CS387 Project Proposal

Yash Shah (160050002), Naman Jain (160050025)
Utkarsh Gupta (160050032), Rupesh (160050042)

October 2018

EDGE: Efficient Digital Grading Environment

1 Overview

For our project, we wanted to do something that would prove beneficial for the academic community of IITB. We plan to build an interface for providing an end-to-end solution to the paper correction procedure for large classes, starting right from question paper generation, to distributed grading across TAs, to cribbing (if time permits). Instructors shall be given access to a web interface that would allow them to generate question papers (Latex support), upload scanned answer sheets and model answers, assign TA's for specific questions, and get a bird's eye view of the class performance. Each student will have two roles - 'TA' and 'Student' - of which the former will be activated for a course only when its instructor assigns him that role. Within the 'Student' role, the user can view quiz-wise performance, model answers and his/her standing in the class. As a TA, the user can view and grade specific questions of quizzes, give individual remarks, and view model answers for that question.

2 Goals

2.1 Entity Relation Generation

For the proper working of our project, we will build the ER model for the given problem statement which we are trying to answer.

2.2 Question Paper Generation

Instructor provides the question, weightage and space to be provided for its answer to the app; it then generates a pdf (i.e. the question paper) with each question and required amount of space, along with a proper header mentioning the course code, date, title, maximum marks etc. It also stores the question paper structure in the database, to be used later for splitting the answer sheets into constituent questions.

2.3 Instructor Interface

The web interface for the instructor will provide a means of making question papers, assigning TAs for evaluation and other details as mentioned in the detailed specifications.

2.4 Student and TA Interfaces

Android interface for the student and TA - according to course-dependent role, the user will be able to check test-wise results and crib (student), or evaluate the papers (TA).

2.5 Maintaining and Updating Database

This interface deals with aspects such as enrollment and removal of students from a course, provision of TA privileges to students, and addition/removal of courses and instructors.

3 Detailed specifications

Web App for Instructor

- Create question paper through Latex
- View class performance
- View and edit relevant TA information and duties
- Assign questions to TAs for correction
- Upload model answers
- Grant TA privileges to a student
- Remove and add students to course

Android App for TA

- Correction of assigned questions
- Enter and edit marks for students
- Add answer-specific remarks
- View model answers
- Resolve cribs (optional)

Android App for Student

- View their marks
- View ranks and course statistics
- View model answers
- Interface for cribbing (optional)

4 Software Specifications

- Apache Tomcat server
- Java Servlets backend
- HTML, CSS and JQuery Javascript for web app
- Postgresql for database
- Android Studio and Flutter

5 Testing

We will be creating a dummy database and testing all the deliverable functionalities on it.

6 E-R diagram

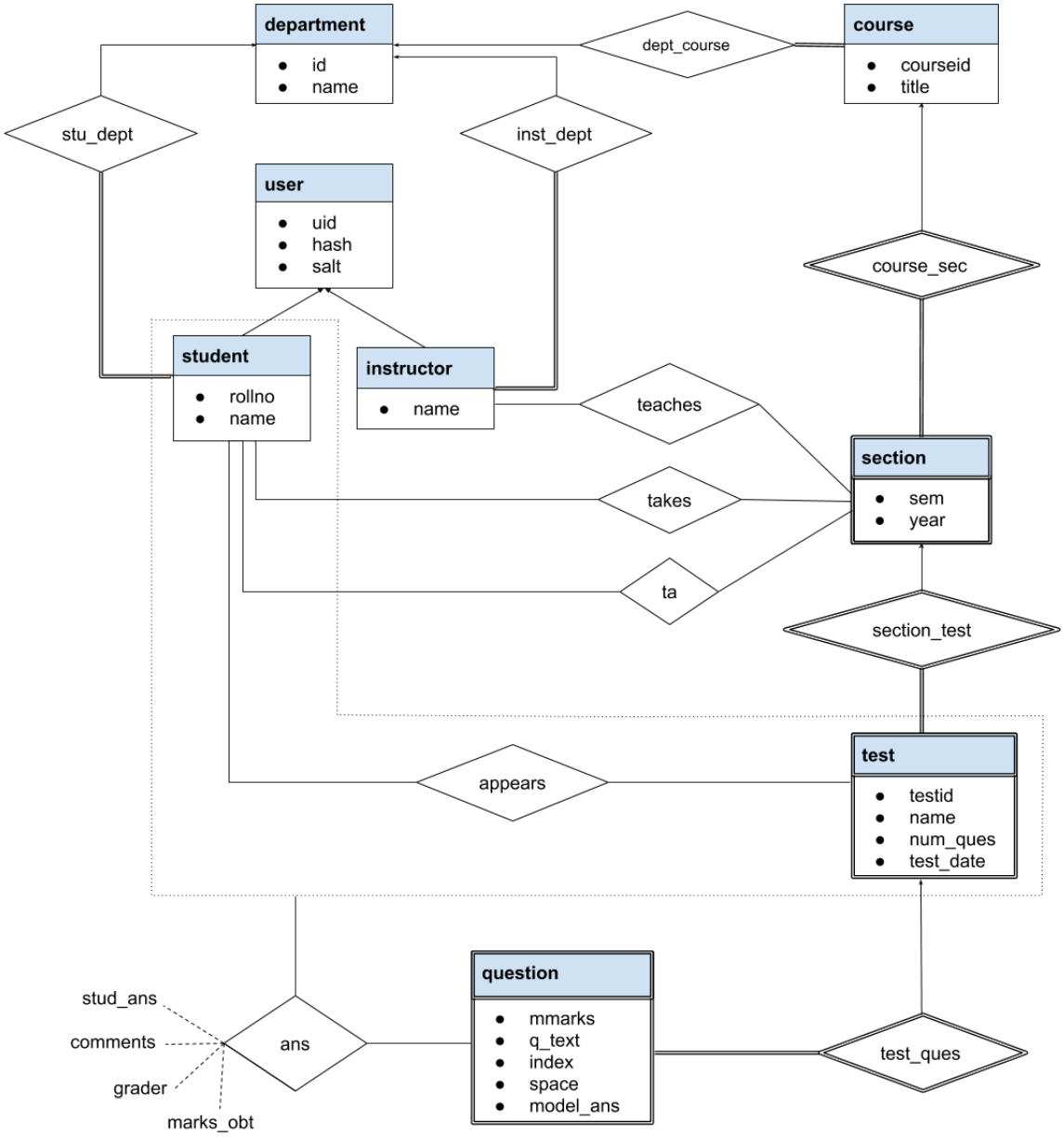

Figure 1: ER diagram for our database

7 Table Design

This is the proposed create.sql

```
create table user
(uid varchar(20),
 hash varchar(256),
 salt varchar(20),
 primary key(uid)
);
```

```
create table department
(dept_name varchar(20),
 primary key (dept_name)
);


create table instructor
(uid  varchar(20),
 name  varchar(20) not null,
 dept_name  varchar(20),
 primary key(uid),
 foreign key(uid) references user
on delete cascade,
 foreign key(dept_name) references department
on delete set null
);

create table student
(rollnumber varchar(20),
 name varchar(20) not null,
 uid varchar(20),
 dept_name varchar(20),
 primary key (rollnumber),
 foreign key (uid) references user
  on delete cascade,
 foreign key (dept_name) references department
  on delete set null
);

create table course
(course_id  varchar(20),
 title      varchar(20),
 dept_name  varchar(20),
 primary key(course_id),
 foreign key(dept_name) references department
on delete set null
);

create table section
(course_id varchar(20),
 semester varchar(20),
 year varchar(20),
 primary key (course_id,semester,year),
 foreign key (course_id) references course
  on delete cascade
);

create table teaches
(uid  varchar(20),
 course_id  varchar(20),
 semester  varchar(20),
 year  varchar(20),
 primary key(uid,course_id,semester,year)
 foreign key(uid) references user
on delete cascade,
 foreign key(course_id,semester,year) references section
on delete cascade,
 );
```

```
create table TA
(rollnumber varchar(20),
 course_id  varchar(20),
 semester  varchar(20),
 year  varchar(20),
 primary key(rollnumber,course_id,semester,year)
 foreign key(course_id,semester,year) references section
on delete cascade,
 foreign key(rollnumber) references student
on delete cascade,
);


create table takes
(rollnumber varchar(20),
 course_id varchar(20),
 semester varchar(20),
 year varchar(20),
 primary key (rollnumber,course_id,semester,year),
 foreign key (rollnumber) references student
  on delete cascade,
 foreign key (course_id,semester,year) references section
  on delete cascade
);

create table test
(course_id varchar(20),
 semester varchar(20),
 year varchar(20),
 test_id varchar(20),
 name varchar(20),
 num_ques int,
 test_date varchar(20),
 primary key (course_id,semester,year,test_id),
 foreign key (course_id,semester,year) references section
  on delete cascade
);

create table appears
(course_id varchar(20),
 semester varchar(20),
 year varchar(20),
 test_id varchar(20),
 rollnumber varchar(20),
 primary key (course_id,semester,year,test_id,rollnumber),
 foreign key (course_id,semester,year,test_id) references test
  on delete cascade,
 foreign key (rollnumber) references student
  on delete cascade
);

create table question
(course_id varchar(20),
 semester varchar(20),
 year varchar(20),
 test_id varchar(20),
 index int,
```

```
 m_marks numeric(5,2),
 q_text varchar(5000),
 space int,
 model_ans varchar(10000),
 primary key (course_id,semester,year,test_id,index),
 foreign key (course_id,semester,year,test_id) references test
  on delete cascade
);

create table ans
(course_id varchar(20),
 semester varchar(20),
 year varchar(20),
 test_id varchar(20),
 rollnumber varchar(20), # Check in appears
 index int, # Should be less than num_ques
 stud_ans blob(10MB),
 marks_obt numeric(5,2), # Should be less than m_marks
 grader varchar(20),
 comments varchar(1000),
 primary key (course_id,semester,year,test_id,index,rollnumber),
 foreign key (course_id,semester,year,test_id,index) references question
  on delete cascade,
 foreign key (rollnumber) references appears
  on delete cascade
);
```

8 Java Servlets

For proper segregation of functionality, we develop most of the servlets for Android backend and web backend separately. Each servlet takes some parameters and returns the result (after executing the query and updates) as a JSON object.

8.1 Common

1. **LoginServlet** - takes in *userid* and *password* and returns whether authentication succeeded or failed for that user
2. **LogoutServlet** - invalidates the current session, thereby logging out the current user

8.2 Android backend

Most of the servlets below are used by both TAs and students, and the functionality depends on whether it was invoked by a TA or a student.

1. **AllCourses** - For students, it returns current and past courses separately (by using the *rollno* of the logged in user); for TAs, it returns a similar list of the courses he/she is a TA of.
2. **AllExams** - For a course specified by *courseid*, *semester*, and *year*, it returns all the exams conducted in that course till date. Same functionality for TA, except that it returns only those courses for which he/she is/was a TA.
3. **ExamDetails** - For a student, it returns the marks obtained in the exam (if graded) along with all the questions asked in the exam and marks obtained in each question out of the total marks of that question (only if question has been graded). For a TA it returns the list of questions that he has been assigned for correction in the respective exam.
4. **AllAnswers** - This servlet is only for TAs for a question in an exam which the TA is supposed to correct, and it returns the list of answers that the TA is supposed to correct for that question.

5. **AnswerDetails** - This is also a TA-exclusive servlet — it returns an answersheet for the TA to grade

6. **Grade** - Servlet used by TAs to assign/update the marks of an answer, and add comments for an answer.

7. **QuestionDetails** - For a question that a student answered in an exam, it returns the marks that he obtained in the question (if graded), his answersheet, class statistics, and comments by examiner.

8.3 Web backend

All of the servlets below are used for the interface for instructors. These servlets return JSON objects to GET/POST requests which will be used further by the web application.

1. **GetSemCourse** - For an instructor and a semester, it outputs the courses undertaken by that instructor in that semester

2. **GetPastSemList** - For a given instructor, this servlet will return the past semesters in which the instructor took atleast one course

3. **GetCourseMetaData** - For given *course_id*, *semester* and *year*, it will return all metadata of a given instance of course i.e. section.

4. **AddTA** - Given TA's *rollnumber*, *course_id*, *semester*, *year*, this servlet will assign that TA to the specified section of a course.

5. **MakeTest** - This servlet will be called with all the data related to test (all sets of questions, marks and pages alloted per question, and any other data needed). After entering the test in the database, it will return the status of changes made.

6. **OldTestData** - This servlet will help the instructor look at old tests taken as part of the section he/she is teaching.