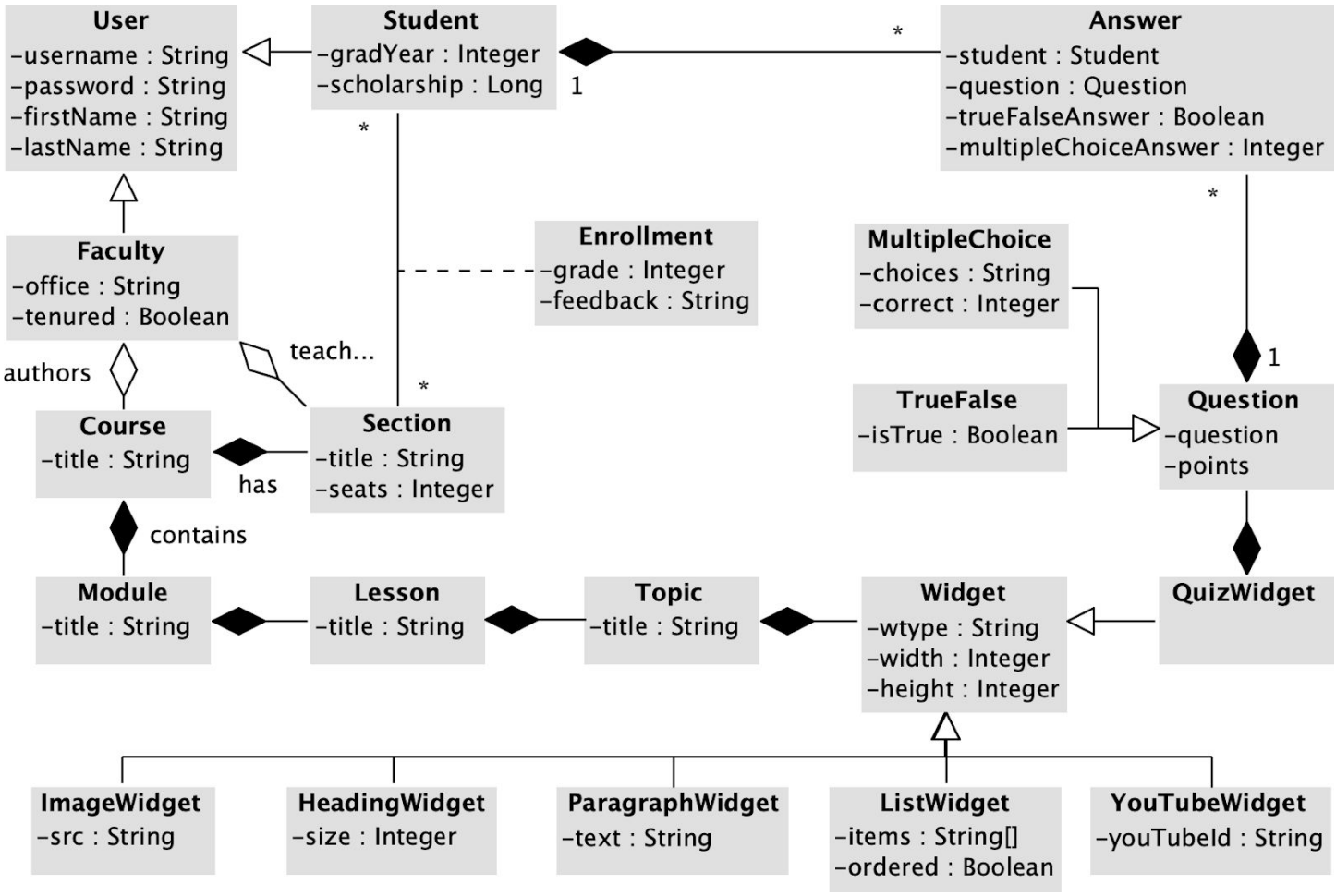
MongoDB Assignment

Introduction

Consider the class diagram below. This assignment will map the following classes to an equivalent Mongo database: **Student**, **Answer**, **Question**, **MultipleChoice**, **TrueFalse**, **Question**, **QuizWidget**. Ignore all other classes. Feel free to add additional fields, but do not rename the existing variable names.



Create and connect to a MongoDB database

In file **/data/db.js** use Mongoose.js to connect to a database called **white-board**.

Implement Mongoose schemas

It is a good practice to implement Mongoose schemas in their own files allowing them to be reused by and defined in terms of other schemas. Create the following schemas in directory **/data/models/** based on the class diagram provided. Instead of letting MongoDB create the **_id** unique identifiers for you, instead declare your own **_id** attribute in your schemas as Number. This will allow using our own IDs which will

simplify testing. Use equivalent JavaScript and Mongoose data types where appropriate. Feel free to make any changes/additions where necessary.

Class	Mongoose schema file	MongoDB collection
Student	student.schema.server.js	students
MultipleChoice	multiple-choice.schema.server.js	N/A
TrueFalse	true-false.schema.server.js	N/A

Implement inheritance schema

Implement base class **Question** and derived classes **TrueFalse** and **MultipleChoice** as an inheritance relationship implemented using a denormalized strategy, i.e., use a single schema to represent all three classes in file **/data/models/question.schema.server.js**. Use a discrimination field called **questionType** with required enumerated values '**MULTIPLE_CHOICE**' and '**TRUE_FALSE**' to distinguish between the question types. Instead of the schema for the Question class having the fields of **MultipleChoice** and **TrueFalse** classes, model these as embedded schemas accessible through schema properties **multipleChoice** and **trueFalse**. Store question instances in a collection called questions. Note the schema is declaring an **_id** attribute to allow providing our own ID for testing. Feel free to use/modify/ignore the example below as an implementation of the schema.

/data/models/question.schema.server.js

```
const mongoose = require('mongoose')
const TrueFalseSchema = require('./true-false.schema.server.js')
const MultipleChoiceSchema = require('./multiple-choice.schema.server.js')
modules.exports = mongoose.Schema({
 _id: Number,
 question: String,
 points: Number,
 questionType: String,
 multipleChoice: MultipleChoiceSchema,
 trueFalse: TrueFalseSchema
}, {collection: 'questions'})
```

Implement one to many relationship

In schema file **/data/models/quiz-widget.schema.server.js**, implement the one to many relationship between **QuizWidget** and **Question** classes. Use an array of **ObjectIds** to hold the IDs of the questions that make up the widget. Feel free to use/modify/ignore the example below as an implementation of the schema.

/data/models/quiz-widget.schema.server.js

```
const mongoose = require('mongoose')
```

```
const questionSchema = require('./question.schema.server')
const questionWidgetSchema = mongoose.Schema({
 questions: [{
  type: mongoose.Schema.Types.ObjectId,
  ref: 'QuestionModel'
 }]
}, {collection: 'question-widgets'})
```

Implement many to many

Implement class **Answer** as a many to many association between classes **Student** and **Question** in Mongoose schema file **/data/models/answer.schema.server.js**. Feel free to use/modify/ignore the example below as an implementation of the schema.

/data/models/answer.schema.server.js

```
const mongoose = require('mongoose')
const student = require('./student.schema.server')
const question = require('./question.schema.server')
const answer = mongoose.Schema({
 _id: Number,
 trueFalseAnswer: Boolean,
 multipleChoiceAnswer: Number,
 student: {type: mongoose.Schema.Types.ObjectId, ref: 'StudentModel'},
 question: {type: mongoose.Schema.Types.ObjectId, ref: 'QuestionModel'}
}, {collection: 'answers'})
```

Implement Mongoose data models

It is a good practice to implement Mongoose models for each collection in a separate file. This allow models to be used by and defined in terms of other models. In directory **/data/models/** implement mongoose models for the following classes.

Class	File
Student	student.model.server.js
Answer	answer.model.server.js
Question	question.model.server.js
QuizWidget	quiz-widget.model.server.js

Feel free to use/modify/ignore the example below as an implementation of the student model.

/data/models/student.model.server.js

```
const mongoose = require('mongoose')
const studentSchema = require('./student.schema.server')
module.exports = mongoose.model('StudentModel', studentSchema)
```

Create DAOs for each of the data models

DAOs (Data Access Objects) encapsulate and modularize interaction with a database by providing a high level interface to the database. DAOs allow other parts of the application to interact with the database without bothering with the idiosyncrasies and minutiae of a particular database implementation or vendor.

In file called **/data/daos/university.dao.server.js**, create a DAO that implements the following API. If you prefer, feel free to breakup the DAO into several DAO files. The DAOs must use the Mongoose models implemented earlier. Feel free to modify the signature of the methods if appropriate and create additional methods if needed, i.e., you might prefer to pass the IDs of existing documents instead of document instances. Implement the following API:

1. **truncateDatabase()** - removes all the data from the database. Note that you might need to remove documents in a particular order
2. **populateDatabase()** - populates the database with test data as described in a later section
3. **createStudent(student)** - inserts a student document
4. **deleteStudent(id)** - removes student whose ID is id. Delete does not cascade
5. **createQuestion(question)** - inserts a question document
6. **deleteQuestion(id)** - removes question whose ID is id. Delete does not cascade
7. **answerQuestion(studentId, questionId, answer)** - inserts an answer by student student for question question
8. **deleteAnswer(id)** - removes answer whose ID is id

Implement the following finder methods

1. **findAllStudents()** - retrieves all students
2. **findStudentById(id)** - retrieves a single student document whose ID is id
3. **findAllQuestions()** - retrieves all questions
4. **findQuestionById(id)** - retrieves a single question document whose ID is id
5. **findAllAnswers()** - retrieves all the answers
6. **findAnswerById(id)** - retrieves a single answer document whose ID is id
7. **findAnswersByStudent(studentId)** - retrieves all the answers for a student whose ID is studentId
8. **findAnswersByQuestion(questionId)** - retrieves all the answers for a question whose ID is questionId

Feel free to use/modify/ignore the example below as an implementation of the student related DAO functions

/data/daos/university.dao.server.js

```
const studentModel = require('../models/student.model.server')
```

```
createStudent = student => studentModel.save(student)
findAllStudents = () => studentModel.find()
findStudentById = studentId => studentModel.findById(studentId)
updateStudent = (studentId, student) => studentModel.update({_id: studentId}, {$set: student})
deleteStudent = studentId => studentModel.remove({_id: studentId})
module.exports = {
 createStudent, findAllStudents, findStudentById, updateStudent, deleteStudent }
```

Populate the database with test data

In DAO function **populateDatabase()**, populate the database with the data described in the following sections

Create students

Use the DAO functions to create the following students. Use the IDs in the **_id** column as the IDs for the documents inserted.

_id	First Name	Last Name	Username	Password	Grad Year	Scholarship
123	Alice	Wonderland	alice	alice	2020	15000
234	Bob	Hope	bob	bob	2021	12000

Create questions

Use the DAO functions to create the following questions. Use the IDs in the **_id** column as the IDs for the documents inserted.

_id	Question	Points	Type	Is true	Choices	Correct
321	Is the following schema valid?	10	TRUE_FALSE	false		
432	DAO stands for Dynamic Access Object.	10	TRUE_FALSE	false		
543	What does JPA stand for?	10	MULTIPLE_CHOICE		"Java Persistence API,Java Persisted Application,JavaScript Persistence API,JSON Persistent Associations"	1
654	What does ORM stand for?	10	MULTIPLE_CHOICE		"Object Relational Model,Object Relative Markup,Object Reflexive Model,Object Relational Mapping"	4

Create answers

Use the DAO functions to create the following questions. Use the IDs in the **_id** column as the IDs for the documents inserted. Use the IDs for students and questions listed in the table below.

_id	Student	Question	True false answer	Multiple Choice Answer
123	Alice	Is the following schema valid?	true	
234	Alice	DAO stands for Dynamic Access Object	false	
345	Alice	What does JPA stand for?		1
456	Alice	What does ORM stand for?		2
567	Bob	Is the following schema valid?	false	
678	Bob	DAO stands for Dynamic Access Object	true	
789	Bob	What does JPA stand for?		3
890	Bob	What does ORM stand for?		4

Create a test suite

In a file called **/test.js**, create tests that validate your DAO, schemas, and models. The test suite should use the DAO functions **truncateBatabase()** and **populateDatabase()** to initialize the database to an initial state. It then should implement and use the following functions to validate the database is in the valid state The test.js implement and invoke the following functions in the right order. Note that you might need to use promises to ensure the order of operation.

1. **truncateDatabase()** - uses DAO's truncateDatabase() function to remove all the data from the database
2. **populateDatabase()** - uses DAO's populateDatabase() function to populate the database with test data
3. **testStudentsInitialCount()** - uses DAO to validate there are 2 students initially
4. **testQuestionsInitialCount()** - uses DAO to validate there are 4 questions initially
5. **testAnswersInitialCount()** - uses DAO to validate there are 8 answers initially
6. **testDeleteAnswer()** - uses DAO to remove Bob's answer for the multiple choice question "What does ORM stand for?" and validates the total number of amswers is 7 and Bob's total number of answers is 3
7. **testDeleteQuestion()** - uses DAO to remove true false question "Is the following schema valid?" and validates the total number of questions is 3
8. **testDeleteStudent()** - uses DAO to remove student Bob and validates the total number of students is 1