Faculty Elektrotechnik / Informatik

UNIKASSEL VERSITÄT

Master Thesis

Homework Manage System with Git-Support and hierarchical File database for Data management

Submitted by: Hao Gao
Matriculation number: 33101387

Supervised by: Prof. Dr. Albert Zündorf
Universität Kassel
Prof. Dr. Gerd Stumme
Universität Kassel

Kassel, January 6, 2016

Declaration of Authorship

Deutsch oder English?

Kassel, January 6, 2016

Hao Gao

Contents

1 Chapter 1
Introduction

motivation, why we need this kind of hms?

Chapter 2

Basics

what were used in this project?explain the basics of the techniques

2.1 techniques like framework and jpa?

or?

2.2 Related works

moodle?

3

Chapter 3

Design

The HMS (Homework Management System) is a web platform to manage the activities related to the homework including handing out the related materials to the students,collecting the handed in homework and managing the evaluation of the assignment. In this chapter the detailed design of functions to support those activities will be discussed. Furthermore the problems of the current alternative system will be analyzed and the solutions to those problem will be introduced.

The functions of the "HMS" system are divided into two parts. The first part is to develop the common features which are similar to other web platform,for instance "register a user account" . The second part of the design is to develop the special core features to make the HMS system better platform compare to others, the dynamic data management and the git based homework management.

3.1 Common functions

The main task of "HMS" is handling the process of handing in and handing out the homework between the students and teachers.

As the Figure 3.1 shows,there are several steps within this process from the prospective of the user roll. First considering the user roll of students,the student should be able to register a account of "HMS", after logging into the system the students can browsing all the available courses in the system and subscribing the target course. Then it is possible for the students to view the homepage of the course and download the available homework. and finally uploading the solutions to the homework accordingly. the process of uploading a homework to the system is finished at the student side. now take a look at the side of Teacher and Assistant, besides the registration and logging process, the user group of teacher can create new course and new assignment also collecting the handed in homework and give them back to the students once the evaluation is finished. The assistants however can not creating new course, but should be able to add new assignment and evaluate the handed in homework as well. In summary, in order to realize the main task of the "HMS", the "HMS" system should have following capabilities:

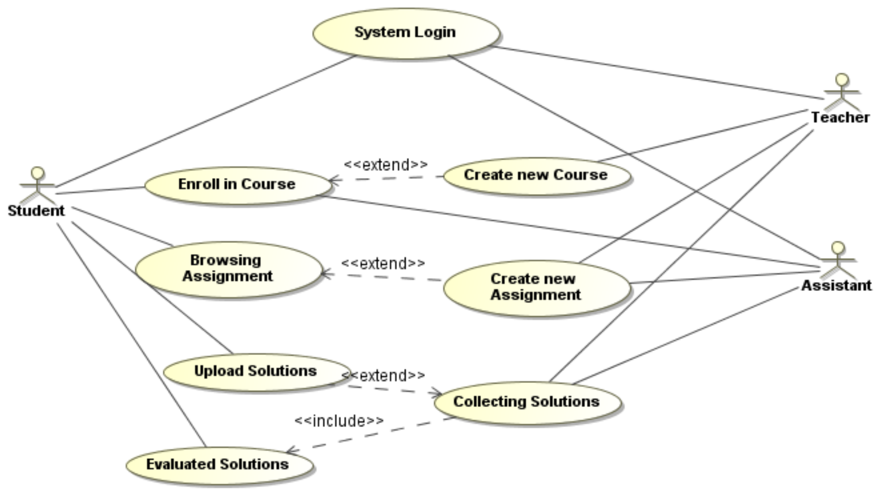


Figure 3.1: The use case from the perspective of different roll of user.

1. User management including "Registration" , "Role based access control","Self management".
2. Course management including "Creation of Course","Modification of Course".
3. Assignment management including "Creation of Assignment","Distribution of Assignment","Collecting of Assignment","Evaluation of Assignment".

Besides the above listed capabilities, the HMS should also provide a way that the students and teachers can communicate with each other. firstly a private message system will be needed for the students and teachers to exchange the information about the problem of assignment or evaluations individually,also the new message system should work as a instant message system,in this way the questions or the problems between the students and teachers can resolve more efficiently. secondly, a course forum is also not a bad idea, a common scenario is that more students may have a same question for a new assignment, if a students write a new post about this question, and the teacher gives a answer to the question, the other students with the same question can also get the answers and avoiding asking the same questions again. This saves the time from both side. So the communications system for the HMS has two parts:

1. Instant private message system.
2. a public course forum for each course.

3.1.1 User management

The user management of the "HMS" system consists of several modules(Figure 3.2), first is the registration module, with this module the user can use their email address to register a account in HMS system. second is the self management module. after the user has logged into the system, they should be able to change their emails and password or other personal details.third is the user role control module, this is necessary for system to arrange the proper functions to the current user based on their user role, the user obtains a startup role at the registration, later on the user role can be changed by the system admin. In the rest of this section, all the modules will be detailed discussed.

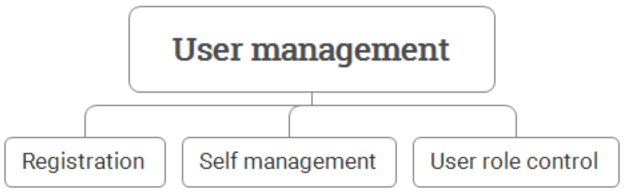


Figure 3.2: The Modules of user management

a. Registration

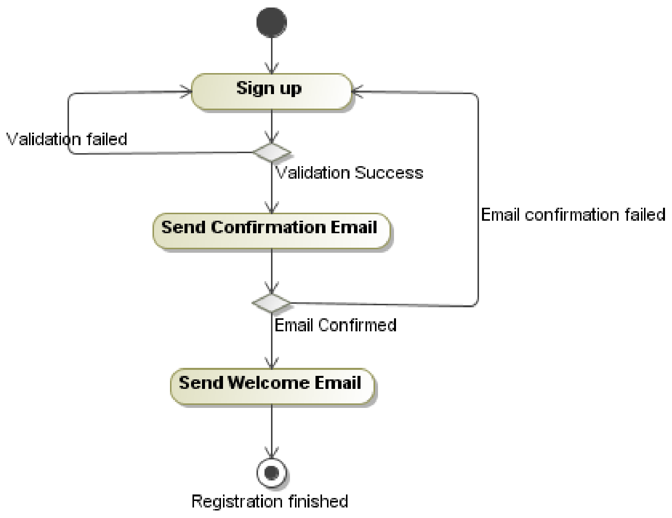


Figure 3.3: Registration activity

Figure 3.3 shows the workflow when a user register a new account. the user first use the registration form to fill in all the relevant information , for instance the password, email, and students number, after clicking the sign up button, the whole process begins, first the validation of the registration form will be performed to check whether the user has fill in all the required field and without error, if user passed the form validation, the HMS system will then send a confirmation email with a confirmation URL to the email address from the registration form, if the user click the confirmation link, the user will be redirect to the website and can directly starting using the account. Otherwise the user has to start over the registration process. The step of email address confirmation is important because this procedure allows the system to check that the user actually signed up for the account and guarantee the email of this user is valid and ready to receive the system information.

b. Self Management

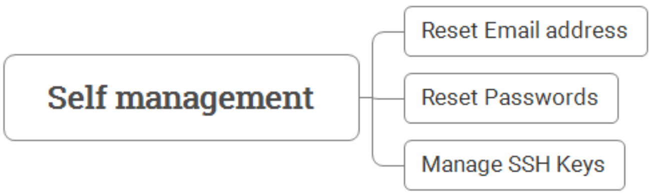


Figure 3.4: The functions of self management

It is very common that user may forget their passwords or even worse their registered email becomes invalid, so it is necessary to develop the functions, user can use to reset their email and passwords. The resetting of email address or the passwords works similar as registration. If user choose to reset their email address, first they will be asked to type in the new email address,after that user click the reset button, the system should send a new confirmation email to the new address, when user click the hyper link in the email, a new web page will be generated and user can confirm the address change. if user choose to reset their passwords, they just need to click the reset password button, the system will send another confirmation email with a hyper link, the user can use this link to type in their new passwords. Besides the modification of passwords and email address,there is another function should be added to self management, thus the HMS system use git server to control all the homework files, and this git server use ssh to authenticate the connections, the system should also provide a function that user can add their ssh public key to the git server, so that they can connect their computer direct to the git server. Figure 3.4gives a overview for all the components of the self management.

	Admin	Default	Students	Assistants	Teachers
System Functions	●	○	○	○	○
Create Lecture	○	○	○	○	●
Create Assignment	○	○	○	○	●
Evaluation	○	○	○	●	●
Join Lecture	○	○	●	●	●
Hand in Homework	○	○	●	○	○
Forum & Message	○	○	●	●	●
Self Manage	○	●	●	●	●

Figure 3.5: Different user role in HMS system

c. Role based access control

The user of the HMS system has different roles to perform different actions, for instance a teacher can create new course but students can not, so it is important to have a subsystem to distinguish the user roles, so that the HMS can serve proper functions to the user.

Figure 3.5 shows all the user role in HMS system,first is the "*system admin*", the job of system admin is to manage other user's role and the system data(backup the database and related files in certain times),any other functions of the HMS is irrelevant to the system admin. second is the default user,the user with this role can not do much things other than updating their personal data(passwords and email). third user role is students, with student account the user can browse all the available course and join the course, download and upload homework, using the communication system like chat and forum. fourth user role is assistants, besides all the functions of students, assistants can review all the student's homework and make a evaluation,last one is the teacher, the teacher account has all the functions of assistant account and additionally the teacher can create new course.

3.1.2 Course management

The typical course provided by the faculty software engineering and their evaluation methods are listed as follows:

- Bachelor Programming Methodology: Evaluation through final exam
- Bachelor Design-Pattern: Evaluation from multiple sub project
- Master Software Engineering 2: Evaluation from semester project
- Master Compiler Construction: Evaluation from multiple sub project
- Master Graph-Model : Evaluation through final exam

Above course can be divided into two categories. in the first type of course the students have to hand in various homework, and the points gained from those homework are usually used as a prerequisite for the final exam. in the second type of course the students will get a semester assignment, normally a whole software project, the points gained from this project usually is the final points for this course. additionally every students will get a git repository after user had signed up the course,this repository will also worked in two modes according to the type of course,the details of git working modes will be discussed later in section 3.2.1.

Features	Precondition to final exam	Git repository Mods	Performance Evaluation
Type I	1. Students have to hand in at least amount of valid homework. 2. A valid homework requires normally for students to gain more than 50% points of a assignment. 3. A student should gain at least 50% of total points for final exam	Git repository works under local mods (Student can only hand in the homeworks through course homepage)	need detail evaluation of assignments (number of valid handin, percentage of gained points)
Type II	None,Students just need to hand in the final project(may consists sub. project)	Git repository works under remote modus (Student can use the course repository as any remote git repository)	Only final evaluation (or multiple sub.evaluation)

Figure 3.6: Features of different types of course

The Table 3.6 shows the different features of different types of course.The function of "creating course" should take all the features from above into consideration.

3.1.3 Assignment management

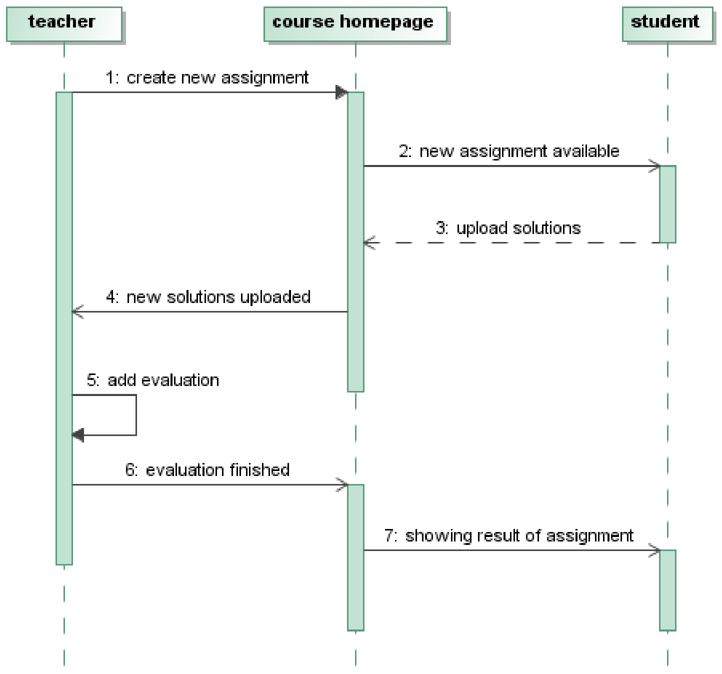


Figure 3.7: Sequence diagram for assignment management

Figure 3.7 shows the sequence of handing in and handing out a assignment, after creating a new course, teacher can start adding assignments to the course, when a new assignment has been successfully added to the course, the course homepage for students will show this new assignment and with a download link, student can download the assignment and start working on the new assignment. later when students finished the assignment,the solution will be uploaded to the according assignment and teacher can review them at the evaluation part of the course's homepage and add a evaluation to the solution. When the evaluation is successfully saved to the assignment, the student will get the results directly at course homepage.

3.1.4 Communication system

a. Forum

The HMS system has a standard client-server structure, the client and server communicate with each other over internet using HTTP protocol.[1] HTTP has a typical

"Request-Response" pattern, the web client sends a request to the web server, web server serves a response according to the web client request. It is a simple but powerful solution to provide a two-way conversation for two party over one channel.[2] The forum function within the HMS system works also after this pattern.

b. Instant message

The standard HTTP protocol however is not suited for the instant message system, because of the "Request-Response" message pattern, user has to manually ask for a content upgrade. But a instant message system needs automatically update the chat contents on both side of a conversations while a new message has been added[3]. therefore a full-duplex communication system "Web Socket"[4]will be used to back up the message module. Figure 3.8 illustrate a simple scenario of a dialog based on web socket. after logging into their HMS account user A and user B are both connecting to the HMS web socket server, later on user A send a new message to user B,first the request from user A are passing to the web socket server,the server processed the request accordingly and served the response not only to the user A but also automatically served the response to the user B,this ensure the both side of the conversation can have their message received in real time.

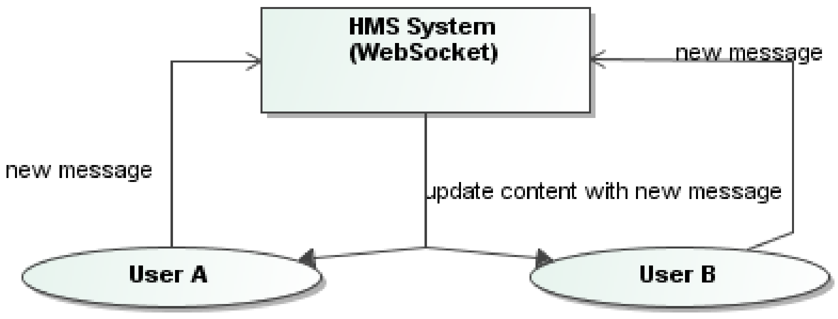


Figure 3.8: Web socket message pattern

3.2 Specific functions

There are three major problems in other homework management platforms:

1. Management of file submission
2. Centralized data persistent(single database)
3. Complicated back up process

the first problem is the management of the student's submission, besides of simply uploading the file to the server ,there will be usually more additional function needs to be added to support different needs related to hand in homework. for instance the documentation of moodle has suggested several submission type related to file submission[5]:

1. Student submit a work and teacher download it later.
2. Student submit a work multiple times.
3. Student submit a work with a response.

and teacher needs to do several settings to make those type work properly, this introduced unnecessary extra cost not only for development but also for the user of the system. on the other hand the this approach didn't take the "Type II" 3.6course into consideration, it is possible to hand in the semester project to the server just using this method, but uploading a whole semester project to the system at the end of semester is sometimes risky for the students(Data loss of the computer before dead line), and the teachers may need to trace the project history to comprehend the work of students(avoiding plagiarism).

the second and third problem are related to each other, because all the data are persistent in a single data base, the backup procedure is complicated and time consuming. all these platform including HMS are designed for a faculty in university, it's very common once in a while the faculty need to archive or backup the old data from the past term. A possible way is query out all the related data based on the semester, then dump these data into file and save the course related files some where else. Moodle also use this approach to make a course backup[6]. since the amount of data will increasing rapidly after years of use, this approach can only consume more times.

The HMS has introduced two new approach two avoid the problem from above:

1. Git based file submission
2. Dynamic data management

this two methods will be detailed introduced in the rest of this section.

3.2.1 Submission management using GIT

Instead uploading a file to a simple folder on the server, the students will obtain a git repository when they first entering the course. using a git repository other than a normal file folder has several advantages. firstly the implementation of different submission type is not needed anymore and don't need user to do extra settings.

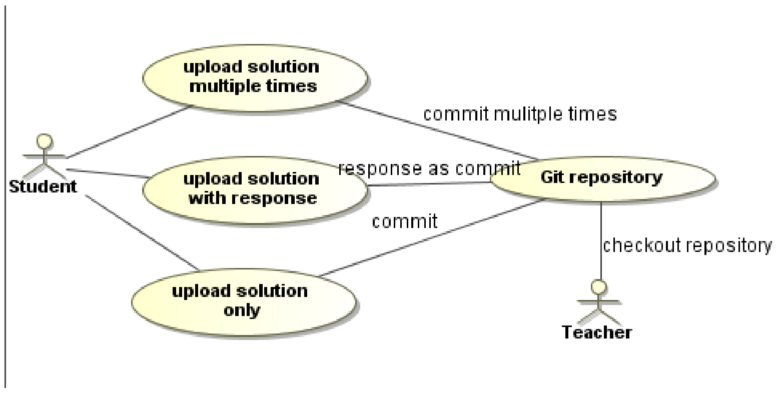


Figure 3.9: Workflow of git

Figure 3.9 shows the use case for all submission type[5] when using the git repository. no matter what submission type was chosen by the user, the only difference is how many commits were made. especially by the submission type "*upload solution multiple times*". since the nature of the git repository is to track the history of file change, teacher can easily review all the changes were made between the uploads without extra efforts. on the teacher side they just need to checkout the students git repository and review the change history and give the evaluation. secondly using a git repository in "*Type II*" course is more convenient for students and teachers to manage the project, since the assignment of "*Type II*" course is usually a semester project, the students can use this repository to host their project to avoid the risk of data loss, or not able to submit whole project before deadline. the teacher can also more easily track the working history of the student's project.

since there are two types of course(Figure: 3.6), the git repository will also work under two modes to fit the property of course:

1. Local modes(for "Type I" course)
2. Remote modes(for "Type II" course)

a. Local modes

for the "*Type I*" course students needs to upload different solutions to different assignments, and every assignment needs to be evaluated individually, so the repository should have a proper file structure to distinguish the student submission based on the assignment. Figure 3.10 shows the file structure inside the local git repository. With this file structure the teacher can manage the student submission more efficiently

In order to maintain this file structure and avoid introducing unnecessary errors, the address of the git repository is hidden from the student, they can only using the course homepage to upload their solutions.

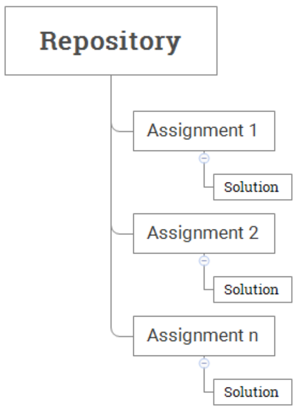


Figure 3.10: Repository file structure in local modes

b. Remote modes

for the "Type II" course students will obtain the repository address at once when they join the course, and use this repository to host their semester project. This part works similar as using a remote repository. students needs to know their the repository address and using the "Push" command of the Git to push the changes from their local machine to the HMS server. Only at the end of the semester or the deadline is due students need to use the course homepage to tag the repository, this can generate a evaluation request at the teacher side so that the teacher can register the evaluation result into the system.

3.2.2 Dynamic data management

In order to make the backup or achieve process as easy as possible. The HMS system introduced a new approach to manage the data. first of all the HMS system use multiple databases instead of one central database.

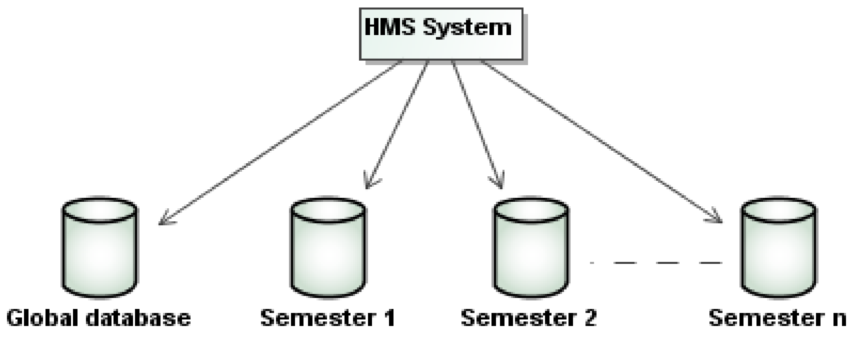


Figure 3.11: multiple database based on semester

Figure 3.11 shows the database structure in the HMS system. The Global database contains all the authentication data including ssh public keys,user name, user email, user role and passwords. this database is only used for the user management(section 3.1.1). another database is semester database. since the data structure of every semester is same, so the data model for semester can be reused in every new semester. every time when a new semester begins, a new clean database for this semester will be generated using the same data structure in real time. therefore in this case there is no need to query the semester data for backing up database.

also the database in this project should work under the embedded modes using file storage, first of all the embedded database runs directly in the application that uses them,it requires no extra server and no maintenance for the database itself. Another advantage of embedded database is the speed, because all the database operations happen inside the application process.[7]since all the relevant data of the database are saved in a single file, backup the semester database only need to copy the database file to some where else.

In this project the H2 database engine will be used as the default database engine. first h2 support the embedded mode(file storage) and it is purely written in java. another reason is that the H2 database support a mixed mode(Figure 3.12),mixed mode is a combination of the embedded mode and server mode, the first application (in this case the HMS) will use the database as embedded mode, but it also starts a server so that the other application(a SQL query tool) can still side load the database. normally embedded database runs within the application, so it is hidden from the end user[7], so there is no way the user can side load the database, but in the real life of maintaining the HMS system, direct accessing and manipulating the database using a SQL query tool sometimes is more efficiency than the usual routine. It is also important to notice that if the application is shut down, the server mode will also close all the connections[8],therefore side loading a database using remote mode can only take place when HMS is still running. However the database file generated by the h2 engine can still be loaded from the h2 web based manage tools without needing HMS system to be on line. this feature is important specially for a archived h2 database, it means that all the contents of the database can still be freely accessed without extra works.

besides backing up the database, the files from student submission, course materials should also be backed up at the same time. using a unified file saving structure can make this process more easier..

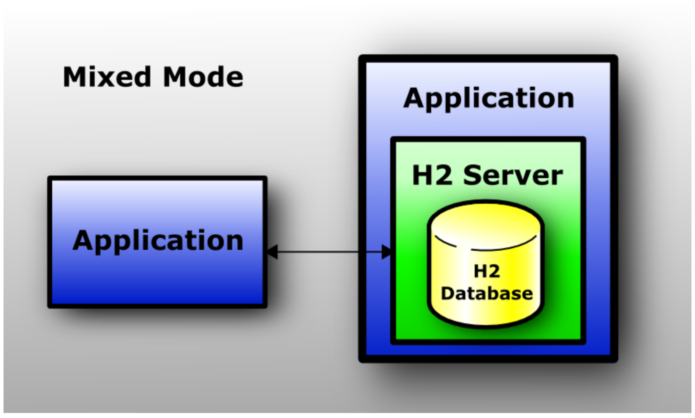


Figure 3.12: Mixed Mode of h2 server[8]

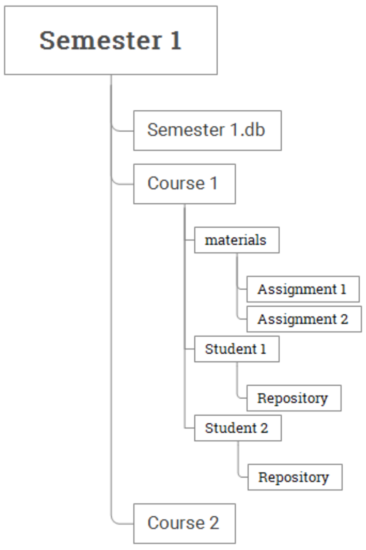


Figure 3.13: Semester file structure

Figure 3.13 shows the file structure of one semester. the uploaded course materials, student's repository are saved with the database file under the same folder. using a clear file structure to host the uploaded files along with the commit history of the student's repository can let the maintainer of the system more easily to distinguish the files without looking into the database. combine these two methods, the backup procedure of the HMS system only need the maintainer to relocate the semester folder and without doing any database operation.

This approach requires the system to create a database in the run time, therefore the standard database configuration of play framework can not be used. The details of implementing the dynamic data management will be demonstrated in the next chapter.

Chapter 4

4 Implementations

In this chapter the concrete implementation of the functions from the chapter 3 will be presented. because the data persistent is the precondition for other function to work, also it is the key features of the HMS system. the implementation of dynamic data management and git based file submission will be first discussed.

4.1 Dynamic Data management

Dynamic data management consists two parts. firstly HMS system can dynamically generate a new database every time when a new semester starts, so that created database will only contain the data related to that semester. secondly the physical files including the students submission, course materials are saved under a unified structure3.13.

4.1.1 Real-time database generation

Referenz: http://ebean-orm.github.io/

Play framework uses Ebean ORM to access the database,ORM is a technique to convert objected oriented programming language(in this case Java) into its persistence as a relational data base, so the data can freely exchange between a java object and database table.[9]By default the configuration of Ebean and the database is done by editing the configuration file of play framework(Figure 4.1). Developer need to define the details of a database and define the java data model for the Ebean server, and Ebean server will handle all the database operation. After the application started, there is noway to modified this configuration again in the run time. therefore this method can not be used

In order to dynamically manage the database, all the configuration has to be done pro-grammatically. Ebean supports database configuration pro-grammatically. but it still need a little modification to make it compatible with play framework.

Figure 4.2 is a static method which create a ebean server in run-time, this method need two parameters as input, first is the name of database, usually the name of a semester

```
# Database configuration
# You can declare as many datasources as you want.
# By convention, the default datasource is named `default`
# db.default.driver=org.h2.Driver
# db.default.url="jdbc:h2:file:~/data/playdb"
# db.default.user=sa
# db.default.password=""
# Ebean configuration
# You can declare as many Ebean servers as you want.
# By convention, the default server is named `default`
# ebean.default="models.*"
```

Figure 4.1: Database configuration

```
public static void createServer(String name, List<Class> entity) {
    ServerConfig config = new ServerConfig();
    config.setName(name);

    DataSourceConfig h2Db = new DataSourceConfig();
    h2Db.setDriver("org.h2.Driver");
    h2Db.setUsername("hms");
    h2Db.setPassword("test");
    h2Db.setUrl("jdbc:h2:tcp://localhost/~/data_dynamic/" + name + "/" + name)
    config.setDataSourceConfig(h2Db);

    Path p= Paths.get(System.getProperty("user.home"),"data_dynamic",name,name+".
        h2.db");
    File f = p.toFile();
    if(f.exists()){
    config.setDdlGenerate(false);
    config.setDdlRun(false);}
    else
    {
    config.setDdlGenerate(true);
    config.setDdlRun(true);
    }
    ...
    for(int i=0;i<entity.size();i++){
    config.addClass(entity.get(i));
    EbeanServer server = EbeanServerFactory.create(config);
    }
```

Figure 4.2: Create Ebean Server in run-time

for instance "*WS2016*", second is a list of java class, this list contains all the java data model related to this database, ebean needs to use this to create the table for the database. the configuration of the ebean server and database is strait forward, after setting all the parameters to the ebean server and data source, the ebean server will be created by the class *EbeanServerFactory*. this static method can be used when there is a need to create a new database. In the HMS, the decision of creating new database is made when teacher creates new course. creating a new course needs teacher to give the name of semester which this course belongs to, after the creating new course request is sent to the server, the server will first check whether this semester was registered already in the semester table of global database, if it is already registered, the course will be simply written to that semester database, if the inputted semester can not be found in the global database, a new database with the inputted semester name will be first created then the new course will be written in this database. after finishing the part of creating database in run time. there is only one thing needed to be added to make it work with play framework. since the databases are configured during the run time, when the application restarted, it will not restore those database automatically, because the play framework was originally designed to use the configuration file to track the database configuration, and in this case the configuration file is empty. The workaround for this issue is to track the database file, and reload all these database before the application restarted. Play framework has already provided a method to do some actions before actually starting the application, this can be used to reload the database. Kleines anderes Beispiel mit einer Klasse link zu Sourcedateien

```java
public class Global extends GlobalSettings{
    @Override
    public void onStart(Application application) {
        super.onStart(application);
        List<Class> entity = new ArrayList<Class>();
        entity.add(User.class);
        ...
        entity.add(SSH.class);

        List<Class> entity1 = new ArrayList<Class>();
        entity1.add(Semesteruser.class);
        ...
        entity1.add(Conversation.class);

        try {
            Server h2server = Server.createTcpServer("-tcpAllowOthers");
            h2server.start();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        createServer("global", entity);
        List<Semester> database == Semester.getallsemester();
        for (int i = 0; i < database.size(); i++) {
          createServer(database.get(i).semester, entity1);
        }
        }
    }
}
```

Figure 4.3: Reload the database at application start

Figure 4.3 demonstrate the process of reloading the database. list entity contains the data model for the "Global" database and list entity1 contains the data model for the "Semester" database(Figure 3.11), first the global database will be first reconstructed, because the name of other semester database are saved within the global database, then using a loop to reconstruct other semester database. it is also should be noticed that between the line of 12 and 21 in Figure 4.2 has an additionally implementations of database file detection, if the database file is present, the ebean server should not regenerate table relations because this action will erase all the data previously saved within this database.

It also should notice that because there are multiple databases, the name of the database should always be given every time when there is a CRUD(Create, Read, Update, Delete) operation taken place. for instance, saving a new course to the semester "WS2016", it should look like newcourse.save("WS2016");

4.1.2 Data models

There are two types database in HMS. one is a global database, it is used for authentication and additionally tracking the creation of semester database. another is semester database, this database saves all the data that related to the Course, Assignment, Evaluation, and Communication. so the design of data models for these two type database are also different.

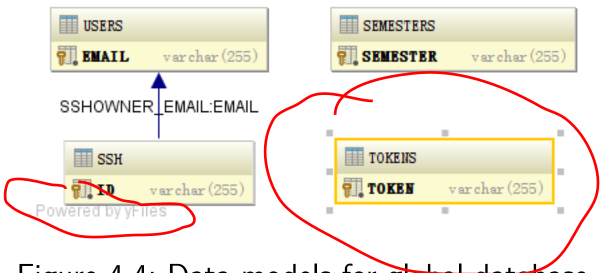


Figure 4.4: Data models for global database

Figure 4.4shows the data model for the global database, there are 4 class defined, first is the *Users*, this class saves all the data from the user registration, for instance the user id, email address, password, etc. second is the *SSH*, this class saves all the ssh keys for the user, and because one user may have multiple ssh keys, there is a "*One-To-Many*" relations between the *Users* and SSH. the third is Semesters, this class contains all the name of the created semester database, the last one is Tokens, this class is used to temporarily save the confirmation token for registration, change email and change password actions.

Figure 4.5 represent the data models of the semester database, because there are more activities using semester database, the model is more complicated than the global database. first is the *Lecture* class, all the activities around the homework management is about the lecture, a lecture includes assignment (Assignment), git repositories(Repos), a forum(Thread) and the lecture evaluation(Evaluation) for students. and each assignments should generate a lot of hand ins(Handins) from the students. besides the lecture related data, the data of chat system is semester related and independently of the lecture. at last all these data are related to the Semesteruser. Semesteruser and User are both subclass extends from super class *Abstractuser*(Figure 4.6).

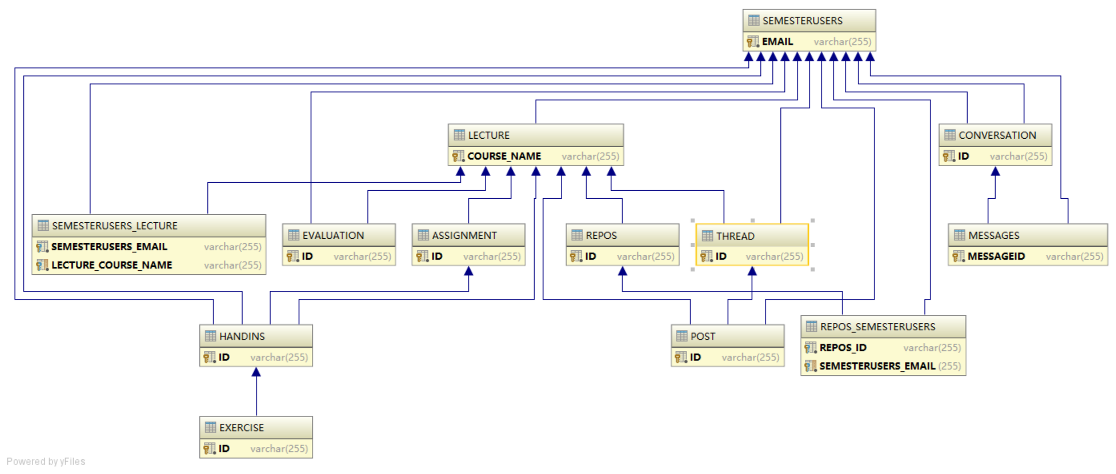


Figure 4.5: Data models for semester database

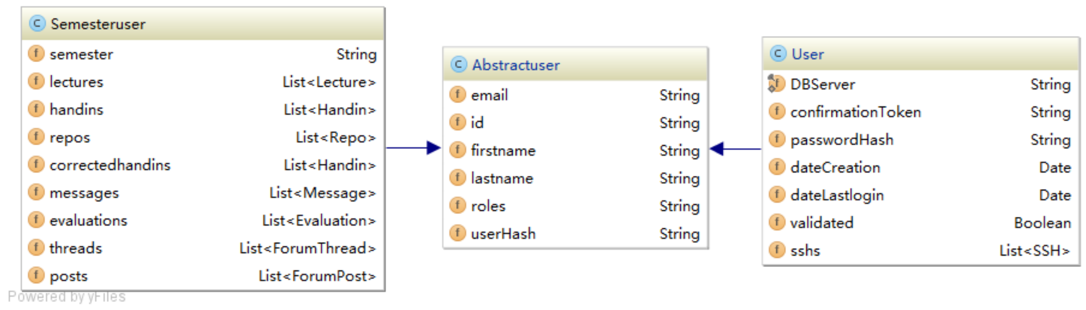


Figure 4.6: Class Diagram Semesteruser

Ebean supports the JPA (Java Persistence API) annotation "*@MappedSuperclass*", this annotation designates a class whose mapping information is applied to the classes that inherit from it, but will be no table generated for the mapped superclass itself. [10]. The common data which will be used both for User and Semesteruser are defined in the class "*Abstractuser*", the database specific data will be defined in the subclass. using this method can first avoid the redundant code, second the semester data can be completely separate from the other data. when a register user want to sing up a new course in a semester, his user information should be first transformed from the "User" to "Semesteruser".

as Figure 4.7shows, when a user want to sign up a course, the system first find his *Semesteruser* information, if the *Semesteruser* object for the current user was not found, a new *Semesteruser* object will be generated in the semester database based on the user information from the global database.

```
public static Semesteruser getSemesteruserfromUser(String database,User user){
        Semesteruser semesteruser = null;
        try {
            semesteruser = Semesteruser.findByEmail(user.email, database);
        } catch (Exception e) {
            semesteruser = null;
        }

        if (semesteruser == null) {
            semesteruser = new Semesteruser();
            semesteruser.email = user.email;
            semesteruser.firstname = user.firstname;
            semesteruser.id = user.id;
            semesteruser.lastname = user.lastname;
            semesteruser.roles = user.roles;
            semesteruser.userHash=user.userHash;
            semesteruser.semester = database;
            semesteruser.save(database);
            return semesteruser;
        }
        else{
            return semesteruser;
        }
    }
```

Figure 4.7: User To Semesteruser

4.1.3 File structure

Besides the database design, the semester related files should also be saved under a unified structure(Figure 3.13). define the upload path within the play framework is strait forward, after the file was uploaded to the server, then the file can be moved to the desired location by the method `FileUtils.moveFile();`.

saving the course repository of students is a little complicated, HMS system using the gitolite to manage the git repository, and repository created by the gitolite is bare repository which doesn't contain a working directory[11], so it is pointless to just copy a bare repository to other place. also the gitolite has its own file structure and can not be changed. so a possible way is using git Java API Jgit[12] to make a clone of this bare repository to the desired destination.

```
Repo repo=Repo.findRepoByLectureAndOwner(assignment.semester,semesteruser,
    assignment.lecture);
Git git = Git.cloneRepository()
 git .setURI(repo.repofilepath);
     .setDirectory(localPath);
 git .call();
```

Figure 4.8: Clone repository using JGit

Figure 4.8shows using JGit(reference here) to clone a bare repository from gitolite to a "local path", which fits the file structure of HMS. This procedure also makes submission a file to gitolite repository possible. this will be closely discussed in the next section.

4.2 Integration of Git

The HMS system using git repository to manage the student's submission. The implementation of this function should consider following requirements:

1. Access control: students can only have access to their own repository, but the teacher or assistant of the course can have the access but limited to read to all student's repository.
2. Local and Remote modes: to support two types of course3.6

The most important requirement is the access control over the git repository. because Git by itself does not do any access control, it relies on the transport medium(authentication of HMS system) to do authentication, and file permissions of the operation system to do authorization(read or write permission)[13]. without a proper access control over the git repository, it is then impossible to integrate the git into the HMS. since the basic needs of managing homework is to keep the student's submission only between the marker and students themselves.

The first part of this section will present a method using "Gitolite" and "Java-Gitolite-manager" to solve the access control problem of git repository. and the usage of git repository for both course type will be introduced in the rest part of this section.

4.2.1 Access control using Gitolite

The access control over the student's repository is pretty simple.As Figure 4.9 shows, only the students them self have fully access rights for their course repository and can not get inside other repository. teacher and assistants have to grant a read right only for evaluation.

Repository Access level	Read	Write
Student	●	●
Other Student	○	○
Teacher	●	○
Assistant	●	○

Figure 4.9: Access level for different user

This access rules can be easily managed by Gitolite. After properly installing the gitolite in the server, it will generate a "gitlite-admin" repository under current user's home folder. within this folder there is a plain text file which will be used by Gitolite to specify the access rules.

```
repo foo
 RW = alice bob
 R  = carol david
```

Figure 4.10: Configuration of access rules with gitolite

Figure 4.10 is a simple configuration of the access rules for the repository "foo".for this repository, Alice and Bob both have read and write access, but Carol and David can only read the content of the repository. The server maintainer only need to modify this configuration file and push the changes using a git command back to the Gitolite all the changes will be adopted automatically by gitolite. additionally the ssh public keys of the user of this repository are also needed to be copied into "gitolite-admin" repository. since gitolite use ssh mechanism to authenticate the user.

Java-Gitolite-Manager

Gitolite gives the possibility to add access control to git repository, but it is still need system maintainer to edit the configuration file manually, therefore is still can not be directly used in the HMS system. Delft University of Technology has developed a java library which enables developer to manage the gitolite configuration direct from Java.

```java
ConfigManager manager = ConfigManager.create("/home/gitolite-admin");
Config config = manager.get();

User user = config.createUser("alice");
user.setKey("desktop", "ssh-rsa AAAB3Nz...");

config.createRepository("foo")
     .setPermission(user, Permission.ALL);
manager.apply();
```

Figure 4.11: Gitolite configuration from java

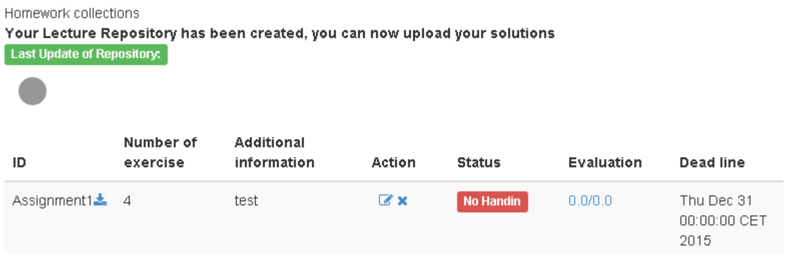
Figure 4.11 is a example to use "Java-Gitolite-Manage" to configure the gitolite from java. first an instance of "ConfigManage" is created with the path of repository "gitolite-admin", then a user with user name "Alice" is created alone with the ssh public key of user Alice. after user has been created, a new repository "foo" will also be created and user Alice will be added to this repository with all permission.

HMS system use the same procedure as the example in Figure 4.11, every time when a student joins a course, no matter what this course type is,a repository will be generated for this student if the ssh public of this students is present. and when a teacher want to review a repository of a student, HMS system will first get out the repository and automatically set a correct permission for teacher using `setPermission(teacher, Permission.READ_ONLY);`.

4.2.2 Local mode

Under local mode, students need use course homepage to upload their solutions and actual repository address is hidden from the student. Figure 4.12 shows the homework submission part for the course under local mode. The gray dot above shows the current status of the repository, under the indicator of repository status, is the homework area, student can download the assignment's material and use the predefined actions: commit and revert.

The left icon under action column is used for committing changes, student need to use this action to upload the solutions. the icon on the right is the predefined action for deleting last submission, students can use this action to delete their last submission


Homework collections
Your Lecture Repository has been created, you can now upload your solutions
Last Update of Repository:

ID	Number of exercise	Additional information	Action	Status	Evaluation	Dead line
Assignment1	4	test		No Handin	0.0/0.0	Thu Dec 31 00:00:00 CET 2015

Figure 4.12: Homework submission for course under local mode

The repository created by the gitolite is a bare repository, a bare repository is an named file directory with a .git suffix which contains only the administrative and control files of the repository, and it doesn't have any copy of the files that present in a normal git repository[14]. The student's submission then can not be directly uploaded to this repository. also HMS system requires a unified file structure to save the physical files. The solutions to this problem has already been presented in the section 4.1.3, that is using JGit to clone the bare repository to a normal repository at a desired location.

JGit

JGit is a Java library for working with git repository, with JGit all the operations for a git repository can be realized from Java.[12]Besides only clone the bare repository presented in Figure 4.8, HMS system needs also upload the students submission to the cloned repository and using JGit to commit the changes and push the changes back to the gitolite repository, so that the teacher can clone the student's repository remotely.

Commit

After student choosing the commit action, a dialog(Figure 4.13) will pop up. Using this form student can upload their files with a commit message, if the commit message is empty, a default commit will be made by the HMS system.

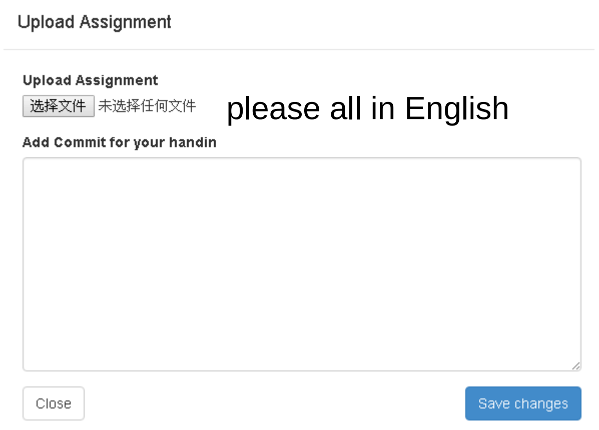


Figure 4.13: Commit dialog

Figure 4.14 shows what happened after submission file and commit message were passed to the HMS system. first the gitolite repository will be cloned to the *localPath*, after the clone operation finished, the uploaded file from the student will be copied into the newly created local repository, then using jgit to commit the changes, after that all the changes will be pushed to the master branch of the gitolite repository. in the end a new "hand-in" object will be generated and saved into the database under the name of student. this will be used by teacher to evaluate the submission.

```
public static Result handinhomework(...){
    ...
try{
MultipartFormData body = request().body().asMultipartFormData();
FilePart homeworkfile = body.getFile("homeworkfile");
if (homeworkfile != null) {
    String fileName = homeworkfile.getFilename();
    Git git = Git.cloneRepository()
    .setDirectory(localPath);
    ...
    FileUtils.moveFile(file, new File(localPath, des+fileName));
    git.add().addFilepattern(des+fileName).call();
    git.commit().setMessage(commit).setAuthor(...).call();
    RefSpec refSpec = new RefSpec("master");
    git.push().setRemote("origin").setRefSpecs(refSpec).call();
    git.getRepository().close();
    ...
    Handin handin= new Handin();
    ...
    handin.save(semester);}
```

Figure 4.14: Commit submission

Delete submission

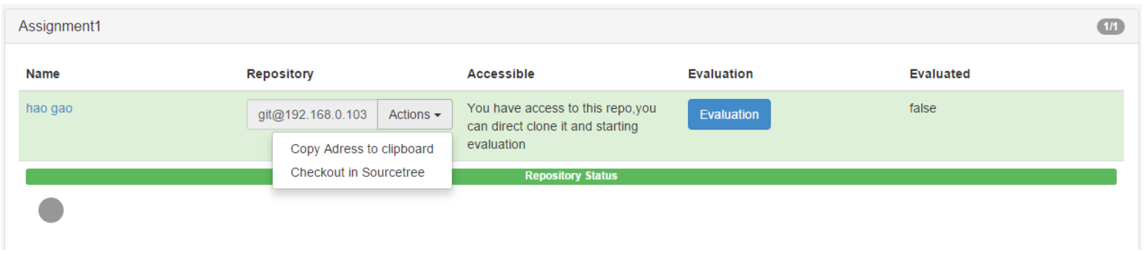
If a student didn't satisfy the first submission for an assignment, the last submission can also be deleted. the procedure is almost the same as submission, first the local repository created by the submission action will be updated to the latest state, second since the submission under the local modes are saved under a unified structure(Figure 3.10), the folder of relevant assignments will be deleted. in the end the delete action will also be committed by the HMS system and the changes will be pushed back to the gitolite repository. at last the related hand-in object from the submission will also be deleted and regenerate by the next submission action.

```
Git git = Git.cloneRepository();
        String subfolder=assignment.title;
        git.rm().addFilepattern(subfolder).call();
        git.commit()
    .setMessage(commit)
    .setAuthor(semesteruser.lastname,semesteruser.email)
    .call();

Handin handin=Handin
        .getHandinofassignmentofstudentinlecture();
    if(handin!=null){
    handin.delete(semester);}
```

Figure 4.15: Delete last submission

Check out by teacher

After student have submit their solutions to the repository, teachers can check out the students repository for evaluation.



Assignment1 1/1

Name	Repository	Accessible	Evaluation	Evaluated
hao gao	git@192.168.0.103 Actions	You have access to this repo,you can direct clone it and starting evaluation	Evaluation	false

Figure 4.16: Checkout student repository

Figure 4.16is the user interface of evaluating students submission. it shows the address of the student repository and its status, and whether the teacher has already the access to this repository. if user has already submit a solution the row of this user will be in color green otherwise it is red. teacher can use the action button to choose an action towards the students repository. the student repository can be directly checked out into the git client "*Source tree*", or teacher can just copy the address of the repository and check out in a favorite git client.

The precondition of checkout the students repository is to grand an access. when teacher first time opens an evaluation tab(Figure 4.16), the method *grandaccess()* in Figure 4.17 will be automatically executed.

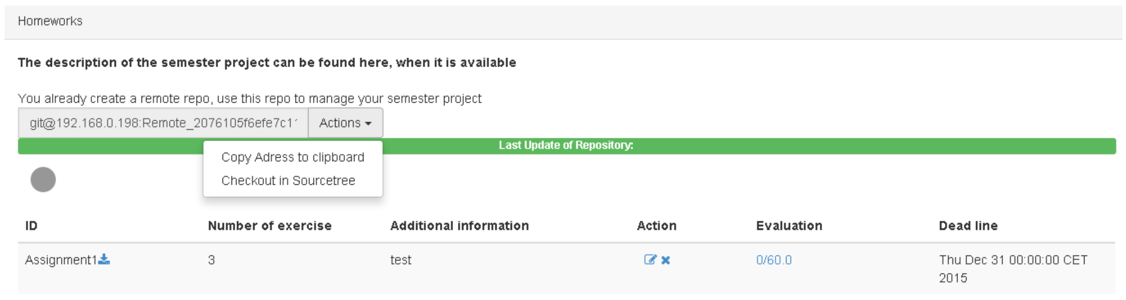
```
public static boolean grandaccess(...) {
if(!admincredential.sshs.isEmpty()
    &&!studentrepo.owner.contains(currentadmin)) {
    User teacher = config.ensureUserExists(teacher.userHash);
    String reponame = lecture.courseName + "_" + student.userHash;

    Repository repository = config.ensureRepositoryExists(reponame);
    repository.setPermission(teacher, Permission.READ_ONLY);

    manager.applyAsync(config);
    studentrepo.owner.add(currentadmin);
    ...
  }
```

Figure 4.17: Grand access to students repository

first the ssh key of teacher must be present, since Gitolite needs ssh key to authenticate the user. second if the teacher already has the access to this repository, system should avoid to run this method again. if the preconditions are fulfilled the teacher will be added to the repository with a read only permission. also the information about teacher already has the access to this repository will be saved into the database.

4.2.3 Remote mode

Remote mode is much simpler than local mode. Figure 4.18 is the user interface of homework area in a "Type II" 3.6course under remote mode. the address of git repository will be directly given to the students. students will use this repository to host their semester project, like using a normal remote git repository. so there will be no more submission over HMS system. only before the dead line the students have to login into their account to use the predefined hand in action to hand in their project. so that the teacher can use the HMS system to give a final evaluation to the project and saved this into the database. and at the same time a copy of the student's repository will be saved under the unified file structure, this is same as the submission action(Figure 4.14) from the local modes. the only difference is that remote mode will not upload any files.


Homeworks

The description of the semester project can be found here, when it is available

You already create a remote repo, use this repo to manage your semester project

git@192.168.0.198:Remote_2076105f6efe7c1

ID	Number of exercise	Additional information	Action	Evaluation	Dead line
Assignment1	3	test		0/60.0	Thu Dec 31 00:00:00 CET 2015

Figure 4.18: Remote mode

4.3 User management

The implementation of user management system consists following parts:

1. Registration system
2. Authentication and Authorization system
3. Self manage system

4.3.1 Registration system

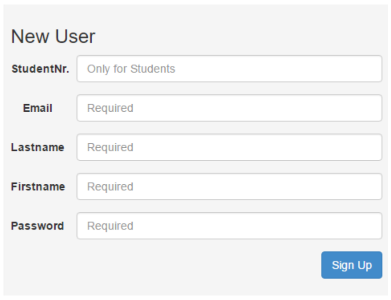


Figure 4.19: Registration form on home page

Figure 4.19 is the registration form on the HMS home page. a new user need to at least type in the email address,first name,last name, and the password. The student number is optional because the multi user roll management of the system. only students needs to give the students number and get a student user role as default. other user will get a random user id and "default user" user role, and need the system admin to change the user role afterwards. Figure 4.20 shows what happens when user click the sing up button in the registration form.

The sign up button is bonded with the function "*Signup.save()*", at first the data from the *registerForm* will be checked whether all the required field has been filled, if there is a error the user will be redirected to the home page and do the registration process again. second if the data are correct, the emails from the *registerForm* will be checked by the function *checkBeforeSave()* to insure that the email address has not be taken by other users. if both tests are proofed, the next step is to check the user id, if the user id has been given then this user should be saved with the user role of students, if the user id is empty, the user will obtain a user role of default user. after saving the user registration data alone with the confirmation token to the database, a confirmation email with user confirmation token will be sent to the user email address by the function *sendMailAskForConfirmation()*. At this step the user data are successfully saved into the database and waited to be confirmed by the user.

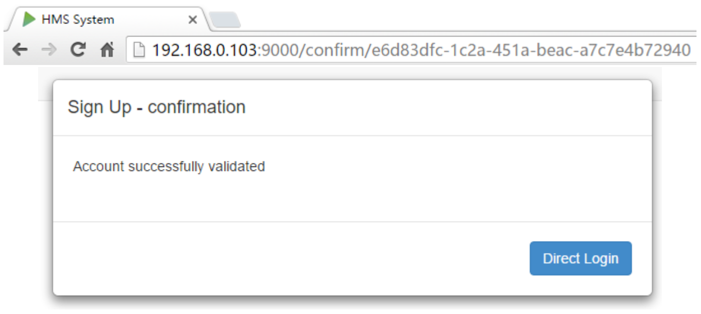
```
public static Result save() {
        Form<Application.Register> registerForm = form(Application.Register.class)
            .bindFromRequest();
        if (registerForm.hasErrors()) {
            return badRequest(index.render());
        }

        Application.Register register = registerForm.get();
        Result resultError = checkBeforeSave(registerForm, register.email);

        if (resultError != null) {
            return resultError;
        }

        try {
            User user = new User();
            if(register.id==null||register.id.isEmpty()){
                user.id= CreateExternalId.generateId();
            }
            else{
            user.id=register.id;}
            ...
            user.roles=UserRoll.Students.toString();
            user.confirmationToken = UUID.randomUUID().toString();
            user.save("global");
            sendMailAskForConfirmation(user);
            ...
    }
```

Figure 4.20: Save user registration

```
private static void sendMailAskForConfirmation(User user){
        String subject = Messages.get("mail.confirm.subject");
        ...
        urlString += "/confirm/" + user.confirmationToken;
        URL url = new URL(urlString);
        String message = Messages.get("mail.confirm.message", url.toString());

        Mail.Envelop envelop = new Mail.Envelop(subject, message, user.email);
        Mail.sendMail(envelop);
    }

public static Result confirm(String token) {
        User user = User.findByConfirmationToken(token,"global");
        ...
        if (User.confirm(user,"global")) {
            sendMailConfirmation(user);
            user.dateCreation=new Date();
            user.save("global");
            return ok(views.html.account.signup.confirm.render(user));
        }
    }
```

Figure 4.21: Confirm registration

The user need to use the hyper link(contains confirmation token) to confirm their registration, after user clicked the confirmation link, the confirmation token will be passed to the method *confirm()*, the system then using this token to find the correct user record within the database, if the user was found then a welcome email will be sent and mark this user as confirmed in the database.


HMS System

192.168.0.103:9000/confirm/e6d83dfc-1c2a-451a-beac-a7c7e4b72940

Sign Up - confirmation

Account successfully validated

Figure 4.22: Registration successful

if the details of the user successfully updated in the database, the system will render a modal and user can direct log into their account and start using the system. Figure 4.22 shows the result of a successful registration, the URL from above is the confirmation link with the user token.

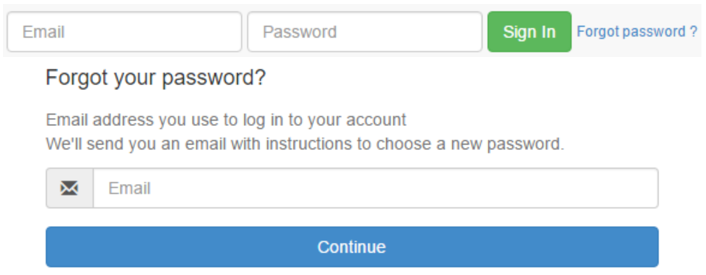
4.3.2 Authentication and Authorization system

Authentication

The authentication system consists two parts:

1. Authentication of HMS system
2. Authentication of Git server

The first part is the authentication of the HMS system, this part controls the user login activities and protect the system from unauthorized user actions, for instance a user with students role can not create a new course or make evaluation to a assignment.



Forgot your password?

Email address you use to log in to your account
We'll send you an email with instructions to choose a new password.

Figure 4.23: Login and Password recovery

Figure 4.23 shows the component of authentication of a user login. user used the login form to input their email and password into the system, system will first find the user based on their email address, and then check the password from the database record and the password from user input, since the password is saved as SHA-256 in database for security reasons, the clear password of user input should first convert into SHA-256, then both passwords will be compared, if they are identical, then user will be redirect to their account homepage.

another function of authentication system is to recover the user password. if the user click the link " Forgot password? " a new page will be generated and ask user to input their email address. after user input their email address, the method from Figure 4.24 will executed.

```
public static Result runAsk() {
        Form<AskForm> askForm = form(AskForm.class).bindFromRequest();
        User user = User.findByEmail(email,"global");
        if (user == null) {
            sendFailedPasswordResetAttempt(email);
            return ok(views.html.account.reset.runAsk.render());
        }
        try {
            Token.sendMailResetPassword(user,"global");
            return ok(views.html.account.reset.runAsk.render());
        } catch (MalformedURLException e) {
            Logger.error("Cannot␣validate␣URL", e);
        }
        return badRequest(ask.render(askForm));
    }
```

Figure 4.24: Reset Password

runAsk() first check whether a user with the input email exist in the database. for security reason, if the user did not exist, the system shouldn't expose any result to the user, instead sending a email to notify the person that the reset has failed. if the user do find with the input email, the rest actions will be same as the registration, the user will receive a email with a hyperlink with reset token, click this hyperlink will generate a password reset page, at this page the user can modify their passwords.

The second part of authentication related to the git server, because the git function is using gitolite to manage the access rights of the student's repository, it has a separate authentication system using SSH public key mechanism. The detail implementation has already been discussed in the section 4.2.

Authorization

Beside the authentication of a user, HMS system still need to authorized the user a proper access rights based on the user role(Figure 3.5). The play framework use actions to serve HTTP request, an action is basically a Java method that processes the data from the HTTP request.[15]if an unauthorized person can mock a correct HTTP request, this person can basically execute any actions implemented in server logic. in order to prevent an unauthorized actions, play framework come with a built in authenticator action called *Secured*. [15] in the case of HMS system, some actions can only be executed by user role of teacher, the additionally secured class implementation should distinguish the current user role and decide whether the actions should be executed for the current user.

```
public class Securedteacher extends Security.Authenticator{
@Override
public String getUsername(Http.Context ctx) {
    User current=User.findByEmail(ctx.session().get("email"), "global");
    if(current!=null) {
        if (current.roles.equals(UserRoll.Teachers.toString())) {
            return ctx.session().get("email");
        } else {
            return null;
        }
    }else{
        return null;
    }
        }

@Override
public Result onUnauthorized(Http.Context ctx) {
    User current=User.findByEmail(ctx.session().get("email"), "global");
    return ok(forbidden.render(current));
        }
}

@Security.Authenticated(Securedteacher.class)
public static Result createlecture() {
              ...
         }
```

Figure 4.25: Secured actions

Figure 4.25 shows the secured class implementation and usage for user role teacher. at first using the *HTTP.Context* session to get current user, then compare the current user role to the required user role, if they are identical, the email address of the current user will be returned, otherwise *null* will be returned. after the definition of the play framework documentation. if method *getUsername()* returns a string, tagged action *createlecture()* will be executed for the current user. if *getUsername()* returns value *null*, the method *onUnauthorized()* will be executed, in this example, a web page "*forbidden*" will be generated for the current user, and notice the user that this action can not be accomplished with his current user role.

4.3.3 Self management

Figure 4.26 shows the component of the self management system, at the settings page of user, user can update their ssh, password and email. updating password and email using the same mechanisms from the section 4.3.2, if a user want to change their email or password, the HMS system will always sending a confirmation email with a confirmation token, only after user confirming the changes from the URL within the email, the changes then can be saved into the database, it is a necessary step to offer

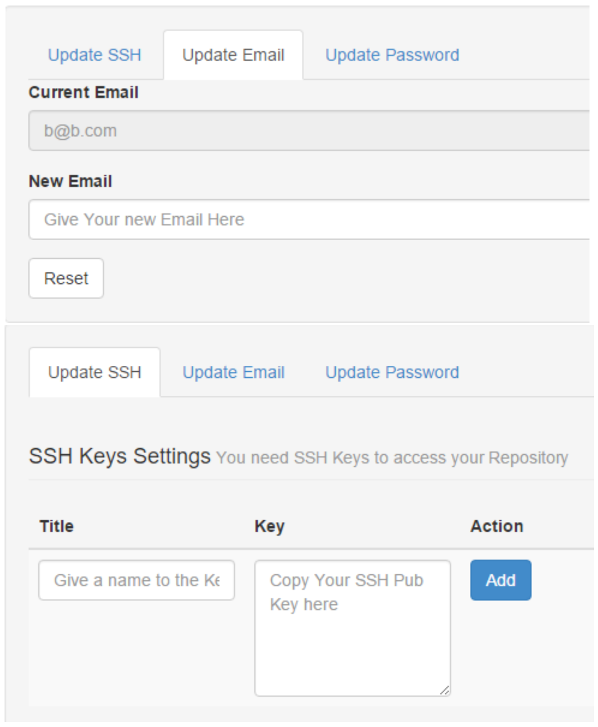


Figure 4.26: Self management

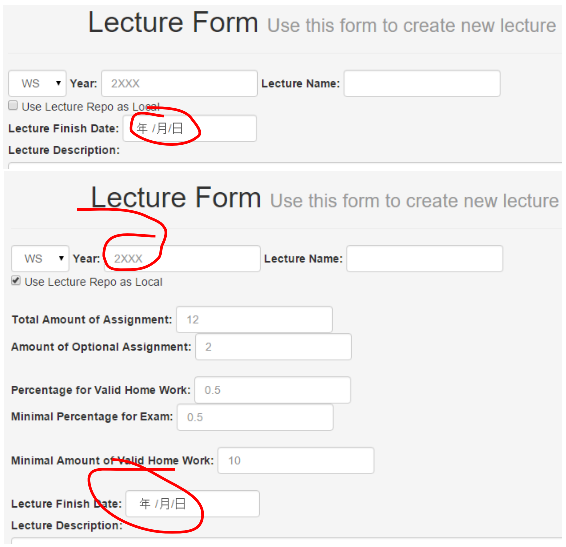
more security to sensitive data. and in order to use the git repository, user also need to add at least one ssh public key from their working terminals. because the HMS system using the gitolite to manage the git repository, and gitolite using ssh keys to manage the access rights of the git repository. the implementation of integration gitolite has already be showed in the section 4.2.

4.4 Course management

The management of a course including two parts: creating new course and the enrollment of the course participants.

4.4.1 Creating new course

Because the different types and features of a course3.6, the function of creating new course must be able to cover this two type of course. also creating a new course is related to the database generation, it should also have the ability to decide whether a new semester database should be generated.


Lecture Form Use this form to create new lecture

WS Year: 2XXX Lecture Name:

Use Lecture Repo as Local

Lecture Finish Date: 年 /月/日

Lecture Description:

Lecture Form Use this form to create new lecture

WS Year: 2XXX Lecture Name:

✓ Use Lecture Repo as Local

Total Amount of Assignment: 12

Amount of Optional Assignment: 2

Percentage for Valid Home Work: 0.5

Minimal Percentage for Exam: 0.5

Minimal Amount of Valid Home Work: 10

Lecture Finish Date: 年 /月/日

Lecture Description:

Figure 4.27: GUI of creating new course

User interface

Figure 4.27 shows the user interface of creating a new course. the form above is used to create the "Type II" course with a remote git repository, this type of course only contains a semester project and didn't have a normal final exam so there isn't much information needed for creating the "Type II" course, only the deadline and a description of the course are needed. another form from under is for the "Type I" course. in the "Type I" course student need to do various homework and earn enough points to enter the final exam. so when creating this kind of course, teacher should provide the precondition to the final exam, like the number of assignments and how much points are needed to enter the final exam. All these settings can still be modified after course has been created

Server-side logic

After sending the course creation form to the server, the data from the from will be process by the method *createlecture()* .Figure 4.28 shows the process that the newly created are saved into a semester database.

the first thing to check is the semester name of this course, if the semester name of the course didn't present in the semester tracking database, it simply means a new semester has already begun, so the first thing system will do is to generate a new database for this course and then saving the new semester into the semester tracking table. after that the course will be saved under the correct semester database.

if the semester name of a new course do present, the course will be simply saved under this course name.

```
if(!createlectureForm.hasErrors()) {
        String semester = createlectureForm.get().yearprefix
                    + createlectureForm.get().year;
        User globaluser=User.findByEmail(ctx().session().get("email"),"global");

        if (Semester.findsemester(semester) == null) {
            List<Class> entity = new ArrayList<Class>();
            entity.add(Semesteruser.class);
            ...
            entity.add(Conversation.class);
            createServer(semester, entity);
            Semester addsemester = new Semester();
            addsemester.semester = semester;
            addsemester.save("global");
        }
        Lecture lecture = new Lecture();
        lecture.semester = semester;
        lecture.courseName = createlectureForm.get().coursename;
        ...
        lecture.closingdate = createlectureForm.get().closingdate;

        Semesteruser semesteruser=Semesteruser.getSemesteruserfomrUser(semester,
            globaluser);
        lecture.lasteditor = semesteruser;
        if (!lecture.attendent.contains(lecture.lasteditor)) {
            lecture.attendent.add(lecture.lasteditor);
        }
        try {
        lecture.save(lecture.semester);
        ...
    }
```

Figure 4.28: Logic of creating new course

4.4.2 Course enrollment

After creating the course, it is time to let the students or other teacher and assistant to join the course. the first part of course enrollment is to list all the available course to the user of HMS.

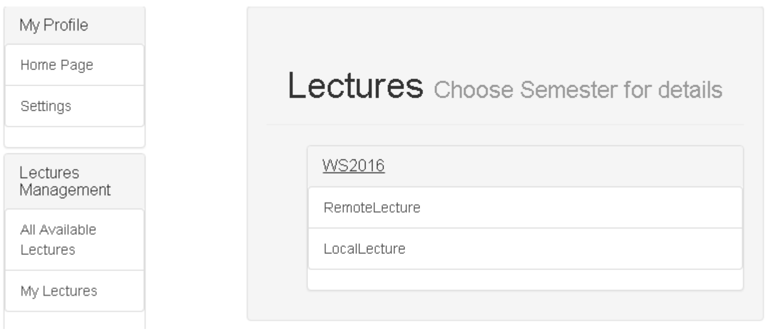
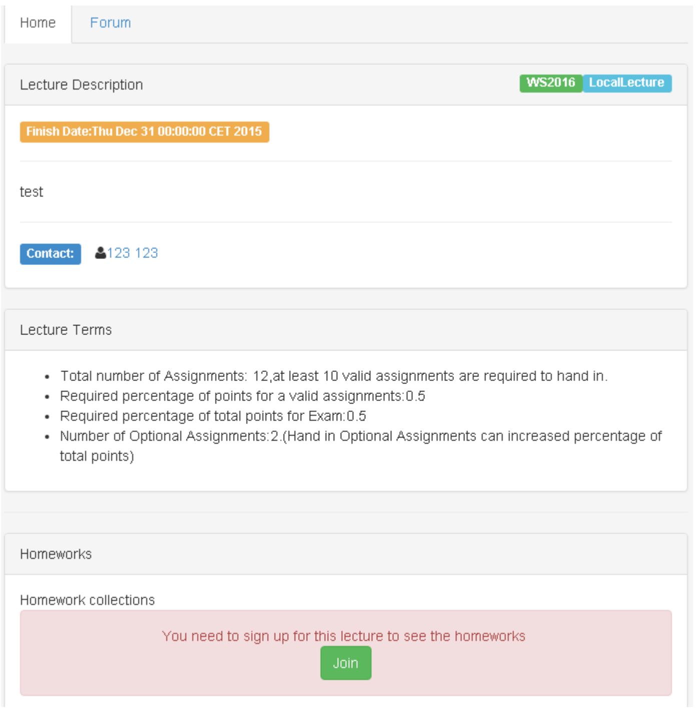


Figure 4.29: Browsing the course

Figure 4.29 shows all the available course under the semester "WS2016", when user click one of the courses, the homepage of the course will show up and ask user to join the course(Figure 4.30).


Home Forum

Lecture Description WS2016 LocalLecture

Finish Date:Thu Dec 31 00:00:00 CET 2015

test

Contact: 123 123

Lecture Terms

- Total number of Assignments: 12,at least 10 valid assignments are required to hand in.
- Required percentage of points for a valid assignments:0.5
- Required percentage of total points for Exam:0.5
- Number of Optional Assignments:2.(Hand in Optional Assignments can increased percentage of total points)

Homeworks

Homework collections

You need to sign up for this lecture to see the homeworks

Join

Figure 4.30: Course homepage before sign up

before actually signing up the course, the students or other user can only viewing the lecture description, the lecture terms. the lecture part of homework and forum are blocked.

Join the new course

if a user has a user role above "Defaultuser", this user can join a course. if the user has student user role,after clicking the "Join" button, three things will happen.

Figure 4.31 shows the tree processes when a student joins the course. first the current student user will be added to the participants list of the target lecture. if the last step passed, then a new *Evaluation* object will be generated for this students, the *Evaluation* object will be used to save the student's performance for this lecture. at last, the student will get a gitolite repository generated by the static method *createRemoteRepo()*, so that the student can submit the solutions.4.2.2

on the other hand if a user has teacher or assistant role. the system only need to add the user to the course participants list, since the evaluation and repository are only needed by the student.

```
public static Result addSemesterusertoLecture(String user, String semester,String
    lecturename){
...
if(Lecture.addSemesterusertoLecture(semester, semesteruser, lecture)){
    if(semesteruser.roles.equals(UserRoll.Students.toString())){
        Evaluation eval= new Evaluation();
        eval.lecture=lecture;
        eval.student=semesteruser;
        eval.save(semester);
        semesteruser.update(semester);
    try{
        String repopath= RepoManager
                        .createRemoteRepo(currentuser, lecture, request().
                            getHeader("Host"));
        if(repopath!=null){
            Repo newrepo = new Repo();
            newrepo.course=lecture;
            ...
            newrepo.save(lecture.semester);
            }
        else{
            ...
        }
    }catch(Exception e){
     ...}
    }
```

Figure 4.31: Add student to course

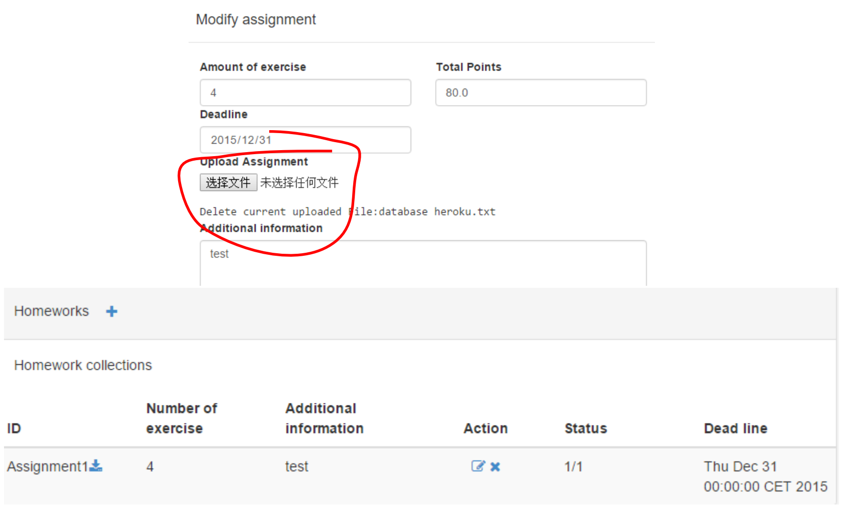
4.5 Assignment management

The management of assignment consists of three parts. first part is creating a assignment, second part is collecting and evaluating the student submission, third part is returning the assignments result back to the student. the second part of assignment management has already been discussed in the section 4.2.2. this section will focused on the creating assignment and distributing the result.

Creating assignment

Figure 4.32 shows the homework area at teacher's course homepage. The form from above is used to modify the details of an assignment, there are several things need teacher to input: first is the details of the exercise, second is the deadline of this assignment, at last teacher can upload the related materials to the assignment. teacher can also gives an additional information to clarify the problem inside the assignment. created assignments will be shown below at the homework section on the homepage, the details of the assignment can still be modified and be deleted after the creation, but after a student uploading a solution, the assignment can not be deleted anymore.

on the student side, the created assignment will be immediately appeared · on the course homepage and ready to be worked on(Figure 4.12).


Modify assignment

Amount of exercise: 4
Total Points: 80.0
Deadline: 2015/12/31
Upload Assignment
选择文件 未选择任何文件
Delete current uploaded file:database heroku.txt
Additional information: test

Homeworks +

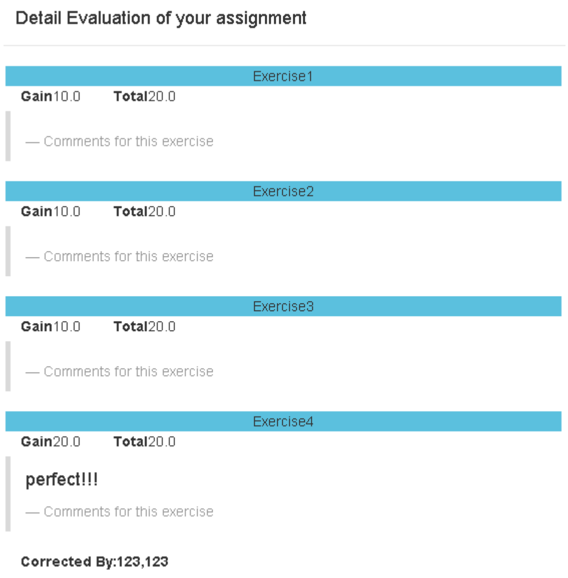
Homework collections

ID	Number of exercise	Additional information	Action	Status	Dead line
Assignment1	4	test		1/1	Thu Dec 31 00:00:00 CET 2015

Figure 4.32: Creation of assignment

Feedback of result

after evaluation the student's solution, the result of the submission should be given back the students. the students can direct click the points under in the column under evaluation(Figure 4.12) for details which was showed in Figure 4.33.


Detail Evaluation of your assignment

Exercise1
Gain10.0 **Total**20.0
— Comments for this exercise

Exercise2
Gain10.0 **Total**20.0
— Comments for this exercise

Exercise3
Gain10.0 **Total**20.0
— Comments for this exercise

Exercise4
Gain20.0 **Total**20.0
perfect!!!
— Comments for this exercise

Corrected By:123,123

Figure 4.33: Assignment results

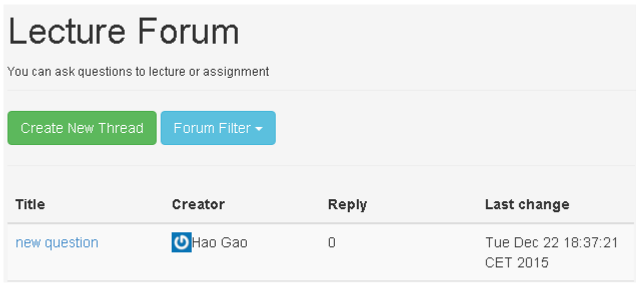
the evaluation of each exercise comes with the points and an comments from the marker. also the name of the marker are showed on left corner, so the student can directly contact the marker, if there is a problem with the evaluation.

4.6 Communication system

There are two way for the user of HMS to communicate with each other: a public course forum and a private instant message system.

4.6.1 Public Forum

Forum is used for students to share their questions about the assignments or the course, using forum to discuss the question can avoid same questions been repeatedly asked by different students. Figure 4.34 is the main page of a course forum. the area above are the forum functions, and the forum thread will be listed under. the latest modified thread will be always listed at the top.



Title	Creator	Reply	Last change
new question	Hao Gao	0	Tue Dec 22 18:37:21 CET 2015

Figure 4.34: Course Forum

4.6.2 Private Instant Message

The HMS system use the Web Socket protocol(Section 3.1.4) to realize the instant message system. Play framework support HMS natively. but Web Sockets can't be directly handled by standard Play framework actions. a possible way using Web socket in Play is using function callbacks.

```java
@Security.Authenticated(Securedstudents.class)
@BodyParser.Of(BodyParser.Json.class)
public static WebSocket<String> socket(){
  User currentuser=User.findByEmail(ctx().session().get("email"),"global");
  if(currentuser!=null) {
      return WebSocket.whenReady((in, out) -> {
          Chatsocket.start(currentuser.email, in, out);
      });
  }
}
```

Figure 4.35: Web socket callbacks

Figure 4.35 is the Web Socket callback function. When the Web Socket is ready, both in and out channels of this connection will be obtained by the system. the channels and the email of the current user will be passed on to the method *Chatsocket.start()*(Figure 4.36).

```
public static HashMap<String,WebSocket.Out<String>> connections;
public static void start(String useremail,WebSocket.In<String> in, WebSocket.Out<
    String> out){
    connections.put(useremail,out);
    in.onMessage(new F.Callback<String>() {
    @Override
    public void invoke(String event) throws Throwable {
    JsonNode inmsg = Json.parse(event);
    if(inmsg.findPath("event").asText().equals("allconversations")){
                ...
                out.write(result.toString());
            }
    }

    if(inmsg.findPath("event").asText().equals("chatcontent")){
                ...
                out.write(result.toString());
        }
    }

    if(inmsg.findPath("event").asText().equals("newmessage")){
                ...
                conversation.update(semester);
                out.write(result.toString());
                notification.put("event","noti");
                ...
                connections.get(other).write(notification.toString());
        }
    }
}
});
}
```

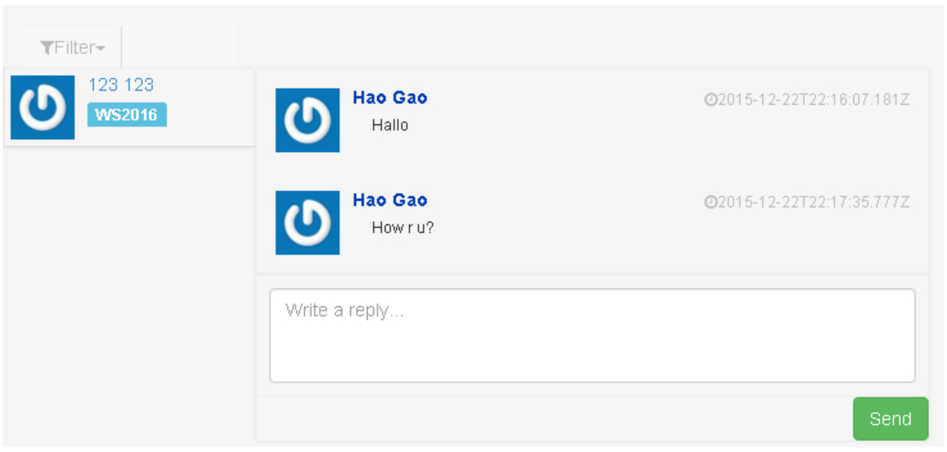
Figure 4.36: Process Web Socket data

The data from the in channel will be processed in the method from Figure 4.36. first the outgoing channel will be saved with its username inside a hashmap, so that an outgoing channel can be easily picked up later. then the inbound message will be processed by the invoke() method. since the inbound messages are saved in a Json string, the first thing to do is to parse the inbound message. the inbound message has two keys: event and data. Event defined the message type, and Data store the actual data from the client. Three different messages are listed below.

1. `{"event":"allconversations","data":{"semester":"WS2016","email":"b@b.com"}}`
2. `{"event":"chatcontent","data":{"convid":"1","semester":"WS2016"}}`
3. `{"event":"newmessage","data":{"semester":"WS2016","convid":"1","content":"How r u?","other":"a@a.com"}}`

according the data from the inbound messages, the server can serve different outgoing message. for instance, the first type message sending a request to the server for all the conversations of user "b@b.com" in semester "WS2016". the second type message requesting the chat content from the conversation with id "1" in "WS2016". the third type of message will be generated when a user send a new message to another user.

for the first two type of message, the system will simply return the result through the out going channel of same user. for the third type of message, the system first will first save the content of the new message into related conversations and finally save the messages into the base, then the same conversation with the new content will be sent back to both participants of the conversation, as long as the other participant has connected to the Web Socket. The out going channel of the other participant will be picked up from the hashmap defined in the first line. additionally a notification message will also be sent to the other user.



Hao Gao 2015-12-22T22:16:07.181Z
Hallo

Hao Gao 2015-12-22T22:17:35.777Z
How r u?

Figure 4.37: GUI of instant message

Figure 4.37 show the user interface of instant message at client side. When user choose the semester number under filter the first type message will be sent to the server. If user click the name of another user in the conversation, a second type message will be sent and the conversation content will be showed. sending the new reply will generate a third type message and the chat content will be immediately updated.

Chapter 5

5 System tests

D[illegible]ntstandende Source code ist zu xxx % getestet. Das Testen spaltet sich in zwei teile auf. Zum einen das Testen des Datenmodells und der Server-Programmlogik und zum anderen der GUI mit den entsprechenden Controllern

Testing a web application is challenging due to the nature of web applications. First, web application have a client/server structure, with asynchronous HTTP calls and response to synchronize the state of each side. Second, web application is a mix of different technology and programming languages, for instance, in the HMS system, JAVA was used on the server side and HTML5, CSS,JavaScript, Scala Template on the client side. Third, the dynamic content on the client side of a web application. web applications have to manipulate the Document Object Model on the client side to serve different content[16]. Therefore, only using the unit test can not test a web application meaningful. In this project Fluentlenium and Junit Test will be used to test the HMS system.

5.1 Fluentlenium

Fluentlenium is a framework that helps developer to write Selenium tests[17]. Selenium is an in-browser programming system, which allows developer to directly drive the real web browser programmatically. It also has the direct access to the DOM elements of web page on the client side, and can assert expected client behavior defined by the developer. since selenium works within the browser, it can be used to test the dynamic behavior of JavaScript and server respond through the interaction between browser and user[18, 19]. Fluentlenium provides a fluent interface to the selenium web driver, so that the programming of selenium tests will be much easier and the code will be more readable. Figure 5.1 demonstrates one of many test scenarios writing with the Fluentlenium: registering a user account in HMS system and getting a confirmation email. First ,the web browser open a page at "*http://localhost:9000*". Second, a HTML form will be filled out and submitted to the server. At last, after receiving the respond from the server, the excepted behavior , in this case , a success label with text "*You will receive a confirmation email soon. Check your email to activate your account.*" should be displayed correctly. otherwise the test will fail.

other scenarios, for instance, "creating a new course by teacher", "uploading a new solution to the assignment by student" are similarly programmed as Figure 5.1.

Ausblick: Später ist doch ein Testen mit einem Continues INtegration System erforderlich, wo die Tests mittels Docker

```
@Test
public void a_testRegistration(){
goTo("http://localhost:9000");
 fill("#SignUpEmail").with("123@123.com");
 fill("#SignUpLastname").with("123");
 fill("#SignUpFirstname").with("123");
 fill("#SignUpPassword").with("123");
 click("#SignUpSubmit");
 await().atMost(5, TimeUnit.SECONDS).until(".label-success").areDisplayed();
 assertThat(find(".label-success").getText()).isEqualTo("You␣will␣receive␣a␣
     confirmation␣email␣soon.␣Check␣your␣email␣to␣activate␣your␣account.");
}
```

Einrückung

Figure 5.1: A Test Scenario of Fluentlenium

5.2 JUnit

Model und Logiktest mittels Java JUnit

Since the server side of HMS system are written in Java, the server logic will be tested by the JUnit. JUnit is a unit testing framework under Java. In order to save the testing time, it is important to test the server logic without starting the whole HMS system.

```
@Test
public void testDeleteSSH() {
User owner= new User();
...
owner.save("global");
SSH ssh= new SSH();
...
ssh.save("global");
FakeRequest request=new FakeRequest("POST","/settings/ssh_delete?sshid=1");
Result result = route(request);
assertThat(status(result)).isEqualTo(OK);
}
```

Einrückung

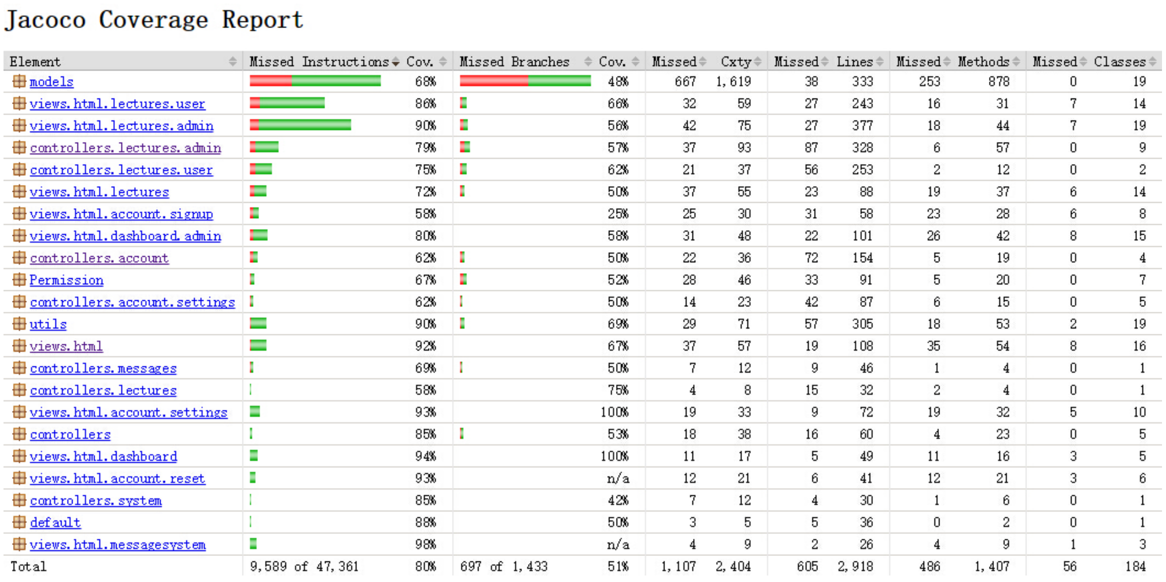
Figure 5.2: Controller Test

Figure 5.2 is a unit test for controller "*deleteSSH*", using the class "*FakeRequest*" from the Play framework, developer can easily test the behavior of the server logic without actually starting the application. In the Figure 5.2 a fake HTTP post request will be sent to the controller with a URL query "sshid=1", since before this request a new ssh is already saved into the database, the request for "*deleting a ssh with id=1*" should be successful, in this case request status should equal to "200" or "OK".

5.3 Result

Figure 5.3shows the test coverage from the test procedure from above, which calculated by the Jacoco for SBT [20].

Merge beide Test-Methoden



Jacoco Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
models		68%		48%	667	1,619	38	333	253	878	0	19
views.html.lectures.user		86%		66%	32	59	27	243	16	31	7	14
views.html.lectures.admin		90%		56%	42	75	27	377	18	44	7	19
controllers.lectures.admin		79%		57%	37	93	87	328	6	57	0	9
controllers.lectures.user		75%		62%	21	37	56	253	2	12	0	2
views.html.lectures		72%		50%	37	55	23	88	19	37	6	14
views.html.account.signup		58%		25%	25	30	31	58	23	28	6	8
views.html.dashboard.admin		80%		58%	31	48	22	101	26	42	8	15
controllers.account		62%		50%	22	36	72	154	5	19	0	4
Permission		67%		52%	28	46	33	91	5	20	0	7
controllers.account.settings		62%		50%	14	23	42	87	6	15	0	5
utils		90%		69%	29	71	57	305	18	53	2	19
views.html		92%		67%	37	57	19	108	35	54	8	16
controllers.messages		69%		50%	7	12	9	46	1	4	0	1
controllers.lectures		58%		75%	4	8	15	32	2	4	0	1
views.html.account.settings		93%		100%	19	33	9	72	19	32	5	10
controllers		85%		53%	18	38	16	60	4	23	0	5
views.html.dashboard		94%		100%	11	17	5	49	11	16	3	5
views.html.account.reset		93%		n/a	12	21	6	41	12	21	3	6
controllers.system		85%		42%	7	12	4	30	1	6	0	1
default		88%		50%	3	5	5	36	0	2	0	1
views.html.messagesystem		98%		n/a	4	9	2	26	4	9	1	3
Total	9,589 of 47,361	80%	697 of 1,433	51%	1,107	2,404	605	2,918	486	1,407	56	184

Figure 5.3: Test Coverage

Chapter 6
System usage

7 Chapter 7
Summary and Future work

Nomenclature

HMS	Homework Management System
ORM	Object-relational mapping

Bibliography

[1] MICROSYSTEM, Sun: *Distributed Application Architecture*. 06 2009

[2] HOHPE, G. ; WOOLF, B.: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Pearson Education, 2012 (Addison-Wesley Signature Series (Fowler)). https://books.google.de/books?id=qqB7nrrna_sC. – ISBN 9780133065107

[3] DAY, Mark ; ROSENBERG, Jonathan ; SUGANO, Hiroyasu: A model for presence and instant messaging. 2000. – Forschungsbericht

[4] MOZILLA: Glossary:WebSockets. (2015)

[5] *Documentation of moodle*. : *Documentation of moodle*

[6] *Moodle documentation:Course Backup*. https://docs.moodle.org/30/en/Course_backup

[7] CHAUDHRI, A.B. ; RASHID, A. ; ZICARI, R.: *XML Data Management: Native XML and XML-enabled Database Systems*. Addison-Wesley, 2003 https://books.google.de/books?id=7LNhd0eQulQC. – ISBN 9780201844528

[8] *H2 Documentation*. http://www.h2database.com/html/features.html#embedded_databases

[9] K, S.K.: *Spring And Hibernate*. McGraw-Hill Education (India) Pvt Limited, 2009 https://books.google.de/books?id=NfNbbhBRc0kC. – ISBN 9780070077652

[10] *ObjectDB JPA Reference*. : *ObjectDB JPA Reference*, http://www.objectdb.com/api/java/jpa/

[11] *Git on the Server*. https://git-scm.com/book/en/v2/Git-on-the-Server-Getting-Git-on-a-Server

[12] *JGit User Guide*. http://wiki.eclipse.org/JGit/User_Guide

[13] *Gitolite all-in-one documentation*. http://gitolite.com/gitolite/gitolite.html

[14] LOELIGER, Jon: Collaborating with GIT. In: *Linux Magazine, June* (2006)

[15] https://playframework.com/documentation/2.3.x/

[16] Garousi, Vahid ; Mesbah, Ali ; Betin-Can, Aysu ; Mirshokraie, Shabnam: A systematic mapping study of web application testing. In: *Information and Software Technology* 55 (2013), Nr. 8, S. 1374–1396

[17] *Fluentlenium Github Documentation*. https://github.com/FluentLenium/FluentLenium

[18] Van Deursen, Arie ; Mesbah, Ali: Research issues in the automated testing of ajax applications. In: *SOFSEM 2010: Theory and Practice of Computer Science*. Springer, 2010, S. 16–28

[19] Brown, C T. ; Gheorghiu, Gheorghe ; Huggins, Jason: *An introduction to testing web applications with twill and selenium*. " O'Reilly Media, Inc.", 2007

[20] *Jacoco for Sbt*. https://github.com/sbt/jacoco4sbt

List of Figures