Simon Fraser University
School of Computing Science
Cmpt 275 Project

Date: August 5, 2015

Team name: Fast and Furious 9 (Group 9)

Project Deliverable #: 5

Project Deliverable Name: Design

Phase leader(s): Maddy Jones

Grade:

Revision History

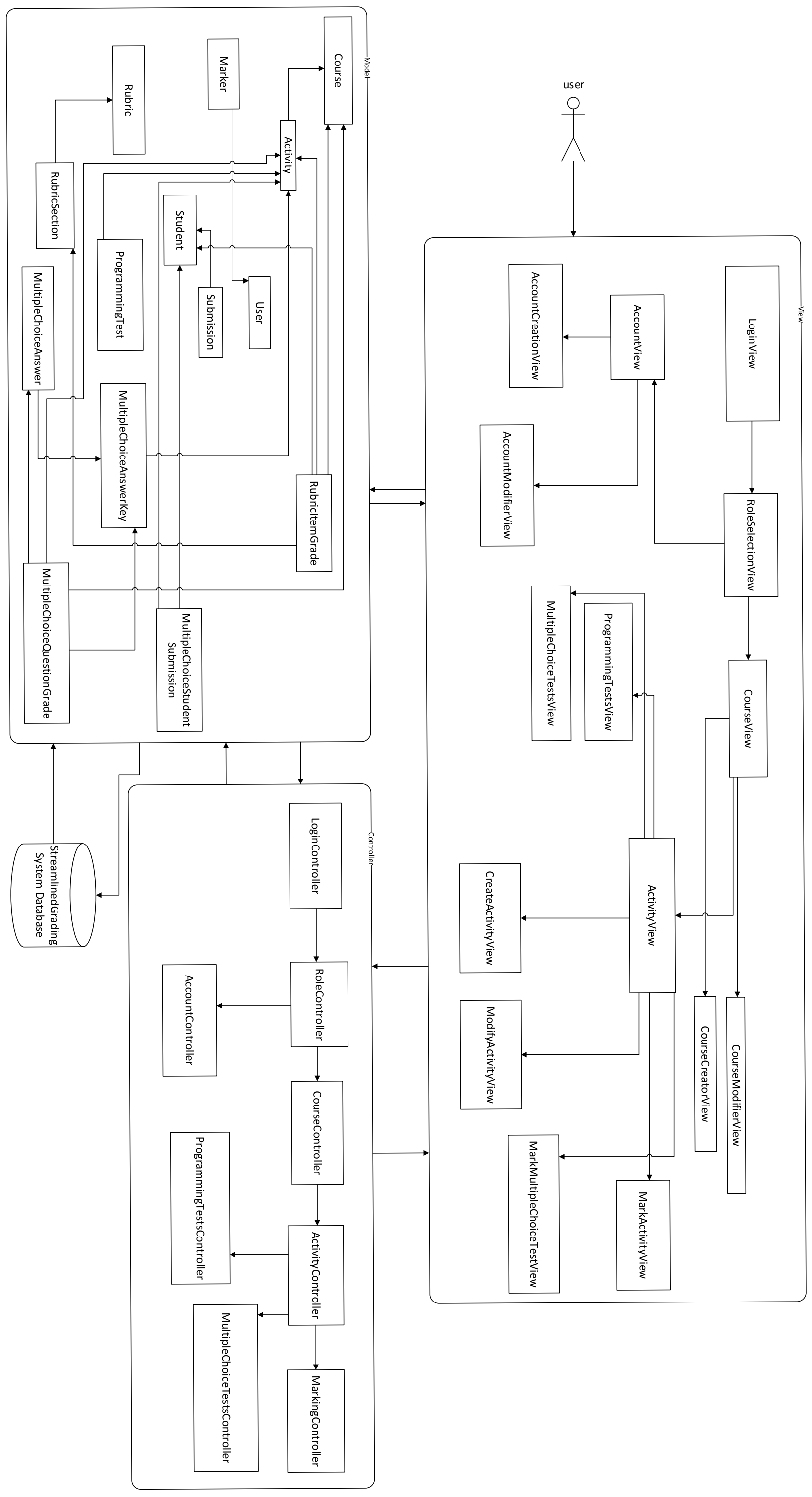
Revision	Status	Publication/Revision Date	By:
1.0	Created	Thursday, June 25 2015	Maddy Jones
1.1	Updated class diagram	Friday, July 3 2015	Connor MacLeod
1.2	Updated use case 1	Saturday, July 4 2015	Janice Mardjuki
2.0	Added collaboration diagram Added sequence diagram Added Architecture Diagram	Saturday, July 4 2015	Samnang Sok German Villarreal David Chow Ching Lam Maddy Jones
2.1	Updated class diagram (attributes & methods)	Saturday, July 4 2015	Joshua Campbell David Chow German Villarreal Connor MacLeod
2.2	Updated and unified class attribute and methods section	Sunday, July 5 2015	Joshua Campbell Connor MacLeod
3.0	Added use case 2 Added table design Added sub-system description Updated use case 1 and 2	Sunday, July 5 2015	Rob Cornall Joshua Campbell Maddy Jones David Chow Ching Lam German Villarreal
3.1	Reviewed and updated all sections	Sunday, July 5 2015	Maddy Jones Joshua Campbell Rob Cornall Samnang Sok David Chow German Villarreal
3.2	Updated class diagram, attributes and methods updated	Wednesday, August 5 2015	David Chow Rob Cornall

Table of Contents

Revision History 2
Table of Contents 3
High Level Design 4
    Architecture Diagram 4
    Sub-System Description 5
    Refined Use Cases 9
Low Level Design 11
    Interaction Diagram 1: Sequence Diagram 11
    Interaction Diagram 2: Collaboration Diagram 12
    Class Diagram 13
Data Persistence 21
    Table Design 21

High Level Design

Architecture Diagram



Sub-System Description

View

LoginView:

- users may log into the system from

RoleSelectoinView:

- users may select one of any roles that they were assigned

AccountView:

- provides account manipulation options to users depending on their role
- system administrators have options to add/modify/delete accounts, while other users may only change their own password

AccountCreationView:

- system administrators can create accounts from this view

AccountModifierView:

- system administrators can edit users' account information, or block/unblock an account from this view

CourseView

- Users can view course information available for them to see from here
- Users' chosen role will affect the possible functions they can perform from this view

CourseModifierView

- Administrative assistant can modify courses from here

CourseCreatorView

- Administrative assistant can create courses here

ActivityView

- Administrator, instructor or TA assigned to the course can choose to perform various actions related to activities from here

CreateActivityView

- Instructors can create activities from here

ModifyActivityView

- Instructors can modify activities from here

MarkActivityView

- Instructors or TAs can mark activities from this view

- can show submission/rubric/solution/test results/etc.

MarkMultipleChoiceTestView
- Instructors or TAs can Mark MC activities here

MultipleChoiceTestsView
- Instructors can view MC tests, and MC test answers here
- viewing MC question statistics is also possible in this view

ProgrammingTestsView
- Instructors can view programming tests here

Controller

LoginController:
- takes username and password input from the LoginView, and checks that these match information from the User model
- creates a session for the user to use the system

RoleController:
- obtains all roles associated with a user (from the User model) and populates the RoleSelectionView with these roles

AccountController:
- takes and can edit data in the User model to get/modify account information
- passes information to the AccountView

CourseController:
- checks user privileges to see which features can be seen in the CourseView
- uses the Course model to obtain/edit course information

ActivityController:
- edits the rubric, rubric section, or activity models if changes to an activity or an activity's rubric are made
- gets the data to fill lists of activities, or information of a specific activity in the ActivityView

ProgrammingTestsController:
- interacts with the ProgrammingTest and Activity models to get information about programming tests, or change various information about programming tests
- provides information to the ProgrammingTestsView

MultipleChoiceTestsController:

- passes information to the MultipleChoiceTestsView
- when changing/obtaining MC data, it is obtained from/changed in the following models: Activity, MultipleChoiceAnswerKey, and MultipleChoiceStudentSubmission

MarkingController:

- passes display information to the MarkActivityView or MarkMultipleChoiceView
- the following models may have their data accessed/changed depending on the type of activity: Marker, Rubric, RubricSection, RubricItemGrade, Submission, Activity, MultipleChoiceQuestionGrade, MultipleChoiceAnswer, MultipleChoiceAnswerKey, MultipleChoiceStudentSubmission

Model

Course:

- stores information about courses
- interfaces with the database for course information

Activity:

- stores information about activities
- interfaces with the database for activity information

User:

- stores information about user accounts
- interfaces with the database for account information

Marker:

- stores information about markers': privilege level and assigned courses
- interfaces with the database for marker information

Student:

- stores information about students and which courses they are taking
- interfaces with the database for student information

Submission:

- stores information about student/group submissions for a particular activity
- interfaces with the database for submission information

Rubric:

- stores information about rubrics for activities
- interfaces with the database for rubric information

RubricSection:
- stores information about weight/expectation pairs from a rubric
- interfaces with the database for information on the sections of a rubric

RubricItemGrade:
- stores the marks students received for specific sections on a rubric of an activity
- interfaces with the database for information on marks given for a section of a rubric

ProgrammingTest:
- stores information about programming tests for certain activities
- interfaces with the database for information programming test information

MultipleChioceAnswerKey:
- stores information about unique multiple choice tests
- interfaces with the database for MC answer key information

MultipleChioceAnswer:
- stores information about the answers for multiple choice tests
- interfaces with the database for MC answer information

MultipleChoiceStudentSubmission:
- stores information about students' answers to multiple choice tests
- interfaces with the database for student MC answers

MultipleChoiceQuestionGrade:
- stores information about the mark a student received for a multiple choice question
- interfaces with the database for the mark a student received for a MC question

Refined Use Cases

Adding a programming activity
Actors: Instructor
Functional Requirements: 18
Preconditions:

- Instructor has logged in to the system
- Instructor has been assigned to the related course
- Instructor has chosen a class
- StreamlinedGradingSystem database is online

Flow of events:

1. Instructors clicks on the “Create Activity” button to add new activity
2. Instructor chooses programming activity
3. The instructor will be required to fill in the relevant field of the programming activity
    a. Instructor inputs the name of the programming activity
    b. Instructor specifies path of the solution
    c. Instructor enters the due date of the programming activity
    d. Instructor chooses the path of the student works
    e. Instructor specifies the programming language
    f. Instructor needs to specify the compiler
    g. Instructor may choose to add a rubric
        i. System will follow the Create Rubric use case
4. Instructor will need to create the programming test
    a. Instructor chooses the path to the test input and output files
    b. Instructor specifies the path to console input/output files
5. Instructor clicks the “Save” to save this programming activity
6. A new entry in the Activity table is created
7. New entries in the ProgrammingTest table are created for each programming test created
8. If a rubric was added, an entry in the Rubric table is created, and multiple entries in the RubricSection table are added for each section on the rubric
9. A window displaying "Activity Created" will appear on the screen
10. Instructor presses the "OK" button to return to the course view

Post conditions:

- Return back to the “View Course”
- A new programming activity has been added on the right table

Exceptional flow of events:

- Path(s) could not be found
- Invalid name/due date/language entered

Run one (previously specified) test on a students submitted code and display the results of that test and the instructor's solution ready to be compared

Initiating Actors: Marker
Functional Requirements: 32
Preconditions:

- Marker has logged into the system
- Marker has been assigned to the related course
- Marker has chosen a class
- Marker has chosen an activity (programming activity) and which test to run on it
- Streamlined Grading System (SGS) database online
- System has selected a student submission
- System has selected a test to run on the student submission

Flow of events:

1. Marker presses "Run Tests" button
2. System accesses ProgrammingTest table to get the tests to run, running all the tests
3. System accesses Rubric table and RubricSection table to get weights and descriptions for the rubric of this activity
4. A view showing the student's output, the expected output, and the related rubric section for that test appear on screen
    a. Marker can enter values into the rubric section to give the student a mark
5. Marker can select another test from a drop down menu to see the student's output, expected output, and rubric section for that test
6. Marker can click the "solution” button to see the instructor's code
    a. System will access Activity table to get the path to the solution, then follow the path to get the solution
7. Marker can click the “rubric” button to view the entire rubric
8. Marker can click the “student submission” to view the student's code
    a. Marker can add comments to the code on different lines
9. Marker indicates they are finished marking by pressing the "done" button
10. System updates the weight attribute for entries in the RubricSection table
11. System accesses the Activity table to get the path to student work, and saves comments in that directory onto a new file

Post conditions:

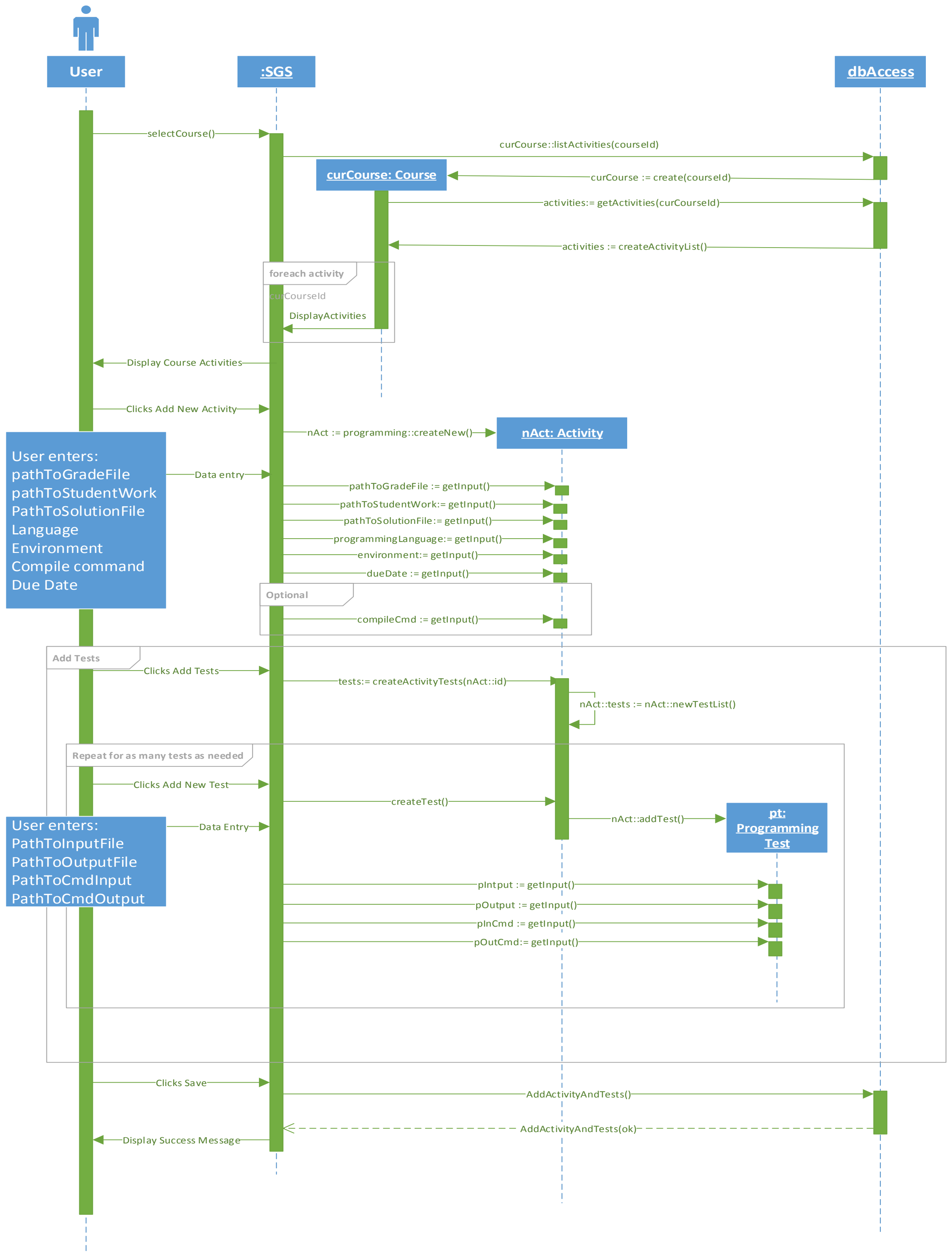
- Marker is brought to the activity view and may select another student submission to test/mark
- Database entries have been updated

Exceptional flow of events:

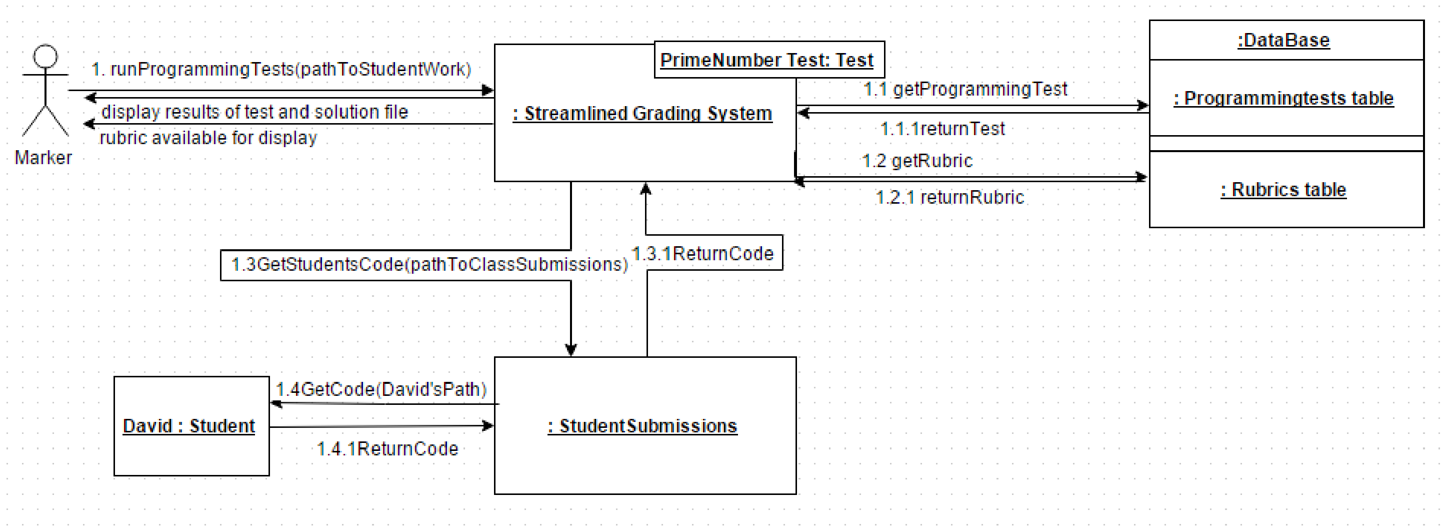
- The instructor left some required fields empty before testing
- Path(s) could not be found
- Students code didn’t run properly (error message)

Low Level Design

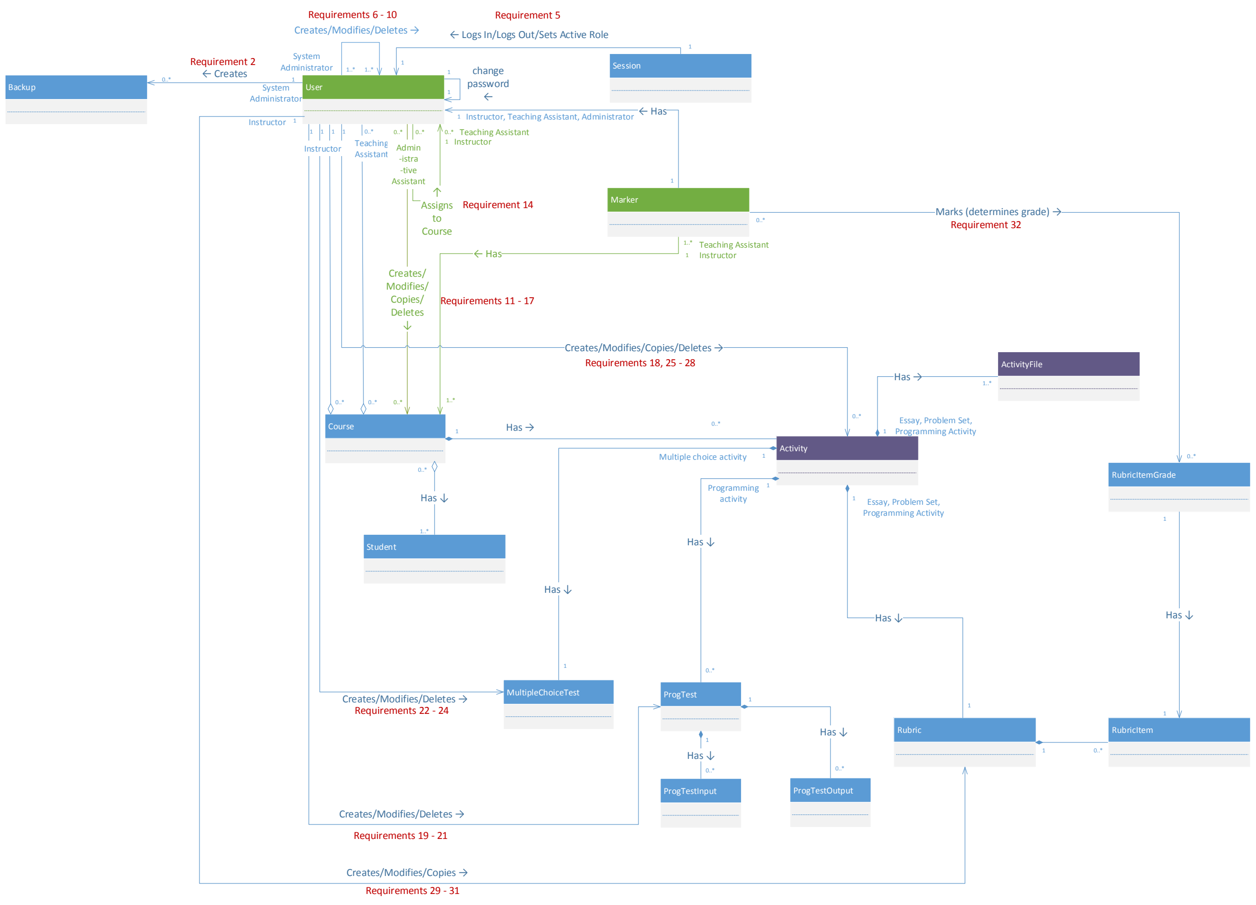
Interaction Diagram 1: Sequence Diagram



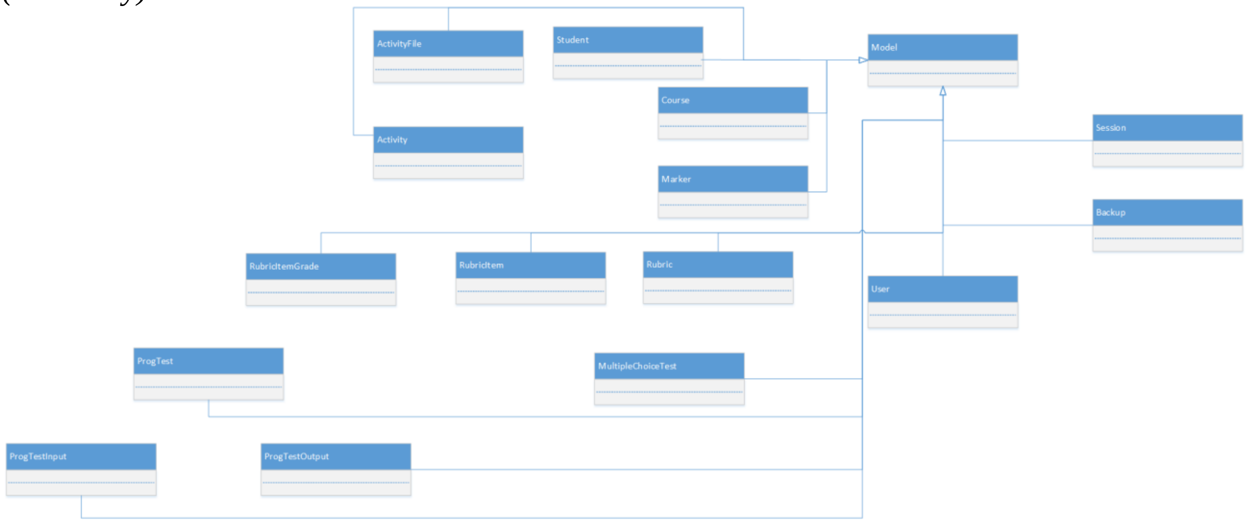
Interaction Diagram 2: Collaboration Diagram



Class Diagram



(hierarchy)



Attributes and Methods:

Activity:

+getRubric(): QList<QMap<QString, QString>>
+getRubric(activityID: QString): QList<QMap<QString, QString>>
+GetActivityByID(_activityID: QString):   QMap<QString, QString>
+InsertActivity(bool returnInsertedItem = true): QMap<QString, QString>
+InsertActivity(): boolean
+getID(): QString
+SetParameter(name: QString, value: QString): void
+SetParameters(_params: QMap<QString, QString>): void
+updateActivity(activityID: QString, valuesToChange: QMap<QString, QString>): bool
+getByCourseID(QStringListcourseIDs):QList<QMap<QString, QString>>
+getByCourseID(courseID: QString): QList<QMap<QString, QString>>
+getByCourseIDIfCanGrade(courseID: QString, role: int): QList<QMap<QString, QString>>
+getRubriclessByCourseID(currentCourseID: QString): QList<QMap<QString, QString>>
+delActivity(_activityID: QString): bool
-params: QMap<QString, QString>

ActivityFile:

+InsertActivityFile(): bool
+getID(): QString
+SetParameter(name QString,value: QString): void
+getByActivityID(activityID: QString): QList<QMap<QString, QString>>
+updateActivityFile(fileNumber: QString,valuesToChange: QMap<QString, QString>): bool
+deleteActivityFiles(activityID: QString): bool
-params: QMap<QString, QString>

Course:

+getID(): QString
+InsertCourse(): bool
+getAllCourses(): QList<QMap<QString, QString>>
+getCoursesByUser(userID: QString, role: int): QList<QMap<QString, QString>>
+createCourse(): QMap<QString, QString>
+insertCourse(*course:QMap<QString, QString>, chckInstructor = false: bool): bool
+updateCourse(courseID: QString, valuesToChange: QMap<QString, QString>): bool
+updateInstructor(currID: QString,newID: QString): bool
+deleteCourse(courseID: QString): bool
+ verifyInstructorExists(course:QMap<QString, QString>):bool
+verifyInstructorExists(instructorID: QString): bool
+removeInstructor(instructorID:QString, courseID: QString):bool
+removeInstructorFromAllCourses(instructorID: QString): bool
+assignInstructorToCourses(instructorID: QString, courseIDs: QStringList): bool
+ assignInstructorToCourses(instructorID: QString, courses:QList<QMap<QString, QString>>
): bool
+ getCourseID(* constcourse:QMap<QString, QString>const): QString

+ getCourseNumber (* constcourse:QMap<QString, QString>const): QString
+ getCourseName(* constcourse:QMap<QString, QString>const): QString
+ getStartDate(* constcourse:QMap<QString, QString>const): QDate
+ getEndDate(* constcourse:QMap<QString, QString>const): QDate
+ getInstructorID (* constcourse:QMap<QString, QString>const):QString
+ getDateFormat(): QString
+ setCourseID(newValue:QString, *course: QMap<QString, QString>, verify = false:bool): bool
+ setCourseNumber (newValue: QString, *course: QMap<QString, QString> , verify = false: bool): bool
+ setCourseName(newValue: QString, *course: QMap<QString, QString>, verify = false: bool): bool
+ setStartDate(newValue: QString, *course: QMap<QString, QString>,verify = false: bool): bool
+ setEndDate (newValue:QString, *course:QMap<QString, QString>, verify = false: bool): bool
+ setInstructorID (newValue: QString, *course: QMap<QString, QString> , verify = false: bool): bool
+ verifyCourseID(newValue:QString)bool
+ verifyCourseNumber(newValue:QString): bool
+ verifyCourseName (newValue:QString):bool
+ verifyDates (* const course: QMap<QString, QString>const):bool
+ verifyInstructorID (newValue: QString): bool
-setParameter(parameter:QString, newVal:QString , *course: QMap<QString, QString>): bool
-getParameter(parameter:QString,* const course: QMap<QString, QString>const): QString
-params: QMap<QString, QString>

Marker:
-params: QList<QMap<QString, QString>>
+insertMarker(*marker:QMap<QString, QString>): bool
+removeMarker(userID:QString): bool
+removeCourse(courseID:QString): bool
+InsertMarker(): bool
+SetParameter(name:QString, value:QString): void
+getMarkerIDsByCourse(courseID:QString): QStringList
+getMarkersByCourse(courseID:QString): QList<QMap<QString, QString>>
+getCoursesByMarker(userID:QString): QList<QMap<QString, QString>>
+createMarker(): QMap<QString, QString>
+setCourseTA(userID:QString, courseID:QString): bool
+setCourseTAs(userIDs:QStringList, courseID:QString): bool
+assignTACourses(userID:QString, courses:QList<QMap<QString, QString>>): bool
+assignTACourses(userID:QString, courseIDs:QStringList): bool
+updateTA(currID:QString, newID:QString): bool
+setPrivilege(newValue:QString, *marker:QMap<QString, QString>, verify = false:bool): bool
+verifyPrivilege(newValue: QString): bool
+getPrivilege(* const marker:QMap<QString, QString>const) int

Model:
#pKeyIndex: int

#pKey: QString
#tableName: QString
#parameters: QStringList
#getDBItems(keyValues: QMap<QString, QString>, andor: QString): QList<QMap<QString, QString>>
#getDBItems(parameter: QString, values: QStringList): QList<QMap<QString, QString>>
#getDBItems(parameter: QString, value: QString):QList<QMap<QString, QString>
#getDBItem(getKey: QString) : QMap<QString, QString>
#insertDBItems(items: QList<QMap<QString, QString>>): bool
#updateItemKey(oldVal: QString, newVal: QString) : bool
#updateDBItem(pKeys: QMap<QString, QString>, item: QMap<QString, QString>): bool
#updateDBItems(currValues: QList<QMap<QString, QString>>, newValues: QMap<QString, QString>): bool
#updateDBItems(parameter: QString, currValue: QString, newValue: QString): bool
#deleteDBItem(delKey: QString) :bool
# deleteDBItem(item: QMap<QString, QString>): bool
#deleteDBItems(rows: QList<QMap<QString, QString>>): bool
#deleteDBItems(parameter: QString, value: QString): bool
#deleteDBItems(parameter: QString, values: QStringList): bool
+getAll(): QList<QMap<QString, QString>>
+getEmptyRecord(): QMap<QString, QString>
+insertDBItem(item: QMap<QString, QString>) :bool
+insertDBItem(item: QMap<QString, QString>, returnInsertedObject: bool) :QMap<QString, QString>
+parseResult(*query: QSqlQuery):QList<QMap<QString, QString>>
+getTableName() const { return tableName: QString}
+getKey() const { return this->pKey: Qstring }
+getParameters() const { return parameters: QStringList }
+getNumberOfParameters() const { return parameters.count(): int }
+pKeyExists(pKey: QString): bool
+ getAll():QList<QMap<QString, QString>>
+getEmptyRecord(): QMap<QString, QString>
+insertDBItem(item: QMap<QString, QString>) : bool
+insertDBItem(item: QMap<QString, QString>, returnInsertedObject: bool) :QMap<QString, QString>
+parseResult(*query: QSqlQuery): QList<QMap<QString, QString>>

MultipleChoiceTest
+ insertMultipleChoiceTest(test: QMap<QString, QString>): QMap<QString, QString>
+ insertMultipleChoiceTestAnswer(testAnswer: QMap<QString, QString>): QMap<QString, QString>
+QList<QString> generateGradeFile(_activityID: QString);
+ getTestAnswers(_activityID: QString): QList<QMap<QString, QString>>
+deleteMCAnswers(_activityID: QString): void
+studentID: QString

+answers: QList<QString>
+ params: QMap<QString, QString>

ProgTest:
+insertProgTest(test: ProgrammingTest *, activityID: QString): QString
+getProgTests(activityID: QString): QList<ProgrammingTest>
+deleteProgTests(activityID: QString):bool
-params:QMap<QString, QString>

ProgTestInput:
+insertProgTest(* test: ProgrammingTest, activityID: QString): QString
+getProgTests(activityID: QString): QList<ProgrammingTest>
+deleteProgTests(activityID: QString): bool
-Params: QMap<QString, QString>

ProgTestoutput:
+insertOutputFiles(outputs: QList<QMap<QString, QString>>): void
+getOutputFiles(testID: QString): QList<QMap<QString, QString>>
+deleteProgTestOutputs(testID: QString): bool
-params: QMap<QString, QString>

Rubric:
+getRubric(_rubricID: QString):QList<QMap<QString, QString>>
+InsertRubric(_activityID: QString): void
+createRubricSection(_itemNumber: QString, _expectedOutcome: QString, _weight: QString): void
+InsertRubricSection(_rubricID: QString, _section: QMap<QString, QString>) : void
+getRubricID(_activityID: QString): QString
delRubric(QString _activityID): bool
+InsertRubric(returnInsertedItem = true: bool): QMap<QString, QString>
+getID():QString
+SetParameter(name: QString, value: QString): void
-rubric_sections: QList<QMap<QString,QString>>
-activityID: QString
-params: QMap<QString, QString>

RubricItem:
+InsertRubricItem():bool
+getID():QString
+SetParameter(name: QString, value: QString): void
+generateGradeSummaryCSV(rubricID: QString): bool
+delRubricItems(_rubricID: QString): bool
-params: QMap<QString, QString>

RubricItemGrade:

+InsertRubricItemGrade(_grade: QMap<QString, QString>): void
+UpdateRubricItemGrade(_grade: QMap<QString, QString>): void
+GenerateGradeFile(_activityID: QString): QList<QString>
+GetGradesByItemID(_rubricItemID: QString): QList<QMap<QString, QString>>
+CheckIfCanBeRemarked(_activityID: QString, _studentID: QString, _markerPrivilegeLevel: int): bool
+ CheckIfHasBeenMarked(_activityID: QString, _studentID: QString): bool
+ GetActivityGradesByStudentID(_activityID: QString, _studentID: QString): QList<QMap<QString, QString>>
+ delRubricItemGrade(_rubricItems: QStringList): bool
-params: QMap<QString, QString>

Session:

-userID: string
-sessionID: QString
-sessionCode: int
-userRole: int
-coursed: QString
-activityID: QString
-courseTitle: QString
-studentID: QString
-activityType: QString
+logout(** _session: Session): bool
+SetRole(_role: int): bool
+getUserID(): string
+getSessionID(): QString
+getSessionCode(): int
+getUserRole(): int
+setCourseID(cid: QString): int
+getCourseID(): QString
+getActivityID(): QString
+setActivityID(QString aid): void
+setCourseTitle(title: QString): void
+getCourseTitle(): QString
+setStudentID(sid: QString): void
+getStudentID(): QString
+setActivityType(type: QString): void
+getActivityType(): QString

Student:

+processStudentListFile(filename: QString) : QList<QMap<QString, QString>>
+setCourseStudentList(coursed: QString, newStudentList: QList<QMap<QString, QString>>): bool
+ enrollStudentCourses(userID: QString, courseIDs: QStringList): bool
+ removeAllFromCourse(coursed: QString): bool
+ createStudent():QMap<QString, QString>

+ deleteStudent(studentID: QString): bool
+ getEnrolledStudents(coursed: QString): QList<QMap<QString, QString>>
+ getEnrolledStudentIDs(coursed: QString): QList<QString>
+ addToCourse(studentID: QString, coursed: QString): bool
+ removeFromCourse(studentID: QString, coursed: QString): bool
+ setStudentID(newValue: QString, *student: QMap<QString, QString>, verify = false: bool): bool
+ verifyStudentID(newValue: QString): bool
+ getStudentID(* const student: const QMap<QString, QString>: QString

User

+LEN_USERID:static const int
+MAX_LOGIN_ATTEMPTS:static const int
+getAllUsers():QList<QMap<QString, QString>>
+getAllTAs():QList<QMap<QString, QString>>
+getAllInstructors():QList<QMap<QString, QString>>
+getAllInstructorsAndTAs:QList<QMap<QString, QString>>
+getAllSystemAdmins():QList<QMap<QString, QString>>
+isLastSystemAdministrator(* const user: QMap<QString, QString> const): bool
+createUser():QMap<QString, QString>
+createUser(userID: QString): QMap<QString, QString>
+updateUserID(QString oldID, QString newID): bool
+insertUser(user: QMap<QString, QString>) : bool
+deleteUser(user: QMap<QString, QString>) : bool
+deleteUser(userID: QString) : bool
+verify (* const user: QMap<QString, QString> const): bool
+updateUser(user: QMap<QString, QString>): bool
+ getUserAccounts(start: int, end: int): QStringList
+login(username: QString, password: QString) : QMap<QString, QString>
+getUser(const username: QString): QMap<QString, QString>
+getUserID(* const user: const QMap<QString, QString>): QString
+getEmployeeID(*const user: const QMap<QString, QString>): QString
+getPassword(* const user: const QMap<QString, QString>): QString
+getEmail(* const user: const QMap<QString, QString>): QString
+getFirstName(* const user: const QMap<QString, QString>): QString
+getMiddleName(* const user: const QMap<QString, QString>): QString
+getLastName(* const user: const QMap<QString, QString>): QString
+getRoles(* const user: const QMap<QString, QString>): QList
+isInstructor(* const user: const QMap<QString, QString>): bool
+isSystemAdministrator(* const user: const QMap<QString, QString>): bool
+isAdministrativeAssistant(* const user: const QMap<QString, QString>): bool
+isTeachingAssistant(* const user: const QMap<QString, QString>): bool
+isAdministrator(* const user: const QMap<QString, QString>): bool
+isRole(* const user: const QMap<QString, QString>): bool
+isRole(* const user: const QMap<QString, QString>): bool
+isBlocked(* const user: const QMap<QString, QString>): bool

+isPwdResetNeeded(* const user: const QMap<QString, QString>): bool
+getLoginAttempts(* const user: const QMap<QString, QString>): int
+setUserID(newID: QString, *user: QMap<QString, QString>, verify = false: bool): bool
+setEmployeeID(newID: QString, *user: QMap<QString, QString>, verify = false: bool): bool
+setPassword(password: QString, *user: QMap<QString, QString>, verify = false: bool): bool
+setEmail(email: QString, *user: QMap<QString, QString>, verify = false: bool): bool
+setFirstName(FirstName: QString, *user: QMap<QString, QString>, verify = false: bool): bool
+setMiddleName(MiddleName: QString, *user: QMap<QString, QString>, verify = false: bool): bool
+setLastName(LastName: QString, *user: QMap<QString, QString>, verify = false: bool): bool
+setInstructor(Instructor: bool, *user: QMap<QString, QString>, verify = false: bool): bool
+setSystemAdministrator(sysAdmin: bool, *user: QMap<QString, QString>, verify = false: bool): bool
+setAdministrativeAssistant(adminAssist: bool, *user: QMap<QString, QString>, verify = false: bool): bool
+setTeachingAssistant(ta: bool, *user: QMap<QString, QString>, verify = false: bool): bool
+setAdministrator(admin: bool, *user: QMap<QString, QString>, verify = false: bool): bool
+setBlocked(blocked: bool, *user: QMap<QString, QString>): void
+setPwdResetNeeded(pwdReset: bool, *user: QMap<QString, QString>): void
+setLoginAttempts(totalAttempts: int, *user: QMap<QString, QString>): void
+verifyUserID(const newID: QString): bool
+verifyEmployeeID(const newID: QString): bool
+verifyPassword(const password: QString): bool
+verifyEmail(const email: QString): bool
+verifyFirstName(const fName: QString): bool
+verifyMiddleName(const mName: QString): bool
+verifyLastName(const lName: QString): bool
+verifyRoles(*const user: const QMap<QString< QString>): bool
+verifyLoginAttempts(const totalAttempts: int): bool

Backup:

+creationDate:QDateTime
+systemAdministratorID:QString
+getBackupTables(bDate: QDateTime): QList<QString>
+setBackupTables(bDate: QDateTime): QList<QString>
+createBackup():void
+restoreSystem(backupTables:QList<QString>): void
-tableNames: QList<QString>
-params: QMap<QString, QString>

Data Persistence

Table Design

Course Table

courseID	courseNumber	courseName	startDate	endDate	instructorID

Activity Table

activityID	courseID	activityName	pathToGradeFile	pathToWorkToMark	pathToSolutionFile	language	dueDate	isProgrammingActivity	isEssayActivity

User Table

userID	employeeID	password	email	firstName	middleName	lastName	isInstructor	isSystemAdministrator	isAdministrativeAssistant	isTeachingAssistant	isAdministrator	isBlocked	resetPassword

Marker Table

userID	courseID	privilegeLevel

Programming Activity Table

activityID	programmingLanguage	compilerEnvironment

Student Table

studentID	courseID

Rubric Table

rubricID	activityID

Rubric Section Table

rubricID	sectionID	expectedOutcome	weight

Programming Test Table

programmingTestID	activityID	compileCommand	pathToTestingScript

Multiple Choice Answer Key Table

multipleChoiceAnswerKeyID	activityID

Multiple Choice Answer Table

answerID	multipleChoiceAnswerKeyID	value	weight

Programming Test Item Table

testItemID	programmingTestID	pathToInputFile	PathToOutputFile	pathToConsoleOutputFile