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, 8th 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.

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.

```
1   # Create user "module"
2   sudo -u postgres \
3           psql --command "CREATE USER module WITH PASSWORD 'fmms';"
4
5   # Create database "modulemanagement"
6   createdb -O module modulemanagement
7
8   for f in scripts/*.sql; do
9           sudo -u postgres psql -U module -d modulemanagement -f "$f";
10  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.

```
1  cd database
2  # Build container
3  docker build -t fmms-database .
4  # Run container
5  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

```
1 cd backend
2 # Build container
3 docker build -t fmms-backend .
4 # Run container
5 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]

```
1 cd frontend
2 # Build container
3 docker build -t fmms-frontend .
4 # Run container
5 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.