286`

Deze problemen manifesteerden zich vooral in de grafische interface.

Aanvankelijk leek het ingewikkeld om het thema van Visual Studio over te nemen in de codegenerator, maar na extra onderzoek bleek dit niet moeilijk door van impliciete styling gebruik te maken. Bij impliciete styling wordt een stijl op alle elementen van een bepaald type toegepast, in plaats van expliciet voor elk element dat gebruikt wordt. Door impliciet de Visual Studio stijl in te stellen op een aantal basiselementen werd de stijl van de hele applicatie in één keer aangepast.

Op het gebied van de grafische interface was er ook een meevaller. Het bleek dat het Telerik UI WPF controls pakket, een product dat door Fenêtre al aangeschaft was, ook binnen een Visual Studio extensie werkt. Hierdoor kwam een groot aantal geavanceerde WPF elementen beschikbaar voor de ontwikkelaars.

Een idee was om de tabel met beschikbare kolommen te kunnen reorganiseren door rijen boven en onder elkaar te kunnen slepen. In een gewone WPF applicatie is dit een niet heel ingewikkelde uitbreiding. In een Visual Studio extensie bleek dit echter zeer complex te zijn. De reden hiervoor is dat Visual Studio zelf het slepen van elementen probeert te regelen, waardoor dit niet op de manier gebeurt zoals het in de applicatie gewenst is. Dit probleem is uiteindelijk niet opgelost, maar op een andere manier uitgevoerd. Het is nu mogelijk om kolommen te reorganiseren door het gewenste rijnummer in een tekstveld in te vullen.

Het grootste voordeel van een Visual Studio extensie bleek te zijn dat het binnen de Visual Studio omgeving mogelijk is om allerlei gegevens van de openstaande solution en projecten op te vragen. Ook biedt Visual Studio veel standaard operaties op projecten aan, zoals het maken van nieuwe projecten en het toevoegen van elementen aan bestaande projecten.

8 Evaluatie

Zoals in de werkwijze beschreven staat, is er door het hele project heen aan de kwaliteit van de code gedacht. Het logisch plaatsen van functionaliteit, kort houden van methodes en toevoegen van commentaar is constant gedaan. Twee keer tijdens het project is de code naar SIG (Software Improvement Group) gestuurd voor evaluatie. Met de resultaten is gekeken hoe de implementatie beter gemaakt kan worden om onder andere de onderhoudbaarheid hoog te houden.

8.1 SIG evaluatie 1

Door het constant bijhouden van de code kwaliteit, was de voorbereiding voor de eerste SIG deadline niet veel werk. Het resultaat van net vier sterren (van de vijf) geeft dan ook aan dat het niet voor niets is geweest. Wel was er commentaar op de volgende punten: Duplicatie, Unit Size en Module Coupling. De volledige evaluatie van SIG is te vinden in Appendix I.

Duplicatie zat voornamelijk in grote stukken SQL queries voor het ophalen van de metadata voor het tabellen model. Dit is opgelost door het selectie deel van de query in een aparte variabele te plaatsen. In de rest van de code is nagenoeg geen duplicatie gevonden.

Unit Size was een probleem bij de zeer specifieke helpers (bijv. TypeMappingHelper) die onder andere een csv bestand uitlezen en in een (JSON) model converteren. Deze helper is grondig geanalyseerd en versimpeld op verscheidene punten. Daarna zijn, waar mogelijk, de overgebleven lange methodes opgesplitst in kleinere methodes die beter te begrijpen zijn. Naast deze helper zijn ook de andere helpers gecontroleerd en zo nodig verbeterd op dit punt.

Bij Module Coupling werd vooral verwezen naar de ProjectHelper en TemplateHelper. Beide bevatten methodes voor het lezen, checken en schrijven van bestanden. Er werd aangeraden om hier een aparte helper voor te maken (FileHelper bijvoorbeeld). Echter zijn wij het niet eens met de aangeraden acties. De file methodes in zowel de Project- als de TemplateHelper maken al gebruik van de ingebouwde bestands functionaliteiten. Het enige wat er hier wordt toegevoegd, is het absolute pad naar of het Project of de Template. Dit heeft als duidelijk voordeel dat de gebruikers van deze helpers na het initialiseren alleen nog maar relatieve paden hoeven op te geven. Er zal dan automatisch verwezen worden naar de correcte Project of Template mappen/bestanden. Het toevoegen van een extra helper voor deze bestands methodes heeft geen toegevoegde waarde omdat de methodes die de relatieve paden accepteren in de Project en Template helper moeten blijven bestaan. Tevens is deze methode ook beter onderhoudbaar op het moment dat het absolute pad of de logica er achter veranderd. Hierom is er besloten dat het voor onderhoudbaarheid en werkbaarheid beter is om de helpers te laten zoals ze zijn.

8.2 SIG evaluatie 2

Net voor het einde van het project heeft de tweede SIG evaluatie plaatsgevonden. Er is hierbij voornamelijk vergeleken met de eerste evaluatie. Zij zijn hierbij tot de conclusie gekomen dat het code volume is toegenomen met 30% terwijl de onderhoudbaarheid ook licht is toegenomen. Alleen de Module Coupling is door de hoeveelheid helper classes slechter geworden. De volledige evaluatie van SIG is te vinden in Appendix I.

Sinds de eerste evaluatie is de functionaliteit van de software meer dan verdubbeld. Het licht toenemen van de code met maar 30% duidt er naar ons idee dan ook op dat de implementatie van de nieuwe functionaliteiten goed gebruik heeft gemaakt van bestaande code en niet onnodig complex is. Dit wordt versterkt door de toename van de onderhoudbaarheid volgens de SIG evaluatie.

Het commentaar over de Module Coupling komt overeen met wat is geschreven bij de eerste evaluatie. Er is bewust gekozen om de aanmerking niet mee te nemen in de verbeteringen. Dit is dan ook terug te zien in deze evaluatie. Daarnaast is er nu een nieuw punt naar voren gekomen, namelijk de hoeveelheid helper classes binnen de gehele code. Tijdens de ontwerp fase was dit al een punt van discussie en is er twijfel geweest over het gebruik van helpers. Echter is besloten dat met goed testen en een technisch correcte opzet (static classes) dit toch de beste keuze was. Door onder andere het testen is de kwaliteit en werking van de helpers gewaarborgd. Daarnaast hebben de helpers duidelijk bijgedragen aan het verlagen van de complexiteit en dus het verhogen van de begrijpbaarheid van de meest belangrijke classes (voornamelijk de generator). De Module Coupling in de helper classes heeft naar ons idee dan ook weinig nadelige effecten voor de uiteindelijke onderhoudbaarheid door de klant.

8.3 Werkwijze evaluatie

De geplande werkwijze bleek in de meeste gevallen succesvol, vooral de dagelijkse overleggen en het laagdrempelige contact met medewerkers van het bedrijf bleken essentieel voor het slagen van het project. Hierdoor konden problemen of misverstanden tijdig gesignaleerd en opgelost worden.

Bij problemen die grote invloed konden hebben op het verloop van het project is er tijdig geëscaleerd naar de opdrachtgever. Dit is niet vaak voorgekomen, meestal konden problemen opgelost worden door het team zelf, of na kort overleg met medewerkers van het bedrijf. Een voorbeeld waarbij wel geëscaleerd is, was toen de performance problemen met de RazorEngine zich voordeden. Na eerst zelf een aantal oplossingen geprobeerd te hebben, is er geëscaleerd naar de opdrachtgever. Hierdoor kon er snel met nieuwe inzichten gewerkt worden en werd er binnen afzienbare tijd een oplossing gevonden. Zonder deze escalatie had dit veel meer tijd gekost. Ook de opdrachtgever was zeer tevreden over deze manier van werken.

De wekelijkse sprints waren niet heel sterk afgebakend, daar is afgeweken van het plan. In de praktijk is het meer een doorlopend proces geweest met wekelijkse terugkoppeling naar de opdrachtgever. Er zijn hier geen nadelige gevolgen van ondervonden. Bij een groter team had dit wel voor problemen kunnen zorgen. Bij een volgend project is dit iets om een bewustere keuze in te maken.

De breakdown en planning bleek een onmisbare hulp om de voortgang te bewaken. Hier werd in de dagelijkse Scrum naar gekeken. Door het gebruik van de breakdown viel het snel op als taken langer duurden dan verwacht. Het verdelen van de componenten bleek een goede keuze omdat dit ervoor zorgde dat de ontwikkelaars elkaar niet in de weg zaten en precies wisten wat de verschillende delen deden. Ook het integreren van componenten verliep hierdoor soepel omdat beide ontwikkelaars goed wisten wat de gevolgen waren van wijzigingen in hun code.

Het gebruik van tools om de codekwaliteit al tijdens het programmeren te analyseren bleek een goede keuze. Hierdoor werden potentiële problemen al tijdens het ontwikkelen ontdekt en hoefde er maar weinig extra werk te gebeuren voordat de code naar SIG gestuurd kon worden.

9 Analyse

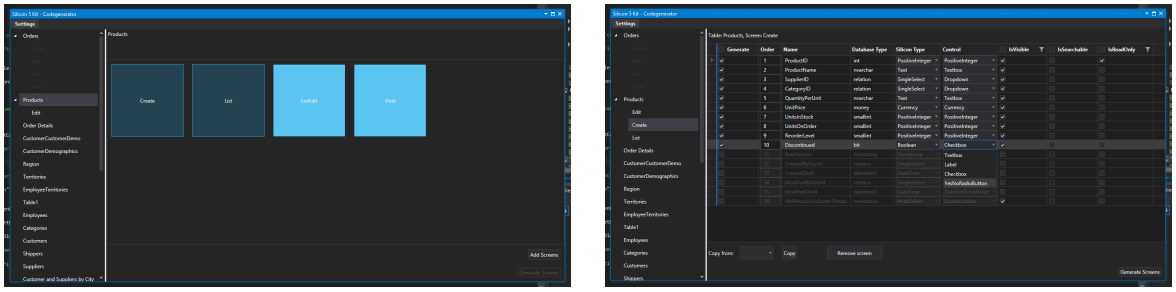
Dit hoofdstuk zal het gemaakte product analyseren. Uit deze analyse moet worden afgeleid of het probleem zoals beschreven in hoofdstuk 3 ook daadwerkelijk is opgelost. Er zal worden gekeken naar het uiteindelijke product en de gebruiksvriendelijkheid daarvan. Daarna zullen de eisen worden bekeken en dan vooral degene die niet behaald zijn. Voordat er een eindoordeel wordt gegeven, zal ook nog gekeken worden naar de bevindingen van de toekomstige gebruikers tijdens een gebruikerstest.

9.1 Product

Het eindproduct kan op twee verschillende manieren bekeken worden. Ten eerste zal gekeken worden naar de interface en dan voornamelijk de gebruiksvriendelijkheid er van. Als tweede wordt gekeken naar de bestanden die gegenereerd worden en de templates die daarvoor worden gebruikt. Dit laatste moet voornamelijk goed onderhoudbaar zijn.

De interface is vrijwel zo geworden als tijdens het ontwerpen (hoofdstuk 6.2 gepland is. Enkele weergaven van de interface zijn in figuur 12 te zien, ook zijn er extra screenshots in Appendix L toegevoegd. Enkele elementen uit de ontwerpfase zijn uiteindelijk niet verwerkt in de interface, zoals het snel toevoegen van extra schermen met knoppen rechts boven in het scherm. Deze missende elementen hebben geen negatieve effecten op de gebruiksvriendelijkheid. De gebruikelijke flow die gevolgd zal worden tijdens het gebruik van de generator, bestaat uit het eerst toevoegen van alle nodige schermen om daarna per scherm alle instellingen maken. Dit is met zeer weinig stappen te doen door de gebruiker en wordt versterkt door een groot aantal standaard instellingen waardoor er in de meeste gevallen weinig aangepast hoeft te worden. Een duidelijk bewijs van een gebruiksvriendelijke interface.

Het correct genereren van schermen in een web project is het tweede sub-product van het eindproduct. Dit gaat goed omdat voor alle vijf verschillende schermen die gemaakt kunnen worden, direct werkende code wordt gegenereerd. Nadat de generator klaar is, kan het aangepast project zonder fouten worden uitgevoerd en zijn de schermen direct beschikbaar op de website. Ook de acties als het aanpassen of verwijderen van items werkt zoals het zou moeten. Daarnaast zijn ook de templates van waaruit gegenereerd wordt, gestructureerd en overzichtelijk. Alle templates zijn gegroepeerd en in gestructureerde mappen geplaatst. Voor bijvoorbeeld het toevoegen van een controller element is het alleen nodig een kleine template toe te voegen en de koppeling in de typemapping (hoofdstuk 7.3 te maken. Hiermee is de generator dus flexibel en onderhoudbaar.



Figuur 12: Eind product: de generator interface

Voor beide onderdelen van het eindproduct is duidelijk voldaan aan gebruiksvriendelijkheid en onderhoudbaarheid. Dit zijn twee van de hoofdeisen die door de opdrachtgever gesteld waren (voor alle eisen zie Appendix G). Het product is dus makkelijk te gebruiken en genereert de verwachte resultaten. Door het gebruik van duidelijke templates is de onderhoudbaarheid groot gehouden. Het product voldoet dan ook goed genoeg aan de opdracht en lost het probleem op.

9.2 Behaalde eisen

De meeste eisen die in de onderzoeksfase zijn opgesteld, zijn behaald. De originele eisen kunnen in Appendix G worden gevonden. De niet behaalde Must, Should en Could have's zullen hier worden uitgelegd met onderbouwing. Over het algemeen zijn er meer eisen behaald dat aan het begin werd verwacht. Van de totaal 34 geformuleerde eisen zijn er 11 niet behaald of aangepast. De meeste van de 11 niet behaalde eisen zijn onnodige functionaliteiten die door meer kennis en ervaring geen toegevoegde waarde bleken te hebben.

Daarnaast zijn er ook een aantal eisen aangepast of niet realiseerbaar. Dit is een logisch gevolg van de nog lopende ontwikkelingen binnen het framework. Als een functionaliteit in het Silicon 5 framework nog niet (volledig) gerealiseerd is, is het niet mogelijk om er iets voor te genereren.

Niet gerealiseerde eisen

- Per scherm in te stellen: filter, extra validatie en annotaties
  Dit bleek uiteindelijk te complex om binnen de tijd te realiseren en er is nog geen goede basis in het framework om onder andere de filter te kunnen genereren. Daarnaast bleek het ook minder belangrijk dan eerst gedacht.

Aangepaste eisen

- Generatie projectskelet (inloggen, dashboard, etc)
  Het model project wordt niet standaard bijgeleverd. Er wordt vanuit gegaan dat dit project al bestaat of dat hij met de andere generator wordt gemaakt. Hier is echter geen rekening gehouden met de modellen die nodig zijn voor de standaard inlog acties. Deze zullen nog door de gebruiker toegevoegd of gegenereerd moeten worden met de andere generator.

- Lijsten kunnen ook gebaseerd zijn op database views (keuze basistabel)
  Het maken van lijst schermen op basis van views is geen probleem als hier ook een model/manager voor is. Er is echter geen basistabel constructie mogelijk en per rij is er ook geen primary key bekend. Om meer functionaliteit dan een lijst scherm te implementeren zal het model uitgebreid moeten worden om correct met views om te kunnen gaan.

- Onthouden gegenereerde code en bestanden voor analyse
  Dit wordt niet onthouden, maar direct uitgelezen uit de bestandsstructuur. Wanneer de bestanden voor een scherm al aanwezig zijn, zal deze grijs en niet aanklikbaar terugkomen in de tabellen en schermen boom (links in de interface).

Onnodige functionaliteiten

- Per scherm in te stellen: Veld validatie keuze gebaseerd op metadata uit de database
  Deze basis validatie wordt al gedaan door het datamodel dat gegenereerd is met een andere generator. Dit was bij het opstellen van de eisen niet volledig duidelijk waardoor het op een later moment is komen te vervallen.

- Standaard teksten/vertalingen direct toepassen bij genereren resource files
  Dit bleek een overbodige functionaliteit die niet of nauwelijks gevolgen zou hebben op de werking en gebruiksvriendelijkheid. Resource bestanden hebben in Visual Studio namelijk al een eenvoudige editor en bevatten over het algemeen maar drie vertalingen.

- Terughalen gebruikte instellingen van eerder gegenereerde code
  Ook dit bleek minder prioriteit dan eerste gedacht. Daarnaast is het vrij veel werk voor zeer weinig toegevoegde waarde. In de meeste gevallen zullen alle schermen in één keer gegenereerd worden.

- Logboek van acties bijhouden
  Dit had te weinig prioriteit en wordt voor het grootste deel opgevangen door direct na het genereren een generatie log weer te geven. Daarin zijn alle problemen tijdens het genereren vermeld.

- Rollen toe kennen voor bestaande schermen
  Deze functionaliteit is al ingebouwd in het framework en kan direct na het maken van een project via de browser gebruikt worden. Voor een beheergebruiker worden de rechten automatisch toegekend na het genereren van een scherm, voor eerder gegenereerde schermen hoeft dit op een later moment niet meer te gebeuren.

- Vertalingen direct bij het genereren kunnen invullen
  Ook dit bleek een onnodige functionaliteit te zijn aangezien resource bestanden al heel makkelijk bewerkt kunnen worden en het over het algemeen maar om 3 vertalingen per scherm gaat.

Niet mogelijk/niet in framework

- Export mogelijkheid bij lijst schermen met keuze voor welke velden
  Dit is komen te vervallen omdat de implementatie nog niet in het framework is gebouwd.

In deze lijst met niet behaalde eisen, zijn nagenoeg geen belangrijke eisen genoemd. Daarnaast zijn een aantal laagstaande eisen, die juist heel complex zijn, wel behaald. Dit zijn voornamelijk de eisen die te maken hadden met het vinden en genereren van relaties tussen tabellen. Hieruit is te concluderen dat, wat betreft de eisen, naar ons idee aan de opdracht is voldaan.

9.3 Gebruikerstest

Zoals beschreven is in de werkwijze (hoofdstuk 5.2) zou er een gebruikerstest worden gedaan. Richting het einde van het project is er met één van de medewerkers van Fenêtre de uitgebreide test gedaan. Deze medewerker werkt normaal met het Silicon framework en zal de generator dan ook gaan gebruiken in de toekomst.

De medewerker kreeg de opdracht om alle mogelijke schermen te genereren voor de "Product"tabel uit de database. Hij begon met een bestaand Silicon project en moest vanaf daar alle stappen zelf

uitvoeren. Aan het begin was nog niet helemaal duidelijk in welke context hij aan het werken was. Bij echt gebruik van generator weet de ontwikkelaar precies waarvoor hij wil genereren. Met de test is dat logischerwijs niet direct duidelijk, waardoor een korte uitleg de uitkomst bood. Het kiezen van de schermen ging soepel en gemakkelijk. Echter was de medewerker eerst in de veronderstelling dat de "scherm knoppen" aan en uit te zetten waren, waardoor het selecteren van meerder schermen in eerste instantie niet zo soepel ging. Toen duidelijk werd dat dit wel mogelijk was met het inhouden van de control-toets op het toetsenbord, ging het testen vlot verder. Het aanpassen van alle schermen naar de eisen van de medewerker ging verder zonder problemen en na het genereren kon de medewerker direct het resultaat bekijken in de browser. De medewerker was tevreden over de werking van de applicatie en had als enige opmerking dat het wellicht verstandig is om bij het selecteren van schermen een uitleg over de werking te plaatsen. De test was een groter succes dan verwacht en bevestigt de gebruiksvriendelijkheid die in de product analyse genoemd is.

In de onderzoeksfase is er ook een onderzoek onder de medewerkers gedaan (zie Appendix F) naar de eisen en verwachtingen van de nieuwe generator. Dit onderzoek heeft een set van aangepaste eisen opgeleverd die verwerkt waren in het programma van eisen. De analyse van de behaalde eisen hierboven laat zien dat ook deze eisen zijn behaald. Daarnaast was het snel kunnen aanmaken van nieuwe schermen een van de grote verwachtingen. Dit is behaald zoals uitgewerkt in de bovenstaande analyse van het product.

Naar aanleiding van de test en het eerdere onderzoek naar de eisen en wensen van de medewerkers, kan geconcludeerd worden dat het product tot nu toe voldoet aan hun verwachtingen. Totdat het product echt in gebruik is genomen, kan dit niet volledig met zekerheid gezegd worden. Waarschijnlijk zullen er uiteindelijk een aantal details naar boven komen die aanpassingen vereisen om de gebruiksvriendelijkheid te blijven garanderen. Daarnaast zal het Silicon framework in de komende tijd ook nog groeien en aangepast worden, hierdoor zal de codegenerator naar verwachting ook aanpassingen nodig hebben.

9.4 Eind oordeel

Aan de hand van de conclusies over de geanalyseerde onderdelen in dit hoofdstuk, kan gezegd worden dat de opdracht is behaald en het probleem is opgelost. In grote lijnen voldoet alles aan wat er werd gevraagd. Hier en daar zitten nog wat fouten, maar die lijken de gebruiksvriendelijkheid, functionaliteit en onderhoudbaarheid nauwelijks te beïnvloeden. Het project is dus geslaagd.

10 Conclusie

In dit project is voor Fenêtre een codegenerator ontwikkeld om webapplicaties te kunnen genereren die gebruik maken van het Silicon 5 framework. De generator kan op basis van metadata uit een database opties tonen aan de gebruiker die daarna door de standaard mapping en beperkte wijziging van instellingen heel snel werkende schermen kan genereren. Ook is het mogelijk om hele nieuwe webapplicatie projecten aan te maken.

Het proces is goed doorlopen en aan de meeste wensen van de gebruiker is voldaan. De oorspronkelijke wensen waar niet aan voldaan is, zijn door nieuwe inzichten komen te vervallen of kunnen pas aan de generator worden toegevoegd op het moment dat het Silicon 5 framework is aangepast. Ook is de code van de generator door de Software Improvement Group als goed onderhoudbaar bevonden. Gezien de conclusies in het Analyse hoofdstuk, kan gezegd worden dat het project en de opdracht geslaagd zijn. De opdrachtgever is tevreden over het verrichte werk. Hieronder volgen nog enkele aanbevelingen voor verdere ontwikkeling van de generator.

10.1 Aanbeveling

Naar aanleiding van bevindingen tijdens het proces en het maken van de implementatie, zijn een aantal inzichten naar boven gekomen. Deze inzichten bieden mogelijkheden tot betere of vernieuwende implementatie van bepaalde onderdelen. De belangrijkste bevindingen zijn hieronder opgenomen in een aantal aanbevelingen. Daarnaast is een lijst van missende functionaliteiten in het framework of de generator gemaakt die buiten dit rapport om aan de opdrachtgever wordt aangeboden.

Generatoren samenvoegen

Op dit moment moeten er, zoals in het rapport beschreven, twee generatoren gebruikt worden om een werkend scherm te krijgen. Ondanks dat het samenvoegen al op de planning van de opdrachtgever staat, is dit alsnog als aanbeveling opgenomen. De nieuwe generator biedt vele mogelijkheden voor zowel de interface als de generatie op het gebiedt van relaties tussen tabellen. Dit kan onder andere goed benut worden voor het genereren van descriptors in de modellen. Daarnaast is het belangrijk dat de modellen en de formulieren aan de voorkant consistent blijven. Dit kan alleen gegarandeerd worden als de beide generatoren samengevoegd zijn. Het laatste positieve gevolg is een vermindering van het aantal gebruikersacties en dus een betere gebruiksvriendelijkheid.

Gecombineerde Primary Keys

Het framework zoals het nu is, biedt geen standaard ondersteuning voor modellen die meerdere kolommen als identifier hebben(Primary Keys). Ondersteuning hiervoor is voornamelijk handig bij een "veel-op-veel"koppeling tussen twee tabellen. Daarnaast is het ook een vereiste als van samengestelde views (database views) een bewerkbare view gemaakt moet worden. Het toevoegen van deze mogelijkheden aan het framework en de generator biedt vele nieuwe mogelijkheden. Echter moet het bedrijf wel bepalen of dit in de praktijk veel voorkomt, dus of standaardiseren de moeite waard is.

Gebruik van templates bij lijst schermen

Zoals de lijst schermen nu in elkaar zitten, wordt er gebruik gemaakt van een standaard grid uit een ander framework. Hierbij kan elke kolom een bepaald formaat bevatten wat door dit framework beheerd wordt. Echter om deze weergaven consistent te houden met de andere schermen (edit, view) is het van belang dat ook per kolom standaard een template gebruikt gaat worden. Hierbij is formaat niet meer nodig, omdat dit door de template gewaarborgd zal worden. Daarnaast vergroot dit de onderhoudbaarheid omdat de templates voor alle schermen op een plek zullen staan.

Enum database type

In de modellen generator zit geen support voor het enum database type. Deze aanbeveling gaat dan ook samen met de eerste aanbeveling over het samenvoegen. Hierbij kan dan direct gekeken worden naar missende database types in de ondersteuning. Het enum type kan belangrijk zijn bij sommige projecten omdat het een vooraf ingestelde categorisering kan bieden. Wanneer support in de modellen is gebouwd, kan dit ook aan de generator en dus aan de schermen toegevoegd worden.

Mapping verbeteren

Voornamelijk bij de lijst schermen zijn nog een aantal instellingen die niet door de generator ingesteld worden. Onder andere de standaard sortering per kolom is hier een voorbeeld van. Veel van deze instellingen hebben over het algemeen een 1-op-1 verband met het database type en zijn dan ook geschikt om op te nemen in de type mapping. Het is aan te bevelen om de mapping en de schermen nog eens goed te bekijken en de mapping uit te breiden en verder te standaardiseren om hergebruik makkelijker te maken in de toekomst.

10.2 Reflectie

Over het algemeen vinden wij dat het project en proces goed zijn verlopen. Tussen de teamleden was er weinig tot geen frictie, waardoor het werken en overleg altijd soepel is gegaan. Dit is hebben we als een groot voordeel ervaren en we beseffen ons dat dit niet altijd vanzelfsprekend is. Desalniettemin is er ook op dit gebied veel geleerd wat betreft communicatie en het bijhouden van taken.

Op sommige momenten hadden wij andere ideeën of inzichten dan de opdrachtgever, wat onenigheid gaf. Dan moesten we onze eigen meningen even opzij zetten voor de eisen van de opdrachtgever. Dit was bijvoorbeeld het geval bij de opslag methode voor de typemapping. Het meemaken van zulke onenigheden was een goed leermoment voor ons. Het heeft ons geholpen om beter vanuit het perspectief van de opdrachtgever te denken en onnodige functionaliteiten snel te signaleren voordat daar meer tijd aan besteed is dan nodig.

Tijdens het implementeren waren er een aantal onderdelen die veel tijd hebben gekost. Dit is onder andere voor gekomen bij het maken van de layout en bindings in de interface. Hier waren wij niet altijd strikt genoeg voor onszelf om te stoppen op de momenten dat het zinloos werd. Vaak is het op zulke momenten goed om even afstand te nemen en iets anders te doen. De volgende dag is het dan in een mum van tijd opgelost met een frisse blik. Ondanks dat wij het op dat moment graag willen oplossen, is het verstandig om hier strikter op te zijn bij volgende projecten.

Ondanks deze lessen zijn we trots op een goed verlopen project. Het was zeer leerzaam om dagelijks in een echte werkomgeving op een kantoor te werken. Het project is zonder grote problemen verlopen en we zijn veel ervaring rijker.

Verklarende woordenlijst

In dit rapport wordt er uitgegaan dat de lezer enige basis kennis heeft van software ontwikkeling. Alleen de project specifieke termen zullen hier kort verklaard worden.

.NET

Het .NET framework van Microsoft biedt gebruikers veel standaard code zodat de ontwikkelaar zich niet meer druk hoeft te maken over veelgebruikte functies zoals geheugenbeheer en foutafhandeling.

AngularJS

Dit is een javascript framework dat html pagina's zeer dynamisch maakt met behulp van direct verbindingen tussen visuele elementen en data modellen. Kijk voor meer informatie in de AngularJS Appendix C.

Annotatie

Attribuut dat aan een datamodel wordt toegevoegd om eigenschappen aan delen van het model toe te voegen. Bijvoorbeeld regels voor validatie of weergave. [3]

ASP.NET MVC 5

Dit is een framework voor het bouwen van standaard en schaalbare web applicaties. Er wordt gebruik gemaakt van veel gebruikte design patronen en van ASP.NET en het .NET Framework.[4]

[3] `https://msdn.microsoft.com/en-us/library/dd901590(VS.95).aspx`
[4] `http://www.asp.net/mvc/mvc5`

ASP.NET Web Forms

Web Forms voor ASP.NET is een manier om formulieren in web browsers. Het biedt onder andere speciale "drag-and-drop"functionaliteit. Er wordt daarnaast gebruik gemaakt van een model dat gestuurd wordt door handelingen.[5]

Directive

Dit zijn attributen die aan HTML tags toegevoegd kunnen worden en hiermee automatisch extra functionaliteiten en opties aan de tag koppelt. Het voegt onder andere Javascript en HTML generatie mogelijkheden toe.

Entity Framework

Dit is onderdeel van het .NET framework om data modellen te mappen op database tabellen. Zie voor een volledige uitleg de Appendix C.

IDE

IDE staat voor "Integrated Development Environment". Het is een software applicatie dat een editor en vele tools bevat voor het ontwikkelen van software door programmeurs.[6]

MoSCoW

Methode om een lijst van functionaliteiten op prioriteit te ordenen. MoSCoW is een acroniem en staat voor **M**ust have, **S**hould have, **C**ould have en **W**on't have. De mogelijke functionaliteiten worden onder deze kopjes georganiseerd.[7]

Scrum

Een processtructuur voor het ontwikkelen van (software) producten. [5]

Silicon

Dit is het framework dat door Fenêtre zelf gebouwd wordt en voldoet aan de specifieke eisen die het bedrijf stelt aan hun internet applicaties. Zie voor meer informatie hierover het Silicon hoofdstuk 4.2.

Visual Studio

Dit is een door Microsoft ontwikkelde IDE. Het wordt onder andere gebruikt voor het ontwikkelen in de talen C#, .NET, Visual Basic, et cetera. Fenêtre gebruikt voornamelijk deze omgeving voor het maken van hun applicaties.

WPF

Windows Presentation Foundation, applicatie framework van Microsoft. [6]

[5] `http://www.asp.net/web-forms`
[6] `http://en.wikipedia.org/wiki/Integrated_development_environment`
[7] `http://en.wikipedia.org/wiki/MoSCoW_method`

Referenties

[1] (resources) Microsoft. Ado.net entity framework at-a-glance, 2015. https://msdn.microsoft.com/en-us/data/aa937709, (bezocht: 28 april 2015).

[2] (resources) Microsoft. .edmx file overview (entity framework), 2015. https://msdn.microsoft.com/en-us/library/vstudio/cc982042%28v=vs.100%29.aspx, (bezocht: 28 april 2015).

[3] (artikel) Julie Lerman. Looking ahead to entity framework 7. *MSDN Magazine*, 30(1), januari 2015.

[4] Josh Smith. Wpf apps with the model-view-viewmodel design pattern, 2009. https://msdn.microsoft.com/en-us/magazine/dd419663.aspx, (bezocht: 12 mei 2015).

[5] Ken Schwaber & Jeff Sutherland. The scrum guide, 2013. http://www.scrumguides.org/scrum-guide.html, (bezocht:29 mei 2015).

[6] Microsoft. Introduction to wpf. https://msdn.microsoft.com/en-us/library/vstudio/aa970268(v=vs.100).aspx, (bezocht:18 juni 2015).

[7] (tutorial) Entity Framework Tutorial. What is entity framework?, 2014. http://www.entityframeworktutorial.net/what-is-entityframework.aspx, (bezocht: 23 april 2015).

[8] (tutorial) W3Schools. The mvc programming model, 2012. http://www.w3schools.com/aspnet/mvc_intro.asp, (bezocht: 24 april 2015).

[9] (resources) Microsoft MSDN. Model-view-controller, 2003. https://msdn.microsoft.com/en-us/library/ff649643.aspx, (bezocht: 29 april 2015).

[10] (boek) R. Branas. *AngularJS Essentials*. Community Experience Distilled. Packt Publishing, 2014.

[11] (documentatie) AngularJS Development team. Creating custom directives, 2010. https://docs.angularjs.org/guide/directive, (bezocht: 30 april 2015).

[12] (blog) Microsoft MSDN Blogs. How to get razor syntax support in visual studio 2010, 2011. http://blogs.msdn.com/b/webdev/archive/2011/01/12/how-to-get-razor-syntax-support-in-visual-stud aspx, (bezocht: 28 april 2015).

[13] (blog) Scott Guthrie. Introducing razor a new view engine for asp.net, 2010. http://weblogs.asp.net/scottgu/introducing-razor, (bezocht: 22 april 2015).

[14] (blog) Phill Haack. C# razor syntax quick reference, 2011. http://haacked.com/archive/2011/01/06/razor-syntax-quick-reference.aspx/, (bezocht: 22 april 2015).

[15] (resources) Microsoft MSDN. Code generation and t4 text templates, 2013. https://msdn.microsoft.com/en-us/library/bb126445.aspx, (bezocht: 29 april 2015).

[16] (documentatie) .NET Foundation. .net compiler platform ("roslyn"), 2014. https://github.com/dotnet/roslyn, (bezocht: 22 april 2015).

[17] (blog) Mike Bennett. Code generation with roslyn fields and properties, 2014. http://dogschasingsquirrels.com/2014/08/04/code-generation-with-roslyn-fields-and-properties, (bezocht: 22 april 2015).

11 Appendices

A Originele project beschrijving

Onderstaand de beschrijvingen die voor het project op Bebsys (bachelor eind project management systeem van de studie Technische Informatie) zijn geplaatst. Daarna nog de onderzoeksvragen die op basis van deze beschrijvingen en de uitgewerkte opdracht in hoofdstuk 3 zijn opgesteld voor het onderzoeksrapport.

Project beschrijving van Bebsys

Fenêtre werkt sinds 2006 aan zijn software framework genaamd Silicon. Voor Silicon bestaat op dit moment een software generator voor het database layer en ASP.NET schermen. Recent is Silicon opnieuw opgezet op basis van de technieken MVC en AngularJS. De huidige generator dient daarom vervangen te worden door een geheel nieuwe. Daarnaast heeft Microsoft zijn code generatie tools uitgebreid met onder andere Roslyn, een open source compiler met bijbehorende tooling. De opdracht start met een onderzoek om te bepalen of code generatie binnen Fentre het beste kan worden uitgevoerd met Roslyn, T4 templates (de huidige methode) of MVC/Razor. Daarna zal de front end generator worden ontwikkeld op basis van de gemaakte keuze. Er zal gebruik worden gemaakt van de Agile projectaanpak, unit testing en technieken als AngularJS, C#, .NET, MVC, en Visual Studio. Deze opdracht is zeer interessant en uitdagend voor studenten. Door de complexiteit is deze opdracht alleen geschikt voor studenten met een goede ontwikkel ervaring.

Bedrijfsbeschrijving van Bebsys

Fenêtre is een full service leverancier van app en websites en online business applicaties. Wij helpen klanten met advies, ontwerp, ontwikkeling, implementatie en/of beheer. Wat kan jij van ons verwachten? - Een goed uitgeruste werkplek met moderne PC op leuke locatie (`http://www.cabfab.nl`) - Goede begeleiding - Veel leermogelijkheden - Een marktconforme stagevergoeding - Een dagelijkse lunch incl. een borrel op de vrijdagmiddag - Maandelijkse kennissessie en uit eten met je collegas - Als je er op dat moment werkt: 1x per jaar een lang weekend naar een leuke plek in het buitenland

Onderzoeksvragen onderzoeksrapport

Voor dit onderzoek zijn twee onderzoeksvragen gedefinieerd, deze vragen zijn vervolgens weer onder te verdelen in enkele subvragen. Dit rapport is opgebouwd zodat een antwoord gegeven kan worden op deze vragen, wat vervolgens zal resulteren in een advies. De onderzoeksvragen worden hieronder beschreven.

- Welke codegeneratietechniek is geschikt voor het genereren van applicaties met het Silicon framework?
  - Wat moet er gegenereerd worden?
  - Hoe werkt het Silicon framework?
  - Wat zijn de wensen van de gebruikers?
- Wat is de beste soort applicatie voor de Silicon codegenerator?
  - Welke soorten zijn mogelijk?
  - Welke soort past het beste bij de wensen van de gebruikers?
- Wat is de concrete lijst met eisen en verwachtingen?

B Infosheet

Titel van het project: Silicon 5 Codegeneratie
Naam bedrijf: Fenêtre b.v.
Presentatiedatum: 26 juni 2015

Beschrijving

Het doel van het project was de ontwikkeling van een codegenerator om snel nieuwe webprojecten op te kunnen starten. Het bedrijf Fenêtre ontwikkelt complexe internetapplicaties met behulp van een eigen framework. De projecten die met dit framework, Silicon 5, worden gemaakt bevatten vaak project-specifieke entiteiten. Voor deze entiteiten zijn bijna altijd onderhoudsschermen nodig. Elk project is uniek, daarom zullen deze schermen voor elk project op een andere manier gebouwd moeten worden. Dit is een tijdrovend proces. Omdat alle schermen volgens dezelfde structuur zijn gemaakt kan het bouwen van de schermen significant versneld worden met een codegenerator.

De uitdaging zat in het zo goed mogelijk benutten van de beschikbare informatie over de tabellen in het framework. Door een reeds bestaande generator is van elke databasetabel een model gemaakt. De combinatie van de metadata uit de database en de beschikbare modellen, biedt veel mogelijkheden. Voornamelijk het analyseren en genereren van de elementen voor relaties tussen verschillende tabellen en views (en dus tussen schermen) is een uitdaging. Samen met het bedrijf zijn op dit gebied veel complexe problemen bediscussieerd en aangepakt.

Het eindproduct is een volledig werkende applicatie die aan bijna alle gestelde eisen voldoet. De applicatie biedt de mogelijkheid om met enkele klikken een serie schermen te genereren op basis van een databasetabel. Deze nieuwe schermen zijn daarna direct te gebruiken in de webapplicatie, inclusief bijkomende functionaliteiten zoals het bewerken van data. Wensen en mogelijkheden die tijdens het project zijn ontdekt, zijn geformuleerd in een lijst van aanbevelingen.

De ontwikkelaars bij Fenêtre zullen deze generator gaan gebruiken bij het opstarten van projecten. Een aantal schermen die standaard in het framework zitten, zijn zelfs al gemaakt met de generator. Daarnaast zal de generator ook met zekerheid nog verder uitgebreid worden met extra functionaliteiten.

Teamleden

Naam: Mirko Dunnewind
Interesse: Ontwikkeling complexe web applicaties, database structuren, project management.
Bijdrage: Focus op de database structuren en het genereren van schermen.

Name: Ewoud van der Heide
Interesse: Interaction design, software design, project management
Bijdrage: Focus op de interface en het genereren van een compleet project.

Beide teamleden hebben bijgedragen aan de rapporten, ontwerpen, implementatie van helpers en de presentatie. Ook zijn alle keuzes altijd samen overlegd.

Contact informatie

Contactpersoon bedrijf: Roger Hendriks, Fenêtre , info@fenetre.nl
Coach TU Delft: Jan Hidders, Web Information Systems, TU Delft

Het eindrapport van dit project kan gevonden worden op: `http://repository.tudelft.nl`

C Betrokken Technieken

Overgenomen uit het onderzoeksrapport

.NET

Het .NET framework is door Microsoft ontwikkeld en bevat veel standaard operaties zodat de ontwikkelaar zich niet meer bezig hoeft te houden met het schrijven van code voor laag-niveau operaties. Zaken zoals geheugenbeheer en foutafhandeling worden door het .NET framework uitgevoerd. .NET is niet taal gebonden maar wordt meestal gebruikt met de talen Visual Basic of C#.

Entity Framework

Het Microsoft ADO.NET Entity Framework (vanaf hier gebruiken we hiervoor Entity Framework) is een Object/Relational Mapping (ORM) framework. Het is bedoeld om het werk voor ontwikkelaars makkelijker te maken omdat het de standaard database operaties bevat. Hierdoor hoeft een ontwikkelaar zich alleen maar bezig te houden met applicatie specifieke logica.[7]

In Figuur 1 is te zien hoe het Entity Framework gepositioneerd staat tussen de applicatie en de data store (data opslag). Hierdoor kan de ontwikkelaar direct met objecten werken zonder eerst SQL queries te moeten schrijven.

MVC

Model-View-Controller (MVC) is een programmeermodel waarin het programma op basis van functionaliteit wordt opgedeeld in drie delen.[8]

Model Representeert de data in het programma.

View Bepaalt hoe de informatie uit het model getoond wordt aan de gebruiker.

Controller Handelt de acties van de gebruiker af en werkt het model en de view bij aan de hand van deze acties.

Het voordeel van dit model is dat de verschillende delen in de code goed van elkaar gescheiden zijn waardoor wijzigingen van het ene deel meestal geen invloed hebben op een ander deel. Ook kunnen er bij een model meerdere views gebruikt worden waardoor dezelfde gegevens op meerdere schermen getoond kan worden.[9] Het Silicon framework van Fenêtre heeft door het gebruik van AngularJS een aangepaste implementatie van het MVC model, voor meer informatie hierover, zie sectie 4.2.2.

AngularJS

AngularJS is een Javascript framework dat helpt bij het maken van web pagina's of hele web applicaties. Het draait aan de client-side en heeft positieve impact op de performance doordat na het laden alleen nog kleine stukken data worden verstuurd. De mate waarin AngularJS gebruikt wordt is door de ontwikkelaar zelf te bepalen. De taal biedt vele opties om statische HTML pagina's uit te breiden met extra modules en dynamische elementen. Het toevoegen hiervan is zeer eenvoudig gehouden waardoor een kleine applicatie in een niet al te lange tijd op te zetten is.

AngularJS introduceert een groot aantal constructies en structuren, hieronder worden twee belangrijke toevoegingen toegelicht. Een van de belangrijkste elementen die AngularJS toevoegt zijn de data bindings [10]. Deze zorgen ervoor dat bij het updaten van een model, ook de HTML view wordt aangepast (en vice versa als het gaat om formulieren). Hiervoor is relatief weinig code nodig wat het

maken dynamische en vloeiende applicatie vereenvoudigd. Daarnaast zijn de directives van AngularJS ook een belangrijke toevoeging. Met directives kan HTML en Javascript code door middel van een attribuut aan een HTML tag toegevoegd worden. Ook is het mogelijk om met behulp van directives nieuwe HTML tags toe te voegen.[11]

AngularJS is opgebouwd als een aangepaste versie op het MVC model. Door de bindings en directives in dit framework, is de taak van de controller grotendeels verdwenen. In de plaats daarvan is het aan de ontwikkelaar hoe hij dit niet gespecificeerde gedeelte wil inrichten. Veel ontwikkelaars noemen het dan ook het Model-View-Whatever model [10].

D Generatie technieken

Overgenomen uit het onderzoeksrapport

De te genereren onderdelen zijn te splitsen in drie categorien: HTML, Javascript en C# bestanden. Overigens zal bij twee van de drie technieken Javascript en C# samen worden genomen als Code genereren aangezien ze heel vergelijkbaar zijn. Er zijn een aantal mogelijke technieken om de genoemde bestanden te genereren. In dit hoofdstuk worden drie van deze technieken beschouwd en beoordeeld op de bruikbaarheid. Aan het eind van het hoofdstuk worden de mogelijke technieken vergeleken en wordt er een advies gegeven over welke generator techniek de beste keuze is. Door de specifieke eisen van de opdrachtgever aan de applicatie en de afwijkende structuur van Silicon 5 worden bestaande codegeneratoren hier niet beschouwd. Er zal bij het gebruik van een bestaande generator teveel aangepast moeten worden waardoor het eventuele voordeel teniet gedaan wordt.

Razor

Razor is een zogenaamde "template markup syntaxën laat programmeurs C# code in HTML documenten gebruiken om dynamische pagina's te maken. Razor en de bijbehorende Razor view engine zijn ontwikkeld door Microsoft als alternatief voor de bestaande .ASPX view engine en is sinds januari 2011 beschikbaar [12]. Razor wordt volledig ondersteund door Visual Studio, dit betekent dat IntelliSense (code completion) en code highlighting mogelijk zijn. Ook wordt het gebruik van Razor door het Visual Studio 2015 ontwikkelteam aangemoedigd[8].

Een document waarin Razor gebruikt bestaat uit standaard HTML elementen met daartussen C# code voorafgegaan door een '@'. De Razor view engine past de C# code toe op de bijbehorende HTML elementen. Hierdoor kan er bijvoorbeeld een for-loop gemaakt worden die voor iedere iteratie een item in een lijst genereert.[13]

HTML genereren

Razor is oorspronkelijk bedoeld om statische HTML pagina's dynamisch te maken. HTML genereren is dus precies waar Razor zeer geschikt voor is. Hieronder staat een voorbeeld van een HTML lijst met daarin een stuk Razor om de verschillende lijst elementen te genereren.

```
<ul id="products">
        @foreach(var p in products) {
                <li>@p.Name ($@p.Price)</li>
        }
</ul>
```

Code genereren

Razor is afhankelijk van HTML tags om te bepalen waar elementen beginnen of eindigen. Daarom is het in Razor initieel moeilijk om tekst of code zonder tags te genereren. Speciaal hiervoor zijn er twee extra elementen in Razor: '@:' voor enkele regels zonder tags en '<text></text>' voor meerdere regels zonder tags.[14]

Razor kan alleen gebruikt worden in .cshtml bestanden, daarom zal er wat extra moeite gedaan moeten

[8] http://blogs.msdn.com/b/webdev/archive/2014/08/23/how-to-customize-scaffolding-templates-for-asp-net-vnext.aspx

om code te genereren in .js en .cs bestanden. Opties hiervoor zijn het aanpassen van de bestandsextensie na het genereren van de code. Voor Javascript kan het gebruik van een externe tool zoals RazorJS[9] of Javascript inline in .cshtml bestanden plaatsen een uitkomst bieden.

T4 templates

T4 templates worden gebruikt in de oude codegeneratoren. Het is vergelijkbaar met Razor aangezien het voornamelijk gebaseerd is op open en sluit tags voor logica en het printen van variabelen. Visual Studio biedt echter weinig support voor het bewerken van T4 bestanden. Daarnaast komt uit discussies met het Visual Studio ontwikkelteam naar voren dat er twijfels waren over het toevoegen van T4 in de 2015 versie[10]. Het type output bestand wordt aan het begin van de template bepaald door control blokken (`<#@ output extension=".txt"#>`). Voor verschillende doeleinden zijn er verschillende tags. Zoals `<# #>` voor logica, `<#+ #>` voor class features en `<#= #>` voor het printen [15].

HTML genereren

Het genereren van HTML is ongeveer net zo eenvoudig als bij Razor. Onderstaand een voorbeeld die hetzelfde genereert als in het voorbeeld bij Razor.

```
<ul id="products">
        <# foreach(var p in products) { #>
                <li><#= p.Name #> ($<#= p.Price #>)</li>
        <# } #>
</ul>
```

Code genereren

C# en Javascript genereren gaat op ongeveer dezelfde manier als eerder met de HTML is voorgedaan. Wat opvalt, is dat het een grote hoeveelheid open en close tags vergt. Echter, er hoeft geen rekening gehouden te worden met HTML open en sluit tags zoals bij Razor. Daarnaast moet er (net zoals bij de andere template talen) opgelet worden met enters en inspringende delen. De gegenereerde code moet er namelijk wel goed uitzien aangezien het gebruikt gaat worden om handmatig verder aan te passen naar de specifieke eisen van de klant.

Roslyn

Roslyn is het .NET Compiler platform van Microsoft, vanaf begin 2014[11] is dit open source geworden. Het project heeft als doel om de interne werking van de compiler beschikbaar te maken voor de programmeur. In een compiler gebeurt traditioneel veel codeanalyse om te bepalen hoe de code gecompileerd moet worden. Deze analyse zou vervolgens gebruikt kunnen worden om code te genereren. Het voordeel van Roslyn is dat, omdat het analyseren en genereren in de compiler gebeurt, al deze stappen uitgevoerd kunnen worden terwijl het programma draait. Hiermee kan het programma "zichzelf"schrijven.[16]

HTML genereren

Omdat Roslyn een C# compiler is, heeft het geen toegevoegde waarde bij het aanmaken van HTML bestanden. Alles wat hiervoor nodig is kan met standaard C# functionaliteiten gedaan worden of met de eerder besproken generatie technieken. Het enige scenario waarbij Roslyn iets kan toevoegen bij

[9] https://www.nuget.org/packages/RazorJS/
[10] https://github.com/aspnet/Home/issues/272
[11] http://visualstudiomagazine.com/articles/2014/04/03/microsoft-open-sources-roslyn-compiler.aspx

HTML generatie is wanneer een C# klasse geanalyseerd moet worden en aan de hand daarvan een HTML bestand gemaakt moet worden.

Javascript genereren

Het is niet mogelijk om direct Javascript te genereren met Roslyn. Het is uiteraard wel mogelijk om met C# een programma te maken dat als uitvoer een Javascript programma heeft. Als een dergelijk programma gecombineerd wordt met Roslyn kan een programma gemaakt worden dat C# code omzet in Javascript. Er kan dan geprogrammeerd worden in C# en de compiler zet dit om in werkend Javascript. Ook is het mogelijk om aan de hand van een analyse van een C# klasse een Javascript programma te maken.

C# genereren

Het genereren van code met Roslyn gebeurt in de compiler, dit betekent dat het op een laag niveau gebeurt. Elke stap van de te genereren code moet in detail gespecificeerd worden. Een operatie die op een hoger niveau uit enkele commando's bestaat, moet worden weergegeven met enkele regels code. Het volgende is een stuk Roslyn code[17].

```
FieldDeclarationSyntax aField = SF.FieldDeclaration(
        SF.VariableDeclaration(
        SF.ParseTypeName("String"),
        SF.SeparatedList(
                new []{
                        sF.VariableDeclarator( SF.Identifier( "_a" ) )
                }
        )
            )
).AddModifiers( SF.Token( SyntaxKind.PrivateKeyword ) );
@class = @class.AddMembers( aField );
```

De bovenstaande code genereert het volgende.

```
private String _a;
```

Advies generatie techniek

Roslyn valt al direct af als keuze. Ondanks dat het zeer krachtig is, past het niet in dit project. Roslyn biedt veel mogelijkheden voor code analyse en dynamische generatie, de generator zal gebruik maken van statische codegeneratie door middel van templates. Roslyn biedt hiervoor geen toegevoegde functionaliteit ten opzichte van andere generatie technieken. Ook is het niet geschikt om HTML te genereren en zal hier dus ook een andere techniek gebruikt moeten worden. Tenslotte is Roslyn zeer complex waardoor het veel tijd kost om te implementeren. De keuze zal dus gemaakt worden tussen T4 en Razor. Razor is weliswaar meer HTML (tag) gebonden, maar bevat ook goede methodes om tekst en code te genereren. Daarnaast is de syntax van Razor minimalistischer wat resulteert in beter leesbare code. Tenslotte moet nog vermeld worden dat het ontwikkelteam van Visual Studio 2015 om dezelfde reden het gebruik van Razor aanmoedigt[12]. Het is duidelijk dat Razor prettiger is om mee te werken en toekomstbestendiger is in vergelijking met T4. Tet advies is: Razor.

[12] http://blogs.msdn.com/b/webdev/archive/2014/08/23/how-to-customize-scaffolding-templates-for-asp-net-vnext.aspx

E Soort applicatie

Overgenomen uit het onderzoeksrapport

Er zijn verschillende mogelijkheden voor de soort applicatie. We onderscheiden drie mogelijkheden: een losse applicatie, een Visual Studio extensie of een Web applicatie. Om een keuze te maken worden de verschillende soorten beoordeeld op een aantal kenmerken. Het belangrijkste argument is gebruiksgemak; het is een nadrukkelijke eis vanuit de opdrachtgever om het gebruik van de generator zo gebruiksvriendelijk mogelijk te maken. Een medewerker moet met zo min mogelijk handelingen het gewenste resultaat bereiken. Ook moet de applicatie goed op de toekomst voorbereid zijn, dus makkelijk te onderhouden en uit te breiden.

Losse applicatie

Een desktop applicatie, gebaseerd op Windows Forms (WinForms) of Windows Presentation Foundation (WPF), is een losse applicatie die onafhankelijk van andere programma's vanaf het bureaublad gestart kan worden. Het voordeel van een dergelijke onafhankelijke applicatie is dat het volledig naar eigen inzicht gemaakt kan worden, zonder dat er rekening gehouden moet worden met een bestaande omgeving. Dit zorgt er ook voor dat het programma geen last heeft van wijzigingen die veroorzaakt worden door bijvoorbeeld een update van Visual Studio. Uiteraard zal het programma nog wel beïnvloed kunnen worden door wijzigingen in Silicon of de projectstructuur.

Een nadeel van een losse applicatie ten opzichte van een Visual Studio extensie, is het aantal gebruikersacties dat nodig is om tot een resultaat te komen. Om de generator te starten moet de gebruiker een extra applicatie naast Visual Studio starten. Ook moet er in de generator een keuze gemaakt worden voor welk project er gegenereerd moet worden.

Uit gesprekken met medewerkers blijkt dat de Silicon 4 generator, die met Windows Forms werkt, prettig in gebruik is. Als groot voordeel wordt genoemd dat, ondanks dat er verschillende versies van Visual Studio zijn geweest, de applicatie nog steeds stabiel en zonder grote updates functioneert.

Visual Studio extensie

Met Visual Studio extensies is het mogelijk om de bestaande IDE uit te breiden met extra functionaliteit. De techniek die hiervoor gebruikt wordt heet VSPackage. Microsoft heeft Visual Studio zelf opgebouwd uit verschillende VSPackages, het is dus mogelijk om elke functionaliteit van Visual Studio hiermee uit te breiden.

Het voordeel van VSPackages is dat het volledig geïntegreerd is in Visual Studio, en daarmee zeer gebruikersvriendelijk kan zijn. De gebruiker gebruikt namelijk al Visual Studio voor het ontwikkelen van de internetapplicatie, waardoor er geen aparte applicatie gestart hoeft te worden. Ook kunnen bepaalde functies vanuit contextmenu's gestart worden. Een nadeel is dat de integratie met Visual Studio voor toegevoegde complexiteit zorgt die niets te maken heeft met de werking van de applicatie. Ook ontstaat er een afhankelijkheid van Visual Studio waardoor nieuwe versies van Visual Studio er voor kunnen zorgen dat de applicatie niet meer werkt. Tenslotte zorgt de grafische interface van Visual Studio voor extra beperkingen bij het ontwerpen van de layout.

De bestaande generator voor de Silicon 5 modellen is een Visual Studio extensie. Ervaring van medewerkers is dat het maken van een extensie een steile leercurve heeft. Ook moeten er enkele wijzigingen gedaan worden bij het geschikt maken voor een nieuwe versie van Visual Studio. Deze wijzigingen waren echter minimaal en hadden geen invloed op de werking van het programma.

Webapplicatie

De codegenerator kan ook gemaakt worden als web interface. Hij zal op het interne netwerk worden gehost. Ook een connectie met alle test databases is dan goed mogelijk. Dit biedt voornamelijk veel mogelijkheden wat betreft het stijlen van de interface en de gebruiksvriendelijkheid. Daarnaast zal het ook door meerdere gebruikers tegelijkertijd gebruikt kunnen worden en kan er voor iedereen bijgehouden worden welke acties er gedaan zijn. De back-end van de applicatie zal gewoon met C# gemaakt worden, wat ongeveer dezelfde mogelijkheden biedt als de andere soorten applicaties. Het zal ook makkelijk te onderhouden zijn aangezien de kennis voor het bouwen van internet applicaties uiteraard aanwezig is bij het bedrijf en updates hoeven maar op één plek uitgevoerd te worden.

Echter zal de opmaak tijdens het ontwikkelen meer tijd kosten omdat er geen gebruik gemaakt kan worden van simpele uitlijning, wat bijvoorbeeld wel bij Visual Studio kan. Ook zal een web applicatie betekenen dat er veel werk voor de gebruiker is. De gegenereerde bestanden moeten namelijk worden gedownload en handmatig in de al bestaande project structuur worden verwerkt.

Advies applicatie keuze

Een losse applicatie en een Visual Studio extensie hebben veel met elkaar gemeen wat betreft de structuur. De web applicatie heeft een andere structuur, maar biedt niet meer mogelijkheden dan de andere twee soorten applicaties. Een web applicatie zal meer werk eisen van de gebruiker en kan moeilijk inspelen op de structuur van het project dat lokaal op de computer van de gebruiker staat. Dit heeft een negatieve invloed op de gebruiksvriendelijkheid. Hierdoor achten wij een web applicatie ongeschikt. De keuze tussen een losse applicatie of een Visual Studio extensie, is wat moeilijker.

Een losse applicatie is minder afhankelijk van externe software, hierdoor is het ontwikkelen makkelijker, maar biedt minder integratie waardoor er meer gebruikersacties nodig zullen zijn. Daarentegen biedt een Visual Studio extensie veel meer integratie mogelijkheden die de gebruiksvriendelijkheid bevorderen, echter is het wel complexer en moeilijker te onderhouden dan een losse applicatie. Een ander voordeel is dat de generator makkelijker het juiste project kan vinden. Als laatste moet opgemerkt worden dat uit de wensen van de gebruikers is gebleken dat de applicatie vorm niet uitmaakt, zolang het maar zeer gebruiksvriendelijk, snel en stabiel is. Besloten is om het gebruiksgemak zwaarder te laten wegen dan de complexiteit. Hieruit volgt het advies: Visual Studio extensie.

F Onderzoek Mederwerkers

Overgenomen uit het onderzoeksrapport

Om een idee te krijgen wat de medewerkers graag zien en verwachten van de nieuwe code generator, zijn er interviews geweest met een aantal medewerkers van Fenêtre. Deze vragen zijn gesteld om de personeelsleden die er vaak mee werken, uit te laten leggen wat ze goed en verkeerd vinden aan de bestaande generatoren. Ook werd gevraagd naar de huidige manier van werken en de eventuele wensen voor de nieuwe generator. Wat voortvloeide uit deze gesprekken is opgenomen in de lijst met eisen (appendix G) aan het einde van dit rapport die is opgesteld in overleg met de opdrachtgever. In dit hoofdstuk zullen de resultaten van de interviews beschreven worden. Ook de hoofdstukken hierna zullen nog enkele resultaten bevatten die uit het medewerkers onderzoek naar voren kwamen.

Tijdens het onderzoek zijn er verscheidene vragen gesteld, met als doel om meer te weten te komen over de huidige applicatie en manier van werken. De vragen gingen dan ook vooral over Silicon 5, de projectstructuur en aanpak, de verschillende generatoren en de soort applicatie.

Er is eerst gesproken met Medewerker 1[13] en Medewerker 2. Zij zijn verantwoordelijk geweest voor respectievelijk de Silicon 4 en de Silicon 5 code generatoren. Ook zijn ze beide gebruiker van de generatoren en het Silicon framework. Tenslotte heeft Medewerker 1 een grote rol gespeeld bij de ontwikkeling van Silicon 5 en het implementeren van AngularJS in Silicon. Op basis van de verkregen informatie zijn de structuur en wensen uitgewerkt. Hiervan is gebruik gemaakt om een extra interviews op te stellen. Deze zijn gedaan met dezelfde medewerkers en als toevoeging Medewerker 3. Hij heeft in het verleden de Silicon 4 generator veel gebruikt.

Gebruiksgemak

Uit de vragen die er op gericht waren om uit te vinden of er een voorkeur bestaat voor de vorm van de applicatie blijkt dat deze voorkeur er niet is. De medewerkers geven aan dat gebruiksgemak het belangrijkste criterium moet zijn voor het programma. Alles in de nieuwe generator moet er volgens de geïnterviewden op gericht zijn om de ontwikkelaar die het programma gebruikt werk uit handen te nemen. Dit betekend dat er zo min mogelijk handelingen nodig moeten zijn en dat de teksten en foutmeldingen duidelijk moeten zijn. Om het aantal handelingen te beperken wordt aangegeven dat het handig is als gekozen instellingen, zoals bijvoorbeeld sortering, naar verschillende andere aangemaakte schermen te kopiëren zijn. Deze functie bestaat al in de Silicon 4 generator.

Toen er doorgevraagd werd of dit betekend dat een Visual Studio extensie de ideale vorm van de applicatie was, dit is immers de IDE waar de ontwikkelaars het meeste mee werken, werd er gewaarschuwd. Een Visual Studio extensie lijkt optimaal omdat dit het aantal handelingen verminderd, ervaring leert echter dat het ingewikkeld is om een extensie te ontwikkelen. Ook is door de afhankelijkheid van Visual Studio het onderhoud wat intensiever: een nieuwe versie van Visual Studio betekend dat er ook een nieuwe versie van de generator uitgebracht moet worden. De impact van een dergelijke wijziging hangt af van de veranderingen die in Visual Studio geïntroduceerd worden.

Basis structuur

De opdrachtgever heeft in de beschrijving van de opdracht aangegeven wat er minimaal gegenereerd moet worden, dit is meegenomen in de eisen en wordt niet beïnvloed door de wensen van de medewerkers. Om te beoordelen of er behalve de minimale eisen nog andere verwachtingen waren bij de medewerkers is er gevraagd naar eventuele andere opties voor de generator. Omdat dit een behoorlijk

[13] De namen van de medewerkers zijn vanwege privacy niet opgenomen in het verslag.

open vraag is werd de medewerkers gevraagd welke stappen zij uitvoeren bij het aanmaken van een nieuw project. Hieruit bleek dat het zeer op prijs gesteld zou worden als de generator ook de structuur van een nieuw project zou kunnen aanmaken. Op dit moment wordt een nieuw project namelijk vaak gestart door een bestaand project te kopiëren en alle overbodige code er uit te verwijderen. Dit is niet efficiënt en vergt veel tijd wat beter aan het ontwikkelen van de applicatie besteed kan worden.

Genereren

Een andere functie uit de Silicon 4 generator die niet in de minimale eisen wordt genoemd is het genereren van SQL code om voor de net aangemaakte schermen de juiste rechten toe te kennen. Deze functie zou zelfs nog uitgebreid kunnen worden door het ook op eerder gegenereerde schermen toe te passen. Het zou veel handelingen en ontwikkelingstijd schelen aangezien het niet nodig zal zijn om handmatig bewerkingen in de database te doen.

Ook wordt de wens uitgesproken om zoveel mogelijk instellingen van de schermen, zoals validatie en sortering, bij het aanmaken standaard vast te stellen op basis van metadata uit de database. De gebruiker moet wel de mogelijkheid hebben om voor het genereren van deze standaard af te wijken. Ook moet het voor de gebruiker mogelijk zijn om extra opties aan velden toe te voegen. Dit gaat dan voornamelijk om validatie en annotaties die eigenschappen in de controllers en views toevoegen.

Tijdens de gesprekken is ook gevraagd naar het regenereren van code. Er werd aangegeven dat dit vrijwel nooit gebeurd aangezien kleine aanpassingen net zo snel zelf gemaakt kunnen worden. Daarnaast zijn er vaak al handmatige aanpassingen in de code gemaakt. Bij regenereren zouden deze wijzigingen overschreven worden. Er kwam wel naar voren dat er regelmatig wijzigingen in het Silicon framework aangebracht worden. Deze aanpassingen hebben soms invloed op de applicatie specifieke code van alle projecten. Het toepassen van deze updates worden nu nog handmatig gedaan, maar een tool hiervoor zou op prijs gesteld worden.

G Programma van eisen

Overgenomen uit het onderzoeksrapport

Niet functionele eisen

Dit zijn de eisen aan de code en software. Welke voornamelijk vanuit het bedrijf komen en zijn gebaseerd op hoe Fenêtre te werk gaat. Het zijn harde eisen aangezien het anders een negatieve invloed heeft op de onderhoudbaarheid van de generator.

- Gebaseerd op de oude Silicon 4 codegenerator
- Gebruik huidig Silicon 5 framework met bestaande database model generator
- Gebruik aangepast MVC model
- Gebruik AngularJS op Silicon 5 specifieke manier
- Gebruik Visual Studio (2013/2015)
- Gebruik semi scrum project methode (werkende tussenproducten)
- Eindproduct liever met beperkte functionaliteit maar van goede kwaliteit dan veel functionaliteit van beperkte kwaliteit
- Klaar voor de toekomst wat betreft technieken

Functionele eisen

Hieronder volgen de eisen die worden gesteld aan de te maken codegenerator. Deze zijn opgesteld aan de hand van de opdrachtbeschrijving, gesprekken met medewerkers (wensen) en gesprekken met de opdrachtgever zelf. Voor de ordening van de eisen wordt gebruik gemaakt van de MoSCoW methode. Dit heeft tot gevolg dat er een duidelijke afbakening is zonder dat dit ten koste gaat van de kwaliteit van de generator.

Must have

- Gebruiksgemak
- Statische code generatie
- Generatie projectskelet (inloggen, dashboard, etc)
- Data model informatie komt van een bestaande SQL database
- Mogelijkheid tot lijst/bewerk/toevoeg/verwijder schermen genereren per database tabel
  - AngularJS MVC controllers
  - AngularJS MVC views
  - Server-side web api controllers
  - Server-side validatie (type validation mapping)
  - Resource bestanden voor talen
- Talen kunnen kiezen voor resource bestanden
- Per scherm in te stellen:

  - Volgorde en zichtbaarheid velden
  - Zoek mogelijkheden bij lijst schermen
  - Sorteer mogelijkheden bij lijst schermen
  - Keuze voor type weergave (type control mapping)
  - Veld validatie keuze gebaseerd op metadata uit de database
  - Export mogelijkheid bij lijst schermen met keuze voor welke velden
- AngularJS controller file kunnen bijwerken om er nieuwe views aan toe te voegen
- Een goede basis om snel een nieuw project te kunnen starten en uitbreiden
- Gebruik door meerdere ontwikkelaars zonder conflicten te veroorzaken

Should have

- Lijsten kunnen ook gebaseerd zijn op database views (keuze basistabel)
- Genereren en uitvoeren van SQL query's om rollen toe te kennen voor nieuwe schermen
- Kopiëren van settings van ander scherm naar nieuw scherm
- Per scherm in te stellen:
  - Filter mogelijkheden bij lijst schermen
  - Extra validatie handmatig selecteren/toevoegen
  - Keuze in annotaties per veld bij genereren
- Standaard teksten/vertalingen direct toepassen bij genereren source files

Could have

- Extra soorten schermen toevoegen (bijvoorbeeld lijst scherm met inline bewerken)
- Per scherm in te stellen:
  - Tabel relaties in lijst/bewerk/toevoeg schermen
- Terughalen gebruikte instellingen van eerder gegenereerde code
- Logboek van acties bijhouden
- Instellingen/logboek delen met andere ontwikkelaars
- Onthouden gegenereerde code en bestanden voor analyse (waarschuwingen, regenereren)
- Rollen toe kennen voor bestaande schermen
- Vertalingen direct bij het genereren kunnen invullen

Won't have

- Regenereren van code die al handmatig is aangepast
- Validatie van tabel relaties (onder andere interfield validation)
- Alle projecten automatisch aanpassen op Silicon vernieuwingen (in de project specifieke code)

H Use cases

UC1: Generatie projectskelet

Deze use case beschrijft hoe een gebruiker een nieuw Silicon 5 project start.

Actors

Applicatie gebruiker

Preconditions

Visual Studio gestart

Basic flow

1. Gebruiker kiest voor menu optie om Silicon 5 project te genereren.
2. De gebruiker wordt een aantal velden getoond:
   (a) Bestandslocatie van nieuw project kiezen (met optie om dit grafisch te doen)
   (b) Naam van het project
   (c) Keuze: bestaande solution of nieuwe solution
      i. Bij keuze nieuwe solution, ook solution naam invoeren.
3. Gebruiker kiest ok.
4. Een nieuw project met de Silicon 5 structuur wordt op de gekozen locatie aangemaakt met de ingevulde naam.
5. Afhankelijk van de keuze in een nieuwe solution of een bestaande.
6. Er is ook een generator status bestand toegevoegd aan het project/solution waarin relevante informatie voor de generator in opgeslagen wordt.
7. Succesvol einde use case.

Alternative flows

Locatie project bestaat niet
Bij stap 4 wordt geprobeerd het project aan te maken op de opgegeven locatie. Als de opgegeven locatie niet bestaat:

1. Het programma geeft een waarschuwing dat de opgegeven locatie niet bestaat.
2. De gebruiker krijgt de keuze: locatie aanmaken of een andere locatie opgeven.
   (a) Als de gebruiker kiest voor aanmaken wordt de basic flow vanaf stap 4 verder uitgevoerd met deze nieuwe locatie.
   (b) Als de gebruiker een nieuwe locatie wil opgeven wordt terug gegaan naar de basic flow in stap 2.

Niet mogelijk om project aan te maken op gekozen locatie
Bij stap 4 wordt geprobeerd het project aan te maken op de opgegeven locatie. Als het niet mogelijk is om op de gekozen locatie de bestanden toe te voegen (bijvoorbeeld door beperkte rechten of naam/pad te lang)

1. Het programma geeft een waarschuwing dat het niet mogelijk is om op de gekozen locatie een project aan te maken (specifieke foutmelding).
2. Gebruiker klikt de waarschuwing weg.
3. De applicatie gaat terug naar de basic flow in stap 2.

Solution bestaat nog niet op de geselecteerde locatie (gekozen om bestaande solution te gebruiken).
Als bij optie 2c gekozen is voor een bestaande solution en er in stap 4 geen solution gevonden kan worden op de gekozen locatie.

1. Het programma geeft een waarschuwing dat er geen solution gevonden is op de gekozen locatie.
2. Het programma biedt de keuze: doorgaan en een nieuwe solution aanmaken of een andere locatie opgeven
   (a) Als de gebruiker kiest voor het aanmaken van een nieuwe solution zal de applicatie verder gaan met deze keuze vanaf stap 4.
   (b) Als de gebruiker een andere locatie wil opgeven gaat de applicatie terug naar stap 2.

Postconditions

Successful
Een nieuw Silicon 5 project is aangemaakt op de gekozen locatie. De applicatie toont de mogelijkheid om een database te selecteren (UC2). Het statusbestand is bijgewerkt.
Failure
Er is geen Silicon 5 project aangemaakt.

UC2: Koppelen database en project

Deze use case beschrijft hoe een gebruiker een database kan koppelen aan een project.

Actors

Applicatie gebruiker

Preconditions

Silicon 5 project.

Basic flow

1. Start als:
   (a) Zolang er geen database is geselecteerd bij een Silicon 5 project zal de applicatie tonen dat er nog een koppeling gemaakt moet worden.
   (b) Wanneer de gebruiker kiest voor de applicatie menu optie "wijzigen database connectie".
2. De applicatie toont de volgende invoervelden. Wanneer er al database gegevens bekend waren, zijn deze alvast ingevuld.
   (a) Database locatie
   (b) Username
   (c) Password

3. De gebruiker vult deze waarden in en klikt op ök".
4. De applicatie verbindt met de database en slaat instellingen op.
   (a) Database instellingen worden aan het project toegevoegd/bijgewerkt in het database config bestand van de Silicon 5 applicatie.
5. Succesvol einde use case.

Alternative flows

Database niet gevonden
Als de opgegeven database bij stap 3 niet gevonden wordt:

1. Het programma geeft een waarschuwing dat de gekozen database niet gevonden kan worden.
2. Het programma gaat terug naar de basic flow in stap 1 (eventueel met waarden nog ingevuld).

Databasegebruiker informatie incorrect
Als de database in stap 3 wel gevonden wordt maar er niet ingelogd kan worden met de ingevulde gebruikersgegevens.

1. 1. Het programma geeft een waarschuwing dat er niet ingelogd kan worden met de ingevulde gegevens (eventueel extra info als username wel goed is maar wachtwoord niet).
2. 2. Het programma gaat terug naar de basic flow in stap 1 (eventueel met waarden nog ingevuld).

Verbonden database is geen Silicon 5 database.
Het lukt bij stap 3 om met de database te verbinden maar de structuur wijkt af van een standaard Silicon 5 database.

1. Het programma geeft een waarschuwing dat de opgegeven database geen Silicon 5 database is en dat er een andere database gekozen moet worden.
2. Het programma geeft 2 opties
   (a) Het programma gaat terug naar de basic flow in stap 1 (eventueel met waarden nog ingevuld).
   (b) Silicon 5 tabellen genereren (en eventueel overschrijven) met standaard waarden.

Postconditions

Successful
De applicatie heeft verbinding gemaakt met een Silicon 5 database. De beschikbare tabellen worden getoond. Het status bestand is bijgewerkt.
Failure
Er is geen database verbinding gemaakt.

UC3: Genereren scherm bij databasetabel

Deze use case beschrijft hoe een gebruiker een nieuw scherm kan genereren bij een database tabel.

Actors

Applicatie gebruiker.

Preconditions

Silicon 5 project. Database koppeling met het project.

Basic flow

1. De applicatie toont op het basisscherm een lijst van beschikbare tabellen, database-views en opties.
2. De gebruiker kiest bij een tabel voor de optie genereren scherm.
3. Applicatie haalt details van deze tabel op uit de database.
4. Applicatie vraagt naar type scherm (lijst, edit, lijstedit, delete, etc, bij views aantal types niet beschikbaar)
5. Applicatie toont opties voor de tabel per type scherm in tabblad:
   (a) Type bij velden kunnen aangeven.
   (b) Gebaseerd op eerder gegeneerd scherm (toon eerder gegenereerde schermen, dit staat in status bestand).
   (c) Opties voor volgorde en zichtbaarheid van velden.
      i. Bij view
      ii. Bij export (alleen lijst schermen)
   (d) Sortering selecteren.
   (e) Opties voor zoeken selecteren.
   (f) Toon standaard validatie (gebaseerd op database).
   (g) Bij database-view edit schermen extra optie voor selecteren basistabel.
   (h) Kopiren alle of gedeeltelijke instellingen van ander tabblad.
6. De gebruiker vult de velden naar wens in en klikt op genereren.
7. Afhankelijk van het bestaan van eerder gegenereerde schermen bij deze tabel zijn er hier twee paden:
   (a) Eerste keer genereren bij deze tabel
      i. Aanmaken volledige structuur: controller, view etc.
   (b) Er is al eerder gegenereerd bij deze tabel
      i. Aanmaken benodigde views
      ii. Waar nodig toevoegingen bij bestaande controllers.
8. Status bestand wordt bijgewerkt met de geselecteerde acties.
9. De applicatie toont een waarschuwing dat er nog geen rechten zijn toegekend voor het nieuwe scherm: rechten toekennen ja of nee, blokkerend in applicatie zelf.
   (a) Ja $\Rightarrow$ UC4.
   (b) Nee $\Rightarrow$ terug naar basisscherm.
10. Succesvol einde use case.

Alternative flows

Postconditions

Successful
Een nieuw scherm is toegevoegd aan de applicatie en het status bestand is bijgewerkt.
Failure
Er is geen nieuw scherm toegevoegd.

UC4: Toekennen rechten voor rol in database voor gegenereerd scherm

Deze use case beschrijft hoe er rechten toegekend kunnen worden aan een nieuw gegenereerd scherm.

Actors

Applicatie gebruiker

Preconditions

Silicon 5 project. Database koppeling Gegenereerd scherm

Basic flow

1. Er zijn drie manieren om deze flow te starten:
   (a) Via de tabellen overzicht in de applicatie.
   (b) Via context menu op een scherm of controller.
   (c) Direct na het genereren van een scherm (Zie UC3).
2. Het scherm toont de rollen die al rechten hebben. De gebruiker wordt gevraagd welke rollen hij toegang wil geven, de default is standaard toegevoegd in de lijst mits deze niet al rechten heeft.
3. De applicatie geeft een waarschuwing dat de gekozen rollen rechten krijgen om het scherm te zien.
4. Gebruiker bevestigt of annuleert
   (a) Annuleren: Einde use case: Terug naar basisscherm.
   (b) Bevestigen: de applicatie maakt rechten aan in database.
5. Geef melding van succesvol toekennen rechten.
6. Succesvol einde use case.

Alternative flows

Postconditions

Successful
Rechten zijn toegekend voor scherm.
Failure
Er zijn geen rechten toegekend voor dit scherm.

I SIG evaluatie

SIG (Software Improvement Group) heeft op twee momenten tijdens het proces een evaluatie gedaan van de gemaakte code. Onderstaand beide volledige evaluaties die door ons zijn ontvangen.

SIG evaluatie 1

De code van het systeem scoort net vier sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Duplication, Unit Size en Module Coupling.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er duplicatie te vinden binnen, bijvoorbeeld, de 'TableColumnModel'-class. In dit specifieke geval is er een grote overlap tussen de verschillende queries, iets wat ook op andere plekken in het systeem voorkomt. Het is aan te raden om dit soort duplicaten op te sporen en te verwijderen om zo aanpassingen in de toekomst makkelijker te maken.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'CombineDatabaseTypes'-methode, zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '// first find the type in this row' en '// type doesn't exist yet, create it' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Module Coupling wordt er gekeken naar het percentage van de code wat relatief vaak wordt aangeroepen. Normaal gesproken zorgt code die vaak aangeroepen wordt voor een minder stabiel systeem omdat veranderingen binnen dit type code kan leiden tot aanpassingen op veel verschillende plaatsen. Wat opvalt binnen dit systeem is dat er meerdere '*Helper'-classen zijn waar verschillende functionaliteiten in zitten. Een voorbeeld hiervan is de 'ProjectHelper'-class die methoden bevat voor het ophalen van project informatie ('GetSolutionPath', 'GetProjectPath'), maar ook generiekere functionaliteit voor files ('CreateDirectories', 'WriteFile', 'ReadFile). Om zowel de grootte als het aantal aanroepen te verminderen zouden deze functionaliteiten gescheiden kunnen worden, wat er ook toe zou leiden dat de afzonderlijke functionaliteiten makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden worden.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

SIG evaluatie 2

In de tweede upload zien we een stijging van 30 procent in het codevolume, terwijl de score voor onderhoudbaarheid ook licht is gestegen.

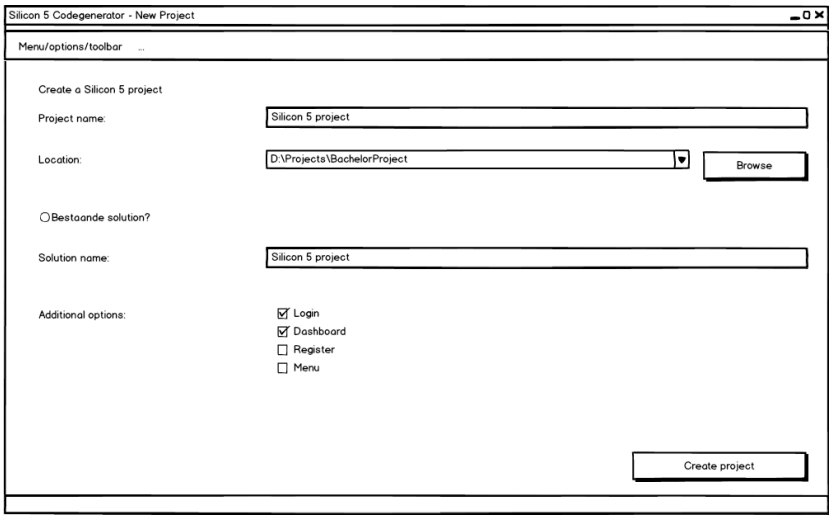
Van de drie deelscores die bij de eerste analyse als verbeterpunt genoemd werden zien we een duidelijke verbetering op Duplication en Unit Size. Dat geldt niet voor Module Coupling, de verwevenheid

is zelfs nog verder toegenomen. Het valt op dat jullie ontzettend veel code in helper-classes hebben zitten (met name TemplateHelper, DatabaseHelper, ProjectHelper, ConfigHelper). Het is volkomen normaal om älgemeneütility-methodes in dit soort classes onder te brengen, maar bij jullie wordt het percentage code dat in helpers zit wel erg hoog (bijna 40 procent).

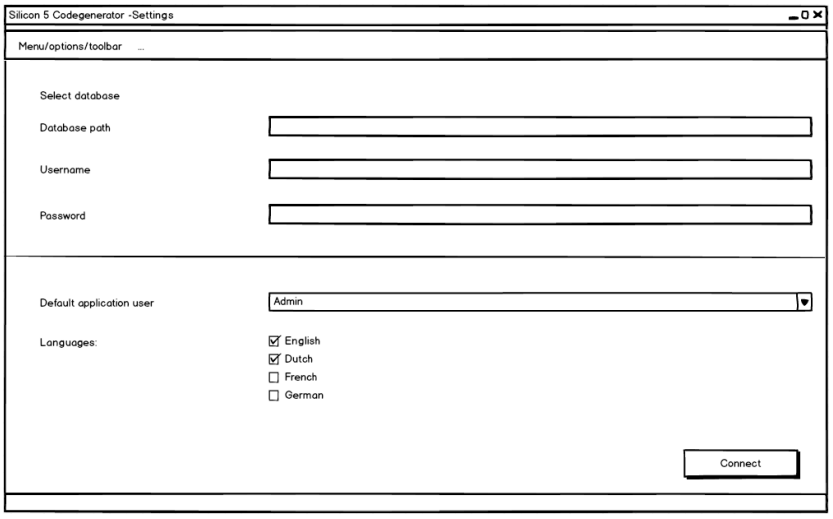
Naast productiecode hebben jullie ook een aanzienlijke hoeveelheid testcode toegevoegd, complimenten daarvoor.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie grotendeels zijn meegenomen in het ontwikkeltraject.

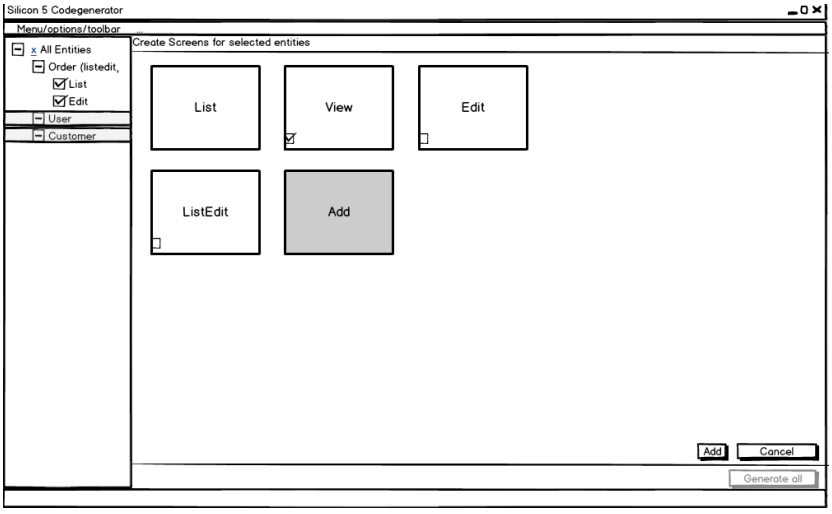
J Afbeeldingen Ontwerp



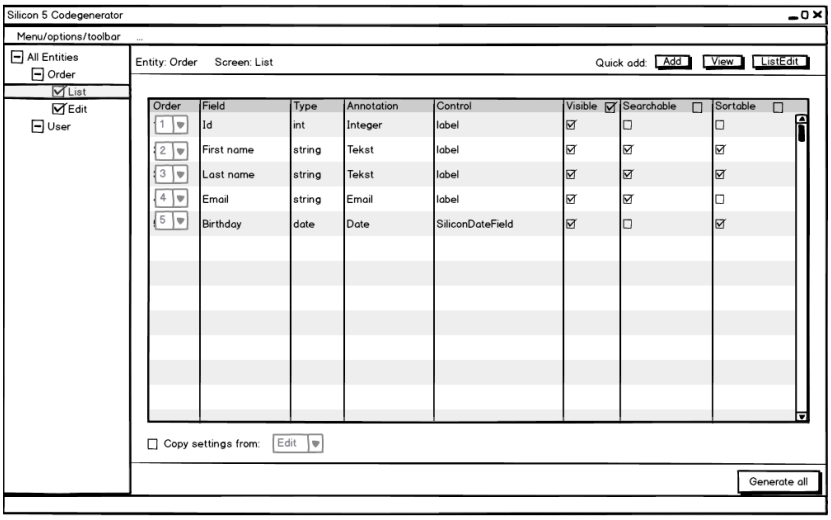
Figuur 13: Mockup: Creëer project scherm



Figuur 14: Mockup: Project settings scherm



Figuur 15: Mockup: Add screen

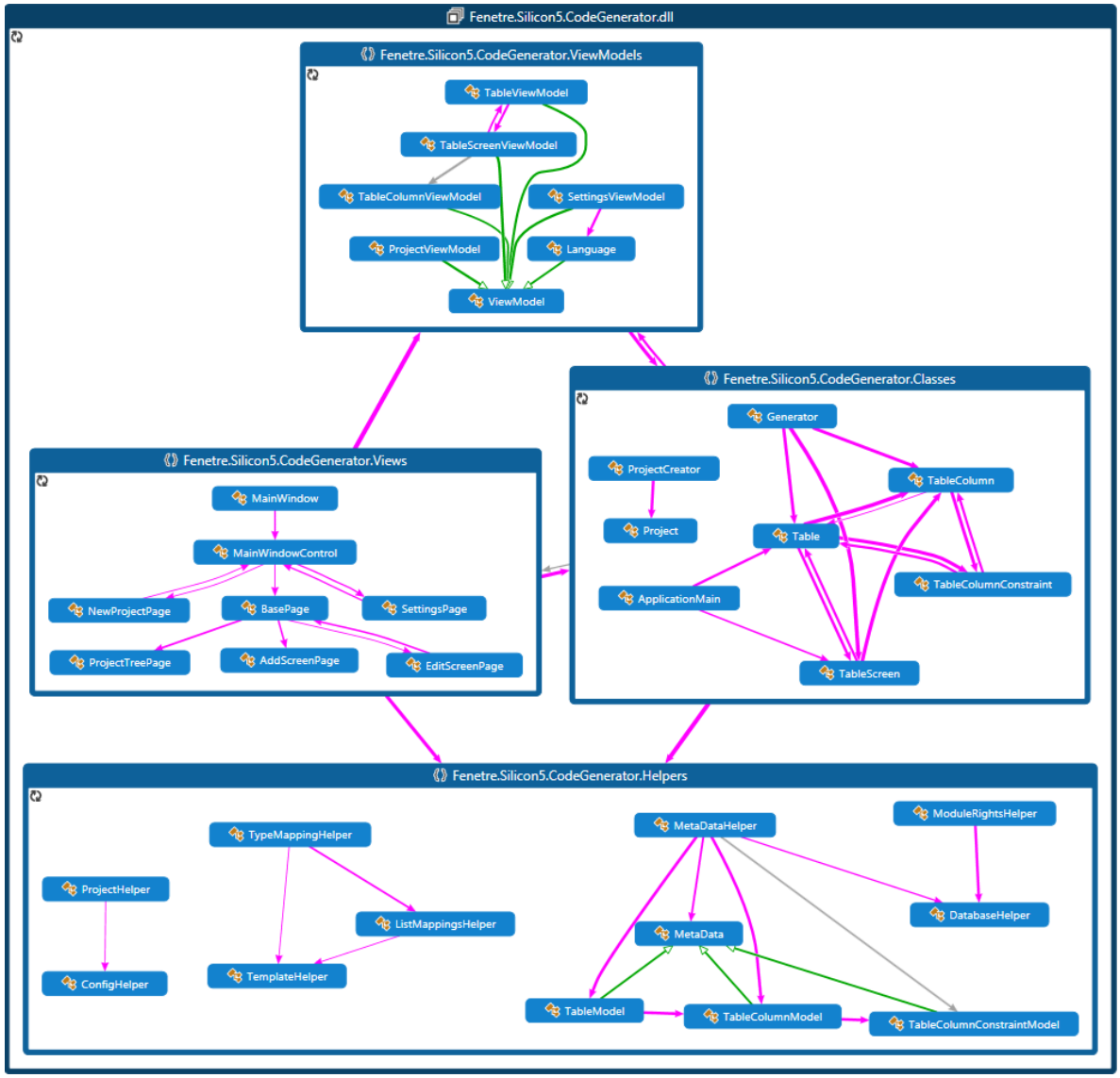


Figuur 16: Mockup: Edit screen

K Codemap

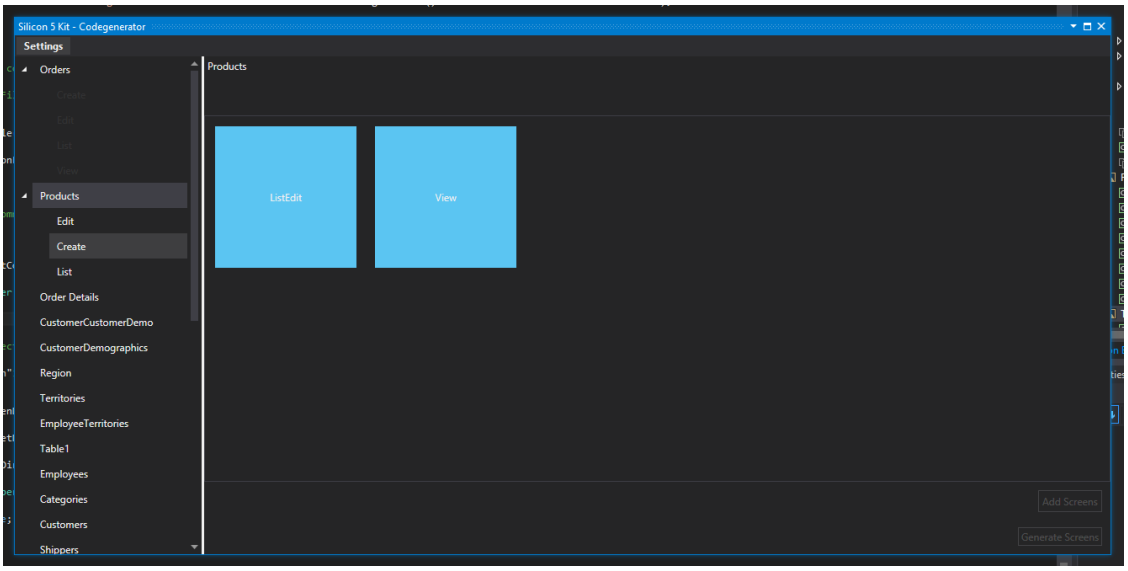
Onderstaand het grootste deel van de door Visual Studio gegenereerde codemap aan het einde van het project. Hierin is duidelijk te zien hoe de verschillende onderdelen tot elkaar verhouden. Zo is te zien hoe de Views en Viewmodels netjes gestructureerd zijn. Ook is te zien dat de Table classes en stuk ingewikkelder in elkaar zitten. Dit is logisch aangezien het hier om complexe methodes met relaties gaat. Als laatste maakt deze code map duidelijk dat de helpers ook echt helpers zijn. Zij worden alleen maar aangeroepen.

Legenda: groene lijn = overerving, paarse lijn = aanroepen, grijze lijn = referentie.

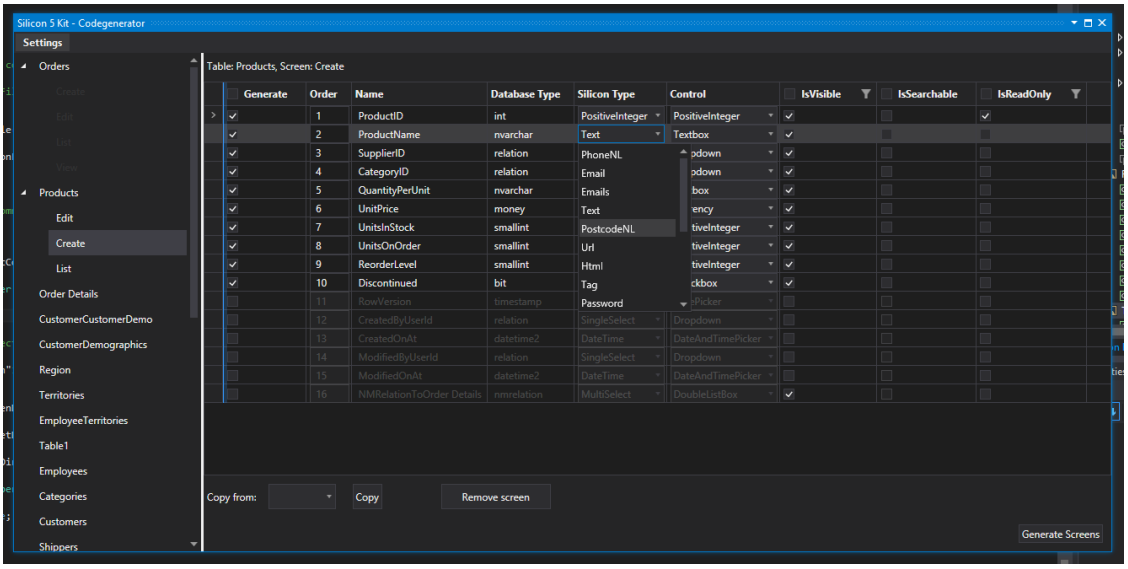


Figuur 17: Codemap van o.a. class, helper en view implementatie

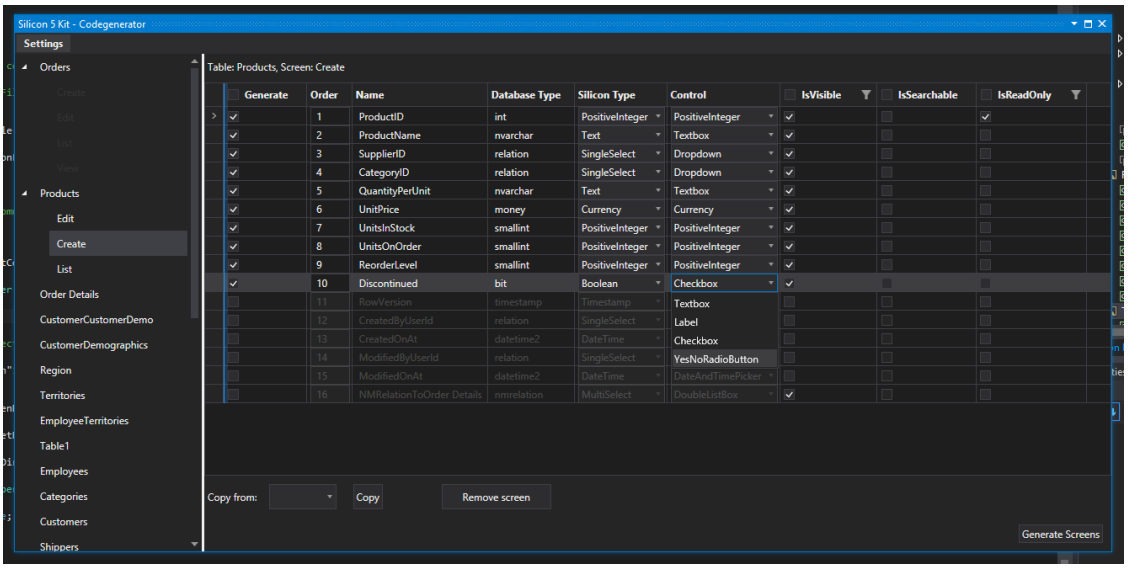
L Afbeeldingen Product



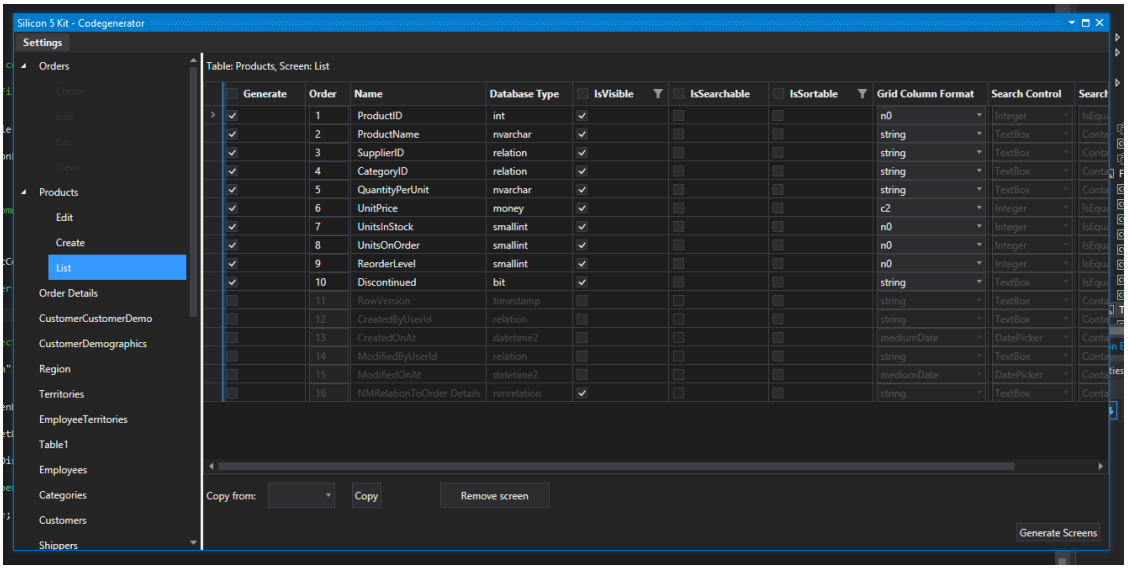
Figuur 18: Product: 3 schermen toegevoegd



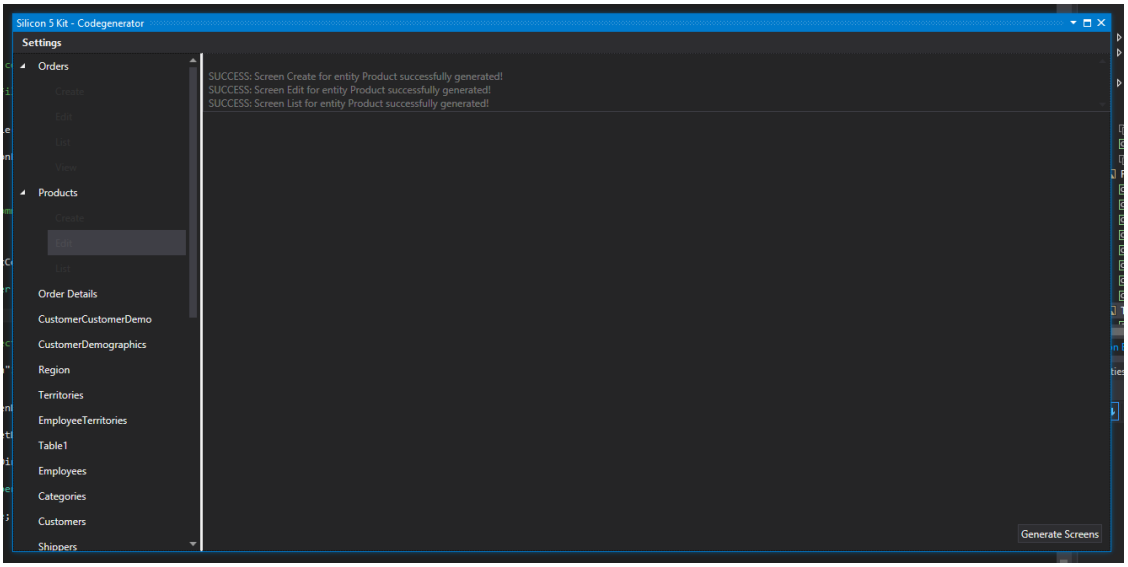
Figuur 19: Product: edit interface, Silicon type selectie



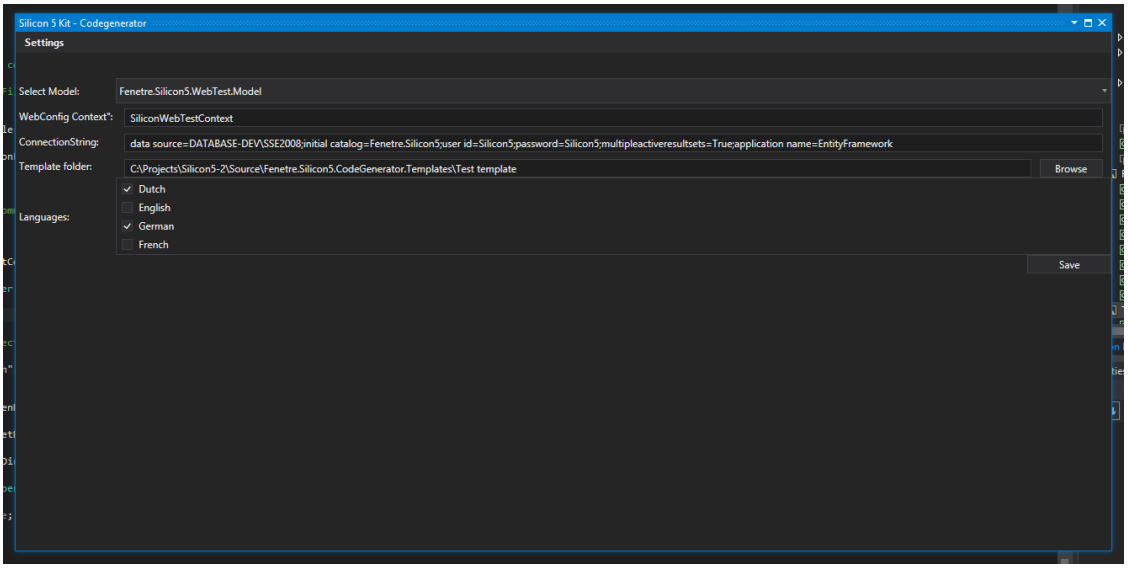
Figuur 20: Product: edit interface, control element selectie



Figuur 21: Product: edit interface voor een lijst scherm



Figuur 22: Product: log na generatie



Figuur 23: Product: settings interface