Research Continuous Integration & Delivery

Fancy subtitle

Tobias Derksen

Fontys School of Technology
Informatics
Module Software Factory (SOFA)

Venlo, 28th December 2017

Contents

Chapter 1

Project Plan

Project Plan

Sjoerd Brauer, Tobias Derksen, Loek Ehren, Nils Nieuwenhuis

September 14, 2017

Contents

Chapter 1

Introduction

This chapter is about general information regarding the Sofa Project "Fontys Module Management System". It includes background information and a small description of the customer and the organisation for which this project is done for.

1.1 Background information

The project belongs to the module SOFA of the seventh semester in the study course Informatics. The project team consists of one business informatic and three software engineering students. A real customer is involved in this project and the main goal of the project is to satisfy the customer and to act as a real company.

1.2 Context

The Project is for the Informatics department of the Fontys University of Applied Sciences in Venlo. The project period is about five months. The overall purpose of this SOFA project is to apply the knowledge and skills gained during the studies in the course Informatics. As students from both study directions business informatics and software engineering are involved in this project, subject-specific knowledge will be shared within the team and applied accordingly.

Chapter 2

Description

In this chapter is a description of the assignment for this project followed by an explanation of the problem which is to be solved. Finally the goals of the assignment will be set.

2.1 Assignment

The assignment of this project is to develop a system to manage module descriptions within the Informatics department of the Fontys University of Applied Science in Venlo.

2.2 Problem

The customer wants a product to easily create, modify and publish module descriptions for the Software Engineering and Business Informatics department of "Fontys Hogeschool Techniek en Logistiek" in the future.

Today there is no standardized way of creating and changing modules and their descriptions.

During the project our company will develop a software product to solve this problem.

2.3 Goals

- Delivering a high quality but not necessarily finished product
- Delivering proper documentation

- Delivering an extensible product

2.4 Approach

To assure a high quality product we stick to the Agile development framework Scrum during the project. A good documentation is achieved with our "SOFA Quality Guidelines" and "Software Requirements Specification" defined at the beginning of the project. Careful consideration will have to be taking into account when designing parts of the system to ensure the end-product is extensible and easily understandable for possible other project teams who could continue with this project.

Chapter 3

Project

In this chapter the project will be described in detail. It starts with a listing of the stakeholders. After that an explanation of the responsibilities will follow. Furthermore a detailed project scope, planning and organisation of this project is described in this chapter.

3.1 Roles and Responsibilities

3.1.1 Stakeholders

Role	**Name**	**Interest**	**Influence**
Customer	Van den Ham, Richard	High	High
Users of the Product	IT Teachers	High	Medium
Users of the Product	Students	Medium	Low
Students who want to pass	SOFA group	High	High
Coach	Jacobs, Jan	High	Medium

Table 3.1: Stakeholders

3.1.2 Team

Project Manager: Nils Nieuwenhuis

The project manager is responsible for the planning and will keep in touch with the customer due to the limited time. The project manager will write and keep the project management plan up to date. He will also keep a tight watch on the available resources. The project manager is responsible for the agenda of each meeting.

Quality Manager: Loek Ehren

The reviewing process of the project deliverables is managed by the quality manager. He has the responsibility to check that nothing leaves the team without testing and for that he writes a quality management plan and keeps it up to date. He also archives the reports produced in the test processes.

Configuration Manager: Sjoerd Brauer

The configuration manager has to document the use and / or modification of any hardware related configuration item as well as the use of the software and document related configuration items. Everything will be documented in the configuration management plan by the configuration manager.

3.2 Project scope

This section is about the scope of the project. It focuses on its functions and data, whereas the deliverables are defined in section 3.3.1 as project products.

The target of the project is to create a system that makes the information collection and deployment of the module informations easier. The most important feature of the system is to display the already available Information about modules and the consistency of this information. Especially their learning goals connected to the hoger beroepsonderwijs.

3.2.1 Functionality

The functionality is described in the Software Requirements Specification document in detail, for more information please refer to this document.

3.2.2 Data

Module	The name of the module
Semester	The semster where the module takes place
Credits	The credits which can be achieved if the module is successfully completed
Valid as of	The date were the descriptions will be valid
Author	The creator of the description
Description	The description of the module
Learning Goals	The learning goals wich will be achieved during the module
Competence Profile	The competence profile a student has after finishing the module
Module Assessment	The assessmentforms of the module
Prior Knowledge	The modules which have to be completed to participate in the module
Additional Information	Additional information to the module

3.3 Planning

In this section the planning will take place and in addition to that it begins with the milestones. After that the project products will be defined. The planning of the project will be performed based on the Scrum framework.

3.3.1 Milestones

By the end of week 4, the project team is familiar with the project and can begin working on the project in a Scrum fashion.

A Scrum sprint will last one week, after which the completed backlog items will be reviewed and the following sprint will be set up. In this way, each week will be its own milestone.

By the end of the project, the mandatory deliverables must be completed.

3.3.2 Project products

- Project plan
- Quality plan
- Configuration management plan
- Documentation
- Prototype
- User manual

3.4 Organisation

3.4.1 Communication plan

Communication medium	**Stakeholders**	**Frequency**	**Deliverable**
WhatsApp	Project team	If necessary	-
Customermeeting	Project team , Customer	Weekly	Meeting minutes
Coachmeeting	Project team, Coach	Weekly	Meeting minutes
Sprint planning meeting	Project team	At the bebinning of a sprint	Sprint Backlog
Daily Scrum	Project team	Daily	-
Sprint review	Project team	At the end of a sprint	-
Sprint retrospective	Project team	At then end of a sprint	-

Table 3.2: Communication plan

3.4.2 Quality assurance

The quality assurance will be dealt with the ”SOFA Quality Guideline” created and maintained by the Quality Manager, for more information please refer to this document.

3.4.3 Risk Management

#	Risk description	Impact	Owner	Likelihood of occurence	Severity of effects	Status	Mitigation action	Reaction time frame
1	A stakeholder is not present during a meeting	Decisions can't be made	Project manager	Low	High	not yet occured	Veto right is only valid if the stakeholder is present	within 1 day
2	Changing requirements	Probably changes in already finished Product features	Customer	High	High	not yet occured	Requirements have to be defined in a signed document	within 1 day
3	At the end of a sprint there are unfinished backlogs	In the next sprint is more work to do	Project team	High	Low	not yet occured	The Backlog items must be graded at the start of a sprint and unfinished Backlogitems will be moved to the next sprint	within 1 day
4	Ressources are not available	Changes the way of developement	Project team	Medium	High	not yet occured	The Resources have to be defined with the customer.	within 1 day

Table 3.3: Riskregister

Chapter 2

Quality Plan

SOFA Quality Guideline

Loek Ehren

January 14, 2018

Date	**Version**
11-09-2017	1.0
14-09-2017	2.0

Contents

1 Documents

This section outlines the quality standards and processes for all documents produced during the SOFA project.

1. All documents shall be written in LaTeX.
2. Every document's source and rendered result shall be committed and versioned.
3. All documents shall be reviewed at least once by another person either inside or outside of the team using the template found at Appendix A Review Template.
4. The results of document reviews shall be stored in the correct folder in the `25reviews/` directory with the following naming convention: **YYDDMM_firstname_documentversion***
5. An improved version of a document shall be reviewed once more to guarantee feedback has been processed and passes standards.
6. If a document does not pass a review, this document shall be improved again.
7. Documents to be reviewed will be added to the week's backlog.

*So the first review of a project plan will have this path: `25reviews/project_plan/20170911_loek_1.0.pdf`

2 Software

This section outlines the quality standards and processes for software development during the SOFA project.

2.1 Code

1. All code committed shall be committed alongside relevant testfiles.
2. All code relevant to a new feature, fix or any other change will be developed in a new branch.
3. Any branch that is to be merged into the master branch will be submitted in a pull request and reviewed by at least one person. Code reviews and any CI tool must also pass before merging.
4. All code shall be 100% tested at all time. If it is not, changes will not be accepted into the master branch.
5. Tests shall be useful, concise and descriptive. Don't test only one scenario, but every scenario.
6. All code shall be documented clearly, either by simple self documenting code or written documentation.

2.2 Commits

1. Commit messages will have a descriptive header describing the content of the commit.
2. A commit message's body could explain more about the change. Why something is changed, what it does etcetera.
3. Commits will remain small and relevant to only one change. If a commit changes more than one thing, it is too big!

3 Others

This section outlines standards for miscellaneous items to be standardized.

- Everyone shall track their Configuration Item in the CI table in `40configuration_management configuration_item_table` folder on GitHub.

4 Appendix A Review Template

Date of Review
Document Title
Document Date
(Optional) Commit ID
Reviewer Name
Reviewer Comments

Chapter 3

Configuration Management Plan

Configuration management plan

Sjoerd Brauer

11-9-2017

Contents

Chapter 1

Strategy

This chapter has a high level overview of what can be expected of configuration management.

1.1 Scope

The configuration management will include all CI's of the projects. The CI are all things beloning to the project. Documents, hardware etcetera. CM also includes how to deliver what CI's to the stakeholders, the frequency and in what way. CM manager will also document how to use hardware/software related to the project. CI's used but not owned by the project will not be documented.

1.2 Responsibility

The CM manager is responsible for maintaining the structure of the CI table(CIT) and this document. Every owner of a CI is responsible for that CI and the documentation of that CI in the CIT.

Chapter 2

Procedures

This chapter describes the procedures the group follows for configuration management.

2.1 CI management

When new CI's are introduced the owner of the CI must make an entry in the CIT and keep this CI item updated at all times. The configuration manager will pulse group members if they do not update their CI.

When delivering a product the CM manager is responsible for the delivery to the stakeholder. This includes the product and documentation

Chapter 3

Deliverables

This chapter mentions the deliverables. More information about the deliverables can be found in the CIT. Like whether it is finished or not. This chapter will be continuously updated.

3.1 Artifacts

Project plan. Planning. Quality Management document. Database design. User Storied. Domain model.

Chapter 4

Requirements Document

"Fontys Module Management System" Software Requirements Specification

Loek Ehren, Nils Nieuwenhuis, Tobias Derksen, Sjoerd Brauer

Fontys Hogeschool Techniek en Logistiek, January 14, 2018

Date	**Version**
11-09-2017	1.0

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document specifies all the requirements of the SOFA project during the seventh semester of the Software Engineering course at Fontys Hogeschool Techniek en Logistiek. The project is named Fontys Module Management System (FMMS).

1.1 Purpose

The purpose of this project is to streamline the process of changing module descriptions for the classes in the Software Engineering course, offering one centralized source of truth for the information contained in the module descriptions and the Onderwijs Examen Regeling (OER), and a controlled way of changing the descriptions if need be.

1.2 Scope

The scope of this document contains all the requirements of the system to be created.

1.3 Overview

This document is divided into three chapters. Chapter one will introduce the document and its purpose. The second chapter will give an overview of the functionality of the system and other interactions. The third chapter will specify all detailed requirements gathered which shall be implemented.

Chapter 2

Overall Description

This chapter will describe the general factors that will influence the product and its requirements.

2.1 Product Perspective

The product will be a software system that interacts with a database and a frontend to deliver and receive data.

Input some general diagram or description of the system's architecture

Describe interfaces

2.2 Product Functions

The general major functions of this system will be

- CRUD operations on module data
- Generating and releasing module descriptions and the OER
- Logging changes

2.3 Users

The users of the system will be

- Teachers
- Department Heads
- Curriculum Owner

2.4 External Interfaces

Any external software interfaces are listed here

Table 2.1: User Interaction Interface

Description	User interface
Purpose	Display data to the user and allow modification
Source of input	User interaction
Destination of output	Backend system

Table 2.2: Database Interface

Description	Database connection to a PostgreSQL database
Purpose	Store and retrieve data from and to the system
Source of input	Backend system
Destination of output	Backend system

Table 2.3: PDF Generator Interface

Description	PDF Generator
Purpose	Generate a PDF file
Source of input	Backend system
Destination of output	Backend system

Chapter 3

Requirements

This chapter will detail all non functional and functional requirements of the system.

3.1 Nonfunctional Requirements

This section will list all nonfunctional requirements of the system.

1. The system should be a web application
2. The system should be able to run in a Docker container
3. The environment should easily connect to different versions of the database
4. The system shall be in English only
5. The system should be available outside office hours
6. The system should be available at normal rates
7. The system should be secure
8. The system should be easily extendable for future developers

3.2 User Stories

User stories

Sjoerd Brauer

11-9-2017

Name	Updating the curriculum
ID	US_003
Actors	Curriculum owner
Story	The curriculum owner changes the curriculum

Name	Publishing the curriculum
ID	US_004
Actors	Curriculum owner
Story	The curriculum owner releases the Curriculum

Name	Login
ID	US_006
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	A system user logs in on the users account.

Name	Logout
ID	US_007
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	A user logs out.

Name	Changing permission
ID	US_008
Actors	Department head
Story	Department head changes an users permission

Name	Adding users
ID	US_009
Actors	Department head
Story	Department head adds users to the system.

Name	Assigning courses
ID	US_011
Actors	Department head
Story	Department head changes teachers to courses

Name	Inputting guidelines
ID	US_012
Actors	Department head
Story	Department head inputs the guidelines for the module description.

Name	Create a module
ID	US_013
Actors	Curriculum owner
Story	Curriculum owner creates a module.

Name	Modifying a module
ID	US_014
Actors	Teacher
Story	Teacher modifies a module.

Name	Checking history
ID	US_015
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	A system user looks into the history of a module description

Name	Track history p2
ID	US_016
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	A system user looks into the history of a module

Name	Display information
ID	US_017
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	The user displays a module with the corresponding HBO-I matrix.

Name	Insert progress data
ID	US_018
Actors	Teacher
Story	A teacher inserts a students progress data into the system

Name	Change progress data
ID	US_018
Actors	Teacher
Story	A teacher changes a students progress data in the system

Name	Update exam information
ID	US_020
Actors	Teacher
Story	A teacher changes a module's exam information

Name	Create exam information
ID	US_021
Actors	Teacher
Story	A teacher creates a module's exam information

Name	Add competences
ID	US_022
Actors	Teacher
Story	A teacher adds competences to the system.

Name	Remove competences
ID	US_023
Actors	Teacher
Story	A teacher removes competences from the system.

Name	Update competences
ID	US_024
Actors	Teacher
Story	A teacher updates competences in the system.

Name	Adding learning goals
ID	US_025
Actors	Teacher
Story	A teacher adds learning goals to a module.

Name	Updating learning goals
ID	US_026
Actors	Teacher
Story	Teacher updates learning goals in the system.

Name	Creating learning goals
ID	US_027
Actors	Teacher
Story	A teacher creates learning goals in the system.

3.3 Use Cases

3.4 Mockups

Chapter 5

User Stories

5.1 Document

User stories

Sjoerd Brauer

11-9-2017

Name	Updating the curriculum
ID	US_003
Actors	Curriculum owner
Story	The curriculum owner changes the curriculum

Name	Publishing the curriculum
ID	US_004
Actors	Curriculum owner
Story	The curriculum owner releases the Curriculum

Name	Login
ID	US_006
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	A system user logs in on the users account.

Name	Logout
ID	US_007
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	A user logs out.

Name	Changing permission
ID	US_008
Actors	Department head
Story	Department head changes an users permission

Name	Adding users
ID	US_009
Actors	Department head
Story	Department head adds users to the system.

Name	Assigning courses
ID	US_011
Actors	Department head
Story	Department head changes teachers to courses

Name	Inputting guidelines
ID	US_012
Actors	Department head
Story	Department head inputs the guidelines for the module description.

Name	Create a module
ID	US_013
Actors	Curriculum owner
Story	Curriculum owner creates a module.

Name	Modifying a module
ID	US_014
Actors	Teacher
Story	Teacher modifies a module.

Name	Checking history
ID	US_015
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	A system user looks into the history of a module description

Name	Track history p2
ID	US_016
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	A system user looks into the history of a module

Name	Display information
ID	US_017
Actors	1. Teacher 2. Department head 3. Curriculum owner
Story	The user displays a module with the corresponding HBO-I matrix.

Name	Insert progress data
ID	US_018
Actors	Teacher
Story	A teacher inserts a students progress data into the system

Name	Change progress data
ID	US_018
Actors	Teacher
Story	A teacher changes a students progress data in the system

Name	Update exam information
ID	US_020
Actors	Teacher
Story	A teacher changes a module's exam information

Name	Create exam information
ID	US_021
Actors	Teacher
Story	A teacher creates a module's exam information

Name	Add competences
ID	US_022
Actors	Teacher
Story	A teacher adds competences to the system.

Name	Remove competences
ID	US_023
Actors	Teacher
Story	A teacher removes competences from the system.

Name	Update competences
ID	US_024
Actors	Teacher
Story	A teacher updates competences in the system.

Name	Adding learning goals
ID	US_025
Actors	Teacher
Story	A teacher adds learning goals to a module.

Name	Updating learning goals
ID	US_026
Actors	Teacher
Story	Teacher updates learning goals in the system.

Name	Creating learning goals
ID	US_027
Actors	Teacher
Story	A teacher creates learning goals in the system.

5.2 Diagram

Chapter 6

Software Architecture Document

Software architecture

1. Interface specifications

This document outlines the specifications of the back-end application. The back-end application has a RESTful interface with which the front-end communicates with.
The data format used will be JSON.
At the beginning of the project we choose that when communicated with the backend the least amount of endpoints should be called. That's why a lot of custom object interfaces can be found in front end.

The JSON objects can be found in the json_objects.ods file in the 60software_architecture\backend_interface directory.

Complete overview of back-end endpoints.	
Uri path with resource	**description**
/curricula	Returns an array of curriculum
/curriculum/:curriculum id/module/:module_code	Returns a module object
/curriculum/:curriculum id/semesters	Returns an array of semester
/curriculum/:curriculum_id/semester/:semester_number	Returns a complete_semester object
/qualications	Returns a filter_qualifications object
curriculum/:curriculum_id/architecturallayer/:id/activity/:id	returns a complete qualications overview semester object
/module/:module_code	Accepts a editable_module_input or returns an editable_module_output.
/curriculum/:curriculum id/module/:module_code/pdf	returns a modules PDF

2. Endpoint definitions

This section describes the urls more specically with functionality, the JSON format and HTTP codes. The dataobjects are described in json objects.pdf in this repository.

Get data for editing module	
URL	/module/:module_code
Method	GET
Returns	Returns a filled editable_module_output
Returns code	200 OK 404 NOT FOUND

Post editing module	
URL	/module/:module_code
Method	POST
Returns	Gets a filled editable_module_input
Returns code	200 OK 404 NOT FOUND

Complete semester	
URL	/curriculum/:curriculum_id/semester/:semester_number
Method	GET
Returns	Returns a filled complete_semester object
Returns code	200 OK 404 NOT FOUND

Filter of qualifications	
URL	curriculum/:curriculum_id/architecturallayer/:id/activity/:id
Method	GET
Returns	Returns a filter_qualifications object
Returns code	200 OK 404 NOT FOUND

complete module	
URL	/curriculum/:curriculum id/module/:module_code
Method	GET
Returns	Returns a filled_module object
Returns code	200 OK 404 NOT FOUND

Semesters of a curriculum	
URL	/curriculum/:curriculum id/semesters
Method	GET
Returns	Returns an array of semester objects
Returns code	200 OK 404 NOT FOUND

Curricula of a student program	
URL	/curricula
Method	GET
Returns	Returns an array of curriculum objects
Returns code	200 OK 404 NOT FOUND

Chapter 7

Reviews

7.1 Project Plan

Date of Review
11-09-2017
Document Title
Project Plan
Document Date
09-09-2017
(Optional) Commit ID
6ef0e1fa27d14ab17f4133610077c6befde22b12
Reviewer Name
Loek Ehren
Reviewer Comments

- Typo 'infomration' in Introduction page 2 line 2
- Typo 'explaination' in Description page 3 line 2
- Typo 'a explanation' in Project page 4 line 2
- Refer to yourself as 'He' not He/She in 3.1.2 page 4 line 3
- Refer to me as 'He' also please in 3.1.3 page 5
- Sjoerd is configuration manager
- Scope cannot change once its defined. It says what must be done and also what must not be done. It has to be defined clearly in the beginning of the project in 3.2
- I think chapter 3.2.2 is not supposed to be in the project plan but in the requirements document or nowhere at all.
- Chapter 3.3 on a new page please
- Typo 'scurm' in 3.3 line 3
- Try to fix the tables and text going off the side of the page.
- In 3.4.2 just refer to the quality standards document and write how we follow the standards etc.

Keep consistency between 'SoFa' or 'SOFA'.

11-9-2017
Project plan
8-11-2017
1
Sjoerd Brauer
Reviewer Comments

1. Introduction - 1.1 Background information
text gives the impression that there are multiple business students. please update.

2. Introduction - 1.2 Background information
Text error in first sentence: ot, should be or.

3. Chapter 2 - Assignment
It's also about planning, but our focus lies on the modules.

4. Chapter 3 - Stakeholders
You forgot the exam commission who is a user of module descriptions as well.

5. Chapter 3 - Roles
Sjoerd is the configuration manager.

6. Chapter 3 - Functionality
Please refer to SRS the SRS contains all functionality/requirement.

7. Chapter 3 - Data
The text goes out of the document.

8. Chapter 3 - Products
add configuration management plan.

9. Chapter 3 - Organisation
the tables are only partly shown on page.

10. Chapter 3 - Organisation
No external communication plan?.

11. missing
Missing stakeholder Analysis.

Date of Review
14-09-2017
Document Title
Project Plan
Document Date
14-09-2017
(Optional) Commit ID
/
Reviewer Name
Loek Ehren
Reviewer Comments

- Grammar error in 2.2 Problem 'there' 'their'
- Describe the problem a bit more. "The modules for the SEBI course in FHTenL" etc. Now it looks so short and abrupt.
- He/She remains in 3.1.2 Quality Manager section
- Typo 'beginns' in 3.3 Planning line 1
- In communication plan, 'retroperspective' should be 'retrospective'

*Adjusted Goal and Approach sections. *Added milestones

7.2 Quality Plan

14-9-2017
SOFA Quality Guideline
11-9-2017
1
Sjoerd Brauer
Reviewer Comments

chapter 1 documents 4th bulletpoint: Document refers to wrong directory, should be 25reviews.

chapter 1 documents 7th bulletpoint. Backlog? where is the backlog.

chapter 1 documents 1th asterix. please properly write down how name should be like name_versionnumber

chapter 2.1 bulletpoint 4 100% line test? thats pretty difficult as you know.

chapter 2.1 bulletpoint 5 Every scenario possible? that's too extreme. Just realistic scenarios.

chapter 2.2 last bulletpoint. if more then 2 things are changed it's too big? please don't define size in this document.

general: please use numbering instead of bulletpoints.

7.3 Configuration Plan

Date of Review 11.09.2017
Document Title Configuration Management Plan
Document Date 11.09.2017
(Optional) Commit ID b5dde48ba02a05d41826e8716a087647795c8e41
Reviewer Name Nils Nieuwenhuis
Reviewer Comments
3.1 CI management:
TYPO: The configuration manager will pulse group members if they do not update their CI for any reason.

7.4 User Stories

Date of Review
11-09-2017
Document Title
User Stories
Document Date
11-09-2017
(Optional) Commit ID
2287fb3e22813472e5bd6ad180b02119d0302eb1
Reviewer Name
Loek Ehren
Reviewer Comments
US_001:

- Random capital at 'Creating'.
- Does the teacher create a module or a module description? They are different things.
- What does retrieve guidelines mean??

US_002:

- Does the teacher create a module or a module description? They are different things.
- What does retrieve guidelines mean??

US_005:

- The teacher makes the module or the module DESCRIPTION public? They are different things.

US_006:

- Saying 'and gets permission according to his role' is already specifying requirements.

US_007:

- 'logout' should start with a capital.
- The story should just be 'A user logs out'.

US_009:

- 'adding users' should start with a capital.

US_012:

- Is this not the same as modifying a module?

US_014:

- Name should start with capital.

Date of Review
12-09-2017
Document Title
User Stories
Document Date
11-09-2017
(Optional) Commit ID
38b159a2161fba6859341c0d2b6ba9c268c1c914
Reviewer Name
Loek Ehren
Reviewer Comments

- Are US14, US12 and US02 all the same? What's the difference?
- Aren't US15 and US16 the same? I thought there was no difference between module and module description? What is the difference?
- US20 / US21: If you're going to specify changing the individual parts of a module as separate user stories you should do all of them. So competences, credits, learning goals, teaching material etc.

Date of Review
12-09-2017
Document Title
User Stories
Document Date
12-09-2017
(Optional) Commit ID
37622c1
Reviewer Name
Loek Ehren
Reviewer Comments

- Random text outside the table at US18
- US20-. Is modifying those things a subset of US14?
- US17 Is there also a display without HBOI matrix? Just displaying it normally?

7.5 User Story Diagram

Date of Review 14.09.2017
Document Title User Story Diagramm
Document Date 14.09.2017
(Optional) Commit ID 4ed4569162594006fa0a4c9688b572bdb4534c90
Reviewer Name Nils Nieuwenhuis
Reviewer Comments

1. Department Head: Please change ”including” to ”include”.
2. System user: ”Login” should *include* ”Logout”.
3. Teacher: There are two use cases with ”Modifying a module”.

Chapter 8

Handover Document

Handover Document
Fontys Module Management System

Nils Nieuwenhuis, Loek Ehren, Sjoerd Brauer, Tobias Derksen

Fontys School of Technology and Logistics
Informatics
Module Software Factory (SOFA)

Venlo, 12th January 2018

Contents

Chapter 1

Introduction

This project started during the Software Factory (SOFA) module. The goal of the project was to develop a module management system. The department head of Software Engineering and Business Informatics is the customer of the project. The project, when fully implemented should be able to support all educations of Fontys though.

More information can be found in the 10projectplan at `https://github.com/FSG1/mgmt/tree/master/10projectplan`.

Chapter 2

Software Architecture

2.1 Frontend

For Frontend a group member did research in multiple technologies for front-end. Then in discussions that followed, the group voted for Angular for front-end. For more information see the research which can be found in the "30analysis/frontend_frameworks" directory. There is also a page breakdown diagram and a page navigation diagram if one needs more information found in "100documentation/frontend".

2.2 Backend

As to lower the cost it is decided to use open source programming languages. The deployment environment supports open source as well. The deployment environment is a Linux server provided by the customer. In this light it is chosen that Java is used for back-end development. The group is well experienced with Java, which made a big impact on the choice as well. See the back-end research for more information found in "30analysis/java_frameworks_backend" directory.

Furthermore a research was done on which frameworks to use in this project. From this research it was decided to use JDBC because complex query's are required. These are not possible with for example Hibernate. Furthermore it was decided that Jersey was to be used for the endpoints. The group preferred small frameworks over big frameworks that do more then one needs.

2.3 Database

To store the relational data, a PostgreSQL database is used. This decision was made by the customer because he already provided an engineered database when the project started. Nevertheless PostgreSQL is a reasonable choice, it provides all necessary feature important for the project.

2.4 Architectural decisions

One of the big decisions made is that the littlest amount of endpoints on a page should be called as possible. The consequence of this is that we have few endpoints but many interfaces (object definitions). The communication between front-end and back-end is thus faster as result. If one is to look through the front-end interface one will see that we almost exclusively use interfaces instead of classes. This is done because creating an object adhering to an interface has a lower memory footprint.

Chapter 3: Hardware Document 73

For more information about the endpoints see the Software "architecture endpoints aesthetic.pdf" document in the 100documentation directory.

Chapter 3

Setup Frontend

1. Clone the GitHub repository of the Frontend application `https://github.com/FSG1/frontend.git`
2. Install the Angular CLI as it is described in the angular "Get Started" tutorial `https://angular.io/guide/quickstart`.
3. Open the cloned frontend repository with a terminal and execute "npm install" to install all necessary node packages for the frontend application.
4. Look at the "page breakdown diagramm.png" to understand the structure and the pages of the frontend. You can find it in the "mgmt" repository on GitHub `https://github.com/FSG1/mgmt/tree/master/100documentation/frontend`.
5. Read the "Software architecture endpoints asthetic.pdf" to understand the connections to the backend application. You can find it also at the "mgmt" repository on GitHub `https://github.com/FSG1/mgmt/tree/master/100documentation`.
6. To understand the source code read the documentation. *RTFM*
7. To start the application open the cloned frontend repository with a terminal and execute "ng serve". Point your browser to `http://localhost:4200` to view the frontend application.
8. If you want to test the frontend application make yourself familiar with the Testing documentation of Angular, you can find it here `https://angular.io/guide/testing`. To start the test, open the cloned frontend repository with a terminal and execute "npm test". A browser window opens which shows you the results of your tests.

Chapter 4

Setup Backend

The backend of FMMS consists of a REST API connected to a PostgreSQL database. The REST API is written in Java using Jersey. Jersey is an open source framework that supports JAX-RS, which is a simple API spec for creating REST APIs. The Jersey documentation can be seen at `https://jersey.github.io/`.

To checkout the project:

1. First install Maven if you do not have it already.
   `https://maven.apache.org/install.html`
2. Clone the GitHub repository with `git clone git@github.com:FSG1/backend.git`

The backend can be run in Docker or standalone from the jar.

- To build and run the backend in Docker, refer to the chapter on Docker deployment.
- To build and run the backend standalone, run `mvn build` followed by `mvn exec:java` in the root of the project, or use your IDEs built in build-and-run functionality.

Chapter 5

Setup Database

The database needs an already installed version of PostgreSQL. To import the database schema, first a new user and a new database have to be created. After that, the SQL files from the "scripts" directory can be imported to the database. Please take care of the order as it is indicated by the numbers.

Listing 5.1 shows how to create a user, a database and import all SQL files. The script is written in bash and only works in Linux systems. For Windows or MacOS please use a database client like PgAdmin or DataGrip.

Please be aware that the database repository does contain only the database schema. To import the data there is a SQL file inside the subversion repository which can be imported the same way as the other files.

```bash
        # Create user "module"
        sudo -u postgres \
                psql --command "CREATE USER module WITH PASSWORD 'fmms';"

        # Create database "modulemanagement"
        createdb -O module modulemanagement

        for f in scripts/*.sql; do
                sudo -u postgres psql -U module -d modulemanagement -f "$f";
        done
```

Listing 5.1: Import database schema

Chapter 6

Run with Docker

For each part of the project there is a docker[1] file which can be used to run the software. The docker file automates the build process and encapsulates it into a container. These containers runs on every operating system and does not need any external dependencies besides the installed docker daemon. You can compose the separate containers to a full service which includes all parts of the project.

6.1 Database

To setup a database server and create the proper users and databases can be very error-prone. Therefore a docker container can be used which automatically sets up the user, the database and tables.

During the creation process, the SQL files in the "scripts" directory are executed, sorted by name (that's why there are numbers in front of the filenames). The database repository only contains the table structure and does not contain any data. Nevertheless, data can be automatically imported by copying a proper SQL file into the "scripts" directory before building the container.

When running the database as a docker container, please make sure that there are no other databases running on port 5432 or change the port mapping.

```
        cd database
        # Build container
        docker build -t fmms-database .
        # Run container
        docker run -d --name fmms-database -p 5432:5432 fmms-database
```

Listing 6.1: Build and run Database Container

6.2 Backend

The configuration of the Backend API can be done without any changes to the source code. During initialization, environment variables [2] are read and the values will be used as configuration. There are basically three important parts to configure:

- The Backend URI containing port and base url
- The database connection

[1] `https://www.docker.com/get-docker`
[2] `https://en.wikipedia.org/wiki/Environment_variable`

- Username and password for restricted actions

Restricted actions are all actions which can change the data in the database. The backend uses HTTP Basic authentication to authenticate users who wants to perform restricted actions. The credentials are currently hard-coded into the configuration and can be set via environment variables.

The default values has been chosen to allow the software to be run locally. For server deployment other values need to be entered. The default database URL contains the default docker host ip address, which implies that a PostgreSQL server is bound to the port 5432 of the host.

The environment configuration can be given into the docker containers using the docker environment functionality (see the Docker documentation [3]).

Name	**Default Value**	
HOST	0.0.0.0	IP Address to bind server socket to. Usually the default value will do the job.
PORT	8080	Server port to listen on
BASE	/fmms	API Base URI
DB	172.17.0.1:5432/modulemanagement	DB URL for JDBC postgres driver format: <IP>:<PORT>/<databasename>
DB_USER	fmms	Username to access the database
DB_PASSWD	fmms	Password to access the database
AUTH_USER	fmms	Username for restricted actions
AUTH_PASSWORD	modulemanagement	Password for restricted actions

Table 6.1: Environment Configuration for Backend

```
        cd backend
        # Build container
        docker build -t fmms-backend .
        # Run container
        docker run -d --name fmms-backend -p 8080:8080 fmms-backend
```

Listing 6.2: Build Backend Container

6.3 Frontend

The frontend configuration needs to be done inside the source code before building the container. The default configuration works only for local use and is not suitable for server deployment. The configuration is done via environment files which are loaded based on cli arguments. [4]

```
        cd frontend
        # Build container
        docker build -t fmms-frontend .
        # Run container
        docker run -d --name fmms-frontend -p 4200:4200 fmms-frontend
```

Listing 6.3: Build Frontend Container

[3] https://docs.docker.com/engine/reference/run/#env-environment-variables
[4] http://tattoocoder.com/angular-cli-using-the-environment-option/

6.4 Compose

To run all parts of the software inside docker containers, Docker Compose[5] can be used to run and supervise the docker containers. Therefore a docker compose file is needed which defines the structure of the application and the needed parameters. The following listing shows a docker compose file which contains all needed configuration to run the project on your local machine.
To use docker compose perform the following steps:

1. Install Docker and Docker-Compose
2. Build Database, Backend and Frontend as explained in sections 6.1, 6.2 and 6.3
3. Put the content of listing 6.4 into a file named „docker-compose.yml“
4. Run shell command „docker-compose up -d“ in the directory with the file created in the previous step

[5]`https://docs.docker.com/compose/install/`

```
version: '2'

networks:
  fmms:
    driver: bridge

services:
  database:
    restart: always
    image: fmms-database
    ports:
      - 5432:5432
    networks:
      - fmms

  backend:
    restart: always
    image: fmms-backend
    ports:
      - 8080:8080
    environment:
      - HOST=0.0.0.0
      - PORT=8080
      - BASE=/fmms
      - DB=database:5432/modulemanagement
      - DB_USER=module
      - DB_PASSWD=fmms
      - AUTH_USER=fmms
      - AUTH_PASSWORD=fmms
    volumes:
      - maven:/root/.m2
    networks:
      - fmms

  frontend:
    restart: always
    image: fmms-frontend
    command: ["--no-live-reload", "--no-watch"]
    depends_on:
      - backend
    ports:
      - 4200:4200
    networks:
      - fmms

volumes:
  maven:
    driver: local
```

Listing 6.4: Docker Compose File

Chapter 7

Staging Area

There is a running instance of the project available at `https://modulemanagement.fontysvenlo.org/`. This instance is mainly a staging and testing instance for the customer and does not guarantee any data persistence. The instance is automatically built and deployed with a Jenkins CI at `https://jenkins.fontysvenlo.org`. The Jenkins installation contains three projects:

- backend
- frontend
- deployment

The backend project is responsible to build the backend code, run unit tests, code style analysis and test coverage checks and, if all checks passes, build a docker container and push the container to a local docker registry.

The frontend project is responsible to build the frontend code, run unit tests and, if the tests run successfully, build a docker container and push the container to a local docker registry.

The deployment project first stops the running frontend and backend containers. After that it starts the build of the frontend and backend projects. When they finished successfully the deployment project pulls the new containers from the registry and starts the frontend and backend with the new versions.

The build configurations are stored in files named "Jenkinsfile". There is exactly one Jenkinsfile for each project which lives in to root of each repository. It uses a special Jenkins pipeline syntax written in Groovy.

To deploy a new version of the project, the build of the deployment project needs to be started manually.

Chapter 9

Research Documents

9.1 Nils

9.2 Loek

9.3 Sjoerd

9.4 Tobias