Software Design Document

Mathieu Reymond, Arno Moonens

Maart 2015

Inhoudsopgave

1 Versiegeschiedenis

Versie	Commentaar
1	Design van eerste iteratie
2	Design van tweede iteratie

2 Definities

Table 1: Definities

API	Application Programming Interface
REST	Representational state transfer
JSON	JavaScript Object Notation
SDD	Software Design Description

3 Introductie

3.1 Doel en Scope

Dit document heeft als doel om de software-architectuur van de WiseLib-applicatie uit te leggen aan de hand van een veralgemeende beschrijving van het systeem.

Dit document zal geraadpleegd worden door de testers en door programmeurs tijdens het implementeren van de applicatie.

WiseLib wordt op een iteratieve manier gemaakt. Dit betekent dat we in de eerste versie beginnen met slechts een beperkt aantal functionaliteiten te implementeren. In volgende iteraties wordt onze applicatie dan telkens uitgebreid en worden de functionaliteiten verbeterd en worden er nieuwe toegevoegd.

Dit document volgt de IEEE Std 1016-2006^{TM}"Standard for information technology — systems design — software design descriptions" standaard.

4 Logica

De Wiselib applicatie is onderverdeeld in twee grote onderdelen: de backend (4.1) en de frontend (4.2). De backend beheert de data van het systeem door gebruik te maken van een database en implementeert de core functionaliteiten van het programma. De frontend toont de data aan de gebruiker, en laat toe om onrechtstreeks met deze data te interageren. De communicatie tussen deze twee onderdelen gebeurt via een op voorhand afgesproken protocol : een RESTful[1] API[2] met als media type het JSON formaat.

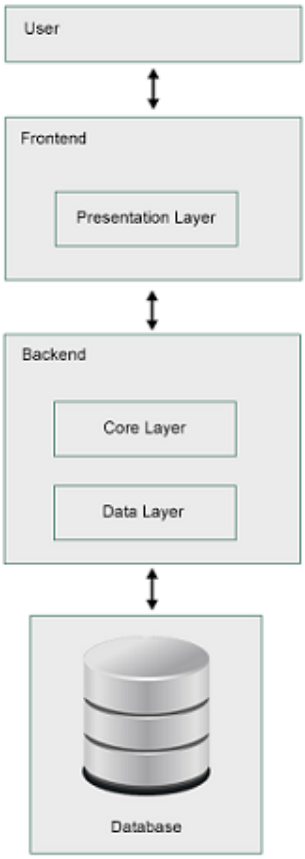

Figure 1: De verschillende layers van de applicatie

4.1 Backend

De backend bestaat uit drie grote modules:

- De database module (zie 4.1.2).
- De core module (zie 4.1.1).
- De communicatie module (zie 4.1.3).

De core module implementeert de functionaliteit van het systeem. De data is hier door verschillende klassen voorgesteld, waarmee de logica van het systeem uitgevoerd kan worden. Deze module is volledig onafhankelijk van de andere modules.

De database module is verantwoordelijk voor de interactie tussen de applicatie en de database. Het zet de gevraagde database-data om naar core-objecten. Het maakt dus gebruik van de core-module. Het interageren met deze module gebeurt ook via de RESTful[1] API[2].

De communicatie module interageert met de buitenwereld door middel van de API. Geldige requests worden behandeld door gebruik te maken van de database module. Eventuele berekeningen gebeuren dan via core-klassen, en het antwoord wordt dan omgezet naar een JSON formaat en teruggestuurd.

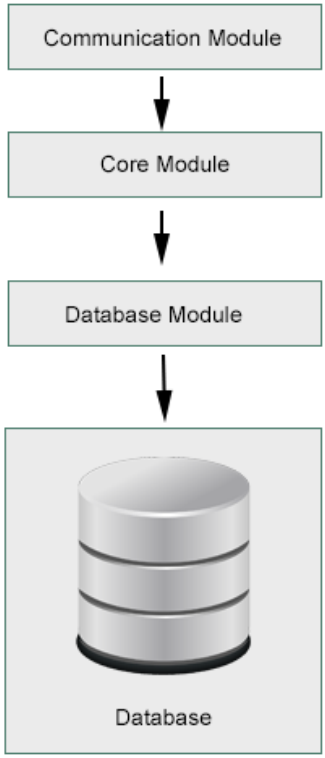

Figure 2: Het Module Schema

4.1.1 Core Module

De core-module bestaat uit verschillende klassen. Gebruikers zijn voorgesteld door de klasse `User`. De applicatie maakt een onderscheid tussen een gebruiker en een persoon `Person`. Het is namelijk mogelijk dat een auteur in de gegevensbank opgeslagen is, maar niet als gebruiker is ingeschreven. Dit kan bijvoorbeeld gebeuren wanneer een gebruiker een publicatie met co-auteurs uploadt. Publicaties zijn voorgesteld door de klasse `Publication`. Er wordt een onderscheid gemaakt tussen twee type publicaties: `JournalPublication` en `ProceedingPublication`. Dit onderscheid is noodzakelijk omdat deze publicaties verschillende karakteristieken hebben. `ProceedingPublication`s hebben bijvoorbeeld publishers en editors, wat niet het geval is bij `JournalPublication`s.

Omdat de rank van een journal of een proceeding een invloed heeft op de rank van een publicatie worden zij ook door hun eigen klasse gerepresenteerd (respectievelijk `Journal` en `Proceeding`).

Journals en proceedings zijn gespecialiseerd in specifieke onderwerpen, voorgesteld door de `Discipline` klasse. Deze kunnen onderdeel zijn van andere onderwerpen, `Discipline`s hebben dus een boom-structuur. Gelijkaardig hebben de affiliaties (`Affiliation` klasse) van auteurs ook een boom-structuur, die structuur is door de klasse `Node` voorgesteld.

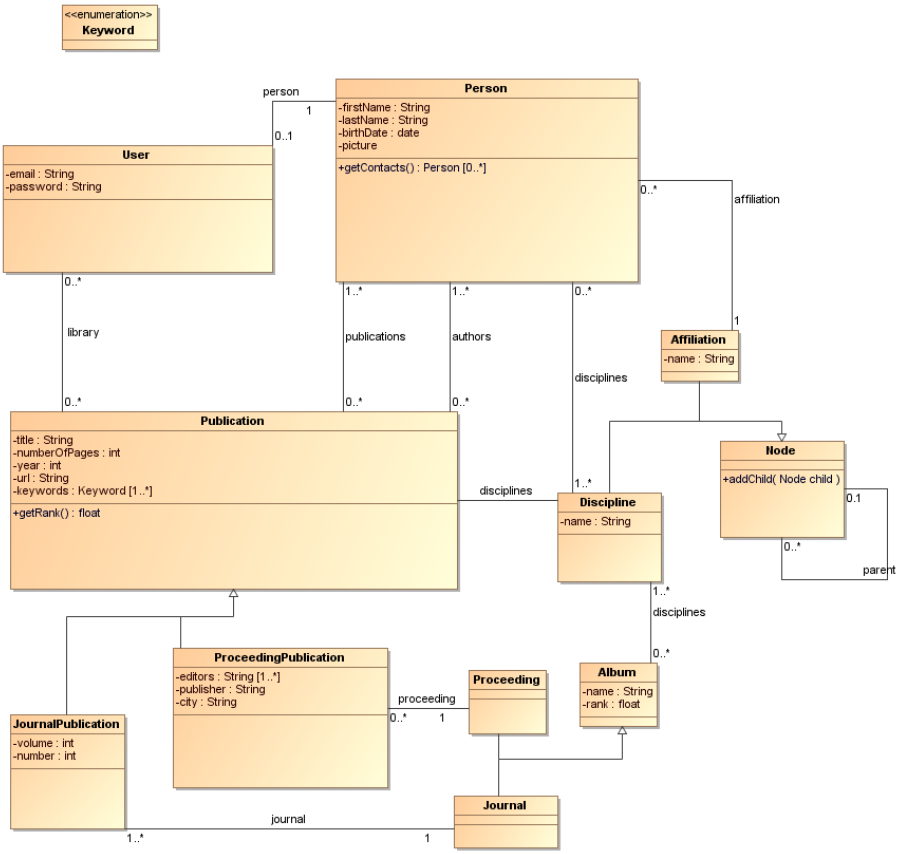

Figure 3: Het Core Klasse diagram

4.1.2 Database Module

De verschillende klassen van de core-module worden in een database opgeslagen. De `Linker` klasse mapt elk JSON object aan zijn database representatie: de verschillende attributen van het object worden gemapt tot hun database velden en in welke tabel ze zich bevinden. Elke `Linker`-representatie heeft ook `get`, `post`, `put` en `delete` attributen die teruggeven welke velden deelnemen aan die queries. Deze worden dan omgezet naar geldige SQL-queries via de `queryGet`, `queryPost`, `queryPut` en `queryDelete` methoden. De `DBManager` maakt gebruik van de linker om met de database te interageren. Deze interactie gebeurt asynchroon. Er is dus nood aan een callback functie, mee te geven aan de methoden van deze klasse. De `DBManager` heeft vier methodes die overeenkomen met de REST architectuur:

- get(jsonObj, linkerRepr, next): Deze methode zoekt in de database.
  - `jsonObj`: de methode zoekt naar data die overeenkomt met de variabelen die in jsonObj gegeven worden.
  - `linkerRepr`: De `Linker` representatie van het JSON object. Deze gaat de correcte SQL queries opstellen die dan in deze methode uigevoerd zullen worden.
  - `next`: De callback functie die opgeroepen wordt aan het einde van de methode. Hij heeft één parameter: een `Array` van core-klassen die matchen op `jsonObj`.
- post(jsonObj, linkerRepr, next): Deze methode voegt nieuwe elementen (rijen, tabellen,... indien nodig) toe aan de database.
  - `jsonObj`: Een JSON object dat volgens de API gedefinieerd wordt. Dit wordt toegevoegd aan de database.
  - `linkerRepr`: De `Linker` representatie van het JSON object. Deze gaat de correcte SQL queries opstellen die dan in deze methode uigevoerd zullen worden.
  - `next`: De callback functie die opgeroepen wordt aan het einde van de methode. Hij heeft één parameter: Het `id` van het toegevoegde `jsonObj`.
- put(jsonObj, linkerRepr, next): Deze methode update bestaande elementen.
  - `jsonObj`: Een JSON object dat volgens de API gedefinieerd wordt. De methode vervangt de velden van de database met diegene die in het JSON object aanwezig zijn.
  - `linkerRepr`: De `Linker` representatie van het JSON object. Deze gaat de correcte SQL queries opstellen die dan in deze methode uitgevoerd zullen worden.

  - `next`: De callback functie die opgeroepen wordt aan het einde van de methode. Hij neemt geen parameters.

- delete(jsonObj, linkerRepr, next): Deze methode verwijdert bestaande elementen.
  - `jsonObj`: Een JSON object dat volgens de API gedefinieerd wordt. De methode verwijdert het object uit de database dat voldoet aan de parameters beschreven in het `JSON` object.
  - `linkerRepr`: De `Linker` representatie van het JSON object. Deze gaat de correcte SQL queries opstellen die dan in deze methode uigevoerd zullen worden.
  - `next`: De callback functie die opgeroepen wordt aan het einde van de methode. Hij neemt geen parameters.

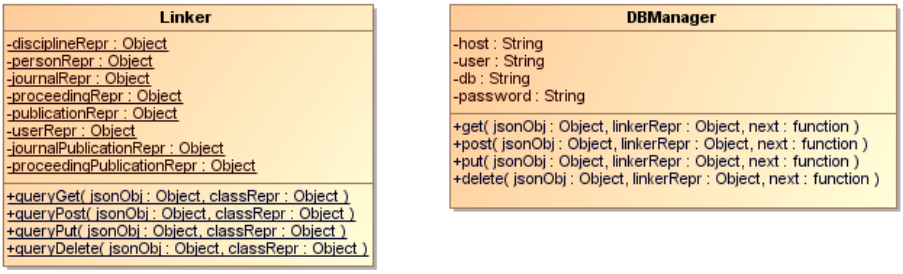

Figure 4: Het Database Klasse diagram

4.1.3 Communicatie Module

De communicatie module handelt requests af van de buitenwereld. De requests zijn gedefineerd door de API[2]. De JSON expressies die binnenkomen kunnen ge*parse*d worden naar objecten en behandeld worden door de database module. Core-objecten kunnen via de `JSONManager` klasse omgezet worden naar een JSON object dat overeenkomt met de API.

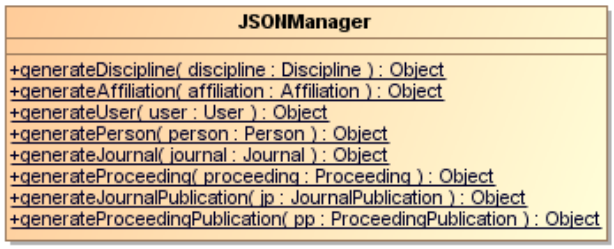

Figure 5: Het Communicatie Klasse diagram

4.2 Frontend

Voor de frontend hebben we gebruik gemaakt van *AngularJS* en *Angular-Material*. De frontend zorgt voor interactie met de gebruiker en bestaat grofweg uit 3 delen: *models*, *views* en *controllers*. Zoals te zien in onderstaand figuur zorgen *controllers* voor de verbinding tussen een *view* en een *model*.
Een *view* bevat html-pagina's en css code. Een *model* (bij *AngularJS* zogenaamde *services* en *factories*) voert bepaalde *bussiness*-logica uit die te complex of uitgebreid is om in een *controller* te definiëren.
De data die meegegeven wordt aan de presentatielaag en ontvangen wordt van de *server* is telkens in *JSON* formaat.
Als een gebruiker een actie uitvoert op een pagina (een *view*) wordt dit doorgegeven aan de bijhorende controller. Deze gaat dan een functie uitvoeren van een bepaald *model*. Dit *model* kan indien nodig communiceren met de *server*. De controller gaat de *view* dan aanpassen om het resultaat van het *model* weer te geven.
Naast gewone *views* (om een bepaalde pagina te tonen), zijn er ook *directives* die op verschillende pagina's kunnen getoond worden. Dit zijn een soort templates waarbij met bijvoorbeeld info over een bepaalde persoon of publicatie kan weergeven zonder elke keer de expliciet alles te formuleren van wat er telkens over moet te zien zijn.

De eerste *controller* die uitgevoerd wordt als de gebruiker de applicatie bezoekt is de *routeProvider*. Deze gaat de juiste *view* (*html*-pagina) voor de presentatielaag laden, samen met de bijpassende *controller*.

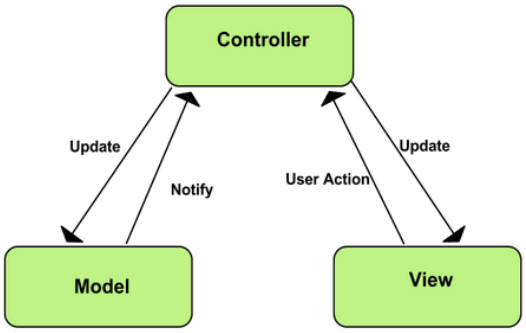

Figure 6: Het *model-view-controller* schema

We beschrijven nu elke pagina apart.

4.2.1 Startpagina en menu

De startpagina zelf toont slechts een welkomsttekst.
Het menu past zich aan naargelang je ingelogd bent of niet. Als je niet ingelogd bent zie je knoppen om je in te loggen, te registreren. Daarbuiten is de knop om een publicatie te uploaden ook aanwezig maar niet aanklikbaar.
Als je wel ingelogd bent zie je naast de reeds genoemde knop (die nu uiteraard wel aanklikbaar is) ook een knop om je uit te loggen, een knop om je accountgegevens te wijzigen en een knop je geüploadete publicaties te bekijken.

4.2.2 Inlogpagina

Hierbij wordt simpelweg het email adres en wachtwoord via de controller en de inlog service naar de gebruiker gestuurd en verwacht men een token of een error als antwoord. De token wordt dan in de huidige sessie opgeslagen. Hierna wordt de gebruiker naar de startpagina gestuurd en wordt er een bericht getoond dat men succesvol is ingelogd. Een error wordt rechtsboven op het scherm getoond ingeval van een foutief emailadress of paswoord.

4.2.3 Registratiepagina

Allereerst wordt er naar een voor- en achternaam gevraagd. Als er voldoende karakters worden ingegeven wordt er naar personen met de ingegeven naam in de database gezocht. Indien er personen gevonden zijn worden deze in een lijst weergegeven en kan men er een van selecteren. Als men geen persoon geselecteerd heeft kan men ook een profiel afbeelding meegeven.
Vervolgens dient men een geldig email adres en paswoord (met minstens acht tekens) mee te geven.
Als er een plobleem was met het registreren krijg je net als bij een error op de loginpagina een bericht te zien met de precieze fout. Anders wordt de gebruiker naar de startpagina gestuurd en krijg je een bericht dat de registratie succesvol was.

4.2.4 Publicatie upload pagina

Deze pagina bevat een formulier om als ingelogde gebruiker een publicatie te kunnen uploaden.
Allereerst moeten de titel, jaar van publicatie, aantal pagina's, url en samenvatting (*abstract*) van de publicatie meegegeven worden.
Hierna kan je co-auteurs meegeven. Deze worden net als een persoon bij het registreren opgezocht terwijl je typt. Je kan gevonden personen dan makkelijk toevoegen of verwijderen door er op te klikken.
Al deze info kan ook zo veel mogelijk automatisch ingevuld worden door het uploaden van de publicatie als pdf bestand.
Daarna kiest men wat voor publicatie men wil uploaden: een *journal* publicatie of een *proceeding* publicatie.

Bij een *journal* publicatie geef je mee in welke *journal* de publicatie is verschenen en in welk volume en nummer precies.
Voor een *proceeding* publicatie geef je mee voor welke *proceeding* de publicatie is verschenen, door welke *editor* en *publisher* en in welke plaats de *proceeding* plaats vond.
Tenslotte kan je referenties toevoegen die in de publicatie gemaakt zijn. Dit kan eveneens automatisch gebeuren door het uploaden van een *bibtex* file.

5 Referenties

References

[1] *RESTful* http://en.wikipedia.org/wiki/Representational_state_transfer

[2] *Wiselib API* http://wilma.vub.ac.be/~se2_1415/api.html