SKRIBL

Vrije Universiteit Brussel

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

Software Test Plan

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*>

2 maart 2015

Versiegeschiedenis

Versie	Datum	Auteurs[1]	Beschrijving
1.0	19/11/2014	Douglas Horemans	Eerste versie
2.0	15/12/2014	Douglas Horemans	Tweede versie
3.0	20/04/2015	Douglas Horemans	Derde versie

[1] Alle versies worden nagelezen door de Document Master (Ivo Vervlimmeren)

Inhoudsopgave

1 Introductie

SKRIBL is een sociaal netwerk voor proffen en PHD studenten.
Dit document is een testplan voor de SKRIBL web-application. Het testplan beschrijft hoe de code getest moet worden opdat alle functionaliteiten van Skribl perfect zouden werken.
Vermits het SKRIBL-team agile werkt, zal dit testplan nog sterk veranderen doorheen de iteraties. Elke test heeft een testversie omdat de tests met de iteraties mee moeten evolueren. Agile werken houdt ook in dat de vorige geslaagde tests, opnieuw getest worden en indien nodig aangepast worden.

2 Referenties

- SDD versie 1.0
- Internal manual `http://wilma.vub.ac.be/~se4_1415/internalManuals/test-framework.pdf`
- Jasmine `http://jasmine.github.io/2.0/introduction.html`
- Postman `https://www.getpostman.com/`
- Selenium `http://www.seleniumhq.org/`
- IEEE 829

3 Definities

HTML Hyper Text Markup Language. Is een taal dat de inhoud van een webpagina beschrijft.

DOM Document Object Model. Is een soort boom dat de content van een webpagine bevat.

GUI Graphical User Interface.

API Application Programming Interface.

MVC Model View Controller. Dit is een software architectuur patroon dat de implementatie opsplitst in drie delen: Model, View en Controller. De controllers worden gebruikt om het model aan te passen, die op hun beurt de view aanpassen.

4 Domein

In het SDD, sectie 3.2, wordt de architectuur van de SKRIBL web-application beschreven. De sectie beschrijft de drie tiers die de webapplicatie gebruikt, namelijk de databasetier, servertier en clienttier. Voor deze drie tiers moeten tests worden geïmplementeerd, den deze worden in de sectie "Test items"beschreven.

5 Test items

Alle geïmplementeerde items & features moeten getest worden. De tests kunnen in verschillende groepen onderverdeeld worden. Deze groepen worden verderop beschreven.

5.1 Databasetier tests

De database houdt alle gegevens bij die te maken hebben met users, publicaties,... In deze tier moet het toevoegen, opvragen en verwijderen van die gegevens getest worden. Het is ook belangrijk dat na het manipuleren van de database deze niet corrupt wordt.

Alle tests bevinden zich in : \server\server\modules\database\jasmine-tests\...

5.1.1 Data-toevoeging tests

Tests die controleren of data op de juiste manier wordt toegevoegd en opgeslagen in de database.

- test adding user
- test adding journal
- test adding proceeding
- test adding library to user
- test adding publication to library
- test updating publication

File : databaseSetupSpecs.js

5.1.2 Data-opvraging tests

Tests die controleren of data op de juiste manier wordt opgehaald uit de database.

- test opvragen gebruiker
- test opvragen pub data
- test opvragen uploader pub
- test opvragen bibliotheken van gebruiker
- test opvragen pubs uit bibliotheek
- test simpel zoeken naar pub
- test geavanceerd zoeken naar pub
- test opvragen pubs van auteur
- test zoeken auteur met naam ...

File : databaseRequestsSpecs.js

5.1.3 Data-verwijdering tests

Tests die controleren of data op de juiste manier verwijderd wordt. Het verwijderen van bijvoorbeeld een user moet meerdere elementen in de database verwijderen, ook dit wordt gecontroleerd.

- test verwijder publicatie uit bibliotheek
- test verwijder bibliotheek van gebruiker
- test verwijder journal
- test verwijder proceeding
- test verwijder user

File : databaseDeletionSpecs.js

5.1.4 Data-persistentie tests

Tests die nagaan of data wel daadwerkelijk bewaard blijft en de database niet corrupt is.

- test user data
- test author data
- test library data
- test journal data
- test proceeding data
- test affiliation data
- test researchdomains
- test keywords

File : databasePersistenceSpecs.js

5.2 Servertier tests

De servertier is een communicatielaag tussen de database en de user. Voor de moeten alle HTTP requests getest worden.

Alle tests bevinden zich in : \server\server\routeTests\...

5.2.1 Authors tests

//TODO NOAH
File : authors.json

5.2.2 Libraries tests

//TODO NOAH
File : libraries.json

5.2.3 Publications tests

//TODO NOAH
File : publications.json

5.2.4 Users Test

//TODO NOAH
File : users.json

5.3 Clienttier

De clienttier zorgt voor een user interface en de communicatie met de server. Deze tier wordt getest aan de hand van de GUI. Om tijdverlies te voorkomen word de GUI automatisch getest, zie sectie "Strategie".

5.3.1 User tests

- Login test
- Logout test
- Register test
- Delete user test

5.3.2 Library managment tests

- Publicatie toevoegen via search test
- Libraries management test
- Downloaden van publicaties test
- Informatie over publicaties aanpassen test
- Informatie over publicaties opvragen test

5.3.3 Upload test

- Publicaties toevoegen(via upload) test

6 Software risico gevallen

De client die de SKRIBL web-application gebruikt komt alleen maar in aanraking met de clienttier. Opdat deze tier zonder probleem zou werken, moeten de servertier en database tier ook werken. Dit impliceert dat een fout in de server of database tier heel de web-application kan verzwakken. Natuurlijk zijn problemen in de web-application voorzien en kan SKRIBL met de meeste problemen leven. Maar om toch zo weinig mogelijk problemen te hebben is het testen van ELKE tier zeer belangrijk. Het grootste risico van onze applicatie is dus het feit dat de onderliggende tiers goed genoeg moeten werken zodat de client de application op een zo goed mogelijke manier kan gebruiken.

7 Features

De features die getest worden zijn:

- Login
- Logout
- Register
- Delete user
- Publicaties toevoegen(via upload)
- Publicatie opzoeken(zowel internal als external) om publicatie toe te voegen
- Libraries managen
- Downloaden van publicaties
- Informatie over publicaties aanpassen
- Informatie over publicaties opvragen

Sommige features die getest worden houden andere features in (die ook getest moeten worden) en worden hier onder beschreven.

Publicaties toevoegen(via upload) = publicatie zoeken + scraping + informatie over publicaties aanpassen + publicatie toevoegen aan library

Libraries managen = user libraries ophalen + publicaties van een library ophalen + publicatie toevoegen aan library + publicatie uit library verwijderen + library aanmaken + library verwijderen.

8 Strategie

Om de code te testen worden drie frameworks gebruikt:

Jasmine 2.0.3 Jasmine is een javascript test framework met heel veel kracht.

Postman Een chrome extension dat HTTP requests kan versturen.

Selenium Een geautomatiseerde GUI tester.

8.1 Jasmine 2.0.3

Om code te testen in Jasmine moet een specification file worden aangemaakt. Deze specification file beschrijft de tests die dienen uitgevoerd te worden.
Om tests te beschrijven worden matchers gebruikt. Matchers kunnen elementen vergelijken, controleren of het element wel gedefinieerd is en nog veel anderen! Het is ook mogelijk om zelf matchers te schrijven. Voorbeelden van matchers :

- expect().toBe();
- expect().toEqual();
- expect().toMatch();
- ...

8.1.1 Tests uitvoeren

Omdat Jasmine een standalone software is, is het uitvoeren van tests heel gemakkelijk. Wanneer een specification bestand geschreven is moet het pad naar deze alleen maar toegevoegd worden in een html bestand. Wanneer dit gedaan is dient het html bestand alleen maar geopend te worden in een webbrowser. In de webbrowser geeft Jasmine de geslaagde en gefaalde tests weer.

8.1.2 Extra

Een manual over Jasmine werd opgesteld door de test manager voor het SKRIBL-team. Deze staat in de referenties en legt het gebruik van Jasmine in detail uit.

8.2 Postman

Postman laat toe om op een heel simpele manier HTTP requests samen te stellen en vervolgens op te sturen aan de hand van een GUI. Het is daarna het werk van de tester om te kijken of de teruggegeven status de verwachte is of niet. In principe zijn alle HTTP responses tussen 200 en 299 juist.

8.2.1 Tests uitvoeren

Postman laat toe om de HTTP requests te schrijven en deze te bewaren. Het is hierna simpel om de HTTP requests (tests) later in te laden in postman om alle requests opnieuw te testen.

8.3 Selenium

Selenium laat toe om GUI testing te reproduceren. Selenium kan op knoppen klikken, veldjes invullen enz. Ook laat het toe om delen van de GUI te vergelijken met bepaalde waarden.

8.3.1 Tests uitvoeren

Het is mogelijk om tests te maken en deze een per een uit te voeren. Ook kunnen de tests opgeslagen worden en later weer geïmporteerd worden en opnieuw uitgevoerd worden.

9 Omgeving

Jasmine heeft geen bepaalde omgeving nodig om tests te kunnen uitvoeren . Het enige wat gedaan moet worden is de library uitbreiden met de gewenste source en specification bestanden.
Daarentegen heeft Postman Google Chrome nodig, en Selenium Mozilla Firefox, vermits het allebei extensions zijn.

10 Verantwoordelijkheden

In het SKRIBL-team worden meerdere programmeertalen en libraries gebruikt afhankelijk van de tier. Elk lid van het SKRIBL team werkt aan een bepaalde tier en kent de libraries en programmeertalen die gebruikt worden voor die tier. Ook kent elk lid natuurlijk de code van zijn tier maar niet alle code van elke tier. Om al deze redenen schrijft elke team lid zijn eigen tests.
Wanneer de code van een lid aan al zijn tests voldoet en alle gevraagde functionaliteiten zijn geprogrammeerd (en getest a.d.h.v zelf geschreven test) controleert de test manager alle tests. De test manager kan dan eventueel vragen om tests toe te voegen indien hij vindt dat er voor sommige functionaliteiten meer nodig zijn, of dat er dieper in de code moet getest worden, enz.

11 Goedkeuring

Wanneer de Test Manager alle tests heeft goedgekeurd zoals beschreven in sectie "Verantwoordelijkheden", moet de Quality Assurance Manager de code bekijken en deze wel of niet valideren. De eisen van de QA worden in volgende sectie besproken. Indien de Quality Assurance Manager de code niet valideert moet de programmeur zijn code aanpassen tot de code aan de eisen van de Test Manager en Quality Assurance Manager voldoet.
Wanneer code wordt gevalideerd kan deze gepushed worden naar de master branch van de SKRIBL repository.

12 Instructies van de QA

De QA is verantwoordelijk voor het bewaken van de kwaliteit van afgeleverde code (documenten daarentegen worden nagekeken door de document master). Dit houdt in dat na het schrijven en testen van de code een extra validatiestap zal plaatsvinden waarbij er niet wordt gekeken naar de functionaliteit zelf (dit gebeurt namelijk bij het testen), maar wel naar de leesbaarheid, performantie en onderhoudbaarheid van de code zoals gedefinieerd door enkele standaard definities.

Om dit proces te sturen werden enkele richtlijnen opgesteld. Deze omvatten onder andere het hanteren van een zekere stijl en het vermijden van minder idiomatische kenmerken van JavaScript. Daarnaast worden enkele metrieken bijgehouden zoals:

- LOC per klasse/functie/module
- nuttige hoeveelheid commentaren (hiervoor wordt tevens gebruik gemaakt van de JSDoc-standaard, cf. SPMP)

- aantal geneste if/for/...-lussen
- ...

Hiervoor wordt intern gebruik gemaakt van CodeClimate, een tool die in staat is om JavaScript code te analyseren en vervolgens enkele vaak voorkomende pitfalls (zoals duplicatie) er uit te halen. Hoewel deze score niet representatief is voor de uiteindelijke codekwaliteit, is het een handige indicator om in een snel tempo de meeste prominente stijlfouten er uit te halen. Elk groepslid wordt dan ook verondersteld om zijn/haar rapport van CodeClimate te analyseren en vervolgens bij te werken waar nodig.

Ten slotte zal de QA aan de hand van een steekproef bepaalde delen van de code zelf bekijken en de nodige feedback waar nodig. Opnieuw wordt hierbij vooral gekeken naar de codestijl en wordt bij het tegengekomen van performantieproblemen de desbetreffende code onderworpen aan profiling (hetzij door de QA, hetzij door developer zelf). De bedoeling is dat de persoon in kwestie uit deze richtlijnen en commentaren dan de nodige principes kan halen om zijn/haar hele codebase bij te werken. Van zodra dit is gebeurd, kan de QA indien nodig dan nog overgaan tot een finale analyse van alle op te leveren code, waarbij opnieuw een gelijkaardige werkwijze en criteria worden gehanteerd.