THE UNIVERSITY OF NEW SOUTH WALES

e-Enterprise project

Project: Code Management

Developer Documentation

Mentor: Angel Lagares

Team members:

Bo Li z3319406

Xiang Xiao z3321525

Ni Xin z3308139

Hailun Zhang z3354270

1. Project Introduction

GitHub is a web-based hosting service for software development projects that uses the Git revision control system [1].

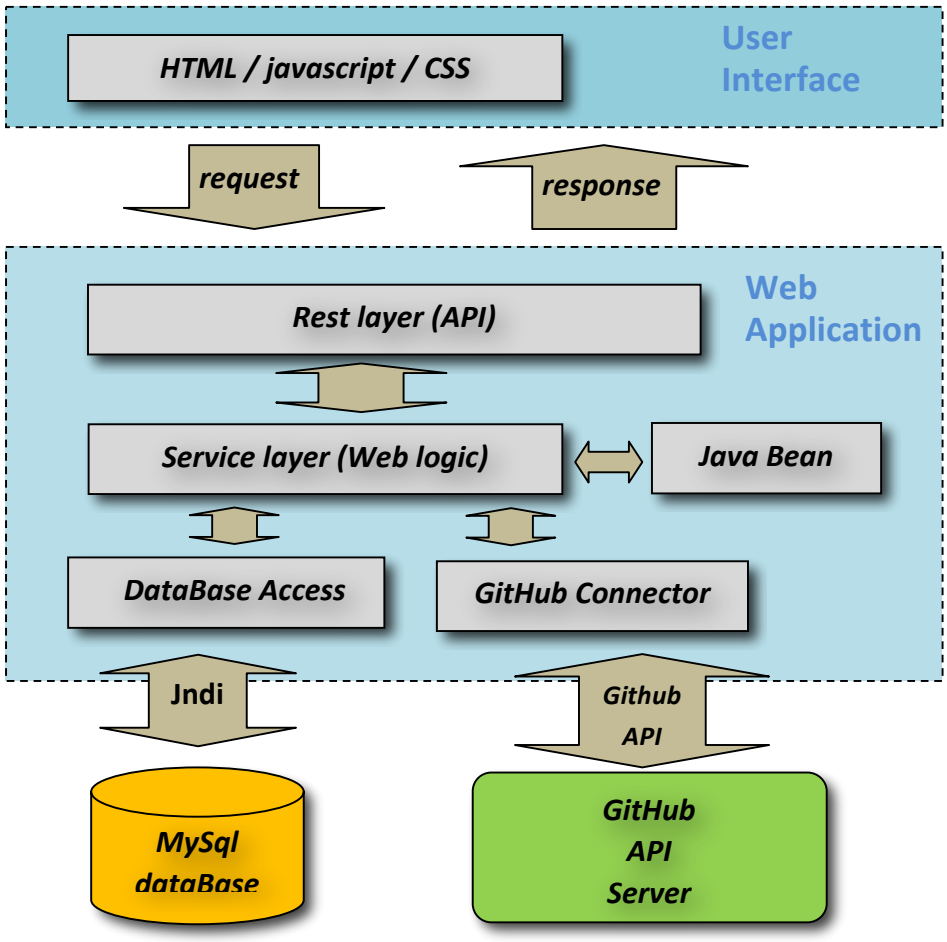
The main goal of our project is to build a code management system for online courses which are based on Github APIs. A restful service is developed to supply APIs which allows the users of our project to achieve online coding and version control.

2. Documentation of the Code

This part of our document can be auto-generated from comments in the code by using javadoc.

3. System Architecture

3.1 Web System Architecture



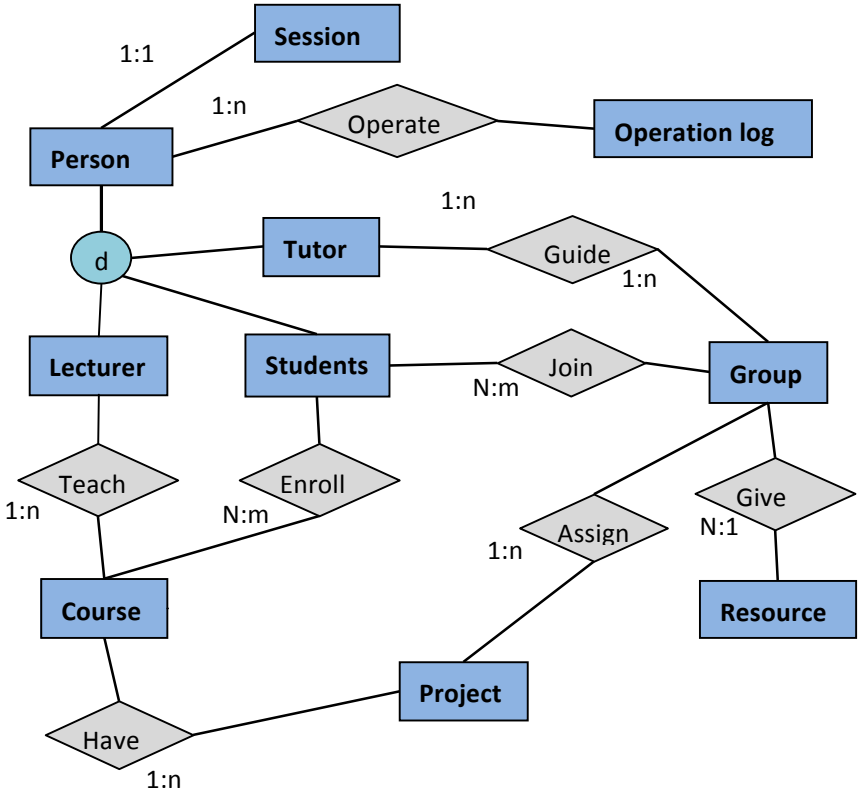
The web Application contacts with UI model through JSON. In addition, it implements the basic functions of online course with the API that offered by Github. JSON is also the default communication form between our web application and Github. We use mysql database to store the user information, resource URI, etc.

We design our web application in four main parts:

- Rest Layer (Restful API) – implement the restful service and offer the restful URL to the UI.
- Service Layer (Web Logic) – integrate the functional modules, such as Authorization function. It controls the workflow and data flow.
- Database Access Layer – connect with the local database via JNDI.
- GitHub Connecter Layer – connect to the GitHub Rest API to get and store data on GitHub server, in that to implement the functions of our project.

4. DATA STRUCTURE DESIGN

4.1 Entity Relational Diagram



4.2 Data base Schema

Person

This table maintained course's information.

Column name	Data type	Description
Person_id	Varchar	Primary Key, identify the person
name	Varchar	Person's name
password	Varchar	Person' s password
role	Varchar	Person's role

Session

Session table record the user's login status, and keep the session.

Column name	Data type	Description
User_id	Varchar	Primary Key
User_token	Varchar	User's access token, generate dynamically
Login_time	Timestamp	Record the user's login time

Course

This table maintains the courses' information

Column name	Data type	Description
Course_id	Varchar	Primary Key, identify the course
Course_name	Varchar	Course's name
Lecturer_id	Varchar	Lecturer's person id(foreign key)

Enrollment_course

This table maintains the enrollment relation between students and courses.

Column name	Data type	Description
Course_id	Varchar	Primary Key, and foreign key from 'course'
Student_id	Varchar	Primary key, and foreign key from 'person'

Group

This table maintains the enrollment relation between students and courses.

Column name	Data type	Description

Group_id	Int	Primary Key, identify the group
Group_name	Varchar	Show the group name
Project_id	Int	Foreign key from 'Project'
Tutor_id	varchar	The tutor's id of this group(Foreign key)

Project

Project table records the relation of project and the course it belongs to.

Column name	Data type	Description
Project_id	Int	Primary Key, identify the project
Project_name	Varchar	Show the project name
Course_id	Int	Foreign key from 'Course'
Project_description	varchar	Description the project

Group_enrollment

This table maintains the enrollment relation between groups of students and their projects.

Column name	Data type	Description
Group_id	Int	Primary key, and foreign key of 'Group'
Student_id	Varchar	Primary key, and foreign key of 'Person'

Operation_log

Operation table records any operations (functions that been called) during application running.

Column name	Data type	Description
Operation_id	Int	Primary Key, identify the operation
Operation_time	Timestamp	Record the operation time
Operater_id	Varchar	Foreign key from 'Person',
Operation_type	Varchar	Indicate the operator's action
Resourse_id	Varchar	Foreign key from 'Resource'

Resource

Resource_id keeps track of the name of the repository which assigned to a Group

Column name	Data type	Description
Group_id	Int	Primary Key.
Resource_id	Varchar	Record the Repo name of the Repository

5. PROJECT FEATURES

5.1 Code

Commit / delete code file

Any operations users have done to their work, they have to commit the changes to make these operations to be executed in Github and generate a new version of the whole project. Users can upload or edit online several pieces of code files, then commit them to the Github directly through our web application, instead of Github platform. After the commission, a new version of "commit" including the latest changes will be generated and stored as current version of "commit". Users could also select one or several files to delete directly through our web application, and a new version of the project which doesn't contain the deleted files will be generated and stored as current version of "commit".

View code

Once users select some specific version of "commit", they can view the code, files and folders contained in that commit version.

Create/delete folders

If users want to organize their code files in different folders, our web application enable them to create new folders under their "project" and commit the files within the folders. Also, users can delete them once they find these folders are no more useful.

Create / delete branches

Our system enables users to continue coding along different branches of code flows. The default branch is called "master". And users could create new branches after any versions of "commit". Users can also select delete one existing branch. Then all the commits and branches followed after this branch would be deleted as well.

View branches

Users can view any branch created under the project. The concept of branching is similar to the structure of tree, which has different versions of "commit" as

child nodes.

View version history

Except the most recent version of the project, people also can have a look at the previous versions.

Roll-back version history

Users can select one history version of commit, and then roll-back to that version. This selected history version would be committed to the Github as a completely new version of “commit” and added after last commit to make sure users that they are free to reuse old versions of the project.

5.2 Comment

Add comment

Users can add their comment to a specific position of code, a code file, a folder or a version of commit.

View comment

Users can view all the comments of a code file, a folder or a version of commit.

5.3 Group

Create group

Users whose roles are lecturer can create groups under their courses by providing group information like group name and group members. The group members can be added by selecting from a list of students who have enrolled into this course and haven’t been assigned to any group yet. Then the lecturer can assign projects to this group. After these, a unique project for this group is created and a corresponded repository is created and initialized in Github.

Modify group

Being a “lecturer”, the user also can modify a existing group by changing the group name, the assigned tutor and group members.

View group

To the users whose roles are lecturer, they can view the list of groups under their courses. While, to the users whose roles are student or tutors, they can

view the list of groups which they are assigned to.

View course list

Users can view the list of courses which they have enrolled in.

View project list

Users can view the list of projects which they have created or been assigned to.

6. SERVICE IMPLEMENTATION

6.1 Interface list

User	**Login**	/user/login
	Registration	/user/register
	Get Unsigned Student List by Project	/user/getUnsignedStudentListByProject/{accountId}/{projectId}
Group	**Get Courses by Account ID**	/group/{accountId}/{token}
	Get Projects by Course ID	/group/{accountId}/{token}/{courseId}
	Get Group List	/group/{accountId}/{token}/{courseId}/{projectId}
	Get Group by Group ID	/group/{accountId}/{token}/{courseId}/{projectId}/{groupId}
	Create Group	/user/group/{accountId}/{token}/{courseId}/{projectId}
	Modify Group	/user/group/{accountId}/{token}/{courseId}/{projectId}
Code	**View Branches**	/repo/{accountId}/{token}/{repoName}
	Create Branch	/repo/{accountId}/{token}/{repoName}
	Delete Branch	/repo/{accountId}/{token}/{repoName}/branch={branchName}
	View Commit Files	/repo/{accountId}/{token}/{repoName}/commitSHA={commitSHA}
	View Root Directory	/rope/{accountId}/{token}/{repoName}/branch={branchName}
	View Directory	/repo/{accountId}/{token}/{repoName}/branch={branchName}/{sha}
	Delete Files	/repo/{accountId}/{token}/{repoName}/branch={branchName}/{path}
	Commit Code	/repo/{accountId}/{token}/{repoName}/branch={branchNam

		e}/code
	View Code	/repo/{accountId}/{token}/{repoName}/branch={branchName}/{sha}/code
	Create Folder	/repo/{accountId}/{token}/{repoName}/branch={branchName}/folder
	Revert Code	/repo/{accountId}/{token}/{repoName}/branch={branchName}/{sha}/revert
	View Versions	/repo/{accountId}/{token}/{repoName}/branch={branchName}/versions
Comment	**View Comment**	/comment/{accountId}/{token}/{repo}/{sha}
	Add Comment	/comment/{accountId}/{token}/{repo}/{branchName}/{sha}

6.2 Interface Description

- **User Interface**

Login:

User login by providing account ID and password, then get token and role as returned value.

URL:
POST /user/login

Parameters:
User - a JSONObject of user loging information constructed according to user's input

Example Input:
```
{
    "username":"boli",
    "password":"123"
}
```

Data Format
JSON

Response:
Status: 201 created
```
{
    "token":"iEhBv0AsjW",
    "role":"lecturer"
```

}

Registration:

User register by providing account ID, username, password and role.

URL:
POST /user/register

Parameters:
User - a JSONObject of user information constructed according to user input

Example Input:
```
{
    "accountId":"boli",
    "password":"123"
    "username":"boli9323",
    "role":"lecturer"
}
```

Data Format
JSON

Response:
Status: 201 Created

Get Unsigned Student List by Project:

Return a list of groups of students who are not assigned to the project yet by providing a project ID.

URL:
GET /user/getUnsignedStudentListByProject/{accountId}/{projectId}

Parameters:
accountId - the id of the current user
token - the token provided by the user
projectId - the id of a project

Data Format
JSON

Response:
Status: 201 Created

```
[{
    "sutdentId":"xx",
    "name":"xiaoxiang"
}]
```

- **Group Interface**

Get Courses by Account ID:

Return a list of courses the user has enrolled by providing user's account ID.

URL:
GET /group/{accountId}/{token}

Parameters:
accountId - the id of the current user
token - the token provided by the user

Data Format
JSON

Response:
Status: 200 OK

```
[{
    "courseId":"9323",
    "courseName":"e-Enterprise project"
}]
```

Get Projects by Course ID:

Return a list of projects under a course by providing course ID.

URL:
GET /group/{accountId}/{token}/{courseId}

Parameters:
accountId - the id of the current user
token - the token provided by the user
courseId - the id of a course

Data Format
JSON

Response:
Status: 200 OK

```
[{
    "projectId":"1",
    "projectName":"code management",
    "projectDescription":"online github"
}]
```


Get Group List:

Return a list of groups which the user has enrolled by providing user's account ID.

URL:
GET /group/{accountId}/{token}/{courseId}/{projectId}

Parameters:
accountId - the id of the current user
token - the token provided by the user
courseId - the id of a course
projectId - the id of a project

Data Format
JSON

Response:
Status: 200 OK

```
[{
    "groupId":"1",
    "groupName":"hzxxgrouop",
    "tutorId":"nixin",
    "tutorName":"nixin",
    "members":
    [{"memberName":"helen","memberId":"hz"},
    {"memberName":"xiaoxiang","memberId":"xx"}]
}]
```


Get Group by Group ID:

Return group information by providing a group ID.

URL:
GET /group/{accountId}/{token}/{courseId}/{projectId}/{groupId}

Parameters:

account_id - the id of the current user
token - the token provided by the user
course_Id - the id of a course
project_Id - the id of a project
group_Id - the id of a group

Data Format
JSON

Response:
Status: 200 OK

```
{
    "groupId":"1",
    "groupName":"hzxxgrouop",
    "tutorId":"nixin",
    "tutorName":"nixin",
    "members":
        [{"memberName":"helen","memberId":"hz"},
        {"memberName":"xiaoxiang","memberId":"xx"}]
}
```

Create Group:

Create a group by providing group information of course ID, project ID, group name, tutor ID and members ID.

URL:
POST /user/group/{accountId}/{token}/{courseId}/{projectId}

Parameters:

account_id - the id of the current user
token - the token provided by the user
course_Id - the id of a course
project_Id - the id of a project
groupInfo - a JSONObject of group information constructed according to user input

Example Input:

```
{
    "groupName":"hzxxgrouop",
    "tutorId":"nixin",
```

```
    "members":
        [{"memberId":"hz"},
        {"memberId":"xx"}]
}
```

Data Format
JSON

Response:
Status: 201 created

Modify Group:
Modify a group by providing group information of group name, tutor ID and members ID.

URL:
PUT /user/group/{accountId}/{token}/{courseId}/{projectId}

Parameters:
account_id - the id of the current user
token - the token provided by the user
course_Id - the id of a course
project_Id - the id of a project
groupInfo - a JSONObject of group information constructed according to user input

Example Input:

```
{
    "groupName":"hzxxgrouop",
    "tutorId":"nixin",
    "members":
        [{"memberId":"hz"},
        {"memberId":"xx"}]
}
```

Data Format
JSON

Response:
Status: 200 OK

- **Code Interface**

View Branches:

Return a list of branches in the searched repository by providing the repository name

URL:
GET /repo/{accountId}/{token}/{repoName}

Parameters:
accountId - the id of the user
repoName - the name of the target repo
token - the token provided by the user

Data Format:
JSON

Response:
Status: 200 OK
[{"branchId":"master"}]

Create Branch:

Create a branch that is not exist in repository

URL:
POST /repo/{accountId}/{token}/{repoName}

Parameters:
accountId - the id of the user
repoName - the name of the target repo
token - the token provided by the user

Data Format:
JSON

Example Input:
{"branchName":"testBranch"}

Response:
Status: 201 Created

Delete Branch:

Delete a branch from repository by providing the repository name and branch

name

URL:
DELETE /repo/{accountId}/{token}/{repoName}/branch={branchName}

Parameters:
branchName - the branch name
accountId - the id of the user
repoName - the name of the target repo
token - the token provided by the user

Data Format:
JSON

Response:
Status: 204 No Content

View Commit Files:

Return the file list of specified commit by providing the commit SHA

URL:
GET /repo/{accountId}/{token}/{repoName}/commitSHA={commitSHA}

Parameters:
commitSHA - the id of the commission
accountId - the id of the user
repoName - the name of the target repo
token - the token provided by the user

Data Format:
JSON

Response:
Status: 200 OK

```
[{
    "filePath":"README",
    "content":"@@ -0,0 +1,2 @@\n+Hello,\n+This is test for git.",
    "sha":"7412341bac44dd88c33c79af5e6067bb7d75c608"
}]
```

View Root Directory:

Return the files and folders in root directory of repository by providing repository name and branch name

URL:
GET /rope/{accountId}/{token}/{repoName}/branch={branchName}

Parameters:
branchName - the name of the target branch
accountId - the id of the user
repoName - the name of the target repo
token - the token provided by the user

Data Format:
JSON

Response:
Status: 200 OK

```
[
    {
        "name":"README",
        "sha":"e69de29bb2d1d6434b8b29ae775ad8c2e48c5391",
        "type":"blob"
    },
    {
        "name":"folder1",
        "sha":"a413a41a95bb7c16e756085d12035b91dafcdee2",
        "type":"tree"
    },
    {
        "name":"home_page_frame.css",
        "sha":"695aabf6b4ffdc7cc9e0fac453a14f885e254eca",
        "type":"blob"
    }
]
```

View Directory:

Return files and folders in any subdirectory by providing the SHA of subdirectory

URL:
GET /repo/{accountId}/{token}/{repoName}/branch={branchName}/{sha}

Parameters:
sha - the id string of current directory

accountId - the id of the user
repoName - the name of the target repo
token - the token provided by the user

Data Format:
JSON

Response:
Status: 200 OK

```
[
    {
        "name":"CommentService.java",
        "sha":"e62ea053902ec9ac0df0f7c3e65fee37e6a01d0e",
        "type":"blob"
    },
    {
        "name":"GroupService.java",
        "sha":"cb0a1236c2fb8360ffe548378f36b90049875377",
        "type":"blob"
    },
    {
        "name":"folder2",
        "sha":"9865b1ee72ea1212d8526b5c3fefee52549b648f",
        "type":"tree"
    }
]
```

Delete Files:

Delete files in repository by providing the path of files

URL:
DELETE
/repo/{accountId}/{token}/{repoName}/branch={branchName}/{path}

Parameters:
path - the path of the file or the folder
branchName - the branch name
accountId - the id of the user
repoName - the name of the target repo
token - the token provided by the user

Data Format:
JSON

Response:
Status: 204 No Content

Commit Code:

Commit code files to the repository by providing file path and file content

URL:
POST /repo/{accountId}/{token}/{repoName}/branch={branchName}/code

Parameters:
- fileList - a list (JSON array) of file objects the user wants to commit
- branchName - the name of the target branch
- accountId - the id of the user
- repoName - the name of the target repo
- token - the token provided by the user

Data Format:
JSON

Example Input:
```
[
    {"filePath":"test1.txt", "content": "test1"},
    {"filePath":"test2.txt", "content": "test2"}
]
```

Response:
Status: 201 Created

View Code:

Return content of searched files

URL:
GET
/repo/{accountId}/{token}/{repoName}/branch={branchName}/{sha}/code

Parameters:
- sha - the id string of a file
- accountId - the id of the user
- repoName - the name of the target repo
- token - the token provided by the user

Data Format:
JSON

Response:
Status: 200 OK
{"This is the content of file."}

Create Folder:

Create a folder in repository by providing the path of folder

URL:
POST /repo/{accountId}/{token}/{repoName}/branch={branchName}/folder

Parameters:
folder - a new folder object (a JSON object)
branchName - the name of the target branch
accountId - the id of the user
repoName - the name of the target repo
token - the token provided by the user

Data Format:
JSON

Example Input:
{"path":"testfolder/testfolder1"}

Response:
Status: 201 Created

Revert Code:

Revert to the previous version of repository by providing the previous commit SHA

URL:
POST
/repo/{accountId}/{token}/{repoName}/branch={branchName}/{sha}/revert

Parameters:
branchName - the name of the target branch
sha - the id string of the commission
accountId - the id of the user
repoName - the name of the target repo

token - the token provided by the user

Data Format:
JSON

Response:
Status: 201 Created

View Versions:

Return all the commits of repository by provide the repository and branch name

URL:
GET /repo/{accountId}/{token}/{repoName}/branch={branchName}/versions

Parameters:
joo - a JSON object which could refer to the path of the repository
accountId - the id of the user
repoName - the name of the target repository
token - the token provided by the user

Data Formate:
JSON

Response:
Status: 200 OK

```
[
    {
        "commiterId":"xx",
        "sha":"771f955d7cbad1e79e0cb0187da4d193bd7b06c5",
        "commitTime":"Sat Oct 06 12:35:07 EST 2012"
    },
    {
        "commiterId":"initial commit",
        "sha":"1f29e8b4c2025b95819873e8a1b803dc936772e8",
        "commitTime":"Fri Aug 24 16:36:04 EST 2012"
    }
]
```

- **Comment Interface**

View Comment:

Return the specified comment on one commit by providing the SHA of commit

URL:
GET /comment/{accountId}/{token}/{repo}/{sha}

Parameters:
account_id - the id of the current user
token - the token provided by the user
repo - the name of the target repository
sha - an id string of the comments

Data Format:
JSON

Response:
Status: 200 OK

```
[
    {
        "commentPath":"hello.txt",
        "commiterId":"xx",
        "commentContent":"Good project.",
        "commentPosition":0,
        "commentDate":"Fri Oct 12 13:02:21 EST 2012"
    }
]
```

Add Comment:

Add the comment to one commit by providing the SHA of commit

URL:
POST /comment/{accountId}/{token}/{repo}/{branchName}/{sha}

Parameters:
account_id - the id of the current user
token - the token provided by the user
repoName - the name of the target repository
branchName - the branch name
sha - an id string of the comments
commentInfo - a comment object encapsulated in a JSON object

Data Format:
JSON

Example Input:

```
{
    "commentContent": "This is comment.",
    "commenterId": "xx",
    "commentPosition": "2",
    "commentPath": "README.md"
}
```

Response:
Status: 201 Created

7. CONCLUSION

To sum up, by using our code management system, users are able to achieve co-programming, online editing and version control.
Compare to Github, there are some advantages for users to choose our project instead of using Github directly:

- Our project provides group management for users so that students are able to co-programming, while lecturers and tutors can follow students' works synchronously.

- Our system allows users to commit or delete code file directly through our application instead of the Github platform. Since Github needs installation of specific desktop application, our application seems more convenience.

- Our system is more brief and clear for user to manage their code, and more helpful with academic background.

Reference

[1]. http://en.wikipedia.org/wiki/GitHub