SKRIBL

Vrije Universiteit Brussel

WE-DINF-6537
Project Software Engineering
Academiejaar 2014-2015

Software Design Document

Douglas Horemans *<dhoreman@vub.ac.be>*
Hannah Pinson *<hpinson@vub.ac.be>*
Ivo Vervlimmeren *<ivervlim@vub.ac.be>*
Noah Van Es *<noahves@vub.ac.be>*
Pieter Steyaert *<psteyaer@vub.ac.be>*

19 november 2014

Versiegeschiedenis

Versie	Datum	Auteurs[1]	Beschrijving
0	19/11/2014	Noah Van Es	Aanmaak eerste versie voor iteratie 0
1	14/12/2014	Noah Van Es	Aanvullingen voor eerste iteratie Aanpassingen gebaseerd op feedback

Inhoudsopgave

1 Introductie

1.1 Doel

Dit document beschrijft de architectuur van de software achter SKRIBL. Het analyseert het ontwerp en haar (sub)compomenten en geeft een beschrijving van de werking van het systeem. Op die manier geeft het dus de nodige richtlijnen voor de implementatie en wordt de filosofie achter het design toegelicht. Daarnaast geeft het ook aan hoe gebruikers het systeem kunnen gebruiken, zowel via de grafische webinterface als via een aparte API voor third-parties.

1.2 Scope

SKRIBL is een webapplicatie die gebruikers toelaat om publicaties met elkaar te delen en te beheren op een interactieve manier. De bedoeling is dus dat men zowel eigen publicaties kan uploaden, alsook op zoek kan gaan naar andere publicaties en zo dus een volledig netwerk kan opbouwen binnen zijn/haar onderzoeksdomein. Het systeem moet hiervoor ook de nodige interface en visualisaties aanbieden, zowel in normale desktop-browsers als voor mobiele apparaten.

Hiervoor zal de software worden uitgerust met een databank en de nodige applicatielogica om aan alle software eisen te voldoen. Ook moet het systeem rekening houden met de webarchitectuur en bekijken hoe de modules en verantwoordelijkheden worden verdeeld onder de client en de server. Voor een volledig overzicht van de beoogde functionaliteit wordt verwezen naar het SRS [9].

1.3 Overzicht

Eerst en vooral wordt er in dit document gekeken naar de globale structuur van de software. Hierbij wordt het systeem eerst in grote lijnen besproken, waarna elk onderdeel vervolgens meer in detail wordt bekeken. Daarnaast wordt ook kort de filosofie van het ontwerp toegelicht. Vervolgens wordt er gekeken naar de organisatie van de informatie binnen het systeem. Hierbij worden onder andere een schema van de database en diagrammen meegegeven om aan te geven welke data hoe wordt opgeslagen en gebruikt. Daarna wordt op een systematische manier bekeken wat elk component binnen het systeem doet. Concreet wordt hier bijvoorbeeld aangegeven hoe een bepaalde methode of algoritme precies werkt.

Nadien volgt een beschrijving van de interfaces, zowel van de grafische webomgeving als van de API. In dit gedeelte kan men terugvinden hoe het systeem gebruikt kan worden en wordt het verloop van de mogelijke handelingen beschreven. Deze handelingen zijn voornamelijk gebaseerd op de requirements zoals opgesteld in het SRS [9].

De opmaak van dit document is gebaseerd op de IEEE 1016-1998 standaard voor SDD's [10].

1.4 Referentiemateriaal

[1] Alle versies worden nagekeken door de Document Manager.

[2] Angularjs. `https://angularjs.org/`.

[3] Express. `http://www.expressjs.com`.

[4] Material design. `http://www.google.com/design/spec/material-design`.

[5] Node.js. http://nodejs.org/.

[6] Orientdb. http://www.orientechnologies.com/orientdb.

[7] Oriento. https://github.com/codemix/oriento.

[8] Software project management plan. Technical report, SKRIBL.

[9] Software requirements specification. Technical report, SKRIBL.

[10] S. North. Software design document (sdd) template. http://cs.kennesaw.edu/snorth/CS_SE/SDD_Template.pdf.

1.5 Definities en Acroniemen

SRS Software Requirements Specification

SPMP Software Project Management Plan

SDD Software Design Document

MVC Model-View-Controller

API Application Programming Interface

GUI Graphical User Interface

SPA Single-Page Application

JSON JavaScript Object Notation

AJAX Asynchronous Javascript and XML

DMBS Database Management System

(E)ER (Extended) Entity-Relation

SQL Structured Query Language

REST Representational State Transfer

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

SSL Secure Sockets Layer

CSS Cascading Style Sheets

JS Javascript

2 System Overview

2.1 Context

De software wordt opgeleverd in de de vorm van een webapplicatie. Dit houdt in dat op de server een HTTP(S)-server zal draaien en de client vanuit de browser hiernaar requests zal sturen via een webinterface, ontworpen voor zowel mobiele als desktop-clients.

2.2 Organisatie

De software zal worden ontwikkeld volgens de principes van het Agile Development Proces. Zo zal de implementatie van de componenten die in dit document worden beschreven verlopen in verschillende iteraties en sprints. Voor een gedetailleerde beschrijving en planning van deze sprints wordt verwezen naar het SPMP [8].

2.3 Gebruikte technologieën

Om het systeem te ontwikkelen wordt gebruik gemaakt van een zuivere **'HTML/CSS/JS'**-stack. Daarnaast zullen ook volgende open-source frameworks en libraries worden gebruikt:

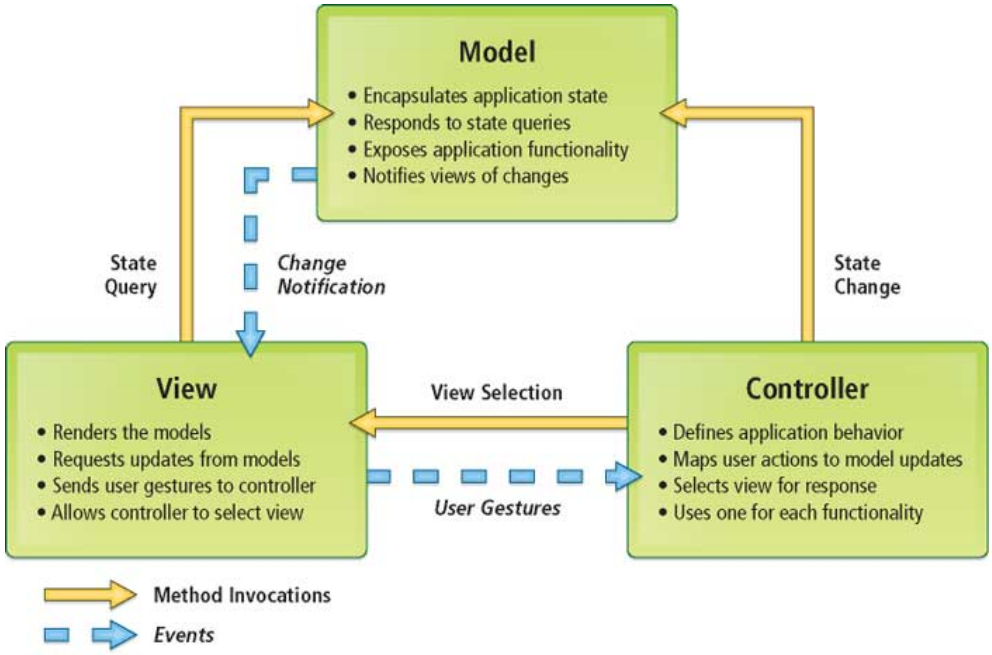
- **ExpressJS** zal worden gebruikt als framework om de webapplicatie te ontwikkelen en zo gemakkelijker een API voor de server te kunnen ontwerpen[3].
- **OrientDB** zal dan weer gebruikt worden als DMBS [6]. OrientDB is een zogeheten NoSQL-database, waarbij data niet wordt georganiseerd in tabellen zoals in een conventionele SQL-database. Specifiek zal gebruik worden gemaakt van de GraphDB-abstractie die OrientDB aanbiedt bovenop hun standaard database. Dit laat toe om om alle informatie rondom gebruikers, publicaties, ... in SKRIBL te modelleren als een graaf. Voor de communicatie over het binary protocol wordt gebruik gemaakt van oriento. [7]
- **AJAX** zal op de client worden gebruikt om op een asynchrone (en dus gebruiksvriendelijke) manier requests te sturen naar de server en vervolgens de UI te updaten. Deze technologie is van cruciaal belang om te voldoen aan de design filosofie die wordt beschreven in 3.1.
- Daarnaast zal voor de front-end gebruik worden gemaakt van **AngularJS**. AngularJS is een zeer flexibel MVC-framework dat HTML uitbreidt met specifieke directives, controllers, ... [2].

Ten slotte zal de server als onderliggende (open-source) runtime gebruik maken van **NodeJS**. NodeJS maakt het mogelijk om ook server-side Javascript te gebruiken met een performant event-driven (m.a.w. asynchroon, non-blocking) model [5].

3 System Architecture

3.1 Design Rationale

Deze sectie bespreekt eerst de algemene filosofie en structuur achter het ontwerp. Veel van deze principes zijn gebaseerd op het concept van 'Web 2.0'-applicaties. Er is gekozen om gebruik te maken van een MVC-structuur om de applicatielogica ('model') en de UI ('view') gescheiden te houden. Hieronder vindt men een grafische voorstelling van dit concept:



Figuur 1: Illustratie van het MVC-patroon

Hierbij zal de webserver voornamelijk de rol van M op zich nemen, terwijl de client de VC-rollen zelf zal beheren. We kunnen hier dus niet langer spreken over een thin, maar wel over een Rich client [1].

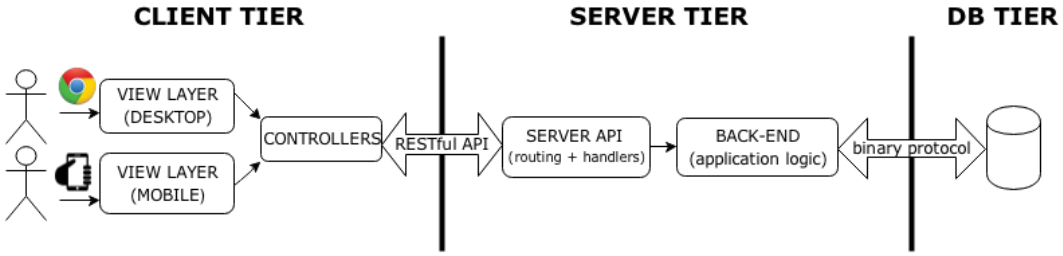
Op die manier zal de applicatie in feite het concept van een Single-Page Application (SPA) volgen, waarbij alle functionaliteit van de server wordt vrijgegeven via een RESTFul API. Hierdoor wordt het gemakkelijker om later aparte desktop en mobiele versies aan te maken en kunnen ook andere developers gebruik maken van SKRIBL.

3.2 Architectuur

Op basis van deze filosofie kan men de architectuur indelen in een three-tier model met een client tier, server tier, en database tier. Op de client tier gebeurt alle user-interactie met de view en

[1] Rich betekent hier niet noodzakelijk 'fat'; onder model wordt ook de applicatielogica rondom dit model verstaan, waardoor de server nog altijd deze belangrijke functies zal moeten implementeren.

wordt de front-end applicatie beheert via controllers. Deze zullen de nodige gegevens halen via de (RESTful) API van de server, die ook de voornaamste services aan de front-end zal leveren.
Voor de uiteindelijke interactie met de data zelf, zal deze server-tier dan ten slotte ook communiceren met de database-tier, waarop de OrientDB-database zich bevindt.
Hieronder vindt men een visualisatie van deze architectuur:



Figuur 2: Verschillende tiers in de webapplicatie

3.3 Decompositie

Hier word een beknopt overzicht gegeven van de verschillende onderdelen uit bovenstaande architectuur. Voor een gedetailleerde beschrijving van hun individuele componenten wordt verwezen naar sectie 5.

3.3.1 Client Tier

Views De views vormen hier de GUI voor de eindgebruiker en verschillen voor desktop en mobiele gebruikers. Om het concept van de Single-Page Application te verwezenlijken, zal de server daarom bij bezoek van de website een zogeheten template aanleveren, afhankelijk van het apparaattype van de gebruiker. Deze template zal gebruik maken van AngularJS en voldoende applicatielogica bevatten (geconfigureerd via de controllers) om tijdens het verloop van de applicatie zelf de interface te laten evolueren. De opmaak en structuur ervan gebeuren aan de hand van HTML & CSS.

Er zullen vanzelfsprekend aparte views nodig zijn om zowel mobiele als desktop-gebruikers te ondersteunen. Hierbij is het wel de bedoeling dat zoveel mogelijk code uit de controllers kan worden hergebruikt, wat mogelijk is dankzij de scheiding tussen view en controller die wordt aangeboden door het MVC-patroon.

Controllers De controllers zullen via de AngularJS-directive 'ng-controller' worden verbonden aan de views en handelingen van de gebruiker verwerken. Meestal houdt dit in dat een AJAX-call naar de server zal worden gemaakt en het resultaat (meestal in JSON-formaat) geparsed en gebruikt zal worden om zo de view te updaten.

De controllers dragen daarnaast ook de verantwoordelijkheid om de 'flow' van de applicatie te beheren. De server is (volgens de REST-principes) stateless en houdt dus geen sessiegegevens bij. Het is dus de taak van de controllers om bijvoorbeeld bij te houden welke gebruiker is ingelogd en waar hij/zij mee bezig is. In feite biedt de Server Tier enkel een interface aan voor de gegevens die in de applicatie moeten worden weergegeven.

3.3.2 Server Tier

Server API Dit gedeelte van de Server API is verantwoordelijk voor het afhandelen van requests van de client. Afhankelijk van het type request en de route (path) zal een bepaalde handler worden opgeroepen die gebruik zal maken van de back-end om de bijhorende applicatielogica uit te voeren en het juiste resultaat naar de client terug te sturen. Als ingangspunt van de server-tier is dit gedeelte dus ook verantwoordelijk voor het opstellen van de RESTful API.

Back-end De back-end voorziet dan weer de meeste functionaliteit van de applicatie. Hoewel de applicatie er voornamelijk in bestaat gegevens in het model correct op te slaan (via de database-tier), is er ook een aparte laag nodig voor de applicatielogica. Deze bevindt zich voor het grootste deel op de server en wordt op de back-end in aparte modules afgehandeld.

Hierbij gaat het vooral over de verwerking van de gegevens zoals gebruikers en publicaties. De back-end zal bijvoorbeeld zorgen voor de server-side validatie van gebruikersinput en zal ook de extractie van gegevens (metadata) uit publicaties op zich nemen. Ook complexere operaties op de database, die nodig zijn om bepaalde features te implementeren, zullen hier terug te vinden zijn.

3.3.3 Database Tier

Ten slotte zal op de database-tier een server worden opgezet met een database door OrientDB. Het volstaat hier om gewoon via de command-line een meegeleverd shell-script uit te voeren om deze server op te starten. Een gedetailleerde beschrijving van het ontwerp van deze database volgt in de volgende sectie.

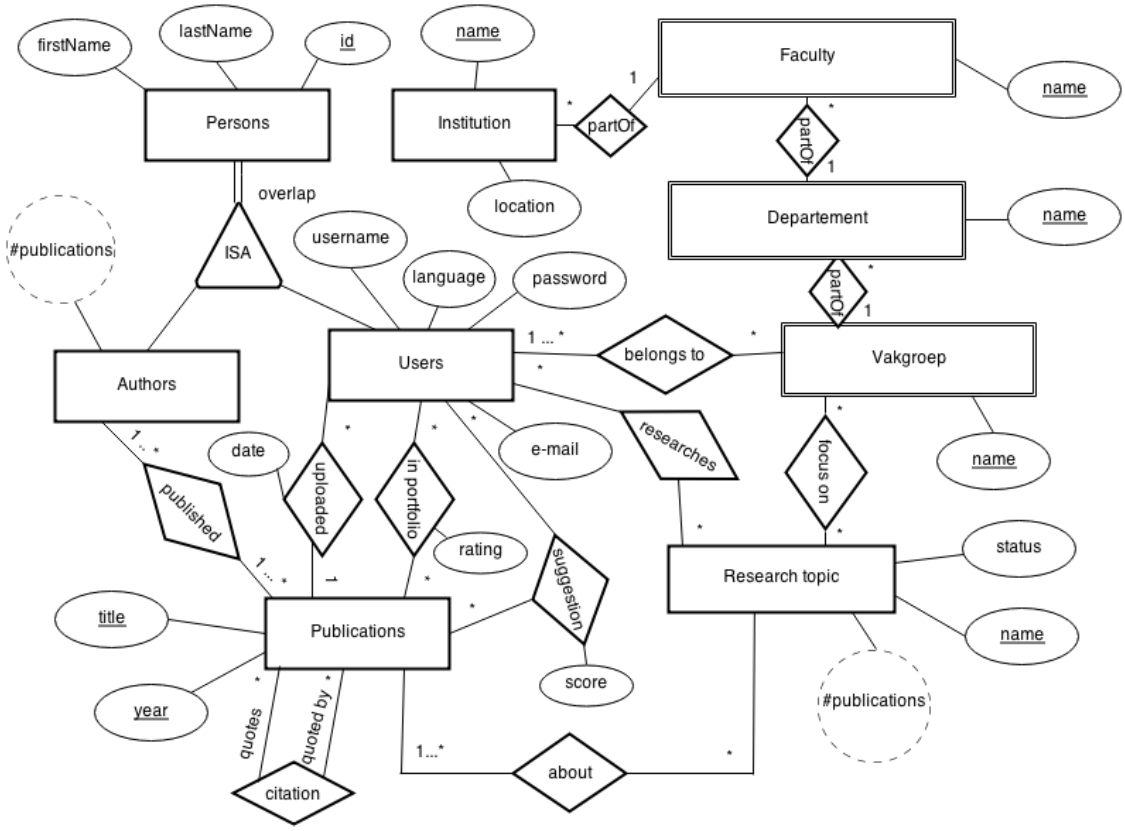
4 Data design

De organisatie van data is cruciaal bij het ontwerp van SKRIBL. Een efficiënte en persistente datastructuur is nodig om alle relaties tussen de verschillende entiteiten (gebruikers, publicaties, onderzoeksdomeinen, ...) in de applicatie te kunnen modelleren.

Om die reden werd gekozen om gebruik te maken van een graph-database. Een graf is veruit de meest natuurlijke representatie van het model en maakt het later ook gemakkelijk om verbanden in de applicatie te verwerken en te visualiseren.

Concreet werd er gekozen voor OrientDB omdat deze bovenop haar NoSQL-database ook een abstractie aanbiedt die specifiek is ontworpen om graph-databases in elkaar te steken. Daarnaast is het ook een zeer performante DBMS met een gebruiksvriendelijke API.

Wat volgt is een (E)ER-model van de huidige structuur in de database:



Figuur 3: Structuur van de graf in een (E)ER-model

[In de komende iteraties zal de entiteit publicatie verder worden uitgebreid in verschillende subklassen en zullen mogelijks meer attributen en entiteiten worden toegevoegd. Hiervoor zullen aparte ER-diagrammen worden gebruikt om het overzicht hier te bewaren.]

5 Component Design

In dit gedeelte worden de verschillende componenten die samen de applicatie vormen, verder toegelicht. Voor triviale componenten wordt enkel een korte beschrijving gegeven, maar indien nodig kan hier ook een volledige specificatie worden uitgewerkt.

[Momenteel zijn het aantal componenten en hun interacties vrij beperkt, later kunnen hier indien nodig eerst hun interfaces en relaties worden geïllustreerd in een klassediagram]

5.1 Server

De server zelf vormt het ingangspunt van de server-tier. Deze module kan geconfigureerd worden met bepaalde routes en zal zo de request naar een bepaalde handler delegeren (met de bedoeling om zo een RESTFul API samen te stellen). De handlers zullen op hun beurt dan weer gebruik maken van de back-end, waar de meeste applicatielogica van de server-tier zich bevindt en waar ook de communicatie met de DB-tier plaatsvindt.

Momenteel bevat dit component vooral methodes om de server te configureren en om ze op te starten met een SSL-certificaat (HTTPS). Daarnaast is er ook een mogelijkheid voorzien om de server te configureren met extra modules en om authenticatie toe te voegen aan bepaalde requests.

5.2 Routes

Routes zullen op de server worden gebruikt om de verschillende requests af te handelen.Op die manier wordt het configureren van de Server API gebruiksvriendelijker en overzichtelijker in de code. Routes kunnen gemakkelijk worden gedefinieerd en bevinden zich in de folder */server/routes*.

Een bepaalde route wordt gekenmerkt door een bepaald path (zoals /users) en verschillende handlers voor zowel GET, POST, PUT & DELETE requests op die route. Een handler is hierbij een functie die een request en response als argumenten nemen en het response-object correct configureert op basis van de request. Een route-object ondersteunt dan properties *get*, *post)*, ... om deze handlers te gebruiken.

5.3 Authentication

Deze module bevat een speciale functie die aan de hand van het HTTP-request zal bepalen welke user zich hier authenticeert (of hij geeft aan dat de credentials ongeldig zijn). Specifiek wordt hier gebruik gemaakt van Basic Authentication, waarbij de gebruiker zijn username:password over HTTPS meegeeft aan de server, geëncodeerd in base64.

Vervolgens kan men dan gebruik maken van de methode *useAuthentication* van de server om deze module te gebruiken authenticatie. Routes kunnen vervolgens aangeven of een geauthenticeerde gebruiker al dan niet toestemming heeft om een bepaalde request uit te voeren. Dit kan door aan de handlers een speciale property *auth* mee te geven. Op die manier wordt het gemakkelijk om een handler te beveiligen, wordt code-duplicatie vermeden en kan men gemakkelijk van authenticatiestrategie veranderen.

5.4 Validation

Deze module wordt in feite niet alleen op de back-end, maar ook op de front-end gebruikt om input van de gebruiker te valideren. Server-side validation is niet alleen aangewezen vanuit een design-standpunt, maar is ook noodzakelijk om veiligheidsredenen. Client-side validation zorgt dan weer voor een aangenamere en efficiëntere gebruikerservaring.

Concreet gaat hier in deze module vooral om functies die worden gebruikt om de geldigheid van een gebruikersnaam, wachtwoord, e-mailadres etc. te controleren. Zo zal men in deze module procedures zoals *validUsername(username)*, *validPassword(pwd)*, ... terugvinden. Dit gebeurt aan de hand van regular expressions, die standaard worden aangeboden door JavaScript. Daarnaast wordt ook een aparte module *researchDomain* gebruikt, waarin mogelijke onderzoeksdomeinen en disciplines, onderverdeeld in majors en minors, wordt geraadpleegd om na te gaan of het opgegeven onderzoeksdomein wel bestaat.

5.5 UserRecord

Deze klasse bevat alle noodzakelijke informatie over een bepaalde gebruiker in het systeem. Zo'n gebruikersinformatie kan ofwel worden geladen uit de database, ofwel kan via een constructor een nieuwe gebruiker worden aangemaakt die eerst alle input zal valideren en het wachtwoord zal hashen.

Het resulterende object kan dan gebruikt worden om gemakkelijk gegevens van de gebruiker op te vragen, of men kan aan de database een nieuw UserRecord geven om aan het systeem toe te voegen. Ook bij het laden van user zal de database zo'n object teruggeven. Naast de gebruikelijke kenmerken zoals (gebruikers)naam, taal, e-mailadres houdt dit record ook de affiliatie en onderzoeksdomeinen van de gebruiker bij.

Opermerking Veel van de geëxporteerde methodes in deze module zijn asynchroon en verwachten dus een callback bij oproep. Dit patroon zal ook in andere modules (zoals dat van de database) voorkomen en is inherent aan web-applicaties in NodeJS. Bij het laden van de gebruiker uit de database via *loadUser(db, name, callback)* zal zo bijvoorbeeld niet meteen een nieuw gebruikersobject worden aangemaakt, maar zal de meegeleverde callback worden opgeroepen met de gebruikersinfo als deze is geladen.

5.6 PublicationRecord

Een PublicationRecord bevat, op deze manier als een userRecord, alle nodige metadata over een publicatie. Opnieuw is er een aparte constructor voorzien die ondere andere een path neemt naar de nieuwe publicatie, hieruit de nodige gegevens extraheert en vervolgens zo volledig mogelijk een record aanmaakt.

Vervolgens kan men met zo'n object specifieke metadata van de publicatie opvragen a.d.h.v. getters en kan men ook vragen om een filepath naar het bestand te genereren. Deze records worden ook gegenereerd door de database wanneer men een publicatie opvraagt en worden eveneens verwacht om nieuwe publicaties aan te maken.

5.7 Database

In deze module vindt men een abstractie voor alle communicatie met de database. Het implementeert een bepaalde interface die onder andere door de Server- en UserInfo-modules wordt verwacht. Concreet is het Database-object verantwoordelijk voor het opzetten van een databaseverbinding en het uitvoeren van de nodige queries voor de interactie met het model. Zo worden er methodes aangeboden die onder andere nieuwe gebruikers toevoegen, gegevens van bestaande gebruikers opvragen of een nieuwe publicatie opslaan.

Om de code hiervoor te organiseren, zijn er voor elke entiteit aparte modules gemaakt die aan de database kunnen worden toegevoegd en die verantwoordelijk zijn voor een bepaald deel van haar functionaliteit. Op die manier wordt vermeden dat deze klasse onoverzichtelijk veel code op zich moet nemen en kan men gemakkelijk een bepaald deel van de databaselogica terugvinden.

Ten slotte bevat de database ook redelijk wat private methodes; deze zijn nodig om alle gegevens correct op te slaan volgens het (E)ER-diagram uit 3. Zo worden in de database ook al alle lagen van de affiliation (researchGroup, department, faculty, institution) correct bijgehouden. Op dezelfde manier worden ook researchDomains gemodelleerd op de database als aparte entiteiten.

De communicatie met de server waarop de database zich bevindt, gebeurt aan de hand van een binary protocol dat OrientDB aanbiedt. Hiervoor wordt gebruik gemaakt van de open-source library Oriento, zodat de queries gewoon kunnen worden uitgevoerd vanuit JS.

6 Software domain design

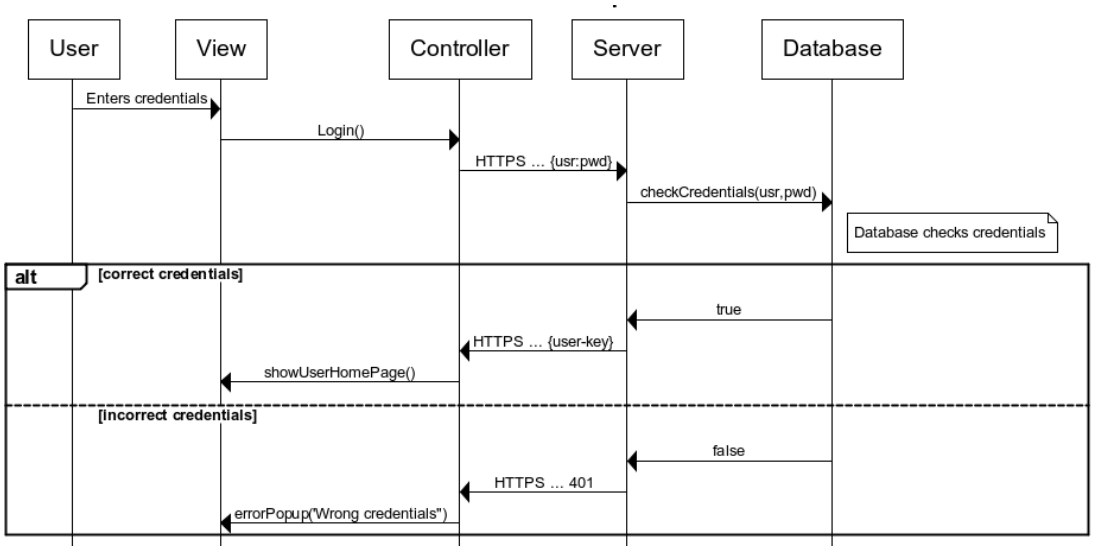
In dit onderdeel wordt bekeken hoe de verschillende componenten in het systeem samenwerken om een bepaalde functionaliteit te verwezenlijken. Op die manier wordt duidelijk gemaakt welke verantwoordelijkheid elk onderdeel in het systeem heeft. Meestal zal de beschrijving worden gegeven aan de hand van een sequentiediagram of (in het geval van een algoritme) pseudocode.

Voor een volledig overzicht van alle benodige functionaliteit wordt verwezen naar het SRS [9].

6.1 Inloggen

Bij het inloggen zal de client eerst een request sturen naar de server met gebruikersnaam en wachtwoord toegevoegd in plaintext. Indien correct, zal de server een id-string terugsturen naar de client die hij later bij elke request die authenticatie vereist moet toevoegen. Volgens de principes van een RESTful ontwerp, mag er echter geen (session) state op de server worden bijgehouden en zal deze unieke identificatiestring dus niet gebonden zijn aan een bepaalde sessie. Momenteel is het gewoon een encodering van de gebruikersnaam en wachtwoord in base64 (= Basic Access Authentication). Uit veiligheidsoverwegingen zal voor de persistente opslag van wachtwoorden in de databasenk wel encryptie worden gebruikt.

Uit deze beschrijving kunnen we ook al afleiden dat het zeker nodig zal zijn HTTPS te gebruiken om zo alle communicatie tussen client en server te encrypteren met SSL (weliswaar met een self-signed key). Zoniet zou het niet veilig zijn om wachtwoorden (geëncodeerd of niet) door te sturen.



Figuur 4: Vereenvoudigd sequentiediagram bij het inloggen

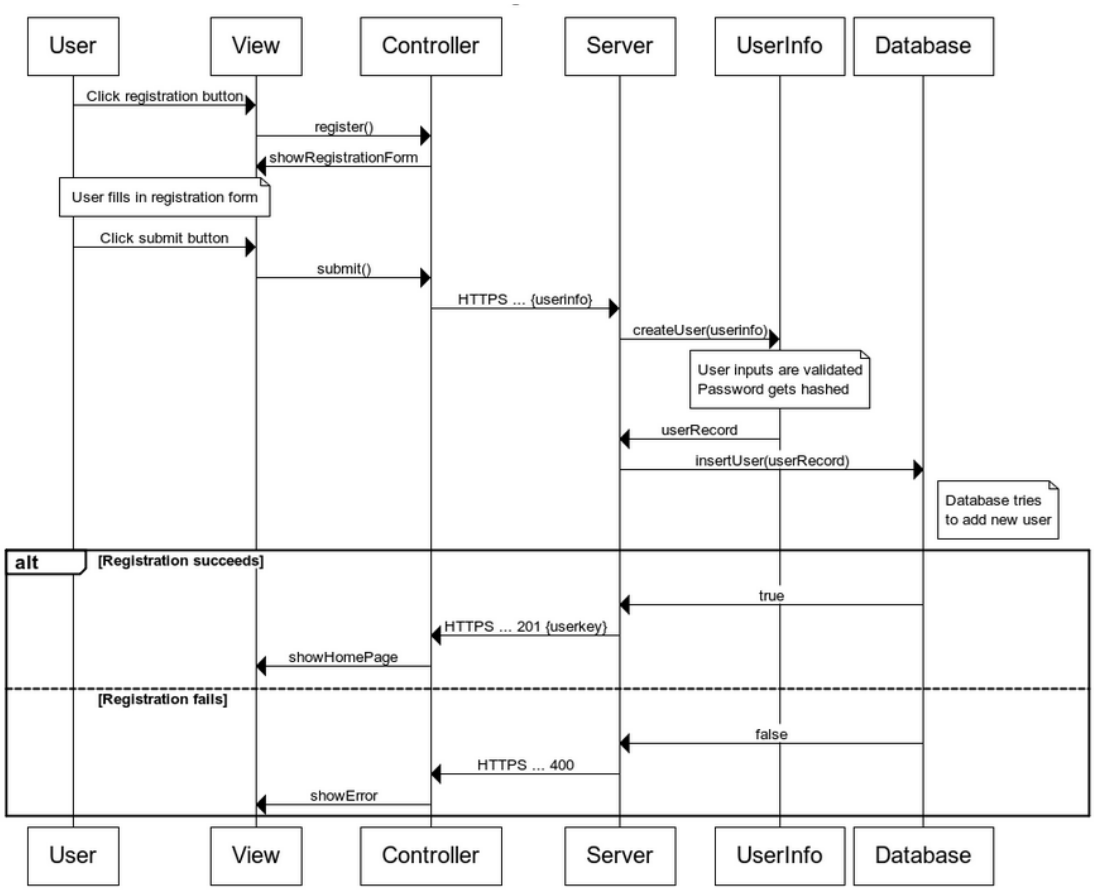
Opmerking Bovenstaand sequentiediagram is slechts een vereenvoudiging van wat er in werkelijkheid gebeurt, zo zijn bijvoorbeeld procedures zoals checkCredentials in werkelijkheid asynchroon, en zullen ze dus geen boolean teruggeven, maar werken ze met callbacks.

Deze opmerking geldt ook voor alle andere sequentiediagrammen in dit onderdeel.

6.2 Registreren

Om een nieuwe gebruiker te registeren, dient men eerst een registratieformulier in te vullen. Om de gebruikerservaring hierbij de verbeteren zal dit gebeuren met client-side validation (naast de server-side validation, die noodzakelijk is).

Van zodra alle informatie is ingevuld, zal de client een identificatiestring terugkrijgen indien de nieuwe gebruiker succesvol is toegevoegd. Zoniet, zal de server een HTTP 40x-error teruggeven.



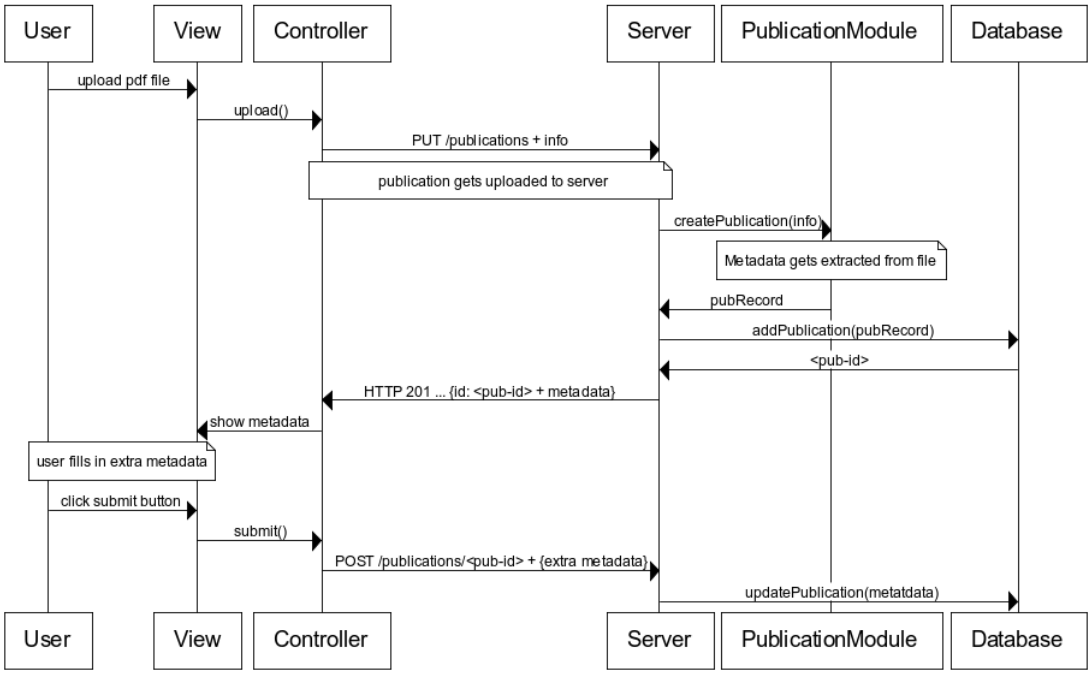
Figuur 5: Vereenvoudigd sequentiediagram bij het registreren

6.3 Publicaties toevoegen

Een nieuwe publicatie toevoegen gebeurt door een combinatie van de HTTP PUT en POST commando's. Eerst en vooral dient men de publicatie zelf te uploaden met HTTP PUT. De server zal aan de hand van dit bestand dan proberen zoveel mogelijk metadata te extraheren en aan te vullen,

alvorens het bestand uiteindelijk op te slaan in de database. Vervolgens zal de client een id en de afgeleide metadata terugkrijgen van deze nieuwe publicatie. Dit id kan hij dan gebruiken met een HTTP POST operatie om daarna de gegevens verder aan te vullen in het systeem.

Hieronder volgt opnieuw een sequentiediagram die dit proces illustreert. Om het overzichtelijk te houden, werden de scenario's waarin er foutmeldingen worden gegenereerd niet gemodelleerd in dit diagram.



Figuur 6: Vereenvoudigd sequentiediagram bij het toevoegen van publicaties

7 Human Interface Design

7.1 Filosofie

Het front-end team van SKRIBL heeft een duidelijke visie over de uiteindelijke gebruikerservaring. Zo zal worden ingezet op een flat, anti-skeumorphistisch ontwerp dat futuristisch oogt. Daarnaast zullen ook de principes van 'interfaceless interface' en de 'three-click rule' worden gevolgd en zal gewerkt worden met vectoriële graphics in plaats van bitmaps. Het eindresultaat zal bijgevolg veel weg hebben van een 'Material Design' [4].

7.2 Overzicht

Aangezien in de eerste sprint enkel inloggen en registreren werden geïmplementeerd, is de functionaliteit van de grafische interface momenteel vrij beperkt.

Hieronder vindt men de voorlopige functionaliteit van de UI:

Inloggen Op de startpagina zal de gebruiker twee inputvelden terugvinden om zijn/haar naam en wachtwoord (niet textueel zichtbaar) in te geven. Daarna dient men enkel op de knop 'inloggen' te klikken en zal bij een correct wachtwoord het dashboard van de gebruiker worden weergegeven, bij een incorrect wachtwoord zal een foutmelding worden getoond.

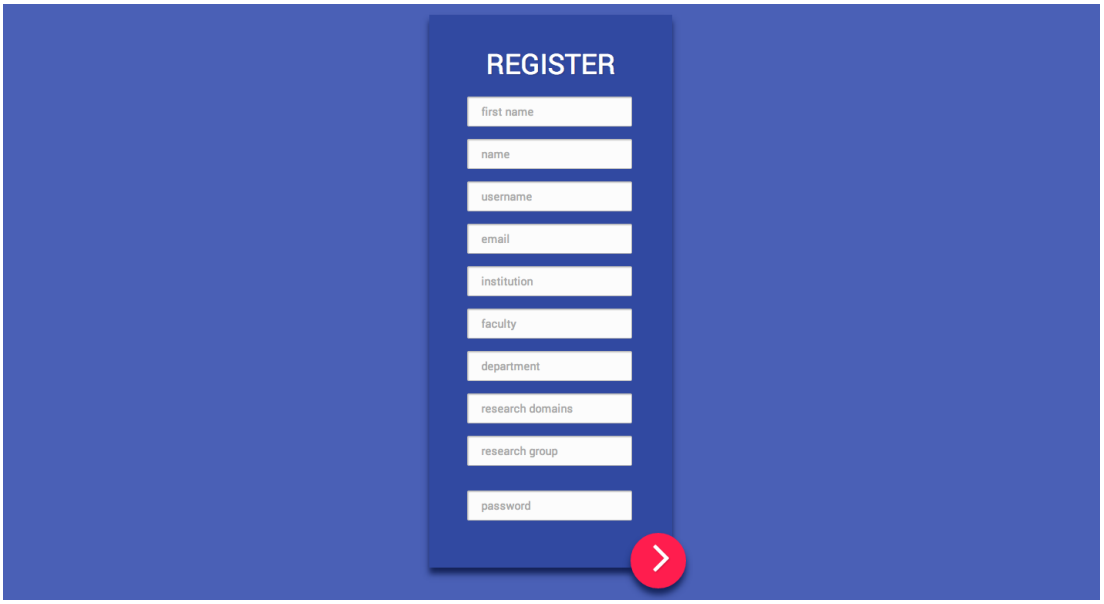
Dashboard Het dashboard is de startpagina van de gebruiker vanwaar men een overzicht kan krijgen van de meest relevante informatie voor de gebruiker (zoals recente publicaties). Momenteel is het enkel voorzien van enkele knoppen waarmee de gebruiker zich ondere andere terug kan uitloggen of zijn account kan verwijderen. Daarnaast is het ook al mogelijk om hier publicaties te uploaden, hoewel er nog geen mogelijkheid is om geüploade publicaties later terug te vinden.

Registreren Registratie gebeurt van zodra de gebruiker op de knop 'registreren' klikt en het registratieformulier invult. Om de gebruikerservaring te verbeteren zal er ook meteen client-side validatie gebeuren voordat men een aanvraag tot registratie kan verzenden. Daarna komt de gebruiker ofwel terecht op zijn nieuw dashboard, ofwel krijgt hij een foutmelding te zien wanneer het toevoegen is mislukt.

7.3 Screenshots

Hieronder worden ten slotte nog enkele screenshots meegegeven om het design te illustreren.

Figuur 7: Screenshot van de startpagina



Figuur 8: Screenshot van het registratieformulier

8 Externe API

Naast de grafische interface, biedt het systeem ook een API aan voor externe ontwikkelaars die hun eigen applicaties willen integreren met SKRIBL. Dit gebeurt met RESTful API over HTTP(S).

POST /login Deze request wordt gebruikt om in te loggen, men geeft de gebruikersnaam en het wachtwoord (over HTTPS) mee in JSON-formaat en krijgt een unieke identificatiestring terug van de server die men later moet meegeven bij bepaalde requests.

GET /user/<username> Op deze manier kan men in JSON-formaat relevante informatie van een bepaalde gebruiker in het systeem opvragen.

PUT /user/<username> Met deze request kan een nieuwe gebruiker op de server worden aangemaakt. Alle nodige gebruikersinformatie dient te worden meegegeven bij de request als JSON-data.

DELETE /user/<username> Hierbij dient men expliciet zijn/haar identificatiestring mee te geven. Het resultaat is dat de aangeduide gebruiker uit het systeem zal worden verwijderd.

PUT /publications Hiermee kan men een nieuwe publicatie uploaden in het systeem. Het volstaat om enkel het bestand en naam mee te geven, later kan metadata verder worden aangevuld. De server zal bij deze request een uniek id van de publicatie teruggeven, samen met de gegevens die werden geëxtraheerd.

GET /publications/<pub-id> Haalt een publicatie van de server met een opgegeven id. In de headers wordt extra metadata worden meegegeven.

POST /publications/<pub-id> Hiermee kan men de metadata van publicaties updaten, deze dient meegeleverd te worden in JSON-formaat. Voor deze operatie dient men zich tevens te authenticeren als oorspronkelijke uploader van de publicatie.

DELETE /publications/<pub-id> Deze methode zal een publicatie van de server verwijderen. Ook hier dient men zich correct te authenticeren.

GET /publications Hierbij wordt verwacht dat men een aparte query-string meegeeft om op zoek te gaan naar een bepaalde publicatie. Het resultaat is dan een JSON-array van publicatie-id's die voldoen aan de criteria.
bijvoorbeeld: /publications?author=hjespers of /publications?topic=Physics&year=2014