# Low-Level Design Architecture

ER Tutoring System

**The A-Team**

University of Massachusetts Amherst

# CONTENTS

# Part I

# INTRODUCTION

# 1 SYSTEM OVERVIEW

The purpose of this system is to be an online tutor for students who are learning to create ER diagrams. This learning tool teaches students to build ER diagrams and allows them to submit the diagrams that they create as responses to assignment questions. The assignments are created by instructors using questions that are taken from the question bank. Each question, that is stored in the question bank, is created by an author and includes any correct answers and necessary feedback that correspond to it. After a student submits a diagram to be graded, he or she is either notified that diagram is correct or given feedback about the incorrectness of the diagram. This allows each student to learn from his or her mistakes. Overall, the benefits of this system are that it serves as an easier and more efficient tool for students who are learning to create ER diagrams and submitting assignments and instructors who are creating assignments and viewing student reports.

# 2

# STAKEHOLDERS

The stakeholders of the system involve the three defined user groups (authors, instructors and students), as well as the client, management, administrators, and developers. The administrators of the system will be involved in managing the three user groups. Administrators will give access rights to each group based on what they need to do (Authors can create questions, but instructors/students can't, etc.). While the system itself has the capability to manage direct user registration the administrators have a stake in ensuring that the system is accessible to those who need it. The users all have a stake in either accessing, modifying, or creating the questions hosted within the system, and the question bank in general. The instructors who have access to the system have a stake in creating groups of questions for students and in a lesser sense creating or editing questions to their needs, and accessing reported student progress on those content clusters. Authors primarily add and edit questions in the bank and will not directly interface with student answers. Students do not create or edit questions, but answer and submit content clusters as assigned by instructors. The client and management have stakes in the system because they are responsible for providing guidance to ensure that the final product is what they want. Naturally, this leaves the developers to have a stake in the system by implementing the wants and needs of the client and management into the system.

# 3 SCOPE OF THE DOCUMENT

This document describes the architectural layout of the online tutoring system. It includes an overview, stakeholders and a glossary to help define and describe the important system attributes. In addition, this document displays many design modules that come together to help form the architecture of the online tutoring system. These design modules include a deployment design, which describes the technologies used for each component of the system, a modular design, which describes the different client and server modules that are involved in transferring data, a client design, which describes all of the methods that are responsible for performing different client actions, and a server design, which describes the server's role as the connection between the client and the data base.

# 4

# DEFINITIONS

- **Answer** - Created by an author (with a corresponding question) and constructed by a student (in the form of an ER diagram) to respond to an assigned question. A student's answer to a specific question in an assignment will be graded when that assignment is submitted.

- **Assignment** - A collection of questions that the instructor selects for students to answer by a certain date.

- **Author** - User who creates the questions, answers and feedback. Also, the author uploads those questions, answers and feedback to the question bank for instructors to use when creating assignments.

- **Draw** - Action that students or author performs to insert the shapes and words from the toolbox to the answer area of the UI. This action occurs when students are answering questions or when the author is creating answers to questions.

- **Draw Space** - Area in which the answer to a question is drawn.

- **Feedback** - Message that is returned to each student after he/she submits an answer. It can be composed of hints, suggestions and messages from the author.
- **Instructor** - User who creates assignments by selecting questions from the question bank for students to answer. Also, the instructor can view the student reports for each assignment.
- **Learning Tool** - System that is used by a student to understand how to create ER diagrams.
- **Online Tutor System** - System that is being created in order to facilitate the instruction and education of constructing ER diagrams.
- **Password** - Secret sequence of characters that a user uses to log into the system.
- **Question** - Created by an author (for an instructor to pull from the question bank), assigned by an instructor (in an assignment) and answered by a student (with an ER diagram).
- **Question Bank** - Database that contains all of the questions that have been created by the author. The instructor selects questions from the question bank to create assignments for students.
- **Student** - A user that is accessing the online tutor system in order to learn and practice making ER diagrams.
- **Student Report** - Displays a unique student's results for a particular assignment

- **Toolbox** - Area of the UI that contains all of the shapes, lines and textboxes for the user or author to drag and drop onto the draw space when drawing.
- **NetID** - Username that a user uses to log into the system.
- **User** - Person using the online tutoring system. There are three different types of users: students, instructors and authors.

Part II

# DIAGRAMS

# 5

# DEPLOYMENT DESIGN

In this high-level deployment diagram, we show the technologies used for each component of the system. The client will use HTML, CSS, and JS, and will be run on the userâĂŹs machine. This will communicated with the server, which will be built with Nodejs and will run on a Unix system. These components will use a Postgres Database in order to store all of the information of the system (user results, assignments, etc.). The database will be on the same machine as the server.

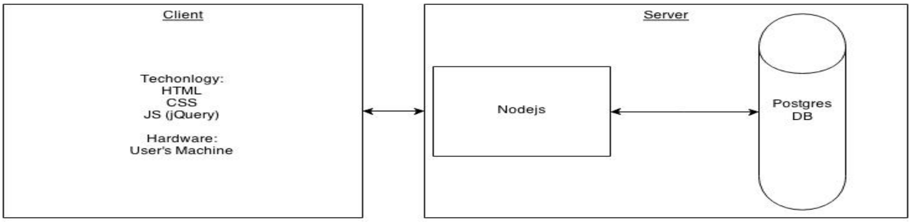


Figure 1: Deployment Diagram

# 6 MODULAR DESIGN

In this more detailed architecture design you can see the different modules in both the client and server sides. The Client is made up of the Drawing Widget, Report Formatter, and Serializer while the Server is made up of the Deserializer, Dispatcher, Validator, DB Accessor and the DB. The Drawing Widget contains everything needed for the user, author or student, to draw out either the question, answer or solution to a question that is displayed on the screen. The Report Formatter is only used by the instructor user and takes the information that it receives from the database to compile a report on the students for the instructor to view. The Serializer, is used for packaging and submitting the question and answer from the user to the server through http sending as a JSON. The Deserializer is used to take what was sent from the server and undo the serialization that was used when sending information to the server. On the Server side, the Dispatcher takes the information from the Server and packages the information to be sent to the client side of the system. The Validator is used when a student submits an answer to a question to check the answer for correctness, and generate feedback to the student. The DB Accessor is responsible for handling all of the requests made to send or receive information from the database. Finally the DB is the database that contains the question bank, the student responses and assignments.

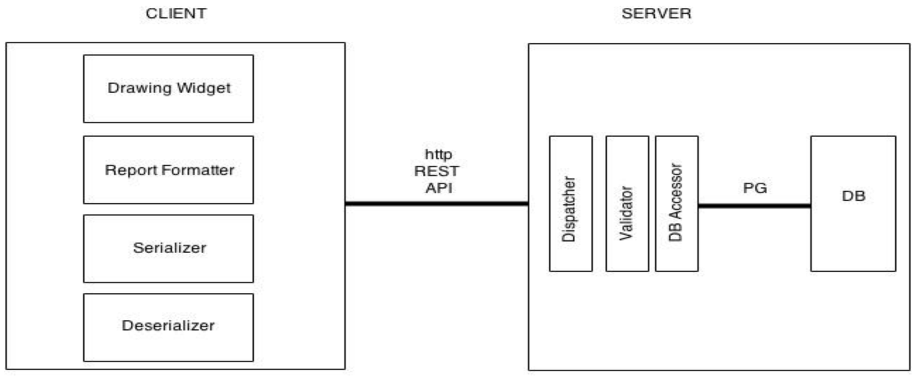


Figure 2: Modular Diagram

# 7 CLIENT

This is the diagram for the Client module. The following are the methods contained in it. openAssignment() is used by both students and instructors to open an assignment. It calls sendPageRequest() and renderPage() to render the page. createAssignment() is used by an instructor to create an assignment. openQuestion() is used by all three users, the authors, the instructors, and the students to open a question. It uses the render methods to render the page. The createQuestion() method is used by an author to create a question. drawDiagram() is used by both authors when creating the diagram and students when answering it. It uses the methods in DrawingWidget. placeEntity() to place the entities in the diagram, placeRelation() to place the relations in the diagram, and addText() to add text to the diagram. submitAnswer() is used to submit a diagram and calls sendToSerial() in DrawingWidget which uses the Serializer methods createSerialJSON() to convert the diagram into a JSON format, and sendJSON() to send this JSON to the server. openReport() is used by an instructor to view the student reports. The methods in ReportFormatter are used, collectStudentReports() retrieves the reports from the server, formatReports() formats the reports to be viewed, and render() renders the webpage. When diagrams are sent back to the client from the server they go through the Deserializer which calls the methods, recieveJSON() when the JSON is passed back from the server and then unpackSerialJSON() to unpack the JSON back into a diagram.

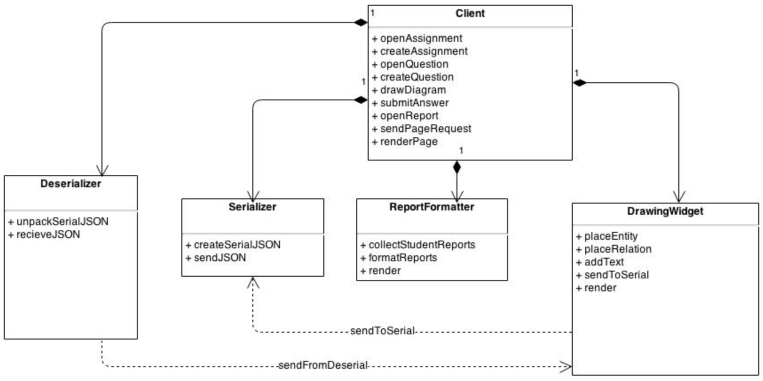


Figure 3: Client Diagram

# 8 SERVER

The Server plays a huge role in our system, it is in charge in keeping connection with the Client and the database. The Dispatcher will process the question submitted by the user and request its validation with the Validator. Then, the Validator checks whether the question submitted by the student is actually right or not, by going to the database accessor. Moreover, when validator check if a question is right or not, it sends a customized feedback for that particular question. The Server also is able to update a particular question in the database that has been saved by the author.

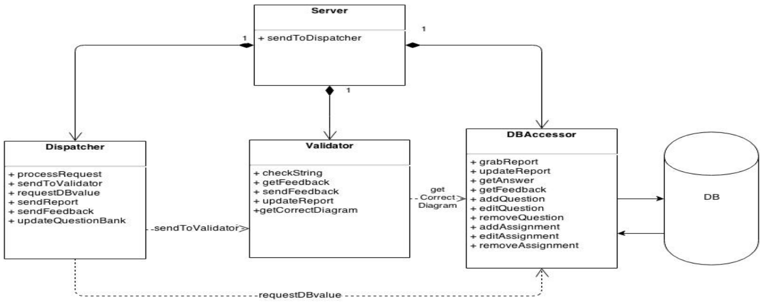


Figure 4: Server Diagram

# Part III

# MODULES

# 9

# SERIALIZER

**Description:** The serializer module provides a way for a client to send an ER diagram to the server by converting the ER diagram into a JSON.

**Interface Specification:**

- JSON createSerialJSON(diagram) - This public method creates a JSON representation of an existing ER diagram.
- void sendJSON() - This public method sends a JSON from a client to the server.

**Assumptions:**

- The ER diagram that is being used by this class is an existing and completed ER diagram.

**Dependencies:** This module is contained by the Client module and used by the DrawingWidget module.

**Internal Structure:**

- **Internal Methods:**
  - None
- **Internal Variables:**

    - None

**Hidden Information:**

- None

# 10

# DESERIALIZER

**Description:** The deserializer module provides a way for a client to receive an ER diagram from a server by converting the JSON representation of an ER diagram back into the actual ER diagram.
**Interface Specification:**

- diagram unpackSerialJSON() - This public method converts the JSON representation of an ER diagram back into the actual ER diagram.
- JSON recieveJSON() - This public method allows a client to receive a JSON from the server.

**Assumptions:**

- The JSON that is being used by this class is the JSON representation of an existing and completed ER diagram.

**Dependencies:** This module is contained by the Client module and uses the DrawingWidget module.
**Internal Structure:**

- **Internal Methods:**
  - None

- **Internal Variables:**
  - None

**Hidden Information:**

- None

# 11 DISPATCHER

**Description:** The dispatcher takes the information from the Server and packages the information to be sent to the client side of the system.
**Interface Specifications:**

- void processRequest - This public method processes the request to be sent to the database.
- void sendToValidator - This public method sends a students answer to a question to the validator.
- value requestDBValue – This public method returns a value that is requested from the database through the DBAccessor module.
- void sendReport – This public method compiles and sends a report to the caller.
- void sendFeedback – This public method compilkes and sends feedback to the caller.
- void updateQuestionBank – This public method updates the question bank contained in the database.

**Assumptions:**

- The modules are online and accessible.

**Dependencies:** This module is contained by the Server module and is dependent on the Validator module and DBAccessor module.
**Internal Structure:**

- **Internal Methods:**
  - None
- **Internal Variables:**
  - None

**Hidden Information:**

- None

# 12

# TOOLBOX

**Description:** The Toolbox contains all of the different entities and relations that are possible for use in the ER Diagram. The Toolbox has drag and drop capabilities to allow the user to click and then drag the desired entity or relation that they wish to insert into the Canvas to create the ER Diagram.
**Interface Specification:**

- void selectEntity(entity) - this method is the user selecting an entity from the Toolbox and dragging it into the Canvas.
- void selectRelation(relation) - this method is the user selecting a relation from the Toolbox and dragging it into the Canvas.
- List<entity> interactiveEntryList - variable that lists all of the possible entities in the Toolbox
- List<relation> interactiveRelationList - variable that lists all of the possible relations in the Toolbox

**Assumptions:**

- The Toolbox contains all of the possible entities and relations possible and the user can drag and drop them.

**Dependencies:** This module is dependent on the Drawing-Widget module, and Question module. This module depends on the Entity module and the Relation module.
**Internal Structure:**

- **Internal Methods:**
  - None
- **Internal Variables:**
  - None

**Hidden Information:**

- None

# 13

# RELATION

**Description:** This module contains all of the relations or lines that connect the different entities. There will be options for both Chen and Crows Foot styles of ER Diagrams. The relations will show if it is many to one, one to many, one to one or many to many.
**Interface Specification:**

- void addLabel(String) - Adding a description to the relation to describe the name and style of relation such as one to one, one to many, many to many, and many to one.
- void relation(type) - creating a relation, the constructor of the class
- Relation relation - the relation variable that is used in the class

**Assumptions:**

- All possible relations are present

**Dependencies:** This module is dependent on the Toolbox module and the DrawingWidget module. It depends on the Entity module.
**Internal Structure:**

- **Internal Methods:**
  - None
- **Internal Variables:**
  - String type - the type of relation

**Hidden Information:**

- None

# 14

# CANVAS

**Description:** This module contains all the necessary components to build a graphical user interface. We want to ensure that all the components are well drawn and easy for the user to use.
**Interface Specification:**

- deleteEntity(entity):void - deletes the specified entity
- moveEntity(entity):void - moves the specified entity
- deleteRelation(relation):void - deletes the specified relation
- moveRelation(relation):void - moves the specified relation

**Assumptions:**

- None

**Dependencies: Canvas depends on the ToolBox and the DrawingWidget. Internal Structure:**

- **Internal Methods:**
  - None

- **Internal Variables:**
  - None

**Hidden Information:**

- None

# 15 DRAWINGWIDGET

**Description:** This will contain everything needed in order to draw and view an ER diagram. This includes a Toolbox for selecting items to draw, and a Canvas to place the ER diagram.

**Interface Specification:**

- Toolbox toolBox - Toolbox instance that will allow users to pick elements to draw
- Canvas canvas - Canvas where the ER diagram will be drawn

**Assumptions:**

- None

**Dependencies:** This module is dependent on the Toolbox and Canvas modules

**Internal Structure:**

- **Internal Methods:**
  - None
- **Internal Variables:**

    - int size - size of the entire widget

**Hidden Information:**

- None

# 16

# ENTITY

**Description:** This module contains all the entities to be connected by relations in the ER diagram. There will be options for both Chen and Crows Foot styles of ER Diagrams.
**Interface Specification:**

- void addText (String) - adding text to be placed inside of the entity.
- void entity(type) - creating an entity, the constructor of the class.
- Entity entity - the entity variable that is used in the class.

**Assumptions:**

- All possible entities are present.

**Dependencies:** This module is dependent on the Toolbox module and DrawingWidget module. It depends on the Relation module.
**Internal Structure:**

- **Internal Methods:**
  - None

- **Internal Variables:**
  - String type - the type of entity

**Hidden Information:**

- None

# 17

# UI UML DIAGRAM

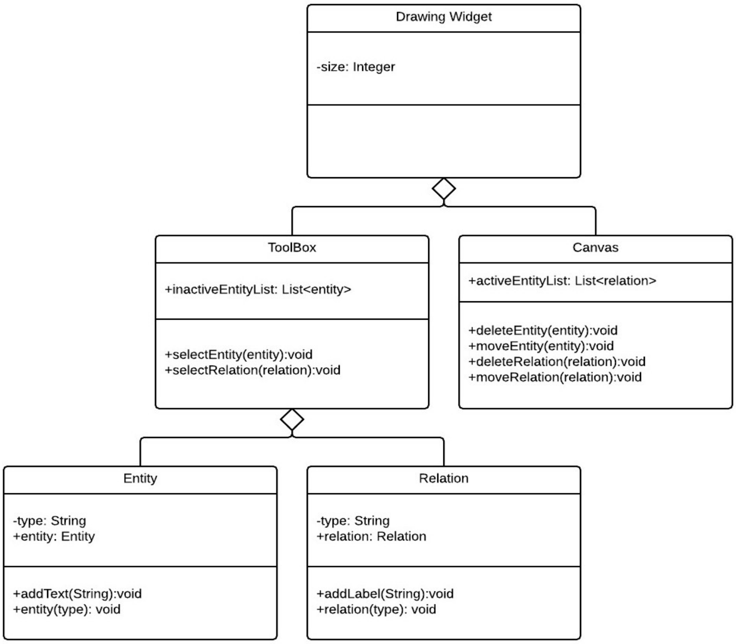


Figure 5: UI UML Diagram

# 18

# REPORTFORMATTER

**Description:** This module formats the student reports to be presented to an instructor.
**Interface Specification:**

- void collectStudentReports() - collects the student reports from the database.
- void formatReports() - formats the student reports to be readable.
- void render() - renders the page and shows the student reports.

**Assumptions:**

- There are reports to format and present

**Dependencies:** This module is dependent on the DBAccessor module.
**Internal Structure:**

- **Internal Methods:**
  - None
- **Internal Variables:**

    - None

**Hidden Information:**

- None

# 19

# USERS UML DIAGRAM

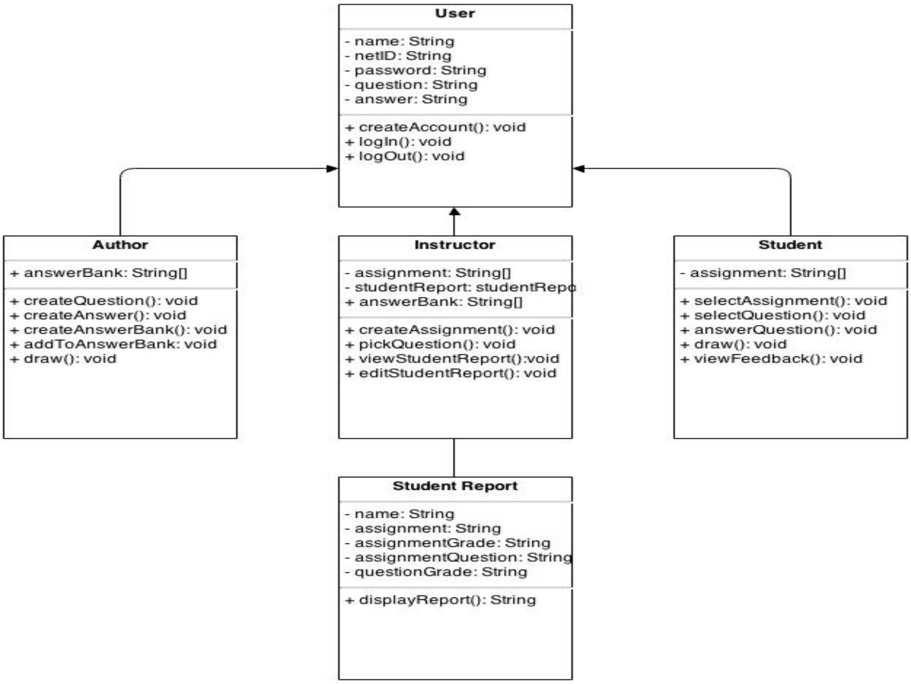


Figure 6: Users UML Diagram

# 20

# USER

User module provides a way to retrieve userâĂŹs information. It handles logging in, logging out and user registration.
**Interface Specifications:**

- void create account()- Creates account, using user’s input. Create a new database entry with user’s ID, account type, name/email/password etc.
- void logIn()- read user’s input authentication input and create a new session.
- void logOut()- finish user’s session.

**Assumptions:**

- None

**Dependencies:** Students, Instructors, Authors depend on User Module
**Internal Structure:**

- **Internal Methods:**
  - void connectToUserDatabase()- Handles database connection.

    - void checkNetIDUnique()- Check whether given ID does not exist in database upon registration.

- **Internal Variables:**
    - name: String
    - netID: String
    - password: String

**Hidden Information:**

- None

# 21

# STUDENT

**Descrioption:** Student Module manages student's account. It handles the access to the assignment list
**Interface Specifications:**

- void selectAssignment()- reads user's input to obtain assignment ID and then retrieves assignment from database
- void displayQuestion()- outputs question based on its ID
- void selectQuestion()- reads user's input to obtain question ID and then retrieves assignment from database
- void answerQuestion()- handles the input provided by Student
- void draw()- handles the input provided by user on the drawing board
- void viewFeedback()- outputs feedback based on the student's answer

**Assumptions:**

- Logging in is handled by User Module

- Account type is provided.

**Dependencies:**

- Depends on Draw
- Assignment
- Question Modules.

**Internal Structure:**

- **Internal Methods:**
  - None
- **Internal Variables:**
  - assignment: String[]

**Hidden Information:**

- None

# 22 INSTRUCTOR

**Description:**Instructor module manages instructorâĂŹs account. It provides interface to create new assignments, add questions to individual assignments, make lists of students that can access a particular assignment and finally to check student report.
**Interface Specifications:**

- void createAssignment(): handler for creating a new assignment, reading assignment name and list of students that instructor provides, and creating list of questions
- void viewStudentReport(): accesses the scores database and outputs the data upon instructors request
- void editStudentReport(): instructor can update student scores and information

**Assumptions:**

- Depends on assignment and question, and drawing tool modules

**Dependencies:** Students, Instructors, Authors depend on User Module
**Internal Structure:**

- **Internal Methods:**
  - void makeStudentList(): instructor makes a list of students for each assignment
  - void makeQuestionList(): instructor makes a list of questions for each assignment
- **Internal Variables:**
  - assignment: String[]
  - studentReport: studentReport
  - answerBank: String[]

**Hidden Information:**

- None

# 23

# AUTHOR

**Description:** Author module manages author's account. It provides interface to create new questions, add possible answers and feedback to individual question.
**Interface Specifications:**

- void createQuestion(): provides interface to make a new question, with question text/diagram options/feedback, uses create answer and create feedback
- void draw(): provides an interface to draw a possible answer, gives an option whether to draw.

**Assumptions:**

- Logging in is handled by User Module, and account type is provided.

**Dependencies:** Depends on assignment and question, and drawing tool modules
**Internal Structure:**

- **Internal Methods:**
  - void createAnswer(): reads the input provided by author in a form of diagram of text

    - void createFeeback(): provides a way to create feedback for a created question

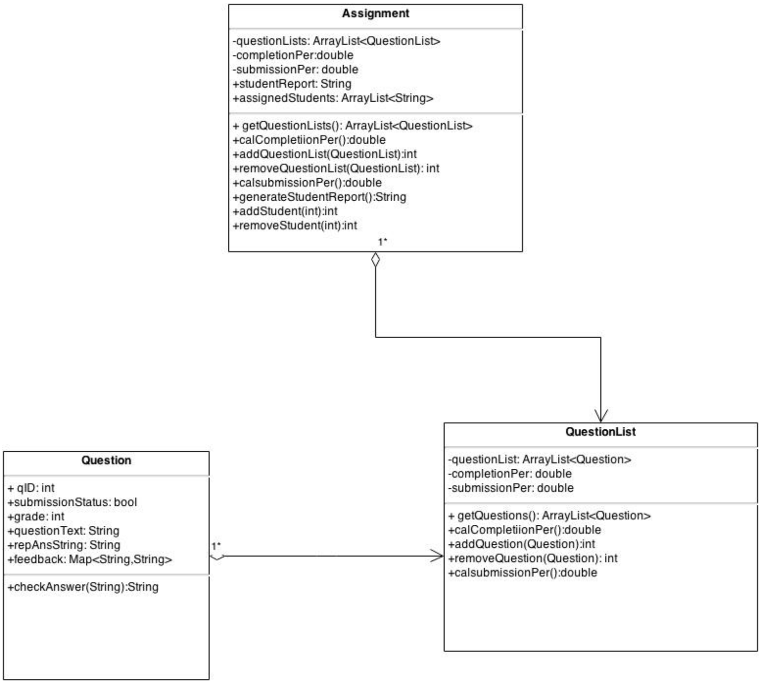
- **Internal Variables:**
    - answerBank: String[]

**Hidden Information:**

- None

# 24

# UML 3



Assignment
-questionLists: ArrayList<QuestionList>
-completionPer:double
-submissionPer: double
+studentReport: String
+assignedStudents: ArrayList<String>
+ getQuestionLists(): ArrayList<QuestionList>
+calCompletiionPer():double
+addQuestionList(QuestionList):int
+removeQuestionList(QuestionList): int
+calsubmissionPer():double
+generateStudentReport():String
+addStudent(int):int
+removeStudent(int):int
1*
QuestionList
-questionList: ArrayList<Question>
-completionPer: double
-submissionPer: double
+ getQuestions(): ArrayList<Question>
+calCompletiionPer():double
+addQuestion(Question):int
+removeQuestion(Question): int
+calsubmissionPer():double
Question
+ qID: int
+submissionStatus: bool
+grade: int
+questionText: String
+repAnsString: String
+feedback: Map<String,String>
+checkAnswer(String):String
1*

Figure 7: Assigments UML Diagram

# 25

# ASSIGNMENT

**Description:** An instructor creates an assignment so that the instructor can add questions or question list to the assignment. Controls what the final assignment looks like before being assigned.
**Interface Specifications:**

- void intialize(String Name) - Initialize an assignment with a specific name.
- void addQuestion(Question) - Add a question that is in the database to the assignment. Checks to see if ″Question″ is in the database.
- void removeQuestion(Question) - Delete a specific question that is part of the assignment Checks to see if ″Question″ is the Question list.
- int count() - Counts the number of questions.
- boolean isQuestionInDB(Question) - Checks if the question is in the question database

**Assumption:**

- None

**Dependencies:** None
**Internal Structure:**

- **Internal Methods:**
  - void addList(questionList) - Add a list of questions to the assignment.
  - void removeList(questionList) - Deletes the list of questions from the assignment. Checks to see if the list is in the assignment.
  - void saveAssignment() - Save assignment to the assignment database
  - String getName() - returns name of the assignment
  - void changeName() - changes the name of the assignment
- **Internal Variables:**
  - ArrayList[] Question
  - String Name
  - int count

**Hidden Information:**

- None

# 26

# QUESTION LIST

**Description:** It′s a list of questions from which an instructor can choose to add the list to an assignment or view the list and then view individual questions. This list is used for a student to pick and choose which questions to answer in a specific assignment.
**Interface Specifications:**

- void initialize() - Initialize a list for questions to be added in the future. This list can be added to assignments. This is called when an instructor wants to make a question list.
- void addQuestion(Question) - Add a question that is in the database to the question list. Checks to see if ″Question″ is in the database.
- void removeQuestion(Question) - Delete a specific question that is part of the question list. Checks to see if ″Question″ is the Question list.
- int count() - Counts the number of questions.
- boolean isQuestionInDB(Question) - Checks if the question is in the question database

**Assumptions:**

- None

**Dependencies:** None
**Internal Structure**

- **Internal Methods**
  - boolean isEmpty() - Checks if the list is empty
  - void find(Question) - Searches for the question in the list
- **Internal Variables**
  - ArrayList[] Questions;
  - int count;

**Hidden Information:**

- None

# 27 QUESTION

**Description:** Questions can be made by the author. These questions will be added to the database and can be copied from the database and used for question list and assignments.

**Interface Specifications:**

- void addQuestion(Question) - adds the question to the database.
- void removeQuestion(Question) - remove question from database.
- boolean isQuestionInDB(Question) - Checks if the question is in the question database

**Assumption:**

- An Answer will be provided with the question.

**Dependencies:** None

**Internal Structure:**

- **Internal Methods:**
  - Question makeQuestion(String Name); Makes a question to be added to the database of questions.

  - Diagram drawAnswer(); Make an answer to be assigned with this question.
  - void UpdateQuestion(); Updates the question, adds question to database

- **Internal Variables:**
  - String Name;
  - Diagram Answer;
  - String Question;

**Hidden Information:**

- None

# 28

# VALIDATOR

**Description:** This module is responsible for validating a diagram provided by a student. compares it to the correct diagram created by the author which it retrieves from the database. Because there are multiple variations of the same diagram available, we also need to make sure to compare these variations to the correct diagram.
**Interface Specification:**

- diagram getCorrectDiagram(int): retrieves the solution from the database given question ID
- boolean validateSolution(diagram, diagram): compares input diagram and correct diagram and outputs whether they are equivalent
- void sendFeedback(): send feedback provided by the author String

- getFeedback(int): retrieve feedback for a given question from database
- void updateReport(): update student report given the result of comparison of studentâĂŹs input with the correct diagram

**Assumptions:**

- We are assuming that the correct answer for the diagram is in database and that the input comes in JSON format

**Dependencies:**
This module depends on the Serializer and DBAccessor modules
**Internal Structure:**

- **Internal Methods:**
  - String[] constructAllVariation(diagram): make all possible variations of JSON strings corresponding to input diagram **Internal Variables:**
- **Internal Variables:**
  - None

**Hidden Information:**

- None

# 29 DBACCESSOR

**Description:** This module is responsible for retrieving required information from the database such as user information, diagram and questions.
**Interface Specification:**

- String getQuestion(int): retrieves question information from database provided ID of the question
- void addQuestion(question): adds question to the database
- void editQuestion(): edit questions
- String getAssignment(int): retrieves assignment information from database provided ID of the assignment
- void addAssignment(): adds assignment to the database
- void editAssignmnet(): edit assignment
- String getSolution(int): retrieves JSON string pf the correct solution given question ID
- String getUserInfo(int): retrieve all the information about the USER given his ID
- String getFeedback(int): retrieves feedback provided by the author given the question ID

- void updateFeedback(String, int): update or create new feedback given question ID
- void getUserInfo(): retrieve userâĂŹs information such as name, ID etc
- void addUserInfo() create a new entry in user database
- void removeQuestion(int): removes question from the database given question ID
- void removeAssignment(): removes assignment from database
- void getStudentReport(): retrieves student report from the database

**Assumptions:**

- Correct name and password of the database are provided for this module

**Dependencies:** This module depends on all modules using the database: User, Validator
**Internal Structure:**

- **Internal Methods:**
  - void connectToDatabase(): establishes a connection to database using name and password provided
- **Internal Variables:**
  - String databaseName: stores database name
  - String databasePassword: store the database password

**Hidden Information:**

- None