HANZEHOGESCHOOL

Informatica

Afstudeer scriptie

11 augustus 2014, Hoogeveen

Auteur:
Marcel Horlings
351254
Opleiding: Informatica

Stagedocent:
Jacob Mulder
Hanzehogeschool Groningen

Stagebedrijf:
Nidaros
Pascal Hakkers
Stoekeplein 1a
7902 HM, Hoogeveen

Nidaros

Hanzehogeschool Groningen
University of Applied Sciences

Woord vooraf

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nidaros

Hanzehogeschool Groningen
University of Applied Sciences

Samenvatting

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nidaros

Hanzehogeschool Groningen
University of Applied Sciences

Inhoud

1 Inleiding **1**
1.1 Nidaros 1
1.2 Probleemstelling 2
1.3 Inhoud van dit rapport 2

2 Plan van aanpak **3**

3 Versiebeheersysteem **5**
3.1 CVCS of DVCS? 5
3.2 Tools 7

4 Ontwerp keuzes **8**
4.1 Besturingsysteem 8
4.2 Communicatie 8
4.3 Frameworks 12

5 Implementatie **13**

6 Conclusie **14**

A Persoonlijke ontwikkeling **15**

B Planning **16**

1 Inleiding

Inleiding

1.1 Nidaros

Nidaros is een bedrijf dat adviseert en ondersteuning biedt bij IT-gerelateerde bedrijfsprocessen. Het doel van Nidaros is het stimuleren van bedrijfsprocessen, het informeren van procesverantwoordelijkheden en het bewust worden van wat IT kan toevoegen aan bedrijven en organisaties. Nidaros is een klein bedrijf met ongeveer twaalf werknemers, die op veel markten in de IT bezig is. Het biedt advies op het gebied van softwareontwikkeling en geven advies op basis van testmanagement en op basis van een informatieanalyse. Nidaros maakt zelf websites en software voor klanten en voeren optimalisaties door in bestaande websites. Daarnaast doet ze ook een stuk ICT-beheer binnen bedrijven, en voeren ze reparaties van computers en iPhones uit.s

Consultancy

In de Consultancy tak van Nidaros worden werknemers gedetacheerd naar andere bedrijven om daar te helpen met het inbrengen van externe systemen, bijdragen van opllossingen binnen die bedrijven en voor het testen van systemen die de bedrijven bouwen.

Solutions

Bij Solutions worden producten gemaakt en verkocht verkocht aan gebruikers. Deze producten kunnen ingekocht zijn, zelf gemaakt zijn of een combinatie hiervan. Zo biedt Nidaros KerkTV aan wat een zelfgemaakt product is waarbij mensen de diensten van de kerk thuis op de tv kunnen bekijken via een kastje wat de stream binnenhaald. Hiervoor levert Nidaros het apparatuur om de dienst mee op te nemen de servers om het met te streamen en de kastjes die bij de mensen thuis voor het ontvangst staat.

Naast KerkTV heeft en werkt Nidaros aan: een urenregistratietool genaamd ProHours, CVanalist een analyste tool en databank van curricu-

lum vitae's. Verder bouwt en host Nidaros meerdere website voor een aantal klanten.

1.2 Probleemstelling

Bij elke aankoop die gedaan wordt in een winkel wordt er een bonnetje meegegeven. Met dit bonnetje kan de garantie van een product verhaald worden wanneer het product het begeeft. Voor veel mensen is het bijhouden van alle bonnetjes die bij elk apparaat verkregen wordt en lastige taak. Ze vergeten wanneer hoe lang het bonnetje nog geldig is of wanneer de garantie afloopt en veel bonnetjes raken ook kwijt. Voor de mensen die hier last van hebben wordt de ScanjeBon app ontwikkeld. Met deze app kunnen de bonnetjes gemakkelijk via de smartphone opgeslagen worden. Waardoor gebruikers de bonnetjes altijd op één centrale plek hebben. Gebruikers van de app zullen ook genotificeerd worden wanneer bonnen aflopen en ze kunnen de bonnetjes overzichtelijk uit elkaar houden.

1.3 Inhoud van dit rapport

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

2 Plan van aanpak

De looptijd van het project bedraagt twintig weken. Deze twintig weken zal verdeeld worden over vier fases zoals beschreven in RUP. De eerste fase is Inception, in deze fase wordt er voor gezorgd dat iedereen betreffende het project hetzelfde beeld krijgt over het project. De tweede fase, Elaboration genaamd, wordt gebruikt als de eerste iteratie.

Zo wordt er een werkend product opgeleverd, maar worden ook dingen gedaan als het opzetten van de ontwikkelomgeving en het versiebeheersysteem. De derde fase, Construction, is waar het ontwikkelen gebeurd. Hier zal het overgrote deel van de Scanjebon app gemaakt worden. Dit wordt gedaan in iteraties en na elke iteratie wordt de app getoond aan de deelnemende stakeholders.

De laatste fase is de Transition, de Scanjebon app is hier klaar voor gebruik en kan in de markt gezet worden. Daarnaast zal hier de overdracht plaats vinden van de broncode, de documentatie en de ontwikkelomgeving. De fases zullen er samen voor zorgen dat er een Scanjebon app voor minimaal één platform gemaakt zal worden met daarnaast een ontwikkelomgeving waar een volgende ontwikkelaar mee verder kan. De planning staat als gantt chart in B.

Overdracht app en broncode.Overdracht app en broncode.

Fase 1: Inception

- Projectgoedkeuring.
- Onderzoek talen / platform.
- Onderzoek ontwikkelomgeving.

Fase 2: Elaboration

- Opzet versiebeheersysteem..
- Wireframes maken
- Bouwen app:
- Foto's maken / laden uit galerij

Fase 3: Construction

Bouwen app:

- Gegevens invullen en meesturen
- Bonnen bekijken in een lijst en apart
- Gegevens automatisch uit de foto halen d.m.v. OCR
- Ontwikkeling REST API:
- Opzetten framework
- REST infrastructuur opzetten.

Fase 4: Transition

- Advies taal platform
- Ontwikkelomgeving / ontwikkelstraat
- Overdracht app en broncode.

3 Versiebeheersysteem

Tijdens het bouwen van de applicatie is er gebruikt van het versiebheersysteem Git. Voor de stageperiode begon werd er binnen Nidaros geen gebruik gemaakt van een dergelijk systeem, daarom moest tijdens de stage uitgezocht worden wat het beste zou werken voor de stage en voor de rest van het bedrijf.

3.1 CVCS of DVCS?

Voor de keuze van van versiebeheersystemen zijn er twee opties een Centralized Version Control System (CVCS) of een Distributed Version Control System (DVCS). Voor CVCS is de bekenste optie Subversion (SVN) en voor de DVCS zijn Git en Mercurial de grootste opties. Aan beide kanten worden er versies bijgehouden van de code die gescrheven wordt en het is bij allebei mogelijk om de code te delen tussen verschillende manieren, ... Maar er is wel een verschil hoe dat gebeurd en dat verschil is ook merkbaar in het gebruik en heeft zijn voor en nadelen.

Servers

Om de code te kunnen delen bij een CVCS is er een server nodig. Op deze server worden alle versies bijgehouden, wat erg handig is want zo kan iedereen bijhouden wat er met de code gebeurd en weet iedereen waar een ander mee bezig is. Het grote probleem met een CVCS server is dat als de server stuk gaat en de data op de server verloren gaat is alle code en vooral de geschiedenis van de code ook weg. Wat dan overblijft is alle code die de mensen op dat moment lokaal op de werkplek hebben staan en geen hele geschiedenis van alle code meer. Bij het werken met CVCS-en wordt er altijd maar een deel van de code naar de locale machine gekopieerd. Wanneer de server voor een tijd offline gaat kan niemand hun code meer delen, opslaan of verder gaan met een ander stuk. Als er een aftakking van de code gemaakt wordt zodat daar nieuwe functies aan toegevoegd kunnen worden heeft de gebruiker alleen toegang tot die code. De code van de nieuwe functies die een collega op dat moment maakt blijven alleen op de server staan.

Bij DVCS is een zelfde opzet mogelijk, er kan een server neergezet worden waar alle code beschikbaar op is en waar wijzegingen bekend blijven. Net als bij de CVCS kan hier de code ingezien worden door iedereen en kan iedereen zien waar een ander mee bezig is. Mocht er bij een DVCS de server stuk gaan of offline, dan onstaat hier wel een verschil met de CVCS. Bij een DVCS staat staat op elke werkplek alle code en wijzegingen. Wijzegingen kunnen opgeslagen worden zonder verbinding met het internet, dit betekent niet alleen dat er door gewerkt kan worden tijdens het offline gaan van de server maar ook wanneer er gewerkt wordt vanuit een trein zonder internet verbinding. Het stukgaan van de server betekent bij DVCS niet dat er geen code meer gedeeld kan worden onderling, want iedereen kan als server fungeren en van iedereen kan de code opgehaald worden. Elke werkplek is hierdoor een backup van de server en van elkaar. Dit kan ook gedaan worden omdat bij een DVCS alle code opgehaald wordt van de server en niet alleen het stukje waar de ontwikkelaar op dat moment mee bezig is. Dit betekent dat als de ontwikkelaar een stuk bij wil dragen aan een stuk code waar hij niet bij betrokken was hij dit in een handomdraai kan doen

VCS -> Git

Een groot verschil tussen de eerder versiebeheersystemen en Git is hoe ze code en vooral veranderingen daarin opslaan. Bij de eerder VCS-en wordt elke keer dat er een stuk code toegevoegd wordt wordt er een nieuwe versie van dat bestand opgeslagen en de wijzeging er naast. Bij het delen van de code worden alleen wijzegingen en bestanden opgeslagen van de bestanden die dan aangepast zijn. Bij Git gaat het opslaan anders, Git kijkt bij elke toevoeging van code naar alle bestanden en onthoud van alle bestanden wat de status op dat moment was. Dit doet Git niet door de bestanden opnieuw op te slaan maar door te verwijzen naar de laatste aanpassing van dat bestand.

Branches

Elke wijzeging die toegevoegd wordt bij een VCS is een commit. Alle commits zijn stukjes code die op elkaar volgen, bij meerdere commits komt er als het ware een lijn van commits. Branches zijn aftakkingen van de reguliere lijn (master branch) aan commits. Een branch wordt gemaakt

met het geven van een naam, deze naam is een beschrijving van wat de wijzegingen gaan doen, een bug-fix of de naam van de functie die in deze branch wordt toegevoegd. De commits in deze branch zijn ook alleen hier zichtbaar en te gebruiken totdat deze branch samengevoegd wordt met de reguliere lijn aan commits, mergen genoemd.

Het maken van branches is een goede manier om mee te werken, omdat de master branch blijft werken en pas wanneer een branch voltooid is die wijzigingen erbij komen. Zo raakt de master branch niet vervuild met versies die maar half werken of nog getest moeten worden maar gebeurd dat al in de branch. Met branches kan er ook tegelijk aan een nieuwe functie van de applicatie gewerkt worden en aan een bug in de code. Wanneer er een bug gemeld wordt in de applicatie kan de ontwikkelaar naar de master branch gaan en vanaf daar een nieuwe branch aanmaken om de bug te maken. Wanneer deze bug gemaakt is kan het getest worden en daarna op de master branch gezet zodat het direct opgeleverd kan worden.

Tussen Git en oudere versiebeheersystemen is er een groot verschil in het maken en gebruiken van branches. In oudere versiebeheersystemen kost het veel tijd en moeite om een branch aan te maken, aangezien het hier vaak betekend dat alle code in de repsoitry verplaatst moet worden naar een nieuwe map. Dit kost tijd wat afhankelijk is van de grote van een repository.

In git is het maken van een branch niet meer dan het wegschrijven van 41 karakters. Doordat het maken van een branch in Git niet meer is dan een verwijzing naar een vorige commit. Dit wordt gedaan in een SHA-1 checksum van 40 karakters die bij de laatste commit hoorden en een enter. Hier hoeven dus verder geen bestanden voor gekopieerd of verplaatst te worden en hoeft er bij Git niet gewacht te worden om een branch aan te maken.

Mergen

3.2 Tools

Stash - github - bitbucket - gitlab - github enteprise

4 Ontwerp keuzes

4.1 Besturingsysteem

Voordat er begonnen kon worden aan het project moest eerst duidelijk zijn op welk platform de applicatie zou draaien. Door de grote overmacht van Android en iOS op het gebied van besturingssystemen voor de smartphone is er in het onderzoek voornamelijk hierop de nadruk gelegd. Hieronder zal uitgelegd worden waarom er voor een iOS applicatie is gekozen en wat de voor en nadelen van beide besturingssystemen zijn

Keuze

4.2 Communicatie

De communicatie tussen de app en de server gebeurt over het internet, om de communicatie goed te laten verlopen moet er afgesproken worden wat de wijze van deze comminucatie is en hoe die zal verlopen. Deze afspraken zitten gebonden in een protocol. Voor het versturen van gegevens tussen de client en de server zijn er twee bekende protocollen REST en SOAP. Hieronder zal worden uitgelegd wat beiden inhouden, waarmee het verschil duidelijk wordt gemaakt tussen de protocollen. En welke keuze er uiteindelijk is gemaakt.

SOAP

Simple Object Access Protocol (SOAP) is een protocol waarmee berichten tussen systemen verstuurd worden. SOAP is bedacht voor Microsoft en wordt tegenwoordig door veel bedrijven en software oplossingen gebruikt. Het protocol bestaat uit drie onderdelen, Het eerste deel beschrijft wat er in het bericht staat en hoe het verwerkt moet worden. Het tweede deel is een set met regels wat aangeeft welke hoe de gegevens verwerkt worden op basis van de applicatie specefieke gegevenstypes. Het derde en laatste onderdeel is een afspraak voor het gebruik van remote procedure calls. Dit is hoe SOAP berichten verstuurd worden.

SOAP gebruikt alleen XML om gegevens mee te versturen, maar dit kan wel over alle protocollen die gebruikt worden op het internet zoals:

HTTP, SMTP en FTP. Het is gemaakt om oudere technologien te vervangen die niet goed werkten op het internet. Toen het gemaakt is werd er erg gelet op het mogelijk maken van uitbreidingen op het protocol, hierdoor zijn er veel uitbreidingen[1]gemaakt die allemaal hun eigen doel hebben. Zo is er een uitbreiding die te maken heeft met beveiliging, een voor notificaties en een voor metadata.

In SOAP wordt een Web Service Definition Language (WSDL) gebruikt om de de service te definiëren. De WSDL geeft aan hoe de hoe de service die gegeven wordt omgaat met SOAP berichten, de service die aangeeft welke endpoints er zijn om gegevens over te dragen of op te halen, de 'binding' die wordt gebruikt om het gebruik van portTypes te beschrijven en de portTypes zelf dat een groep van operaties definïeerd. In een WSDL zit veel informatie en bij het juist configureren van een WSDL kan er code worden gegenereerd die communiceren aan de hand van de WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
             xmlns:tns="http://www.tmsws.com/wsdl20sample"
             xmlns:whttp="http://schemas.xmlsoap.org/wsdl/http/"
             xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
             targetNamespace="http://www.tmsws.com/wsdl20sample">

<!-- Abstract type -->
   <types>
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                 xmlns="http://www.tmsws.com/wsdl20sample"
                 targetNamespace="http://www.example.com/wsdl20sample">

         <xs:element name="request"> ... </xs:element>
         <xs:element name="response"> ... </xs:element>
      </xs:schema>
   </types>

<!-- Abstract interfaces -->
   <interface name="Interface1">
      <fault name="Error1" element="tns:response"/>
      <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
         <input messageLabel="In" element="tns:request"/>
         <output messageLabel="Out" element="tns:response"/>
      </operation>
   </interface>

<!-- Concrete Binding Over HTTP -->
   <binding name="HttpBinding" interface="tns:Interface1"
            type="http://www.w3.org/ns/wsdl/http">
      <operation ref="tns:Get" whttp:method="GET"/>
   </binding>

<!-- Concrete Binding with SOAP-->
   <binding name="SoapBinding" interface="tns:Interface1"
            type="http://www.w3.org/ns/wsdl/soap"
            wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
            wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
      <operation ref="tns:Get" />
   </binding>

<!-- Web Service offering endpoints for both bindings-->
   <service name="Service1" interface="tns:Interface1">
      <endpoint name="HttpEndpoint"
                binding="tns:HttpBinding"
                address="http://www.example.com/rest/"/>
      <endpoint name="SoapEndpoint"
                binding="tns:SoapBinding"
                address="http://www.example.com/soap/"/>
   </service>
</description>
```

Figuur 4.1: Voorbeeld van een WSDL bestand.

REST

Representational state transfer (REST) is een webarchitectuur die de basis vormt voor het world wide web (www). REST gebruikt voornamelijk JavaScript Object Notation(JSON) of XML voor de data overdracht.

REST werkt erg simpel. Het gebruikt create, read, update en delete (CRUD) voor zijn acties, via http wordt dit aangeroepen met POST, GET, PUT en DELETE. REST verstuurd zijn CRUD opdrachten naar een Uniform Resource Identifier (URI). Door de combinatie hiervan weet de ser-

[1]Alle gespecificeerde SOAP uitbreidingen in 2007:
`https://www.innoq.com/soa/ws-standards/poster/innoQ%20WS-Standards%20Poster%202007-02.pdf`

ver welke actie ondernomen moet worden en wat er vervolgens moet gebeuren.

Bij een GET op de URI `http://www.scanjebon.nl/receipts` worden alle bonnen verwacht waar de gebruiker bij mag. De data van deze bonnen worden dan in JSON of XML teruggestuurd. Bij een GET op de URI `http://www.scanjebon.nl/receipts/1` wordt de data van het bonnetje verwacht die als 'id' 1 heeft. Naast de data die teruggegeven wordt krijgt de client een HTTP code 200 terug

Wanneer er data opgehaald moet worden voor alle bonnetjes van een specefieke gebruiker kan dit door `http://www.scanjebon.nl/users/1/receipts` aan te roepen. Er wordt eerst aangegeven van welke gebruiker de data nodig is, met behulp van `/users/1` is het duidelijk dat er van de gebruiker met het id 1 data nodig is. Door `/receipts` erachter te plaatsen is het duidelijk dat van die gebruiker data van de bonnen gevraagd wordt.

Bij een POST wordt er geen data opgehaald maar opgeslagen in het framework. Vervoglens wordt de data wel teruggestuurd, omdat bij het aanmaken op de server nog het een en ander is veranderd. Zo krijgt een resource pas een id op het moment dat deze wordt aangemaakt op de server en de client heeft die id nodig om later die resource te kunnen ophalen, aanpassen of verwijderen. Het aanmaken van een resource wordt gedaan door een POST te versturen naar een URI van een resource zoals `www.scanjebon.nl/receipts`. Bij de post worden de gegevens in de vorm van JSON of XML meegestuurd. De server controlleert dit en slaat het vervolgens op en stuurt diezelfde data terug met extra informatie die op de server is aangemaakt. Bij het slagen van de POST wordt de HTTP 200 code meegestuurd.

Om de data aan te passen die is opgeslagen kan er een PUT of PATCH uitgevoerd worden op een object. Deze PUT of PATCH wordt gedaan op een resource met id als `www.scanjebon.nl/receipts/1`. Bij een PUT actie wordt via JSON of XML meegestuurd wat het huidige object moet worden. Dit is waar het verschil wordt gemaakt met een PATCH waar alleen het geen wat veranderd moet worden wordt meegestuurd. In tegenstelling tot een POST krijgen de PUT en PATCH geen data meer terug. De client zal weten dat de opdracht gelukt is wanneer er de HTTP 204 code

teruggestuurd wordt.

Als laatste van de CRUD opdrachten blijft de DELETE over. De DELETE wordt uitgevoerd op een recourse met een id `www.scanjebon.nl/receipts/1` en verwijderd het object met dat id. Bij de uitvoering van een DELETE worden geen gegevens meegestuurd en krijgt dat doorgaans niet teruggestuurd. Wanneer het gelukt is krijgt de client een HTTP 200 code terug wanneer het gelukt is en er data wordt meegestuurd. Wanneer er geen data wordt teruggestuurd maar de opdracht wel gelukt is krijgt de response de 204 HTTP code terug.

Wanneer er fouten optreden tijdens de uitvoering van CRUD methodes op endpoints word dat opgemerkt door de HTTP status die wordt teruggestuurd. Wanneer er een aanvraag wordt gedaan op een endpoint die niet bestaat zal er een 404 teruggestuurd worden dat betekend dat het endpoint niet gevonden is. Voor fouten worden veelal codes in de 400 en 500 range gebruikt. [2]

Keuze

Uiteindelijk is SOAP erg uitgebreid en kan het alle kanten op geconfigureerd worden. Maar het is erg gecompliceerd, het configureren van SOAP binnen een applicatie vergt veel werk. Daarnaast werkt SOAP uitsluitend met XML en binnen iOS zijn daar nauwlijks API's op library's voor die daar mee werken. Dit zou betekenen dat alle parsen van alle binnenkomende gegevens op iOS zelf geparst moet worden. Daarnaast werken er maar heel weinig mensen die iOS development doen met SOAP waardoor ondersteuning op internet heel mager is.

Het werken met REST is voor iOS erg aan te raden. Bijna iedereen werkt hier mee en er zijn veel library's te vinden die hier iets mee kunnen. De API van iOS kan zelf goed overweg met JSON wat het direct naar gegevenstypes van Objective-C om kan zetten. Door dit alles is er ook gekozen om REST te gebruiken met JSON om dat REST erg simpel te maken is in tegenstelling tot SOAP en het voordeel van JSON ten opzichte van XML.

[2] HTTP 1.1 statuscodes en hun betekenis:
`http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`

4.3 Frameworks

5 Implementatie

6 Conclusie

A Persoonlijke ontwikkeling

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

B Planning