# Software Design Document

Mathieu Reymond, Arno Moonens

December 2014

## 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.

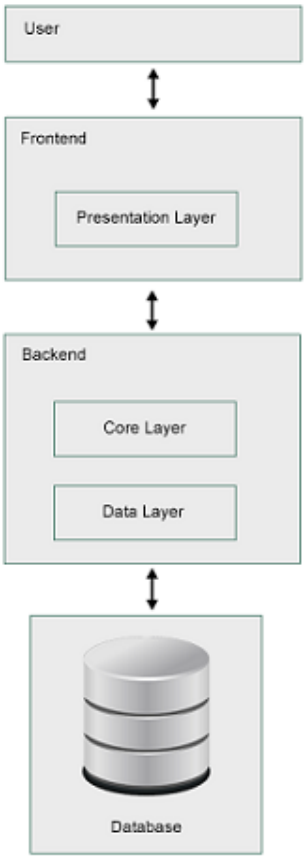


Figure 1: De verschillende layers van de applicatie

## 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.

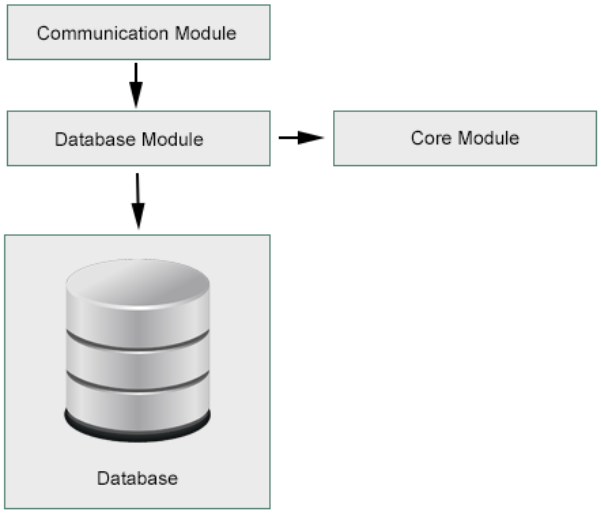


Figure 2: Het Module Schema

### 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`).

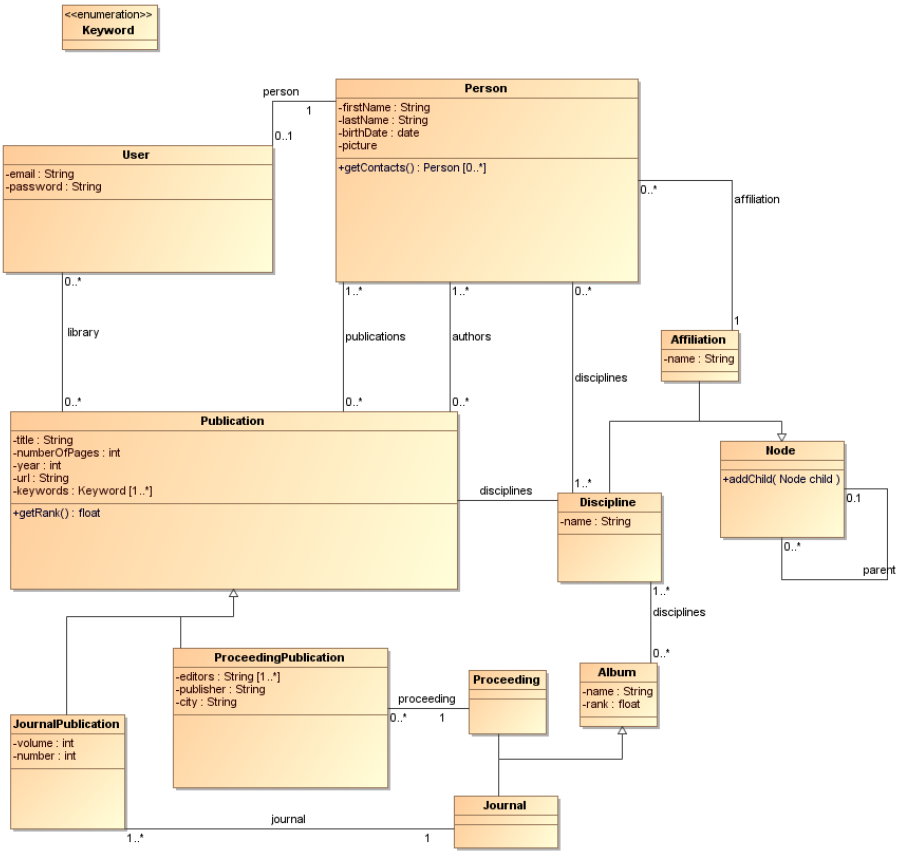


Figure 3: Het Core Klasse diagram

### 4.1.2 Database Module

De verschillende klassen van de core-module worden in een database opgeslagen. De interactie met de database gebeurt via de `DBManager` klasse. De interactie gebeurt asynchroon. Er is dus nood aan een callback functie, mee te geven aan de methoden van deze klasse. Deze heeft voor elke core-klasse vier methodes die overeenkomen met de REST architectuur:

- getClassName(params, next): Deze methode zoekt in de database.
  - `params`: de methode zoekt naar data die overeenkomt met de variabelen die in params gegeven 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 `params`.
- postClassName(jsonObj, 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.
  - `next`: De callback functie die opgeroepen wordt aan het einde van de methode. Hij heeft één parameter: Het `id` van het toegevoegde `jsonObj`.
- putClassName(jsonObj, 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.
  - `next`: De callback functie die opgeroepen wordt aan het einde van de methode. Hij neemt geen parameters.
- deleteClassName(jsonObj, 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 dit `JSON` object.
  - `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.

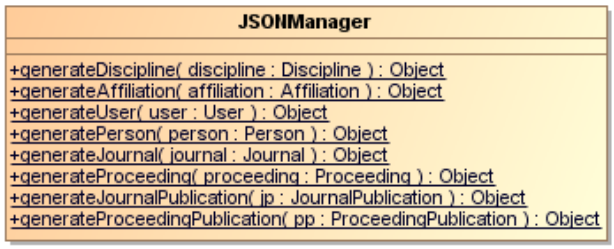


Figure 5: Het Communicatie Klasse diagram

## 4.2 Frontend

De interactie met de gebruiker gebeurt via de presentatie laag. De verbinding tussen de *server* en de presentatie laag wordt verzorgd door de *controller*s. Deze geven mee aan de presentatielaag welke data er moet weergegeven worden en halen die data zo nodig van de *server*. Als de gebruiker een actie uitvoert op de presentatielaag, wordt dit doorgegeven aan de juiste *controller* en gaat deze indien nodig met de server communiceren en de weer te geven data veranderen. De data die meegegeven wordt aan de presentatielaag en ontvangen wordt van de *server* is telkens in *JSON* formaat. De *controller* verwerkt dit dan en zet het om naar tekst om weergegeven te kunnen worden op de presentatielaag.

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*.

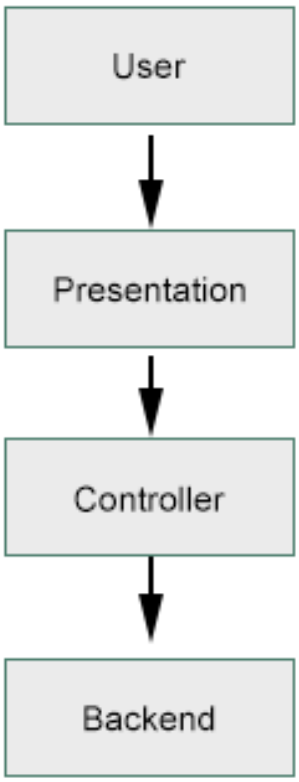


Figure 6: Het Interactie Schema

# 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