A Scalable and Flexible Pedagogy Helper Application

A thesis submitted in fulfillment of the requirements

for the degree of Master of Technology

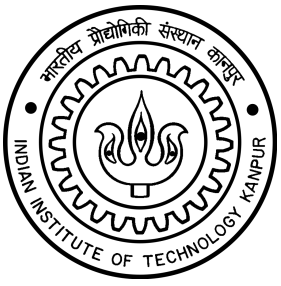
by

Neeraj Kumar

17111055

under the guidance of

Purushottam Kar



to the

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

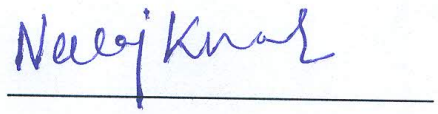
May 2019

Page intentionally left blank

Statement of Thesis Preparation

I, **Neeraj Kumar**, declare that this thesis titled, **"A Scalable and Flexible Pedagogy Helper Application"**, hereby submitted in partial fulfillment of the requirements for the degree of **Master of Technology** and the work contained herein are my own. I further confirm that:

1. The "Thesis Guide" was referred to for preparing the thesis.
2. Specifications regarding thesis format have been closely followed.
3. The contents of the thesis have been organized based on the guidelines.
4. The thesis has been prepared without resorting to plagiarism.
5. All sources used have been cited appropriately.
6. The thesis has not been submitted elsewhere for a degree.



Name: Neeraj Kumar

Roll No.: 17111055

Department of Computer Science and Engineering

May 2019

Page intentionally left blank

Certificate

It is certified that the work contained in the thesis titled **"A Scalable and Flexible Pedagogy Helper Application"** has been carried out under my supervision by **Neeraj Kumar** and that this work has not been submitted elsewhere for a degree.

28 MAY 2019

Purushottam Kar

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

Kanpur, 208016.

May 2019

v

Page intentionally left blank

ABSTRACT

Name of student: **Neeraj Kumar** Roll no: **17111055**

Degree for which submitted: **Master of Technology**

Department: **Department of Computer Science and Engineering**

Thesis title: **A Scalable and Flexible Pedagogy Helper Application**

Name of Thesis Supervisor: **Purushottam Kar**

Month and year of thesis submission: **May 2019**

The task of offering an educational course, whether in the traditional classroom format, or the increasingly popular MOOC format, has student evaluation and feedback as an integral part of the process. Increasing class strengths have made this process time consuming as well as labor intensive. In this work we present SphinX, a scalable and flexible online system that aids the process of managing and executing assignment and examination grading in huge courses. SphinX seeks to streamline and make less tedious, the process of grading by humans but also supports machine grading.

Although there are several existing (commercial) applications with similar goals such as Gradescope, Google Classroom, INGInious, GradeIt, etc, all these systems have one or more drawbacks such as not supporting course organization into sections, tutors etc, requiring users to surrender student submission and performance data to be stored on offshore servers, demanding a heavy (per-student) subscription fee, and being closed source which inhibits inspection of the shortcomings of the system.

SphinX distinguishes itself in (simultaneously) being extremely flexible to suit the needs of various institutions and courses – allowing instructors to seamlessly define new roles in a course which makes enabling constructs such as peer grading an effortless task, being privacy-aware and allowing the end user to choose not to part with student data, being security aware and using best practices to secure the system against attack and in/ex-filtration, and being developed and released in the free-and-open source software model.

Page intentionally left blank

Acknowledgments

It has been a great pleasure working with the professors, staff, and students at IIT Kanpur. I express my sincere gratitude to my thesis adviser, Dr. Purushottam Kar, for his continuous and extremely valuable guidance. I am thankful for his consistent enthusiasm, motivation, and support. Whenever I faced any problems, he was always there to help and support.

I gratefully acknowledge the contributions of Prof. T. V. Prabhakar and Prof. Amey Karkare for their support in taking the architectural decisions of SphinX and sharing their experiences with mooKIT and Prutor. I want to thank Revathy KT for sharing her valuable knowledge of mooKIT. I would like to thank Umair Z Ahmed and Saurabh Srivastava for providing valuable insights into various problems. I would like to thank Shailesh Vishweshwar Nandkule for his contribution to UI development. I would like to thank Amit Chandak for his support in designing the architecture of the application.

I am thankful to my parents for their love and support. I am forever in debt of my two brothers for continuously supporting me at every point of my life.

Neeraj Kumar

Contents

List of Figures

List of Tables

Page intentionally left blank

Chapter 1

Introduction

With an increase in demand for educational services, class strengths have risen significantly in institutions of higher education across the country over the past few years. Scaling up quality education services in course offerings to meet this increased demand is a major challenge for our nation. Not only do we need to train the next generation of students in futuristic technologies, but given the rapidly changing face of technology in several sectors, we also need to offer continuing education or re-education services to existing professionals as much as possible.

This has resulted in several MOOC-based technologies (Massive Open Online Course) being developed for delivering course content to a large number of students residing in disparate geographic regions through videos and other interactive content. In order to be successful, it is essential that these courses offer timely and proper feedback to the students on their performance.

Although consistent and uniform student evaluation is important in any course offering, it becomes even more critical in MOOCs where there is often scant interaction between the instructor and the student. However, evaluation is also a time-consuming, laborious, and tedious task, more so in MOOC-like settings. It is thus, no surprise that MOOC offerings rarely go beyond basic evaluation techniques of MCQ-style quizzes or short answer questions, possibly due to the challenge of very large-scale evaluation and grading.

Consequently, several applications have indeed come up which seek to address the specific challenge of grading and evaluation. Examples include Gradescope [15], Google Classroom [4], Blackboard [1] and mooKIT [6]. We will review existing applications in detail in Chapter 2. All these applications are aimed at either automated evaluation or assisted evaluation i.e. where the application does not perform evaluation itself but merely makes it convenient for a human to perform the evaluation with reduced effort, and some of these, such as mooKIT, are indeed natively integrated into MOOCs. However, these applications do present newer challenges of their own.

1. Several of these applications are designed with a very narrow view of how courses are orga-

nized and conducted and do not recognize the fact that various participants in a course may play various roles at various points of time. A good example is that of peer-grading [12, 17] in which, participants who are otherwise students of the course, temporarily turn into graders for one-or-two submissions made by their classmates.

2. Several of these applications are offered as cloud-based services and hence, in order to cover the hosting costs, end-up charging what is often a high price per enrolled student for using these services. This is frequently hard for instructors to afford, especially in departments that are not so well-endowed monetarily.

3. Another drawback of these online hosted services is that student submission data as well as student performance data, which is often regarded as sensitive and private, has to be surrendered to the service providers that often themselves us third-party service providers (such as AWS, Azure, Google Cloud etc) which could be hosting the data at arbitrary geographical locations. This may make these services unusable should our nation adopt strict data privacy legislation that prohibits student data from being transferred outside the country.

To remedy several of the above concerns, we present SphinX, a scalable, secure, and highly customizable platform that addresses the challenges of large-scale grading and management of courses. Below we enumerate some salient benefits of SphinX over existing systems.

1. SphinX supports multiple offerings of multiple courses, each with several hundreds of students, in a single installation without adversely affecting the application's scalability.

2. SphinX allows significant flexibility to the instructor in designing new roles for every course offering. An instructor may of course use existing roles, such as "student", "grader", etc but may also create new roles with each role being allowed to access an arbitrary subset of the system's non-admin actions.

3. The above flexibility allows SphinX to natively support procedures like peer grading without having to put in any additional architectural support. To enable peer grading, an instructor simply needs to create a new role, that of a student who may also perform grading. Support for other notions such as course auditors, multiple instructors also becomes simple given this flexibility.

4. An individual's role in one (offering of the) a course on SphinX in no way affects their role in other courses. Thus, the same person may be an instructor in one course, a TA in another course, and a student in yet another course with all three courses being simultaneously hosted on SphinX.

5. SphinX directly addresses handles issues of data privacy by providing two different modes of use, namely *local mode* and *server mode*. In the local mode, an instructor may install SphinX on their local machines and use it, thereby not having to share any student data. In server mode, a department or larger entity may create a SphinX installation on a more resourceful platform such as a large server or local cloud and use it to offer a large number of courses.

6. Although we present a helpful and easy-to-use front end, the SphinX back-end actually exposes a largely RESTful [10] API that can be used by anyone to build their own front-end for mobile and other device specific front-ends.

7. SphinX is intended to be released in a free and open source (FOSS) model with no charges begin levied for downloads of the installation sources. However, SphinX may yet be monetized by setting it up on a massive server and offering SphinX as a (paid) service.

In the rest of the chapters we will review related works, present the system architecture details of SphinX as well as describe the back-end API it exposes, and enumerate a few helpful apps we have designed to be used with the platform.

Chapter 2

Related Works

As noted in the previous chapter, the advent of MOOCs (Massive Open Online Courses) has led to the requirement of online student evaluation systems. However, several of these systems support only MCQ (multiple choice question) or else SWA (single word answer) questions, presumably due to the ease of automating grading work for such questions. However, restricting oneself to such question types is neither desirable, nor practiced in courses run at institutions of higher education, many of whom currently host several hundreds of students in every single offering and thus, face similar challenges as MOOCs.

In India, NPTEL[1] [7] uses either MCQ or SWA questions, or else conducts offline exams at various centers in the country. The student evaluation techniques on Coursera[2] [2], globally dominant platform for MOOCs, are also limited to MCQ or SWA. However, we note that several grading platforms have indeed been built to ease and better regulate the process of evaluating student submissions in assignments and examinations in (non-MOOC) courses.

Examples of such grading platforms include Gradescope [3] [15], Crowdmark[4] [3], Prutor[5] [9], INGInious[6], GradeIt [14], and Grade-It[7].

Gradescope [15] is a platform that was developed by erstwhile teaching assistants and tutors at the University of California at Berkley and has gained popularity in the recent years with courses at several institutions, including IIT Kanpur (ESC101, CS771) having used it in large course offerings. Although Gradescope started out as freeware, right now it is freely offered to instructors for only two courses per offering, with a reasonable charge per student later on. The use of additional services such as automated grading incur charges per student. Gradescope is also closed source

[1] https://nptel.ac.in/
[2] https://www.coursera.org/
[3] https://www.gradescope.com/
[4] https://crowdmark.com/
[5] https://prutor.cse.iitk.ac.in/
[6] https://inginious.org/
[7] https://homes.cs.washington.edu/ stepp/gradeit/

and requires instructors to allow student data to be stored on (possibly) offshore locations due to Gradescope's use of Amazon AWS™services.

Crowdmark [3] is another closed source collaborative online grading and analytics platform. It provides a basic workflow without automating any grading tasks. The use of this service also requires a subscription fee.

INGInious is a free and open-source software (FOSS) distributed under the AGPL license. INGInious is integrated with edx[8] [5] to perform programming assessment and provide feedback to the students.

Prutor [9] is a free and open-source platform developed at IIT Kanpur itself that is being used at several Indian universities such as IIT Bombay, IISc, etc. Prutor specilizes for programming courses and supports a variety of languages such as C, C++ and Python. The platform supports online submission and online grading by humans and has recently been augmented with string comparison-based automatic grading. GradeIt [14] is a tool built on top of Prutor that provides automatic grading and Feedback tool using Program Repair for Introductory Programming Courses.

Grade-It (not to be confused with GradeIt) is a system similar to Prutor developed at the University of Washington that handles courses and grading of students' programming assignments.

As discussed in Chapter 1, apart from systems such as Prutor and Grade-It that are specific to programming courses, existing systems are either closed source and require a (heavy) subscription for their use (Gradescope, INGInious) or they do not fully cater to the various organizational and logistical demands of non-MOOC courses being offered at various institutions (mooKIT, moodle, Google Classroom), or they require surrendering student submissions and performance data to a third party. As we also pointed out in that chapter, SphinX remedies several of these drawbacks of existing systems.

[8] https://www.edx.org/

Chapter 3

Entity Description

In this chapter we describe the various entitites that SphinX recognizes within its system. The chapter will also give details of the database schema used by the SphinX. Below we first give a high level use case diagram. SphinX in principle allows all users to perform all actions but create courses and instructors. However, in a real use-case, most users would have roles that would restrict the set of allowed actions to a subset of all possible actions.

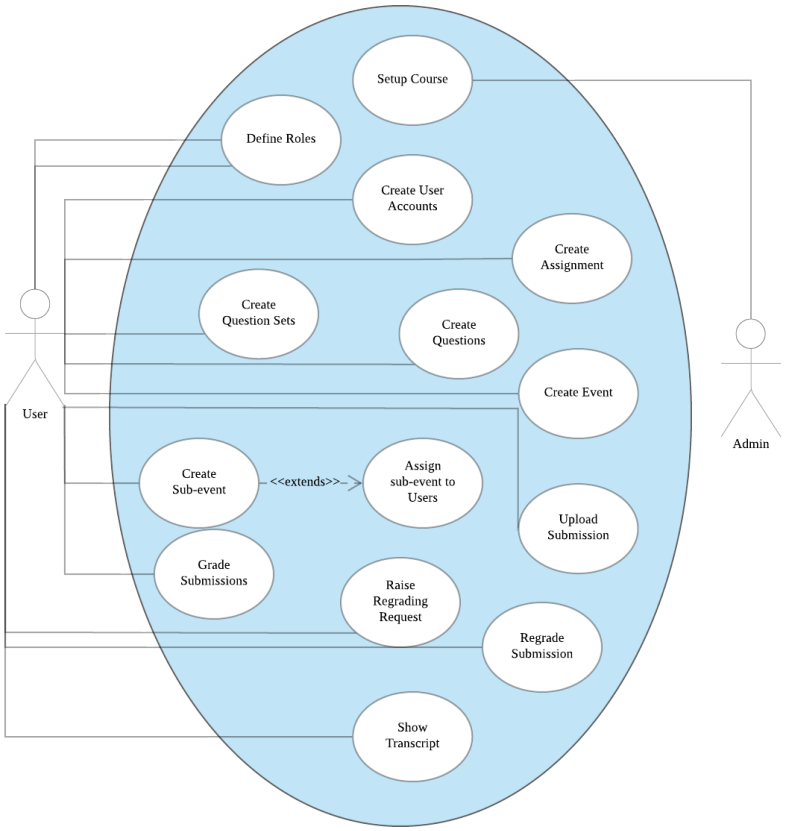


Figure 3.1: High level Use Case Diagram

3.1 Conceptual Framework

We now describe the terms corresponding to various entities in the system. These may correspond to humans interacting with the system, as well as information that they generate in the due course of these interactions.

1. **Appliation/Platform**: SphinX running instance.

2. **User**: A user is an entity who has been assigned a role (a subset of actions) in the application. A single user account is maintained at the global scope and shared with all courses in which the user has enrolled.

3. **Course**: The course is at the heart of application containing activities and resources. Course setup can be done only by an admin. An instructor can only be added by an admin into a course. The instructor can then take any action in the scope of the course such as add students/TAs, given them access privileges, set up assignments, deadlines etc.

4. **Actions**: These are another name we use for a user's entry point into the application. Actions are mapped to endpoints of the application. API End-points which correspond to unit actions, are defined in following chapter. Examples of actions include adding a user into a course, creating an assignment, setting a grading deadline for that assignment, submitting an assignment etc.

5. **Section**: Students may be grouped into sections and assigned a separate set of resources and activities in the course. This helps organize large courses like ESC101 and CS771. Several sections can be created and each assigned separate tutors.

6. **Role**: A role corresponds to an arbitrary subset of actions. Instructors may create new roles and assign them a name. To help the instructor create these roles, we provide a default set of *meta-roles*. The instructor may mix and match these meta roles to create new roles. Examples of roles include student, teaching assistant, tutors. An example of a meta role can be *can-grade* which gives a user all privileges required to perform grading and regrading on submissions assigned to them. Creating a new role by clubbing the role student with the meta role can-grade can enable peer grading.

7. **Topic**: A set of keywords used in grouping questions. These keywords can be a subject name or a topic name. Topics can be hierarchical. A question can be tagged with multiple topic tags.

8. **Assignment**: This is a generic term used to describe a set of questions and accompanying rubric items.

Subevent Types	
Type	**Description**
QVIEW	During what period is the **question paper visible**?
SUPLOAD	During what period are students allowed to make submissions?
SVIEW	During what period is a **student's** own submission visible to them in a read-only fashion?
GUPLOAD	During what period are **graders** allowed to **submit/modify** grades?
GVIEW	During what period are **grades/regrades** visible to **students and graders** in a read-only fashion?
AVIEW	During what period are the **gold/instructor's** solutions visible to students?
RGREQ	During what period are students allowed to make **regrading requests**
RGUPLOAD	During what period are **regraders** allowed to submit regrades?

9. **Question Set**: An assignment can have several question sets. This allows examinations to be conducted in several phases and to maintain separate (and possibly multiple) question sets per phase.

10. **Question**: A logical unit at which student submission is requested and marks assigned. Questions can be hierarchically organized as well. A question can be an actual question if it is at the leaf level in the hierarchy. Questions may be tagged with topics.

11. **Rubric**: These are advanced grading evaluation criteria. A set of rubrics can be created for evaluation of each question. The rubric defines what is expected in the solutions to get a particular grade. This promotes uniformity of grading among graders and also reduces evaluation time.

12. **Question Options**: An option for question in multiple-choice questions (MCQs).

13. **Event**: An assignment (which includes take-home assignments as well as examinations and quizzes) is released to students by creating an event. This enables an instructor to group various subevents such as submission, grading, and regrading associated with that event. There can also be an event that is not associated with any assignment and are called external events.

14. **Sub-Event**: An event comprises of a set of predefined type of activities called sub-events. Various types of sub-event are found in the sub-event table.

15. **Upload/Submission**: Students work submitted on the platform.

16. **Submission Group**: Students can submit an assignment in a group. A group can be formed using submission group scheme and policies.

17. **Submission Group Scheme**

Submission Group schemes[1]

Submission Group Scheme(SGS)	
Schemes	**Description**
INOS	students submit **individually** but **choose** their set on their own
INFS	students submit **individually** according to the set assigned to them (for exams etc)
FGFS	**assign groups** and **fix** the set that they have to submit
FGOS	**assign groups** but allows them to choose their question set
OGOS$max	**allow students** to form groups on their own upto **max group size**, each group **chooses** a question set
OGFS$max	**assign a set** to each student and only **allow groups** to be formed among students who have received the same set

Table 3.1: Submission Group Schemes

Submission Group Scheme Policy	
Policy	**Description**
FG/IN Policy	A user can perform initial upload, as well as update upload for the submission corresponding to their **own group** only. If there has already been an upload, the system returns that file upon the GET request. If there is already an upload, throw a **NODUP** error if the user tries to make a **POST** request. Once an upload has been created, it can only be updated, never deleted.
OG Policy:	If a user is not in a group currently they can either (a) Join an existing group by obtaining the submission_group_id of that group from one of the group members (the submission_group_id is geneated as a cryptographically secure number). (b) Create a new submission group and then make an upload. They can tag other users not currently assigned to any group to this submission group. If a user is already in a group currently, they can i. Update the upload of that group ii. Join another existing group by obtaining that group's submission_group_id and switching their group. This will be allowed only if that group strength has not breached the maximum limit. iii. Create a new group and then make an upload. They can tag other users not currently assigned to any group as well.

Table 3.2: Submission Group Scheme Policy

18. **Responses** Every submission/upload is broken down and mapped to an actual question. This is called a response.

[1]
(a) INOS: individual open set
(b) INFS: individual fixed set
(c) INRS: individual random set
(d) OGOS: open group open set
(e) FGOS, fixed group open set
(f) FGFS: fixed group fixed set
(g) OGFS: open group fixed set

19. **Grading** Rubrics are attached to responses by graders based on the subevent. We will discuss GUPLOAD subevent in the next section.

20. **Grading Duty** A grading task assigned to a grader using GUPLOAD Subevent.

21. **Grading Duty Schemes**

    Various grading duty schemes [2] are defined in the table.

Grading Scheme	
Schemes	**Description**
MQS	For every (question_set,question) tuple, specify >=1 graders, each submission graded by just one grader
MQR$rep	For every set,question, specify >= rep graders, each submission graded independently by rep graders
Note	Only questions with is_actual_question should be assigned graders. offer the instructor a CSV file containing user_id, name, roll number of all TAs/tutors

Table 3.3: Custom Grading Scheme

Grading Scheme	
Schemes	**Description**
RQS	MQS but with random assignment with one grader assigned as few questions as few as possible
RQR$rep	MQR but with random assignment with one grader assigned as few questions as few as possible
RSS	all questions corresponding to each upload is entirely graded by one grader
RSR$rep	all questions corresponding to each upload is entirely graded independently by rep graders
Note	In these schemes, the instructor does not need to specify any tuples

Table 3.4: Random Grading Scheme

22. **Regrading Request**: When students are not satisfied with the feedback, the system provides flexibility to request for a regrade.

23. **Regrading** The action taken in response to regrading request. The re-grader can change the rubrics assigned to the solution and provide feedback.

24. **Regrading Duty Scheme**: Offer the instructor a CSV file containing user_id, name, roll number of all TAs/tutors. various regrading duty scheme [3] are defined in the table

[2]

(a) RQS random question single
(b) *RSS random submission single
(c) *MQR -Manual question Repetition

[3]

(a) SOR: Same or Random
(b) RAN: Random
(c) QRN: Random for each Question

Regrading Scheme	
Schemes	**Description**
SOR	The same person who graded the response will regrade it. If there were multiple graders of a question, a random one will be chosen to regrade it

Table 3.5: Same Regrading Scheme

Regrading Scheme	
Schemes	**Description**
RAN	From a list of regraders, choose a random one for every regrading request
QRN	Same as RAN but specify the list separately for each question

Table 3.6: Custom Regrading Scheme

3.2 ER Model

Columns like id(auto increment key), updated_at, updated_by, created_at, created_by and deleted_at is common in all the tables. Deleted_at column is used in soft delete, and other columns are used to store application monitoring data when running in production.

3.2.1 Global Schema View

User		
Field Name	**Data type**	**Description**
username	CharField	
roll_no	CharField	
first_name	CharField	
last_name	CharField	
email	EmailField	unique
department	CharField	
program	CharField	BT, MT, MS, DD, PhD etc
password	CharField	
last_login	DateTimeField	
last_login_ip	CharField	
is_active	BooleanField	
is_enabled	BooleanField	
is_logged_in	BooleanField	
password_reset_token	CharField	
session_id	CharField	stores the current session id
courses	ManyToManyField	many to many relation to course id

Course		
Field Name	**Data type**	**Description**
name	CharField	
title	CharField	
description	CharField	
semester	CharField	
year	IntegerField	
department	CharField	
is_active	BooleanField	is course active at current moment
image_directory	CharField	use to store course files

UserHasCourse		
Field Name	**Data type**	**Description**
user	ForeignKey user id	
course	ForeignKey course id	
enrollment_id	BigIntegerField	
enrollment_role_id	BigIntegerField	
enrollment_action_list	CharField	
enrollment_section_list	CharField	list of user enrolled sections

Action		
Field Name	**Data type**	**Description**
app	CharField	
url	CharField	
method	CharField	HTTP methods

3.2.2 Course Schema View

Global Logs		
Field Name	**Data type**	**Description**
is_logged_in	BooleanField	Does the session making this request correspond to a logged in account?
user_id	Foreign key to user	Which user made this request?
ip	CharField	user ip address
app	CharField	Which app handled this request? Note, only login/account apps would handle requests that get logged in the global logs.
url	IntegerField	Which URL was targeted in this request?
method	CharField	Which method (GET, PUT, POST) was used?
meta	CharField	
file_path	CharField	Link to any files uploaded as a part of the reqeust

Enrollment		
Field Name	**Data type**	**Description**
user	ForeignKey user	
role	ForeignKey Role	
sections	ManyToManyField Section	
subevents	ManyToManyField	

Role		
Field Name	**Data type**	**Description**
name	CharField	
action_list	CharField	

Topic		
Field Name	**Data type**	**Description**
name	CharField	
super_topic	ForeignKey Topic self referencing foreign key	
description	CharField	

Section		
Field Name	**Data type**	**Description**
name	CharField	should be unique in course

EnrollmentHasSections		
Field Name	**Data type**	**Description**
enrollment	ForeignKey Enrollment	
section	ForeignKey Section	

3.2.3 Assignment Schema View

Course Log		
Field Name	**Data type**	**Description**
is_logged_in	BooleanField	
user_id	CharField	
ip	CharField	
app	CharField	Which app handled this request?
url	CharField	
method	CharField	
file_path	CharField	Link to any files uploaded as a part of the reqeust?
flag_id	CharField	SUCC or one of the failure flags
message_id	CharField	

Assignment		
Field Name	**Data type**	**Description**
name	CharField	
comments	CharField	

Question set		
Field Name	**Data type**	**Description**
name	CharField	
assignment	ForeignKey	
question_file_path	FileField	
supplementary_file_path	FileField	
solution_file_path	FileField	
total_marks	IntegerField	
original_question_file_name	CharField	
original_supplementary _file_name	CharField	
original_solution_file_name	CharField	user's file name
original_solution_file_name	CharField	user's file name
name_coords	CharField	
roll_coords	CharField	
is_class_avg_dirty	CharField	used to save computation

3.2.4 Event Schema View

Question		
Field Name	**Data type**	**Description**
subpart_no	CharField	This should be a non-negative integer i.e. 0, 1, 2 etc
title	CharField	
type	CharField	Is it MCQ, short answer, T/F etc
file_page	IntegerField	On which page of the question paper does this appear
file_cords	CharField	On that file, on that page, where does this question appear
text	CharField	The text of this question in case of live exam.
difficulty_level	CharField	
marks	FloatField	
solution_list	CharField	In case of questions where mutliple solutions may be correct, store all correct solutions in a serialized format
is_autograded	BooleanField	A question may be UNGRADED (the question was created just for sake of structuring) MANUAL (a human user will grade this question) AUTO (autograde this question)
parent	ForeignKey Question	Set as blank for top level questions
grading_duty_scheme	CharField	
is_actual_question	BooleanField	
topics	ManyToManyField Topic	
question_set	ForeignKey	Set as blank for top level questions

QuestionOptions		
Field Name	**Data type**	**Description**
label	CharField	
text	CharField	
is_correct	BooleanField	
image_path	CharField	if options has image, store its path
image_size	IntegerField	
question	ForeignKey	

Rubric		
Field Name	**Data type**	**Description**
question	ForeignKey	
text	CharField	
marks	IntegerField	

QuestionHasTopics		
Field Name	**Data type**	**Description**
question	ForeignKey	
topic	ForeignKey	

Event		
Field Name	**Data type**	**Description**
name	CharField	
assignment	ForeignKey	
grade_aggregation_method	CharField	
is_external	BooleanField	

Subevent		
Field Name	**Data type**	**Description**
name	CharField	
event	ForeignKey	
gen_subevent	ForeignKey Subevent	GUPLOAD subevents are generated by an SUPLOAD subevent. RGREQ subevents are generated by a GUPLOAD subevent. RGUPLOAD subevents are generated by an RGREQ subevent.
type	CharField	(course_open removed as an event type as it is handled using is_active flag in courses) 1. QVIEW: view questions 2. SUPLOAD: make submissions 3. SVIEW: view own submissions as read-only 4. GUPLOAD: grade submissions 5. GVIEW: view grades as read-only 6. AVIEW: view gold solutions 7. RGREQ: make regrading requests 8. RGUPLOAD: service regrading requests
start_time	DateTimeField	
end_time	DateTimeField	
display_end_time	DateTimeField	Instructor may want to display 5PM as end time but have actual end time as 5:05PM
allow_late_ending	BooleanField	
late_end_time	DateTimeField	
display_late_end_time	DateTimeField	
is_blocking	BooleanField	Is this an exam? If so, only the assignment related to this event will be visible during that period in this course.
params	CharField	subevent-specific params. See subevents in entity description for details

UserHasSubevent		
Field Name	**Data type**	**Description**
subevent	ForeignKey	
enrollment	ForeignKey	

Upload		
Field Name	**Data type**	**Description**
file_path	CharField	
file_size	IntegerField	
is_successful_upload	BooleanField	
uploader	ForeignKey	
uploader_at	DateTimeField	
uploader_ip	CharField	
is_bulk_upload	BooleanField	
is_paginated	BooleanField	

Submission Group		
Field Name	**Data type**	**Description**
subevent	ForeignKey	This must be an SUPLOAD event
access_code_gold	CharField	restrict students to access questions and not allow to submit
access_code_submitted	BooleanField	
is_late_submission	BooleanField	Did the student(s) breach the submission deadline and submit into the late submission period
choosen_question_set	ForeignKey	In case student is free to choose their set, store which question set was selected by the student. In case this is a fixed set assignment, this must be the question set id assigned by the instructor. In general this must be a valid question set of the assignment which is linked to the event corresponding to this subevent.
upload_id_main	ForeignKey	
upload_id_supp	ForeignKey	
enrollments	ManyToManyField Enrollment	

SubmissionGroupHasUser		
Field Name	**Data type**	**Description**
submission_group	ForeignKey	
enrollment	ForeignKey	

SubmissionResponse		
Field Name	**Data type**	**Description**
submission_group	ForeignKey	
question	ForeignKey	
upload_page_no	IntegerField	user will mark the page no of this answer
upload_coords	CharField	coordinate in the page where answer is written.
response_text	CharField	
upload	ForeignKey	

GradingDuty		
Field Name	**Data type**	**Description**
subevent	ForeignKey	This must be a GUPLOAD or else RGUPLOAD subevent
response	ForeignKey	
marks_adjustment	IntegerField	
is_regrading	BooleanField	Is this is original grading duty or a regrading duty? Regraders get to see previous grades allotted by other graders whereas original graders do not get to see grades given by other graders.
grader_comment	CharField	
student_comment	CharField	
is_completed	BooleanField	
aggregate_marks	IntegerField	sum of all marks obtained through the rubrics plus any marks adjustment
is_late_grading	BooleanField	Did the (re)grader breach the (re)grading deadline and instead (re)grade into the late (re)grading period
grader	ForeignKey	This field stores whether the aggregate marks need to be recomputed i.e. is it stale. This bit will be set to 1 whenever a rubric is added or removed from this grading duty or else if marks adjustment is done and will be set to 0 whenever recomputation of aggregate marks is done.

GradingDuty		
Field Name	**Data type**	**Description**
grading_duty	ForeignKey	foreign key reference to grading duty primary key
rubric	ForeignKey	foreign key reference to rubric table primary key

Chapter 4

Process Description

In this chapter we describe the back-end processes of SphinX as well as the API that its back-end exposes. We first outline below a schematic describing the way in which API requests eventually lead to accesses from the persistent stores (database and disk) as well as the distributed cache.

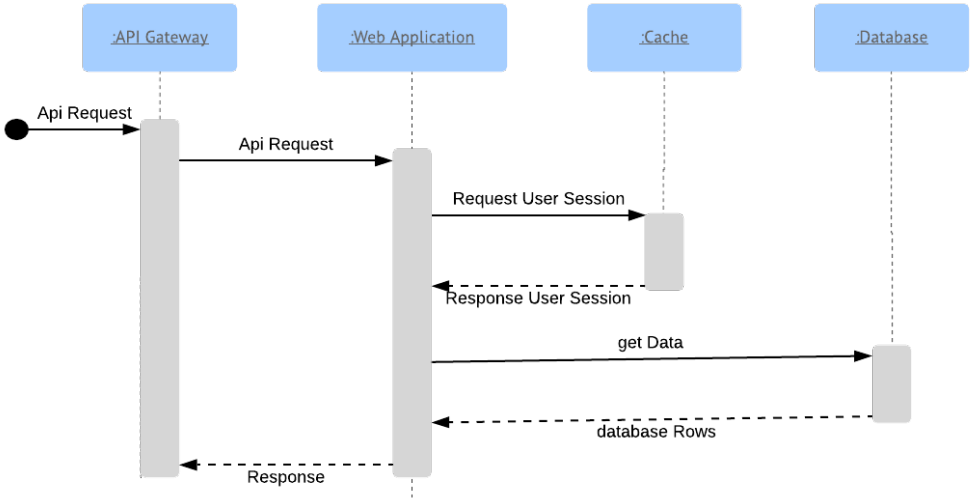


Figure 4.1: Sequence Diagram for SphinX database Access

SphinX processes are distributed among several *Managers*, each of which handle a broad functionality of the system as well as all API calls that relate to that system. Below we described all managers in detail.

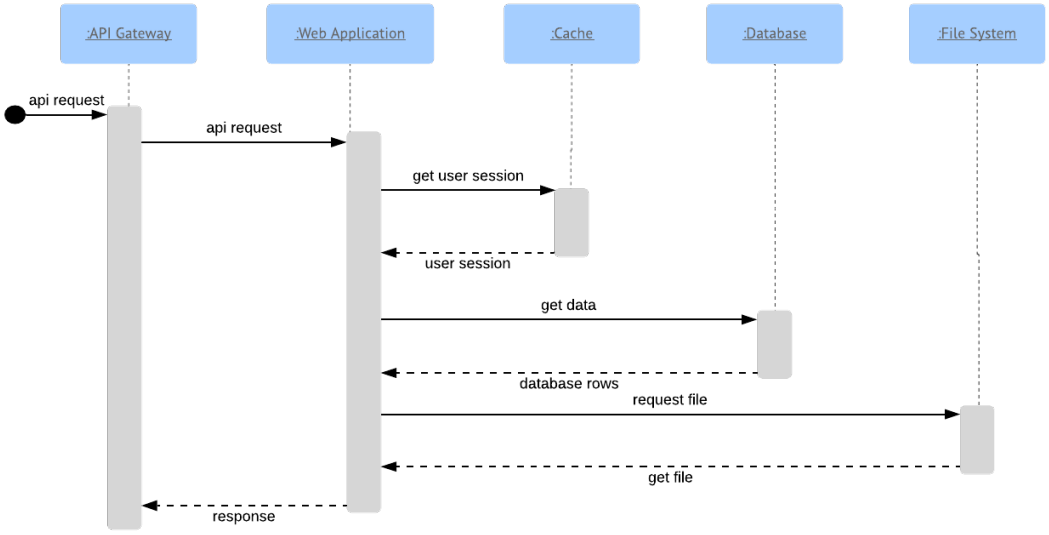


Figure 4.2: Sequence Diagram for SphinX File Access

4.1 Authentication Manager

This manager is responsible for authentication and user account management.

API	
URL	/
HTTP Method	**GET**
Pre-Validations	If this session is already logged in, redirect to /courses/ and finish
Action	Show a splash screen with a nice logo etc and a login form
Return Data	**None**

API	
URL	/login/
Method	**POST** (Note: This is a **POST** request so that a CSRF token is required. If this were a **GET** request then an attacker could perform CSRF to attempt a login into the attacker's account and in the process, log this person out of their account. Thus, the attacker could **GET** the victim to submit an assignment on behalf of the attacker's account if the user is not careful.)
Pre-Validations	If this session is already logged in, redirect to /courses/ and finish.(This can be done in API gateway or middleware). If user.is_active is not true return **NOACTIVE** error.
Action	Authenticate username and password. If username does not exist or if it has been soft deleted, return a **NOUSER** error and finish. If the password is wrong, return a **NOKEY** error and finish. If password is correct, set users.is_logged_in = 1, If this user is already logged in some other session, terminate that session by clearing the value stored in users.session_id And update the Memcached entry for that session. Also populate Memcached with any useful information with respect to this user. Now update values stored in users.session_id with the current session and also update the Memcached entry for the current session. If the user has anything stored in the users.password_reset_token Field, clear that value.
Return	**None**

API	
URL	/logout/
HTTP Method	**POST** (Note: this is a **POST** request so that a CSRF token is required. If this were a **GET** request then an attacker could perform CSRF to log this user out to annoy them)
Pre-Validations	**None**
Action	Clear the values stored in users.[is_logged_in, session_id] so that this session is no longer logged-in or associated with this user and then update Memcached for this session to reflect the change as well. We should flush the session along with the data from Memcached to prevent reuse of same session. It will help us in the cases when same browser can be used by multiple users in labs.
Return Data	**None**

API	
URL	/account/
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	From the user's own account, return users.[**username, roll_no, first_name, last_name, email, department, program, last_login, last_login_ip**]. Also return user_has_courses.[**course_id, course_name, enrollment_role, enrollment_sectionlist**]. In user_has_courses list we should only have courses if course.is_active is 1 and is not soft deleted and user_has_courses row should also be not soft deleted

API	
URL	/account/
HTTP Method	**PUT**
Pre-Validations	**None**
Action	A single form would be sending a whole bunch of fields to be updated. If these are the old values, do nothing, else update to the new values being sent. However, allow updates only to the fields users.[first_name, last_name, email, department, program]. Note that a user may manufacture a malicious **POST** request asking username/roll number to be changed as well - simply ignore the username field in that request and update the allowed fields mentioned above.
Return Data	**None**

API	
URL	/password/change/
HTTP Method	**POST**
Pre-Validations	match the two in**PUT** passwords, if they are not matching return **NOVAL** error
Action	the old password must have been entered correctly else return a **NOKEY** error and finish. Update password
Return Data	**None**

API	
URL	/password/reset/
HTTP Method	**GET**
Pre-Validations	**None**
Action	Show a page where the user must enter their username
Return Data	**None**

API	
URL	/password/reset/
HTTP Method	**POST**
Pre-Validations	**None**
Action	If the username entered by the user is incorrect (i.e. no such user exists), return a **NOUSER** error and finish else generate a new token and store it in **users.password_reset_token**, email the token and the users.id in the form of a URL. If the user is already loggedin with this or some other session, clear users.[**is_logged_in, session_id**] as well as update Memcached for all those sessions so that the user is no longer logged in any session.
Return Data	**None**

API	
URL	/password/reset/confirm/$userid-token
HTTP Method	**PUT**
Pre-Validations	**None**
Action	Verify the userid and the token. If either is wrong, return a **NOUSER** or **NOKEY** error, as appropriate. If both are correct, update users.password with the new (salted) password. If the user is already logged-in with this or some other session, clear users.[**is_logged_in, session_id**] as well as update Memcached for all those sessions so that the user is no longer logged in any session.
Return Data	**None**

4.2 Course Manager

This manager is responsible for several course-level processes such as managing course details, course enrollments, structuring the course into sections, creation and management of roles and topics.

API	
URL	/course/
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return user_has_courses.[**course_id, course_name, enrollment_role, enrollment_sectionlist**] for all courses where the user is enrolled.

API	
URL	/course/$courseid
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return courses.[**name, description, semester, year, department, is_active**].
Note	The frontend may want to show a link to pages (in case the user does have required permissions to access them) for enrollments, roles, topics, assignments, as well as a list of existing events, topics, roles, assignments.

API	
URL	/course/$courseid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	A single form would be sending a whole bunch of fields to be updated. If these are the old values, do nothing, else update to the new values being sent. However, allow updates only to the fields courses.[**name, description, semester, year, department, is_active**]. Dont allow updates to fields like **image_directory**. Note that a user may manufacture a malicious **POST** request asking image_directory to be changed as well - simply ignore the **image_directory** in that request and update the allowed fields mentioned above.
Return Data	**None**

API	
URL	/course/$courseid/sections
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return sections.[**id, name**] for all sections in this course

API	
URL	/course/$courseid/sections
HTTP Method	**POST**
Pre-Validations	If this section name already exists, return a **NODUP** error and finish.
Action	Given a section name, create a new section.
Return Data	sections.[**id, name**] of the newly created section so that the front-end may display it as well

API	
URL	/course/$courseid/section/$sectionid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	Given a new section name, change the section name. If the new section name already exists, return a **NODUP** error and finish.
Return Data	Return sections.[**id, name**] for the newly modified section in this course

API	
URL	/course/$courseid/section/$sectionid
HTTP Method	**DELETE**
Pre-Validations	If there is even one undeleted row in enrollment_has_sections which has the section being deleted, refuse deletion by throwing a **NODEL** error. Ask the instructor to first shift those enrollments to some other section and then delete this section.
Action	If no enrollment exists in this section, **soft-delete** this section.
Return Data	**None**

API	
URL	/course/$courseid/roles
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return roles.[**id, name, actionlist**] for all roles associated with this course

API	
URL	/course/$courseid/roles
HTTP Method	**POST**
Pre-Validations	**None**
Action	Given a role name and actionlist, create a new role. If this role name already exists, return a **NODUP** error and finish.
Return Data	roles.[**id, name, actionlist**] of the newly created role so that the front-end may display it as well
Comment	Instead of a so many actionlist, it may be better to develop a few sub roles e.g. “can create and modify assignments and events” to help instructor state set up roles
Scope	create template roles

API	
URL	/course/$courseid/role/$roleid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	Modify the name and action list of this role. If the new name of this role already exists in this course, return a **NODUP** error and finish.
Return Data	roles.[**id, name, actionlist**] of the newly updated role so that the front-end may display it as well

API	
URL	/course/$courseid/role/$roleid
HTTP Method	**DELETE**
Pre-Validations	If there is even one enrollment where **enrollments.role_id** is this role then deny this deletion by throwing a **NODEL** error. The instructor must set those enrollments a new role before deleting it.
Action	If there is no enrollment with this role then set **roles.is_deleted = 1** for this role
Return Data	**None**

API	
URL	/course/$courseid/topics
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return topics.[**name, super_topic_id, description**] of all topics associated with this course

API	
URL	/course/$courseid/topics
HTTP Method	**POST**
Pre-Validations	If this topic name already exists, return a **NODUP** error and finish.
Action	Given a topicname, description and **super_topic_id** (may be blank), create a new topic.
Return Data	topics.[**name, super_topic_id, description**] of the newly created topic so that the front-end may display this as well.

API	
URL	/course/$courseid/topic/$topicid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	If the new topic name already exists in this course, return a **NODUP** error and finish. If the super topic does not exist (or has been deleted), return a **NOVAL** error and finish. Otherwise modify the name, description, and super topic of this topic.
Return Data	topics.[**name, super_topic_id, description**] of the newly updated topic so that the front-end may display this as well.

API	
URL	/course/$courseid/topic/$topicid
HTTP Method	**DELETE**
Pre-Validations	**None**
Action	Set **topics.is_deleted = 1** for this topic. For all topics that have this topic as a **supertopic**, set **topics.super_topic_id** to be blank to indicate that they do not have a super topic anymore. Also **soft delete** all **question_has_topics** rows with this topic.
Return Data	**None**

API	
URL	/course/$courseid/enrollments
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return enrollments.[**user_id, role_id**] as well as **enrollment_has_sections.section_id and sections.name** of all enrollments in this course. May want to return **users.name and roles.name** as well for easy viewing.

API	
URL	/course/$courseid/enrollments
HTTP Method	**POST**
Pre-Validations	**None**
Action	Given a (list of in a CSV file) (**username, rolename, sectionlist**) tuple(s), create a new **enrollment** in this course. If this **username** has already been enrolled in this course before, throw a **NODUP** error and finish. If this **username** does not exist in the system, create a new user in the system as well(create a new row in 'user has course' with these values columns **enrollment_id, enrollment_section_list, enrollment_role, enrollment_action_list** and **send mail** to the user with**username and password**). If (any of) the rolename(s) supplied do(es) not exist for this course, return a **NOVAL** error. The section list must be a list of sections with each section containing only **alphanumeric** characters and two section names separated by a **pipe** — **symbol**. If this format is violated for any data point, or else if any of the sections mentioned in the list does not exist yet or has been deleted, return a **NOVAL** error and finish. If any of these users are logged in, update their Memcached entries (can **GET** the session_id from the users table) with the new roles they have been assigned. If there is even one row in the **CSV file** that is not correct in all respects, disregard the entire CSV file i.e. first take one pass to confirm that all rows in the CSV file make sense, and in the second pass, create all enrollments/accounts.
Return Data	enrollments.[**users_id,role_id**] as well as enrollment_has_sections.section_id and sections.name of the newly entered enrollments so that the frontend may display these as well.

API	
URL	/course/$courseid/enrollment/$enrollmentid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	Modify the role and the sectionlist of this enrollment. If this role does not exist, return a **NOVAL** error and finish. If the sectionlist is in an invalid format or contains a section that does not exist yet or has been deleted, return a **NOVAL** error and finish. If this user logged in, update their Memcached entries (can **GET** the session_id from the users table) with the new roles they have been assigned.
Return Data	enrollments.[**users_id, role_id**] as well as **enrollment_has_sections.section_id and sections.name** of the newly entered enrollments so that the front-end may display these as well.

API	
URL	/course/$courseid/enrollment/$enrollmentid
HTTP Method	**DELETE**
Pre-Validations	**None**
Action	Set **enrollments.is_deleted** = 1 for this enrollment. If this user is logged in, update their Memcached entries (can **GET** the **session_id** from the users table) and revoke permissions that they were assigned in this course earlier.
Return Data	**None**
Note	we are not going to do cascaded **soft delete** of all actions performed by this enrollment so far since the enrollment deleted may be a student who has previously submitted assignments in a group but other group members are still around so we would not want to delete that submission, or else the enrollment may be a TA who has already graded things so we may not want to delete the grading work they have already done.

4.3 Assignment Manager

This manager is responsible for creation of and structuring of assignments, including managing assignment details, question sets, questions and their subparts, topic assignment to questions, question and solution files, rubrics for each question, and others.

API	
URL	/course/$courseid/assignments
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return assignments.[**id, name, comments**] for all **non-deleted** assignments in this course. Also return how many question sets does this assignment have yet (may be zero if no question sets have been created yet).

API	
URL	/course/$courseid/assignments
HTTP Method	**POST**
Pre-Validations	If this assignment name already exists in this course, throw a **NODUP** and finish.
Action	Given name, comments, create a new assignment. If any of the values entered are invalid () then return a **NOVAL** error and finish.
Return Data	assignments.[**id, name, comments**] of the newly created assignment so that the front-end may display details and a link to this assignment as well.

API	
URL	/course/$courseid/assignment/$assignmentid
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return all columns of the corresponding row in the **assignments** table. Also return all columns of rows of **question_sets** corresponding to this assignment.

API	
URL	/course/$courseid/assignment/$assignmentid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	Allow valid changes to all allowed columns (e.g. name etc). In case of illegal values, return **NOVAL** errors as appropriate.
Return Data	Return all columns of the updated row in the **assignments table** so that the frontend can display them.

API	
URL	/course/$courseid/assignment/$assignmentid
HTTP Method	**DELETE**
Pre-Validations	**None**
Action	Delete this assignment by setting is_deleted = 1. Also **soft delete** 1. All **question_sets** rows associated with this assignment and follow the cascading deletes that follow the deletion of a **question_set** 2. All events rows associated with this assignment and follow the **cascading deletes** that follow the deletion of an event 3. All **gradesheet** rows associated with this assignment. This operation is not required since **gradesheet** rows are associated with events and not assignments. Thus, **gradesheet** rows will automatically **GET** deleted due to cascaded deletes in point ii above. 4. All **gradesheet** rows associated with this assignment. This operation is not required since gradesheet rows are associated with events and not assignments. Thus, **gradesheet** rows will automatically **GET** deleted due to cascaded deletes in point ii above. 5. All the **subevent** rows associated with this event and then all the **submission_gropus , grading_duty** entries will be deleted.
Return Data	**None**
Note	**Warn** the user that such an assignment delete operation is extremely pervasive and may **not necessarily be reversible** even by an admin since it would be hard to distinguish **cascading soft deletes** to, say uploads, performed as a result of question set delete, from actual deletes performed to uploads while editing them.

API	
URL	/course/$courseid/assignment/$assignmentid/questionsets
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return question_sets.[**id, name, total marks, name_coords, roll_coords**] for all question sets associated with this assignment. Also, for each **question set, return the question_file, the supplementary file, and the solution file** if they exist.
Note	Do not return hyperlinks to these files, return the files themselves.

API	
URL	/course/$courseid/assignment/$assignmentid/questionsets
HTTP Method	**POST**
Pre-Validations	The name of the question set must not have been used by any other question set in this assignment before otherwise throw a **NODUP** error and finish.
Action	Given a **name**, and **total**, create a new question set. The total marks must be a **non-negative** integer (may be zero) otherwise throw a **NOVAL** error.
Return Data	Return question_sets.[**id, name, total_marks, name_coords, roll_coords**] for this newly created question set so that the frontend can display a link to this.

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/$questionsetid
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return **question_sets.[id, name, total_marks, name_coords, roll_coords]** for this question set. Also return the **question_file, the supplementary file, and the solution file** - if they exist.
Note	Do not return hyperlinks to these files, return the files themselves.

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/$questionsetid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	Given new values of **name and total_marks**, update them. The name of the **question set** must not have been used by any other question set in this assignment before otherwise throw a **NODUP** error and finish. Also, the **total marks** must be a **non-negative integer** (may be zero) otherwise throw a **NOVAL** error.
Return Data	Return the updated values of **question_sets.[id, name, total_marks, name_coords, roll_coords]** for this question set to be displayed

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/$questionsetid
HTTP Method	**DELETE**
Pre-Validations	**None**
Action	Soft delete this question_set row. Also soft delete all questions rows associated with this question set follow the cascading deletes that follow the deletion of a question.
Return Data	**None**
Note	Might want to warn the user that such an assignment delete operation is extremely pervasive and may not necessarily be reversible even by an admin since it would be hard to distinguish cascading soft deletes to, say uploads, performed as a result of question set delete, from actual deletes performed to uploads while editing them.

API	
URL	/course/$courseid/assignment/$assignmentid/ questionset/$questionsetid/questionFile
HTTP Method	**GET**
Pre-Validations	If no question file has been uploaded for this question set so far, return a **NOEXIST** error and finish.
Action	**None**
Return Data	Return the **file.**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset /$questionsetid/questionFile
HTTP Method	**POST**
Pre-Validations	If there already exists a file for this question set, return a **NODUP** error and finish. The existing file must be updated using a **PUT** query in order to change it.
Action	Create a new question file for this question set and update **question_sets.question_file_path accordingly**. Save the original file name in db and rename the file name with 32 random alphanumeric characters, save this new file name in **question_file_path**.
Return Data	**None**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/questionFile
HTTP Method	**PUT**
Pre-Validations	If no question file has been uploaded for this question set so far, return a **NOEXIST** error and finish.
Action	Update the question file for this question set and update **question_sets.question_file_path** accordingly
Return Data	**None**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/questionFile/images
HTTP Method	**GET**
Pre-Validations	If no question file has been uploaded for this question set so far, return a **NOEXIST** error and finish.
Action	**None**
Return Data	Return the **paginated file** i.e. return the file as a **set of images**.

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/solutionFile/
HTTP Method	**GET**
Pre-Validations	If no solution file has been uploaded for this question set so far, return a **NOEXIST** error and finish.
Action	**None**
Return Data	Return he **file**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/solutionFile/
HTTP Method	**POST**
Pre-Validations	If there already exists a file for this question set, return a **NODUP** error and finish. The existing file must be updated using a **PUT** query in order to change it.
Action	Create a new solution file for this **question set** and update **question_sets.solution_file_path** accordingly
Return Data	**None**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $question-setid/solutionFile/
HTTP Method	**PUT**
Pre-Validations	If no solution file has been uploaded for this question set so far, return a **NOEXIST** error and finish.
Action	Update the solution file for this question set and update question_sets.solution_file_path accordingly
Return Data	**None**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $question-setid/supplementaryFile
HTTP Method	**PUT**
Pre-Validations	If no supplementary file has been uploaded for this question set so far, return a **NOEXIST** error and finish.
Action	**None**
Return Data	Return the file

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $question-setid/supplementaryFile
HTTP Method	**POST**
Pre-Validations	If there already exists a file for this question set, return a **NODUP** error and finish. The existing file must be updated using a **PUT** query in order to change it.
Action	Create a new supplementary file for this question set and update **question_sets.supplementary_file_path accordingly**
Return Data	**None**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $question-setid/supplementaryFile
HTTP Method	**PUT**
Pre-Validations	If no supplementary file has been uploaded for this question set so far, return a **NOEXIST** error and finish.
Action	Update the supplementary file for this question set and update **question_sets.supplementary_file_path** accordingly
Return Data	**None**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $question-setid/questions
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return all columns of the **questions table** for all questions associated with this **question set**. Also, for each question, return a list of all **topics** from **question_has_topics** that are linked to that question. The front end may use all this information to show highlighted portions (if indicated) for each question on the paginated question file.

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/questions
HTTP Method	**POST**
Pre-Validations	**None**
Action	Create a new row in the **questions table**. Do not allow **questions.question_set_id** to be specified or else modified. That must be set automatically by the backend script. Make sure that the following hold else return a **NOVAL** error and finish 1. **Questions.subpartno** must be a non-negative integer like 0, 1, 2. For all questions that have a common parent question, their subpart number must be unique else a **NODUP** error must be thrown (i.e. we cannot have two questions named 2.2). In particular, all **questions** that do not have a parent question must also have unique subpart no. 2. **questions.question_file_page** is a positive number less than or equal to the number of pages in the **question file** 3. **questions.question_file_coords** are valid coordinates that fit inside the paginated image of that page 4. **questions.marks** is a non-negative (but possibly fractional i.e. decimal) number 5. A question for which **questions.is_actual_question** = 0 (i.e. a question created purely to structure questions nicely) must have its marks column blank. Its marks will be derived as a sum of marks of its subquestions. 6. A question with no **subquestions** must have **questions.is_actual_question** = 1 7. A question with **questions.is_actual_question** = 1 must have a non-zero marks value. 8. A question that has **sub_questions** i.e. a question which is the **parent_question** of some other question, must not be an actual question i.e. only non-actual questions can have sub-questions. 9. questions.[**type, difficulty_level, parent_question_id,grading_duty_scheme, is_actual_question**] all take valid values.
Return Data	Return all columns of the newly created row of the questions table.

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/question/$questionid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	Allow modifications to all fields except **questions.question_set_id**. The instructor should not be able to assign a question to a different question set just by changing the question set id. If the modified values do not satisfy the checks mentioned in the **POST** request, return a **NOVAL** error and finish.
Return Data	Return all updated columns of the **questions** table for this question.

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/question/$questionid
HTTP Method	**DELETE**
Pre-Validations	If this question has children questions i.e. if there are questions that have this question as the parent question, then deny **DELETE** operation by throwing a **NODEL** error. A question can be deleted only if it has no children question (or if all children have been deleted first). This is to prevent us from having orphaned questions.
Action	**Soft-delete** this question. Also soft delete 1. All **question_options, and question_has_topics** rows associated with this question. 2. All rows in the **table rubrics** associated with this question and follow the **cascading deletes** that follow the deletion of a rubric. 3. All rows in the table **responses** that correspond to this question and follow the cascading deletes that following deletion of a response.
Return Data	**None**
Note	Warn the user that such a **question delete** operation is extremely pervasive and may not necessarily be reversible even by an admin since it would be hard to distinguish **cascading soft deletes** to, say rubrics, performed as a result of question delete, from actual deletes performed to rubrics while designing them.
Scope	Ban deletion of **questions or assignments** (even by the instructor) if even a single submission has taken place with respect to that question or that assignment.

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/question/$questionid/topic/$topicid
HTTP Method	**POST**
Pre-Validations	If this topic has already been linked to this question before, throw a **NODUP** error and finish. If this **topicid** does **not exist** or else has been soft deleted, return a **NOEXIST** error and finish. If there already exists a soft-deleted row in question_has_topics linking this question to this topic, simply **un-delete** it and finish (no need to create a new row).
Action	Create a new row in **question_has_topics** linking this topic to this question
Return Data	**None**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/question/$questionid/topiclinkTopic/$topicid
HTTP Method	**DELETE**
Pre-Validations	If such a row does not exist in the table question_has_topics, then return a **NOEXIST** error and finish.
Action	Soft delete this row in **question_has_topics** linking this topic to this question
Return Data	**None**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/question/$questionid/rubrics
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return the page from the question paper that contains this **question**, as well as **questions.question_paper_coords** so that the frontend can display this question to the user. Also return all rows of the table **rubrics** that correspond to this question.

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/question/$questionid/rubrics
HTTP Method	**POST**
Pre-Validations	**None**
Action	Given the text and marks, create a new **rubric** for this **question**. Marks can be negative as well. Do not allow **specification or updates** to the **foreign_key rubrics.question_id**. That must be set by the backend script itself.
Return Data	Return all details of the **rubric** just created.
Scope	allow importing **rubrics** from other questions

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/question/$questionid/rubric/$rubricid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	Given updated **marks and text**, update the rubric item.
Return Data	Return details of the updated **rubric**
Scope	allow **reverse rubric search** i.e. allow a grader to retrieve all **grading duty** rows assigned to them where they applied this **rubric**

API	
URL	/course/$courseid/assignment/$assignmentid/questionset/ $questionsetid/question/$questionid/rubric/$rubricid
HTTP Method	**DELETE**
Pre-Validations	**None**
Action	Soft delete this rubric item, also soft delete all grading_duty_has_rubrics rows that contain this rubric.
Return Data	**None**
Return Data	Warn the **user** that such a **rubric** delete operation is extremely pervasive and may not necessarily be reversible even by an admin since it would be hard to distinguish **cascading soft deletes** performed as a result of **rubric** delete, from actual deletes performed to during grading process

4.4 Event Manager

This manager is responsible for creation and management of events within the system and all related tasks including managing submission and grading deadlines, assigning specific grading duties to various graders, managing various submission and grading modes, etc.

API	
URL	/course/$courseid/events
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return events.[**id, name, assignment_id, is_external**] for all relevant events in this course for this user, i.e. all events such that the event has a subevent such that the user has a row in the table **user_has_subevents** with respect to that subevent. Additionally, for each such event, also return subevents.[**id, name, start_time, type, display_end_time, is_blocking, allow_late_ending, display_late_end_time**] for all subevents of this event which are linked to this user. Instructor may want to display 5PM as end time but have actual end time as 5:05PM. In such a case, end_time would be 5:05PM but display_end_time would be 5PM. Similarly for display_late_end_time.
Note	do not show **end_time** to students - this defeats the whole purpose of having a **display_end_time**

API	
URL	/course/$courseid/events
HTTP Method	**POST**
Pre-Validations	If an event exists with the same name, throw a **NODUP** error and finish. For an event where **is_external** is true, **assignment_id** must be blank. If the **assignment_id** is not blank, then **is_external** must be false. If this relation is not satisfied then throw a **NOVAL** error.
Action	Given a **name, assignment_id, grade_aggregation_method, and is_external tag**, create a new event in the course. If this **assignment id** does not exist, then return a **NOEXIST** error and return. **Grade_aggregation_method** must be one of **AVG, MAX, MIN, MED**.
Return Data	Return **events.[id, name, assignment_id, is_external]** for this newly created event so the new event can also be displayed.

API	
URL	/course/$courseid/allEvents
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	Return **events.[id, name, assignment_id, is_external, grade_aggregation_method]** for all events in this **course**, not just the ones linked to this user. Additionally for each event, also return **subevents.[id, name, start_time, type, end_time, is_blocking, display_end_time, allow_late_ending, late_end_time, display_end_time, event_id]** for all subevents of this event, not just the ones linked to this user.
Note	We are showing both display and actual end times here since this method would be available only to the instructor.

API	
URL	/course/$courseid/event/$eventid
HTTP Method	**PUT**
Pre-Validations	For an event where **is_external** is true, **assignment_id** must be blank. If the assignment_id is not blank, then is_external must be false. If this relation is not satisfied then throw a **NOVAL** error.
Action	Given a new **name, assignment_id, is_external tag, grade_aggregation_method**, check if the **assignment_id** is valid else return a **NOVAL** error and finish and then update the respective columns.
Return Data	Return the updated values of **event.[id, name, assignment_id, is_external]**

API	
URL	/course/$courseid/event/$eventid
HTTP Method	**DELETE**
Pre-Validations	**None**
Action	Soft delete this event. Also soft delete all subevents rows associated with those events and follow the cascading deletes that follow the deletion of a subevent.
Return Data	**None**
Note	Warn the user that such an event delete operation is extremely pervasive and may not necessarily be reversible even by an admin since it would be hard to distinguish cascading soft deletes to, say rubrics, performed as a result of event delete, from actual deletes performed to rubrics while designing them.

API	
URL	/course/$courseid/event/$eventid/subevents
HTTP Method	**POST**
Pre-Validations	If a subevent of this event exists with this name, throw a **NODUP** error and finish. Note that two events can each have a subevent with the same name. For example, two events "MidSem" and "EndSem", may both have a subevent called "Morning". However, a single event, say "Quiz", cannot have two subevents with the name "Morning".
Action	Given details of the subevent, create a new subevent of this event following the type-specific instructions given below. If **allow_late_ending** is true, then a valid **late_end_time and display_late_end_time** must be specified else return a **NOVAL** error and finish. The times must satisfy **start_time <= end_time <= late_end_time, as well as end_time start_time <= display_end_time ≤ display_late_end_time** else return a **NOVAL** error and finish. The param attribute should be sent in the request as described in the subevent section of entities .
Return Data	Return **subevents.[id, name, etc]** for the newly created subevent
Note	1. Note that **subevents SVIEW, GVIEW, AVIEW** are not linked to any users since they apply to all enrolled users of the course. However, the events **SUPLOAD, GUPLOAD, RGREQ, RGUPLOAD** must be assigned to users via the table **user_has_subevents** to allow only certain users to participate in this subevent. 2. Also note that there may be multiple subevents within an event of the types **SUPLOAD, GUPLOAD, RGREQ, RGUPLOAD** (e.g. multiple SUPLOAD subevents for phased exams, etc). However, there can be only one subevent of type **QVIEW, SVIEW, GVIEW, AVIEW** within each event. If the user tries to create two **QVIEW subevents**, for example, by manufacturing a POST request, throw a NODUP error. The instructor must modify the existing QVIEW event to change times etc, not create multiple QVIEW events. Various parmameters in the subrequest should be sent based on the type of the sub-request as described in the below tables.

Subevent Specific params for SUPLOAD	
Type specific params	1. **SBM** (submission mode): this must be one of the following: **OLS, OLI, OSS** (online by student(s), online by instructor, onsite by student(s)). If the mode is **OLI**, then the following params are not required. However, if the mode is **OLS or OSS**, then the following params must be specified as one of their valid values. 2. **SGS** (submission group scheme): this must be one of the following: **IN, FG, OG$max (individual, fixed-group, open-group** with maximum of **$max** students in each group where **$max** is a strictly positive integer) If the **SGS = FG or SGS = IN**, we must expect a CSV file with each row giving us a comma-separated list of rollnumbers/user names which belong to that group. 3. **QSS** (question set scheme): this must be one of the following: **OS, FS (open-set, fixed-set) QSS = OS** is allowed only if the assignment linked to the event corresponding to this subevent has at least two question sets associated with it. If **QSS = FS**, the CSV file must have two columns, the second column having the **question_set_name** of the question set associated with that submission group. Future Scope: implement **SGS = OG and QSS = OS**. This can be done but requires a bit more careful bookkeeping 4. **NAC** (need access code): does this submission need a secret code for the system to accept submissions? Allow **NAC = 1** only if **SGS = IN or SGS = FG** (**NAC** must be 0 if **SGS = OG** since **NAC = 1** does not make sense here). If **NAC = 1**, the CSV file must contain a third column as well, the third column giving us the secret access code for each submission group. 5. **MUS (main upload size)**: how large can the **main upload file** be (in MB)? 6. **MUT** (main upload type): what file types are allowed as the **main upload**? (Allow only PDF in general) 7. **SUP** (allow supplementary?): **SUP = 1** means that the student is allowed to upload a supplementary file as well 8. **SUS** (supplementary upload size): how large can the supplementary upload be (in MB)? 9. **SUT** (supplementary upload type): what file types are allowed as the supplementary upload? 10. **DEL** (delay parameter): specify a **mean and standard deviation** (in seconds) for the amount of time the frontend must artficially delay the upload of a submission after the student clicks the submit button. 11. **COL** (color): specify a color in **hex RGB** format for the background color of the submission page for this assignment. This will come in handy during live quizzes

Pre-Validations	If any row in the CSV file is invalid (contains a non-enrolled **roll number/username**, or wrong question set etc), or if the number of columns in the CSV does not match the modes/schemes chosen, throw a **NOVAL** error and do not process the CSV file. This CSV file must be preserved and its filepath must be saved inside the **course_logs** row corresponding to this request. If a **username/roll** number is repeated in the CSV file in the same or two different rows, throw a **NODUP** error and finish. If a **username/roll** number mentioned in the CSV file already has a **user_has_subevents** row corresponding to some subevent of this event of type **SUPLOAD**, then throw a **NODUP** error and finish (e.g. a student can make only one submission per assignment). For every **username/roll** number mentioned in the CSV file, make sure that they have a role in this course that at least allows them to make a submission i.e. they are able to make a **POST/PUT** request to **/event/$eventid/submit** otherwise throw a **NOACCESS** error (for example, do not allow a non-student of this course to be scheduled a submission subevent).
Action	If **SBM = OLI** or else if **SBM = OLS and SGS = OG**, nothing additional required to be done. However, if **SBM = OLS** (**SBM = OSS** not handled for the moment) and **SGS = FG or SGS = IN**, and if everything is okay with the CSV file, process the CSV file and for each row in that file, create a submission group row and corresponding **submission_group_has_users** rows (note, a submission group may have just one member too). Depending on whether **NAC = 1** or **QSS = FS**, fill in the columns **submission_group.[access_code_gold, chosen_question_set_id]** accordingly from the CSV file. In addition, create a **users_has_subevents** row for each user which has a mention in the CSV file.

Subevent Specific params for GUPLOAD	
Type specific params	1. **GDS** (grading duty scheme): how to allot grading duty to students (see top of this document for details)? This must be one of the following: **MQS, MQRrep, RQS, RQRrep, RSS, RSR$rep**, where **$rep** must be a strictly positive integer. In case the grading duty scheme is **MQS or MQR$rep**, the instructor must also upload a CSV file specifying for each question set and each actual question within that question set, one or more graders that are alloted to that question. In case of **MQR$rep graders**, at least **$rep** graders must be specified for each question. 2. **GLIST** (grader list): stores the contents of the CSV file serialized in some nice format. This is required because for **open group (OG) submissions**, the grader allocation will have to be done at the time of a submission so we need to store the contents of the CSV file somewhere so that we can take those decisions to create **grading_duty** rows at the time of submission.
Pre-Validations	When creating this subevent, the instructor must specify one previously created (and non-deleted) **SUPLOAD** subevent within this event which will be stored in **subevent.gen_subevent_id** for this event. Make sure that in the CSV file, each roll **number/username** mentioned of a grader is actually someone enrolled in this course who is capable of grading i.e. making **POST/PUT** requests to **/event/$eventid/gradingduty/$gradingdutyid** otherwise throw a **NOACCESS** error. If any of the questions for which graders are specified are not actual questions but merely dummy questions then also throw a **NOVAL** error.
Action	If **GDS = RQS or RQR$rep or RSS or RSR$rep**, then first collect a list of all graders in the course i.e. all enrolled people who have the ability to perform grading and then create **grading_duty** rows for each grading duty. Also, create a single row in **user_has_subevents** for each grader who has been allotted grading in this subevent and who does not already have a row in the table with respect to this subevent i.e. make sure there is at most one row in **user_has_subevents** for this subevent for any grader.

Subevent Specific params for RGREQ	
Type specific params	**None**
Pre-Validations	When creating this subevent, the instructor must specify one previously created (and non-deleted) **GUPLOAD** subevent within this event which will be stored in **subevent.gen_subevent_id** for this event.
Action	Find out the **SUPLOAD** subevent which generated the **GUPLOAD** subevent which was supplied along with this request. Then find out all students who had **user_had_subevent** rows corresponding to that **SUPLOAD** event. Simply create one **user_has_subevents** row for each such student. However, do not do so if a **username/roll** number mentioned in the CSV file already has a **user_has_subevents** row corresponding to some subevent of this event of type **RGREQ** (i.e. a student can have only one **RGREQ** event per assignment).

Subevent Specific params for RGUPLOAD	
Type specific params	1. **RDS** (regrading duty scheme): this must be one of following: **SOR, RAN, QRN** (see top of document for details) If the choice is **RAN or QRN**, the instructor must specify in a CSV file, the list of regraders (list of regraders per **question_set**, question pair in case of **QRN**). 2. RGLIST (regrader list): stores the contents of the CSV file serialized in some nice format. This is required because **regrader** allocation will be made only when some student raises a regrading request so we need to store the contents of the CSV file somewhere so that we can take those decisions to create **grading_duty** rows at the time of a regrading request.
Pre-Validations	When creating this subevent, the instructor must specify one previously created (and non-deleted) **RGREQ** subevent within this event which will be stored in **subevent.gen_subevent_id** for this event. Make sure that all **usernames/roll** numbers given in the CSV files do correspond to graders enrolled in this course else throw a **NOACCESS** error.
Action	**None**

API	
URL	/course/$courseid/event/$eventid/subevent/$subeventid
HTTP Method	**PUT**
Pre-Validations	**None**
Action	Allow changes to only the following fields subevents.[**name, start_time, end_time, display_end_time, allow_late_ending, late_end_time, display_late_end_time, is_blocking**]. Do not allow any changes to **subevents.params** (instructor must delete and create a new event in order to change params). If the new name is already present as the name of another **subevent** of this very event then throw a **NODUP** error. If times are being modified then verify that the times satisfy **start_time ¡= end_time ¡= late_end_time** as well as **start_time ¡= display_end_time ¡= display_late_end_time** else return a **NOVAL** error and finish. If times are being modified or else a subevent that was earlier non-blocking is now being made blocking, then verify from the **user_has_subevents** table that this change does not create a situation where a user has two blocking events going on simultaneously for this course. If a clash of blocking events will take place as a result of this modification, do not allow this modification by throwing a **NOVAL** error.
Return Data	Return **subevents.[id, name, etc]** for the newly modified subevent.

API	
URL	/course/$courseid/event/$eventid/subevent/$subeventid
HTTP Method	**DELETE**
Pre-Validations	**None**
Action	If it is a subevent of the type **QVIEW, SVIEW, GVIEW, AVIEW**, simply soft delete the row and finish. However, if it is a **SUPLOAD, GUPLOAD, RGREQ, RGUPLOAD** event, then soft-delete all **user_has_subevents** rows associated with this subevent. Further, do the following, depending on the type of subevent being deleted. 1. **SUPLOAD:** Soft delete the following. (a) All **submission_groups** rows associated with this subevent. (b) All **submission_group_has_users** rows associated with those **submission_groups**. (c) All **uploads** rows associated with those submission groups. (d) All **responses** rows that are linked to those submission groups. (e) All **grading_duty rows** that are linked to those responses. (f) All **GUPLOAD** subevents linked to this subevent through **gen_subevent_id**, as well as any cascading deletes that may follow. 2. **GUPLOAD:** Soft delete all the following. (a) All **grading_duty** rows linked to the **subevent** being deleted. (b) All **grading_duty_has_rubrics** rows associated with those **grading_duty** rows. (c) All RGREQ subevents linked to this subevent through **gen_subevent_id**, as well as any cascading deletes that may follow. 3. **RGREQ:** Soft delete all the following. (a) All **RGUPLOAD** subevents linked to this subevent through **gen_subevent_id**, as well as any cascading deletes that may follow. 4. **RGUPLOAD:** Soft delete all the following. (a) All **grading_duty** rows linked to the subevent being deleted. (b) All **grading_duty_has_rubrics** rows associated with those **grading_duty** rows.
Note	Warn the user such that a subevent delete operation is extremely pervasive and may not necessarily be reversible even by an admin since it would be hard to distinguish cascading soft deletes to, say rubrics, performed as a result of subevent delete, from actual deletes performed to rubrics while designing them.
Scope	Need a calendar app to show the duration of all subevents with respect to a course (for instructor), all events related to a particular user (for students, graders), and all events on the entire system (for admin).

API	
URL	/course/$courseid/event/$eventid/gradingEvent/ $subeventid/modifyGrader
HTTP Method	**PUT**
Pre-Validations	Verify that this subevent is indeed one of type **GUPLOAD or RGUPLOAD** else throw a **NOVAL** error and finish.
Action	Given a tuple of the form QUESTION:question_id, ORIG:graderid0, NEW:graderid1,graderid2,...,graderidk, first verify that **question_id** is indeed an actual question in the assignment linked to this event (if not throw a **NOVAL** error). Then verify that graderid1, graderid2, ..., graderidk are indeed graders enrolled in this course capable of grading (if not then throw a **NOACCESS** error). Then, take all **grading_duty** rows that correspond to responses to **question_id** and that were earlier assigned to graderid0 and rewrite the **grader_id** in those rows evenly and randomly among graderid1, graderid2, ..., graderidk. If any of graderid1, graderid2, graderidk does not have a row in **user_has_subevents** with respect to this subevent, then create one to enable them to perform grading. Also modify the **GLIST or RGLIST** parameters of this subevent appropriately to reflect the new grader assignment.
Return Data	Return the updated **GLIST or RGLIST** parameters (as applicable).
Note	This step allows the instructor to modify duty assignment after grading has started by allowing them to redistribute, for every question, replace one grader with one or more graders. Note that this can be implicitly used to remove a grader from a question, as well as add new graders to a question. In this case, store the tuple **QUESTION:**question_id, **ORIG:**graderid0, NEW:graderid1,graderid2,...,graderidk in the logs for safekeeping and future reference :) Data to return: Return the updated **GLIST** or **RGLIST** parameters (as applicable).

4.5 Submission Manager

This manager is responsible for managing the viewing and submission of question papers and student responses.

API	
URL	/course/$courseid/event/$eventid/qpapers
HTTP Method	**GET**
Pre-Validations	If there does not exist even one **QVIEW** subevent linked to this event defined for the user who made this request in the table **users_have_subevents** that is currently going on, return a **NOACCESS** error and finish.
Action	Take the following steps Identify all rows in **users_have_subevents** that are of type **QVIEW** and that are linked to this event and this user and which are currently going on. For every such **QVIEW** subevent, do the following. Define a pile which is initially empty. If the **subevents.gen_subevent_id** for the **QVIEW** event is of type **SUPLOAD**, check the question set scheme **(QSS)** of that **SUPLOAD** event. If it is **OS**, then add all **question_sets** that are a part of the assignment to which this event is linked to the pile which are not already in the pile. However, if the **QSS** is of type **FS**, then find the submission group of this user from **submission_group_has_users** (user can be part of multiple submission groups in multiple assignments, so we should find the submissions groups that are only associated to this subevent, then find out the question set associated with that submission group from the table **submission_group**s and add that **question_set** to the pile if it is not already in the pile. If the **subevents.gen_subevent_id** for the **QVIEW** event is of type **GUPLOAD or RGUPLOAD**, find all **grading_duty** rows linked to that subevent which are assigned to this user, then for all r **esponse_id** linked to those grading duty rows, find the **question_id** of those responses, then from that find out which **question_set** do those **question_id** belong to. Add all such question sets to the pile which are not already in the pile.
Return Data	The **question_set name** and PDF files (actual contents, not links) of **question_file and supplementary_file** of all **question_sets** in the pile.

API	
URL	/course/$courseid/event/$eventid/submissions/
HTTP Method	**GET**
Pre-Validations	If the user does not have an **SUPLOAD** subevent or an **SVIEW** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish.
Action	**None**
Return Data	If the user has a row in the table **submission_group_has_users** with respect to a submission group such that **submission_groups.subevent_id** is an **SUPLOAD** subevent of this event, then return **submission_group_has_users.submission_group_id. Also return users.[first_name,last_name,roll_no]** for all users that are a part of this submission group. Also for this submission group, return **submission_groups.[chosen_question_set_id, access_code_submitted, is_late_submission]**. Also return, for the **SUPLOAD** event, the parameters **DEL** (artificial submission delay) and **COL** (background color) so that the frontend may use them. If there no rows for this user in **submission_group_has_users** then return a **NOEXIST** error.

API	
URL	/course/$courseid/event/$eventid/submissions/
HTTP Method	**POST**
Pre-Validations	If the user does not have an **SUPLOAD** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish. If the **SUPLOAD** subevent has **SBM = OLI**, then also return a **NOACCESS** error and finish (since submission mode being **OLI** implies that instructor will upload the submissions).
Action	If this **SUPLOAD** subevent has parameter **SGS = FG** or **IN**, then a row must have already been created for this submission group so no need to create a new row, just throw a **NODUP** error and finish (should not create a new submission if one already exists). If this **SUPLOAD** subevent has parameter **SGS = OG**, and if the user already has a row in the table **submission_group_has_users** with respect to this event, then throw a **NODUP** error and finish (a user cannot have two submissions in a single assignment). However, if **SGS = OG** and the user does not have currently a row in the table **submission_group_has_users** with respect to this event, then create a new **submission_group** row. Make sure that the id field is not simply an auto-increment value but is instead a hard-to-guess, cryptographically secure, random number (of the kind we use for **CSRF** tokens etc). If this **SUPLOAD** subevent requires an access code i.e. the parameter **NAC = 1**, then take the access code sent by the user and store it in **submission_groups.access_code_submitted**. If this **SUPLOAD** subevent has **QSS = OS**, i.e. open-set submission, take the question set id supplied by the user and store it in **submission_groups.chosen_question_set_id**. However, if the question set supplied is not a question set associated with the assignment linked to this event, then throw a **NOVAL** error. If **QSS = FS**, then the user should not be specifying a question set - if one is still supplied, ignore the supplied value.
Return Data	Return the **submission_groups.id** if we have a newly created submission group so that the student may share this with group members to join the submission group. If the access code provided does not match the gold access code, also return a **NOVAL** error. Until the correct access code is submitted, the student would not be able to submit or modify any submission files.
Scope	May want the user to specify just the question set name and not the question set id for sake of convenience.(this we can handle in ui by showing question name by doing a **GET** request)

API	
URL	/course/$courseid/event/$eventid/submissions/
HTTP Method	**PUT**
Pre-Validations	If the user does not have an SUPLOAD subevent with respect to this event going on at the moment, then return a NOEVENT error and finish.
Action	Allow edits to submission_groups.access_code_submitted if this SUPLOAD subevent requires an access code i.e. the parameter NAC = 1. Also allow edits to submission_groups.chosen_question_set_id if this SUPLOAD subevent has QSS = OS, i.e. open-set submission.
Return Data	**None**
Note	No **DELETE** method is defined. Submissions cannot be deleted by a student once created, only modifications are allowed.

API	
URL	/course/$courseid/event/$eventid/submissions/main
HTTP Method	**GET**
Pre-Validations	If the user does not have an **SUPLOAD** subevent or an **SVIEW** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish.
Action	**None**
Return Data	If the user has a row in the table **submission_group_has_users** with respect to a submission group such that **submission_groups.subevent_id** is an **SUPLOAD** subevent of this event, then corresponding to **submission_groups.upload_id_main**, return the contents (not a link) of the **submitted main file**, else return a **NOEXIST** error. Also return **uploads.is_paginated** for that upload.

API	
URL	/course/$courseid/event/$eventid/submissions/main
HTTP Method	**POST**
Pre-Validations	If the user does not have an **SUPLOAD** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish. If there no rows corresponding to this user in **submission_group_has_users** then return a **NOEXIST** error. If for this **SUPLOAD** event, we have **NAC = 1**, then verify that **submission_groups.access_code_gold === submission_groups.access_code_submitted**. If not then throw a **NOACCESS** error and finish. If the student has already uploaded a main file for this submission i.e. if **submission_groups.upload_id_main** is not **NULL**, then throw a **NODUP** error and finish. A student can submit only one main file per submission. If the **SUPLOAD** subevent has **SBM = OLI**, then also return a **NOACCESS** error and finish (since submission mode being **OLI** implies that instructor will upload the submissions).
Action	If the type of the submitted file is not one of the allowed types specified in the **SUPLOAD** parameter **MUT**, throw a **NOVAL** error. If the size of the submitted file size exceeds that of the **SUPLOAD** parameter **MUS**, throw a **NOVAL** error. The frontend should have submitted a **SHA** hash of the submitted file. Compute the SHA hash of the submitted file and compare the two hashes. If not equal, return a **NOVAL** error. If everything is alright, record the time of submission. If it is within the late submission period, set **submission_groups.is_late_submission** = 1. Then create a new entry in the table uploads and also link this new upload to the submission by setting the value of **submission_groups.upload_id_main**.
Return Data	Return the contents of the just submitted file. Also return **uploads.is_paginated** for that upload.

API	
URL	/course/$courseid/event/$eventid/submissions/main
HTTP Method	**PUT**
Pre-Validations	If the user does not have an **SUPLOAD** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish. If there no rows corresponding to this user in **submission_group_has_users** then return a **NOEXIST** error. If for this **SUPLOAD** event, we have **NAC** = 1, then verify that **submission_groups.access_code_gold === submission_groups.access_code_submitted**. If not then throw a **NOACCESS** error and finish. If the student has not previously uploaded a main file for this submission i.e. if **submission_groups.upload_id_main** is NULL, then throw a **NOEXIST** error and finish. A student can modify a file only after an initial submission has been made. If the **SUPLOAD** subevent has **SBM = OLI**, then also return a **NOACCESS** error and finish (since submission mode being **OLI** implies that instructor will upload the submissions).
Action	If the type of the submitted file is not one of the allowed types specified in the **SUPLOAD** parameter **MUT**, throw a **NOVAL** error. If the size of the submitted file size exceeds that of the **SUPLOAD** parameter **MUS**, throw a **NOVAL** error. The frontend should have submitted a SHA hash of the submitted file. Compute the SHA hash of the submitted file and compare the two hashes. If not equal, return a **NOVAL** error. If everything is alright, record the time of submission. If it is within the late submission period, set **submission_groups.is_late_submission** = 1. Then create a new entry in the table uploads and also link this new upload to the submission by updating **submission_groups.upload_id_supp**.
Return Data	Return the contents of the just submitted file. Also return **uploads.is_paginated** for that upload.
Note	No **DELETE** method is defined. Submissions cannot be deleted by a student once created, only modifications are allowed.

API	
URL	/course/$courseid/event/$eventid/submissions/supplementary
HTTP Method	**GET**
Pre-Validations	If the user does not have an **SUPLOAD** subevent or an **SVIEW** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish.
Action	**None**
Return Data	If the user has a row in the **submission_group_has_users** table with respect to submission group whose **subevent_id** is an **SUPLOAD** subevent of this event, then corresponding to **submission_groups.upload_id_supp**, return the contents (not a link) of the submitted **supplementary file**, else return a **NOEXIST** error. Also return **uploads.is_paginated** for that upload.

API	
URL	/course/$courseid/event/$eventid/submissions/supplementary
HTTP Method	**POST**
Pre-Validations	If the user does not have an **SUPLOAD** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish. If there no rows corresponding to this user in **submission_group_has_users** then return a **NOEXIST** error. If for this **SUPLOAD** event, we have **NAC** = 1, then verify that **submission_groups.access_code_gold === submission_groups.access_code_submitted**. If not then throw a **NOACCESS** error and finish. If this **SUPLOAD** event does not allow supplementary material upload i.e. **SUP** = 0, then throw a **NOVAL** error and finish. If the student has already uploaded a supplementary file for this submission i.e. if **submission_groups.upload_id_supp** is not **NULL**, then throw a **NODUP** error and finish. A student can submit only one supplementary file per submission. If the **SUPLOAD** subevent has **SBM = OLI**, then also return a **NOACCESS** error and finish (since submission mode being **OLI** implies that instructor will upload the submissions).
Action	If the type of the submitted file is not one of the allowed types specified in the SUPLOAD parameter SUT, throw a **NOVAL** error. If the size of the submitted file size exceeds that of the SUPLOAD parameter SUS, throw a **NOVAL** error. The frontend should have submitted a SHA hash of the submitted file. Com**PUT**e the SHA hash of the submitted file and compare the two hashes. If not equal, return a **NOVAL** error. If everything is alright, record the time of submission. If it is within the late submission period, set submission_groups.is_late_submission = 1. Then create a new entry in the table uploads and also link this new upload to the submission by setting the value of submission_groups.upload_id_supp.
Return Data	Return the contents of the just submitted file. Also return uploads.is_paginated for that upload.

API	
URL	/course/$courseid/event/$eventid/submissions/supplementary
HTTP Method	**PUT**
Pre-Validations	If the user does not have an **SUPLOAD** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish. If there no rows corresponding to this user in **submission_group_has_users** then return a **NOEXIST** error. If for this **SUPLOAD** event, we have **NAC** = 1, then verify that **submission_groups.access_code_gold === submission_groups.access_code_submitted**. If not then throw a **NOACCESS** error and finish. If this SUPLOAD event does not allow supplementary material upload i.e. **SUP** = 0, then throw a **NOVAL** error and finish. If the student has not previously uploaded a supplementary file for this submission i.e. if **submission_groups.upload_id_supp** is NULL, then throw a **NOEXIST** error and finish. A student can modify a file only after an initial submission has been made. If the **SUPLOAD** subevent has **SBM = OLI**, then also return a **NOACCESS** error and finish (since submission mode being **OLI** implies that instructor will upload the submissions).
Action	If the type of the submitted file is not one of the allowed types specified in the **SUPLOAD** parameter **SUT**, throw a **NOVAL** error. If the size of the submitted file size exceeds that of the **SUPLOAD** parameter **SUS**, throw a **NOVAL** error. The frontend should have submitted a SHA hash of the submitted file. Com**PUT**e the SHA hash of the submitted file and compare the two hashes. If not equal, return a **NOVAL** error. If everything is alright, record the time of submission. If it is within the late submission period, set **submission_groups.is_late_submission** = 1. Then create a new entry in the table uploads and also link this new upload to the submission by updating **submission_groups.upload_id_supp**.
Return Data	Return the contents of the just updated file. Also return **uploads.is_paginated** for that upload.
Note	No **DELETE** method is defined. Submissions cannot be deleted by a student once created, only modifications are allowed.

API	
URL	/course/$courseid/event/$eventid/submissions/submission_group/ $submission_group_id/join
HTTP Method	**POST**
Pre-Validations	If the user does not have an SUPLOAD subevent with respect to this event going on at the moment, then return a NOEVENT error and finish.
Action	If this **SUPLOAD** subevent has parameter **SGS = FG** or **IN**, then a row must have already been created for this submission group so there is no notion of joining a group - simply throw a **NODUP** error and finish. If this **SUPLOAD** subevent has parameter **SGS = OG**, and if the user already has a row in the table **submission_group_has_users** with respect to this event, then throw a **NODUP** error and finish (a user cannot have two submissions in a single assignment). However, if **SGS = OG** and the user does not have currently a row in the table **submission_group_has_users** with respect to this event, then check if **$submission_group_id** is a valid submission group with respect to this **SUPLOAD** event. If it is not a valid submission group or else if it is a valid submission group but not of this **SUPLOAD** event (e.g. student tries to join an assignment 2 group for assignment 3), then throw a **NOVAL** error. Otherwise, check how many students are already a part of this group. If this number is already equal to $max (**SGS = OG$max**), then throw a **NOACCESS** error. Else, create a new row in the table **submission_group_has_users** linking this user to the submission group **$submission_group_id**.
Return Data	**None**
Scope	Allow instructors to also link students to existing submission groups even after submission deadline using a separate URL. Also create URLs to allow creation of new submission groups, linking students to them, uploading main and supplementary files, and paginating those files using separate URLs.

API	
URL	/course/$courseid/event/$eventid/allSubmissions
HTTP Method	**GET**
Pre-Validations	**None** (Warning: this is supposed to be an instructor level URL)
Action	**None**
Return Data	Return a list of **SUPLOAD** subevents within this event. For each **SUPLOAD** subevent, return a list of **submission_groups.[id, access_code_gold, access_code_submitted, is_late_submission, chosen_question_set_id]** which got created with respect to that event. For each **submission group**, return the list of students linked to that submission group. Also, for each **SUPLOAD** subevent, return a list of students who were given permission to make submissions, but have not (yet) made any submissions.

API	
URL	/course/$courseid/event/$eventid/allSubmissions/$submission_group_id /main
HTTP Method	**GET**
Pre-Validations	**None** (Warning: this is supposed to be an instructor level URL)
Action	**None**
Return Data	Return the contents of the main upload file of this submission if this is a valid **submission_group_id**. If this is an invalid submission_group_id or else if this **submission group id** is not linked to this event, then throw a **NOEXIST** error.

API	
URL	/course/$courseid/event/$eventid/allSubmissions/$submission_group_id/supplementary
HTTP Method	**GET**
Pre-Validations	**None** (Warning: this is supposed to be an instructor level URL)
Action	**None**
Return Data	Return the contents of the supplementary upload file of this submission if this is a valid **submission_group_id**. If this is an invalid **submission_group_id** or else if this **submission group id** is not linked to this event, then throw a **NOEXIST** error.

API	
URL	/course/$courseid/event/$eventid/sectionSubmissions
HTTP Method	**GET**
Pre-Validations	**None** (Warning: this is supposed to be an instructor level URL)
Action	**None**
Return Data	Return a list of **SUPLOAD** subevents within this event. For each **SUPLOAD** subevent, return a list of **submission_groups.[id, access_code_gold, access_code_submitted, is_late_submission, chosen_question_set_id]** which got created with respect to that event where there is a student in that submission group that belongs to the same section as the user making this query. For each submission group, return the list of students linked to that submission group. Also, for each **SUPLOAD** subevent, return a list of students who were given permission to make submissions, but have not (yet) made any submissions.
Scope	Figure out a decent way to allow a person to be in multiple sections. That way we would be able to **GET** rid of the allSubmissions URLs

API	
URL	/course/$courseid/event/$eventid/sectionSubmissions/ $submission_group_id/main
HTTP Method	**GET**
Pre-Validations	Check if this submission group contains a student in the same section as the user making this query. If not then throw a **NOACCESS** error (**Warning**: this is supposed to be a tutor level URL)
Action	**None**
Return Data	Return the contents of the main upload file of this submission if this is a valid **submission_group_id**. If this is an invalid **submission_group_id** or else if this submission group id is not linked to this event, then throw a **NOEXIST** error.

API	
URL	/course/$courseid/event/$eventid/sectionSubmissions/ $submission_group_id/supplementary
HTTP Method	**GET**
Pre-Validations	Check if this submission group contains a student in the same section as the user making this query. If not then throw a NOACCESS error (Warning: this is supposed to be a tutor level URL)
Action	**None**
Return Data	Return the contents of the supplementary upload file of this submission if this is a valid submission_group_id. If this is an invalid submission_group_id or else if this submission group id is not linked to this event, then throw a **NOEXIST** error.

API	
URL	/course/$courseid/event/$eventid/submissions/paginate/
HTTP Method	**GET**
Pre-Validations	**None**
Action	**None**
Return Data	If the student has not yet chosen a **question_set** yet, return a **NOEXIST** error else return a list of all actual **question_id** with respect to that **question_set**.

API	
URL	/course/$courseid/event/$eventid/submissions/paginate/ $question_id/$pageno/
HTTP Method	**POST**
Pre-Validations	If the user does not have an **SUPLOAD** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish. If the user is not linked to any submission group yet, or else if that submission group has not yet made a main file upload or else if the submission group has no chosen a question_set yet, throw a **NOEXIST** error and finish. If this question_id does not correspond to an actual question in the question set chosen by the user, then throw a **NOVAL** error and finish (only actual questions need to be linked to pages).
Action	If the $pageno parameter is less than 1 or more than the number of pages in the main file linked to this submission group, then throw a **NOVAL** error and fininsh. If there is already a responses row created with respect to this **question_id** and the submission group of which this user is a part, simply update the page no to the one that is supplied. Else, create a new row in the table responses with this information.
Return Data	Return the page-question link that was created/updated so that the frontend may display the same.
Note	allow graders/section tutors/instructors to also perform/correct pagination using a separate URL
Scope	do we need **upload_coords** as well for user submitted assignments or is just upload page enough? The table responses currently has a **upload_coords** field as well which does not seem necessary at the moment.

API	
URL	/course/$courseid/event/$eventid/submissions/paginate/$question_id/
HTTP Method	**DELETE**
Pre-Validations	If the user does not have an **SUPLOAD** subevent with respect to this event going on at the moment, then return a **NOEVENT** error and finish. If there is no responses row for this **question_id** which is linked to a submission_group of which this user is a part, throw a **NOEXIST** error and finish.
Action	Soft delete the **responses** row corresponding to this question and this user's submission group. Also soft delete all **grading_duty** rows that correspond to that response and all **grading_duty_has_rubrics** that correspond to those **grading_duty_rows**.
Return Data	None

4.6 Grading Manager

This manager is responsible for all activities related to scheduling of grading duties, actual grading and regrading requests.

API	
URL	/course/$courseid/event/$eventid/gradingEvent/ $subeventid/doGraderAssignment
HTTP Method	**PUT**
Pre-Validations	Verify that this subevent is indeed one of type **GUPLOAD** else throw a **NOVAL** error and finish.
Action	Collect all response_id rows that are linked to the **SUPLOAD** subevent that is the generator subevent of this **GUPLOAD** subevent which have not been assigned graders i.e. such that there does not exist even a single **grading_duty** row. Create grading duty rows for all these orphaned responses according to the grading policy specified for this **GUPLOAD** event. Also, for every grader who is allotted grading_duty rows as a result, check if they have a user_has_subevents row with respect to this **GUPLOAD** subevent or not. If not then create a **user_has_subevents** row that would allow them to do grading (i.e. make sure that the same grader does not have two rows in the table **user_has_subevents** with respect to the same subevent). Also make sure that no grader **GET**s more than one **grading_duty** rows for the same **response_id** (a grader cannot grade the same submission more than once - they can regrade it more than once though).
Return Data	**None**
Note	The instructor may call this URL again and again for example, a few students submit and then the URL **GET**s called which causes those submissions to **GET** assigned graders. Afterward if some more students submit and now this URL **GET**s called again, only the new submissions **GET** assigned graders, old submissions are untouched. The only way to modify graders in **grading_duty** rows that have already been created is to use the URL **/course/$courseid/event/$eventid/gradingEvent/$subeventid/modifyGrader**

API	
URL	/course/$courseid/event/$eventid/gradingEvent/ $subeventid/
HTTP Method	**GET**
Pre-Validations	Verify that this subevent is indeed one of type **GUPLOAD** or **RGUPLOAD** else throw a **NOACCESS** error and finish. Also ensure that this subevent is indeed linked to eventid and not some other event else throw a **NOACCESS** error and finish. Also ensure that this user does have a **user_has_subevents** row corresponding this this subeventid otherwise throw a **NOACCESS** error and finish. Also ensure that this subevent is still going on (possibly in the late period) otherwise throw a **NOEVENT** error and finish.
Action	**None**
Return Data	Return a list of all **grading_duty.[id, is_completed]** rows linked to this subevent that are also linked to this user.

API	
URL	/course/$courseid/event/$eventid/gradingEvent/$subeventid/ gradingDuty/$gradingdutyid
HTTP Method	**GET**
Pre-Validations	Verify that this subevent is indeed one of type **GUPLOAD** or **RGUPLOAD** else throw a **NOACCESS** error and finish. Also ensure that this grading duty row is a valid one and is indeed linked to this subevent as well as that the grading duty is assigned to this very user else throw a **NOACCESS** error and finish. Also ensure that this user does have a **user_has_subevents** row corresponding this this subeventid otherwise throw a **NOACCESS** error and finish. Also ensure that this subevent is still going on (possibly in the late period) otherwise throw a **NOEVENT** error and finish.
Action	Com**PUT**e the aggregate marks of this grading duty row and set the dirty flag to zero.
Return Data	Return **grading_duty.[grader_comment, marks_adjustment, is_completed, aggregate_marks, is_late_grading]**. Also return all **grading_duty_has_rubrics** rows linked to this grading duty. Also, return for the response linked to this grading duty, return all rubrics rows linked to the **question_id** of the response, as well as the appropriate page (image) of the upload that the student linked to that **question_id**.
Scope	Develop a nice way to show the supplementary file to graders. Right now it has no use.

API	
URL	/course/$courseid/event/$eventid/gradingEvent/$subeventid/ gradingDuty/$gradingdutyid
HTTP Method	**PUT**
Pre-Validations	Verify that this subevent is indeed one of type **GUPLOAD or RGUPLOAD** else throw a **NOACCESS** error and finish. Also ensure that this grading duty row is a valid one and is indeed linked to this subevent as well as that the grading duty is assigned to this very user else throw a **NOACCESS** error and finish. Also ensure that this user does have a user_has_subevents row corresponding this this subeventid otherwise throw a **NOACCESS** error and finish. Also ensure that this subevent is still going on (possibly in the late period) otherwise throw a **NOEVENT** error and finish.
Action	Allow changes to **grading_duty.[grader_comment, marks_adjustment]**. If the current server time has breached into the late grading period, set the **is_late_grading** flag to 1. Also, recom**PUT**e the aggregate marks of this grading duty row and set the dirty flag to zero.
Return Data	Return the updated values so that the frontend may display them.

API	
URL	/course/$courseid/event/$eventid/gradingEvent/ $subeventid/gradingDuty/$gradingdutyid/applyRubric/$rubricId
HTTP Method	**POST**
Pre-Validations	Verify that this subevent is indeed one of type **GUPLOAD or RGUPLOAD** else throw a **NOACCESS** error and finish. Also ensure that this grading duty row is a valid one and is indeed linked to this subevent as well as that the grading duty is assigned to this very user else throw a **NOACCESS** error and finish. Also ensure that this user does have a **user_has_subevents** row corresponding this this subeventid otherwise throw a **NOACCESS** error and finish. Also ensure that this subevent is still going on (possibly in the late period) otherwise throw a **NOEVENT** error and finish. Also verify that this **rubric_id** indeed corresponds to the question linked to this response else throw a **NOVAL** error and finish.
Action	If there is already an application of this **rubric** to this grading duty, throw a **NODUP** error and finish (cannot apply a rubric twice). Else, if there is a soft deleted row applying this rubric to this grading duty, just undelete it, else create a new **grading_duty_has_rubrics** row assigning this rubric to this grading duty. Set **grading_duty.is_aggregate_dirty = 1**. If the current server time has breached into the late grading period, set the **is_late_grading** flag to 1.
Return Data	Return this **(grading_duty_id, rubric_id)** pair that was just linked.

API	
URL	/course/$courseid/event/$eventid/gradingEvent/ $subeventid/gradingDuty/$gradingdutyid/applyRubric/$rubricId
HTTP Method	**DELETE**
Pre-Validations	Verify that this subevent is indeed one of type **GUPLOAD or RGUPLOAD** else throw a **NOACCESS** error and finish. Also ensure that this grading duty row is a valid one and is indeed linked to this subevent as well as that the grading duty is assigned to this very user else throw a **NOACCESS** error and finish. Also ensure that this user does have a user_has_subevents row corresponding this this subeventid otherwise throw a **NOACCESS** error and finish. Also ensure that this subevent is still going on (possibly in the late period) otherwise throw a **NOEVENT** error and finish. Also verify that this rubric_id indeed corresponds to the question linked to this response else throw a **NOVAL** error and finish.
Action	If there is no non-deleted grading_duty_has_rubrics row assigning this rubric to this grading duty then throw a **NODEL** error (cannot remove a rubric that has not been applied yet). Else, soft delete the **grading_duty_has_rubrics** row assigning this rubric to this grading duty. Set **grading_duty.is_aggregate_dirty = 1**. If the current server time has breached into the late grading period, set the **is_late_grading flag** to 1.
Return Data	**None**

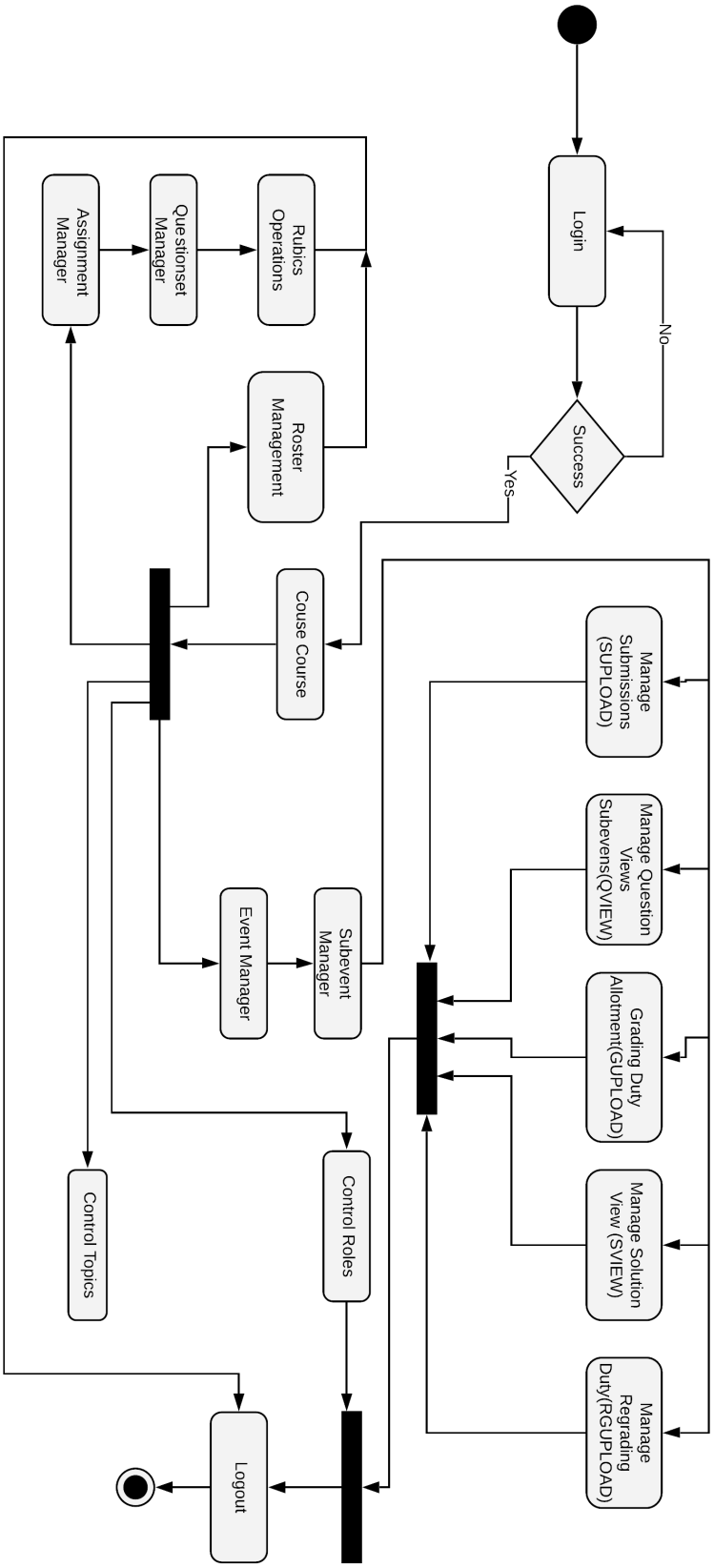


Figure 4.3: Instructor Activity Diagram

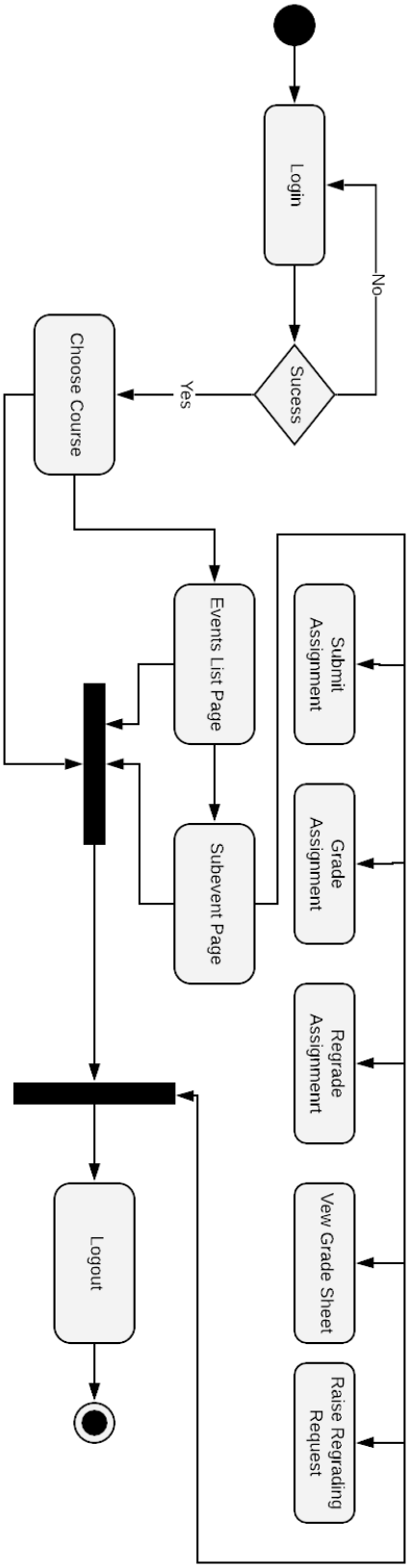


Figure 4.4: User Activity Diagram

Chapter 5

Applications

The SphinX platform is expected to interact with dozens of courses, their instructors, each of whom would be interacting with the system by setting questions, creating assignments and examinations/quizzes, hundreds or thousands of students who would each respond by giving their responses to these assignments/examinations/quizzes, and also dozens or hundreds of graders who will provide grades and/or textual comments. This will generate a large amount of data in various formats. Our database model captures this data in well-defined formats such that it can be accessed, analyzed, and conveniently feed into machine learning algorithms. Researchers can use this data for research in Natural Language Processing and Visual Recognition tasks. This data can also be used to build different applications that can help system users.

In this chapter we present an example of such a useful machine learning-based application that can be built on top of SphinX. Although not a core part of SphinX's basic functionality of enabling scalable management of grading duties in large course, this app nevertheless makes life more convenient for the course administrators. We intend to include several such applications as a part of SphinX in future releases.

5.1 Problem Formulation

SphinX supports online electronic submissions as well as pen and paper submissions where answers are handwritten but then scanned into the system. The latter is especially useful in case of examinations that are carried out within examination halls in the traditional way. The answer sheets are scanned, converted to PDF files, and uploaded onto SphinX, typically by the instructor or a teaching assistant. The system supports bulk uploads to the platform wherein a single PDF file containing submissions of dozens of students can be uploaded at once. Now, whereas convenient, this poses a challenge of mapping the individual submissions to the accounts of the students who

made those submissions.

Since students typically write their names and roll numbers on (each page of the) answer script, SphinX solves the above problem by first asking the instructor to point out coordinates in the answer sheet where the student would have written these identification marks, and then automatically using machine learning and vision techniques to map the student submissions to their accounts. Figure 5.1 gives and example of such a submission. Our task is to use this identification information to map the submission to a student account.

Figure 5.1: A sample identity information box from an actual answer sheet from the course ESC101: Fundamentals of Computing. The name and roll number of the student have been blurred to protect their identity.

5.2 Recent Work

Handwritten character recognition traditionally proceeds in the following following steps: image pre-processing, character segmentation [8], and training a classifier that classifies each character into one of several alphanumeric classes. The segmentation step is a bottleneck in this process since segmentation is predominantly based on multiple heuristics, background image properties, foreground image properties or combination of these. In the past two decades, segmentation-less handwritten character recognition methods are rising in popularity. A recent segmentation-less approach proposed by [11] is to build a classifier capable of distinguishing among 1110 classes so that it can be used to classify up to 3 digit strings. The steps included in this method are pre-processing, length classifier and character classifiers. The length classifier estimates the number of the digits in the image. Separate models are used to classify numerals 0-9, 10-99 and 100-999 respectively.

5.3 Proposed Framework

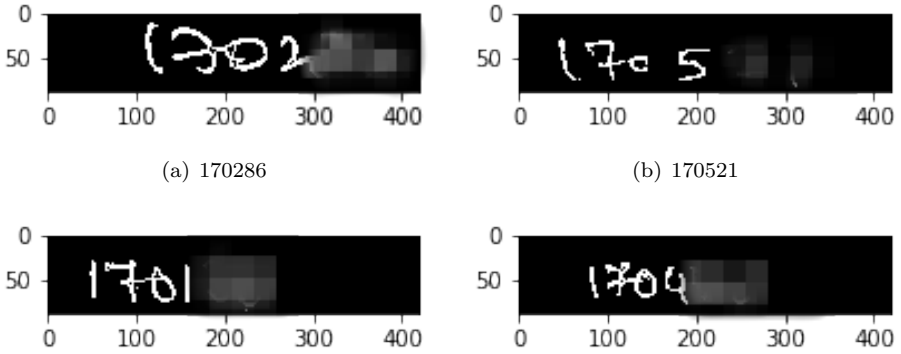
We now describe our proposed approach for SphinX.

5.3.1 Datasets

The MNIST [13] dataset includes 60,000 training examples and 10,000 tests examples of handwritten digits and is a subset of the extensive NIST dataset. MNIST consists of images of single digits which are represented as 20×20 pixel images centered in the frame of a 28×28 pixel image. Because the data is pre-processed and formatted, using this dataset and applying various machine learning techniques is very convenient. We have additionally created our own handwritten character/digits testing dataset from offerings of two courses (CS771: Introduction to Machine Learning 2017-18-I and ESC101: Fundamentals of Computing 2018-2019-I). This includes 469 test examples. Each test example contains handwritten student's name, roll number, section, and department. Our models are tested separately on that dataset as well.

5.3.2 Pre-Processing

The input image first passes through a pre-processing step. The pre-processing steps include RGB to grayscale conversion, thresholding, and morphological transformations. In the first step, the RGB image is converted into a grayscale image. A grayscale image has a single channel where each pixel value is in the range of 0-255. In the thresholding step, each pixel value can be assigned only two values. If a pixel value is higher than the threshold assign one value to it and if it's less than the threshold, then select another value. The last pre-processing step is morphological transformation. In this step, erosion followed by dilation is used to remove noise from the image.



(a) 170286 (b) 170521

(c) 170138 (d) 170476

Figure 5.2: Examples of images after the pre-processing step. The last two digits in the image are blurred to protect the student's identity.

5.3.3 Segmentation

The general goal in segmentation is to assign a label to every pixel in an image such that pixels with the same label share some semantic characteristics. This implicitly parititions an image into several

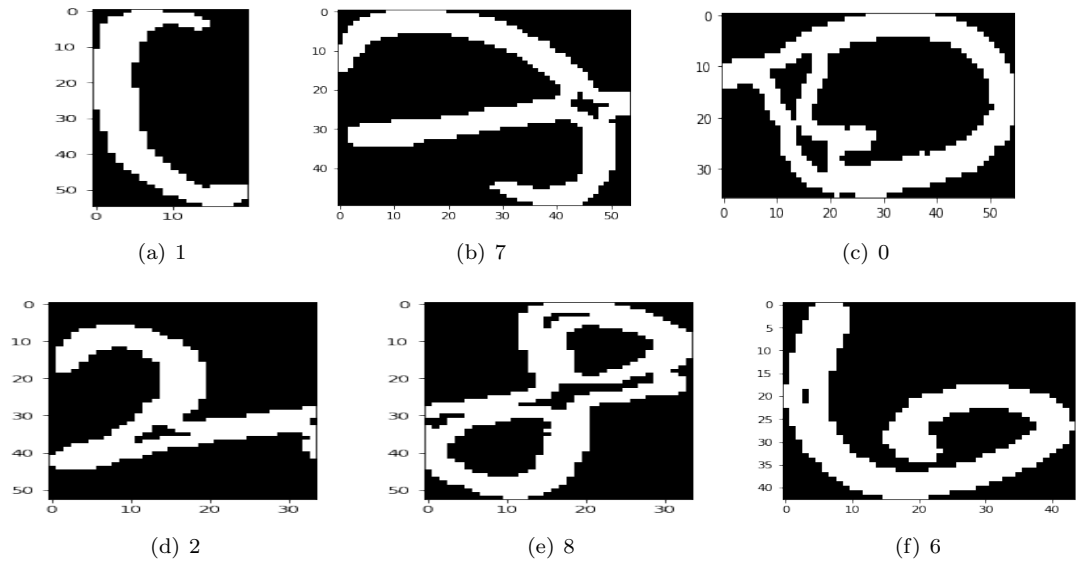
segments sharing common features. Segmentation converts the pre-processed image containing the string of digits into separate digit images. The pseudo-code for our segmentation algorithm used is given below and consists of two main steps: the first step finds out all the contours in the image. A contour is a continuous component having the same color or intensity. In the next step, various heuristically tuned thresholds are used to segment the image.

1. **Contour Finding** Find all the contours in the input image by using the Border following technique [16].

2. **Noise Redution** Remove contours that are less than a threshold area. We regarded contours with an area of less than 50 pixels as noise and removed them from the set.

3. **Character Fragmentation** The goal of this step is to handle characters that touch each other. The number of pixel-wide columns spanned by an image is defined as its pitch. Pitch/string size is defined as the average width of a character. If the width of any contour image exceeds twice the average width of a character, we fragment it horizontally into two equal parts.

4. **Association** Fragments with height less than half of all characters' average height are considered to be broken off from another character. We calculate the bounding boxes of the contours and attach this broken-off fragment with the image of the nearest (in terms of Euclidean distance from the center of the bounding box) contour.



(a) 1 (b) 7 (c) 0

(d) 2 (e) 8 (f) 6

Figure 5.3: Some examples of outputs of the segmentation step.

5.3.4 Convolution Neural Networks

Convolution Neural Networks (CNN) are a deep neural network architecture used to capture strong spatial structure in an image. In a CNN, layers are sparsely connected and perform aggressive parameter sharing, thus reducing the number of parameters in the model. A CNN is a vertical stacking of layers consisting of convolution layers, pooling layers, batch normalization layers, and full-connected layers.

Convolution Layer. Convolution layers consist of a sequence of filters, that are used to capture the local properties of the image. These filters compute the output of the neurons that are connected to a certain local region in the input. The neuron output is a dot product of the filter and local input image pixels. We have four convolution layers in the network. Our Filter size is 3×3. Two convolution layers contain 32 filters, and the other two contain 64 filters.

Max Pooling. Pooling layers reduce the spatial size of the model representation resulting in a reduction in the number of parameters and computation cost of the network. They also control over-fitting in the network. Pooling layers are inserted in between consecutive convolution layers. Two variants used to construct pooling layers are max pooling and average pooling. Our network found max pooling to work well. We use a pooling layer filter of size 2×2 and stride value of two that down samples the representation length and width by a factor of two.

Dropout. Dropout is a regularization technique that prevents the network from over-fitting. At each training stage, nodes in the network are kept with the probability p or dropped out with 1-p where p is a hyper-parameter to the network.

Batch Normalization Layer. Batch normalization is the process of adjusting and scaling the activation in batches. It has been shown to improve the speed, performance, and stability of the network.

Fully Connected Layer. The output of the convolution layer is flattened and feed into a feed-forward neural network with soft-max activation in the output layer. The dimension of the output layer is the same as the number of classes. The actuation used in the hidden nodes is generally ReLU (Rectified Linear Unit).

CNN Architecture. Table 5.1 gives details of the CNN architecture used by SphinX for digit classification.

CNN Architecture								
Input layer	**conv**	**conv**	**maxpooling**	**conv**	**conv**	**maxpooling**	**Fully connected**	**output**
28x28x1	3x3x32	3x3	3x3	3x3x64	3x3x64	2x2	512	10

Table 5.1: The CNN architecture used by SphinX for digit classification. The stride length in convolution layers and pooling layers is 1 and 2 respectively.

Training. Our model was trained using 60K examples from the MNIST dataset using the categorical cross entropy loss. The Adam optimizer was used to update network weights and the mode was trained for 30 epochs. It took 2hrs 30 min to train this model.

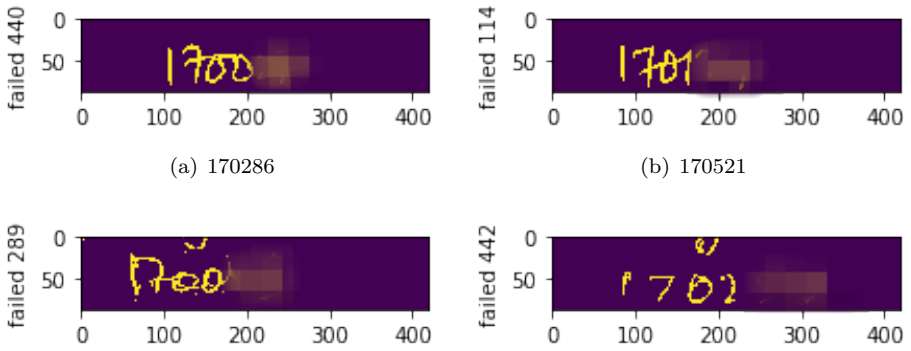
Testing. Testing is done on our own handwritten character/digits testing dataset from offerings of two courses (CS771: Introduction to Machine Learning 2017-18-I and ESC101: Fundamentals of Computing 2018-2019-I). This includes 469 test examples. We have also tested the model on MNIST test data.

5.4 Experiments

Table 5.2 gives the results of our segmentation step. It is clear that our method achieves very high segmentation accuracy. Some failure cases are outlined in Figure 6.4. The next tables respectively give the digit classification accuracy of our method on the MNIST dataset, as well as the ESC101/CS771 datasets. On the latter dataset, our method offers an impressive 99.27% digit recognition accuracy, possibly because the quality of the images may have been slightly better, given that these were digits written during an examination. Our method acheives a final accuracy of 86.78% in recognizing entire roll numbers. This number goes up to 88.67% if we exclude examples where segmentation was faulty. Overall, a near 90% hit rate in roll number recognition is satisfactory for an application like ours although we will try to improve prediction accuracy using side information, in the future.

Segmentation Results			
# Total Examples	#Correctly Segmented	# Segmentation Failed	% Accuracy
469	459	10	0.978

Table 5.2: Segmentation



(a) 170286 (b) 170521

(c) 170138 (d) 170476

Figure 5.4: Examples where our segmentation method failed. The last two digits in the image is blurred to protect the identity of the student.

Character Recognition			
# Total Examples	#Correctly Recognized	# Incorrectly Recognized	% Accuracy
2751	2652	99	96.40

Table 5.3: Character Recognition Results on MNIST Dataset

Character Recognition			
# Total Examples	#Correctly Recognized	#Incorrectly Recognized	# % Accuracy
10000	9927	73	99.27

Table 5.4: Character Recognition Results on the ESC101/CS771 dataset

Roll No. Recognition			
# Total Examples	#Correctly Recognized	#Incorrectly Recognized	# % Accuracy
469	407	62	86.78

Table 5.5: Roll Number Recognition on Our Dataset including the examples with segmentation error

Roll No. Recognition			
# Total Examples	#Correctly Recognized	#Incorrectly Recognized	# % Accuracy
459	407	62	88.67

Table 5.6: Roll Number Recognition on Our Dataset excluding the examples with segmentation error

Roll No. Prediction			
Sl.	Correct Roll No.	Predicted Roll No.	Score or Probability
1	170154	170154	0.9927
2	170563	170563	0.9997
3	170191	170111	0.4739
4	170727	172727	0.3074
5	170007	170007	0.6196

Table 5.7: Examples of Roll Number Prediction by our model

Chapter 6

System Description

The role of SphinX as an online grading management platform that enables instructors to manage courses and provides assistive grading techniques, makes it important for the platform to ensure scalability (with respect to number of courses as well as number of students), availability, access control, security, durability, and consistency of various types of resources submitted by users on the platform. Since SphinX aspires to offer a local mode of installation, the system must be portable as well. Since the system is expected to handle scheduled assignments and examination submissions, the system architecture will have to handle spikes in engagement and requests.

In this chapter, we discuss the SphinX architecture and how it achieves various software architecture quality attributes mentioned above like scalability, performance, security, reliability, portability, maintainability, and usability.

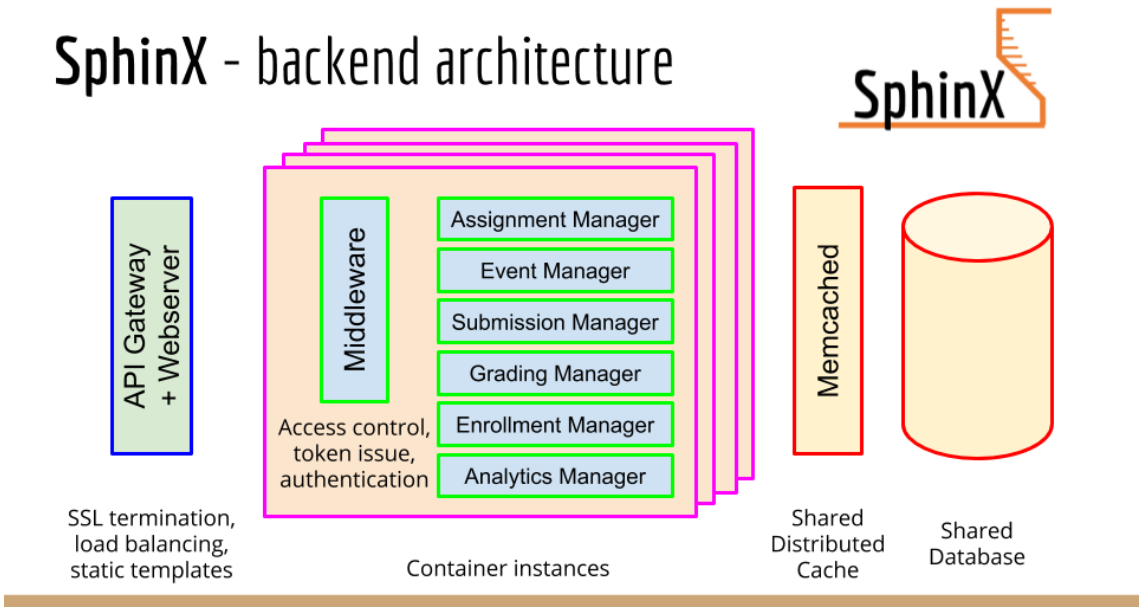


Figure 6.1: An Overview of the SphinX Architecture

6.1 SphinX Architecture Quality Attributes

Scalability and Portability. This is a measure of the ability of a system to handle increased workload without altering its own basic architecture. Scalability can be achieved notably by either horizontal scaling or vertical scaling. Horizontal scaling involves adding parallel machines that can distribute user requests among themselves. In vertical scaling, the resources in a single machine, typically CPU, RAM, Storage, or network devices are upgraded to increase the capacity of the application.

SphinX utilizes containers to enable horizontal scaling as well as portability. Containers are light-weight virtual environments that help in packaging, shipping, and running applications. A separate set of resources such as memory, CPU, disk, etc. is allocated to each container. Containerization helps immensely in scaling an application horizontally. Any number of instances of a container can be created with virtually no creation overhead and very little operational overhead.

The core logic of SphinX resides within containers of which multiple instances can be created dynamically and registered with a load balancer. SphinX uses Docker containers and all its business logic web App, middleware are installed within containers. Systemwide components such as MySQL Database, API Gateway, Nginx server, distributed store (memcached) are also hosted on (separate) containers. To scale horizontally, we need only replicate the dockers containing the business logic of SphinX and register the same with the load balancer. These dockers communicate through a bridge network driver supported by these containers.

Containers also enable SphinX to be extremely portable and be installed on systems with vastly different architectures and OS configurations. Vertical scalability can be achieved in SphinX by efficient use of available hardware and smart caching strategies. Increasing the number of cores, speed of CPUs would increase the number of concurrent processes that SphinX can handle.

Security

Since users are expected to store various course related documents on the platform, it is crucial to prevent unauthorized access to these, either to users registered with the system or external intruders. It is also incumbent on the platform to safeguard users from common attacks such as XSS, CSRF attacks etc. SphinX adopts best practices, often taken from the Open Web Application Security Project (OWASP)[1] project. In particular, it adopts the following security protection measures.

1. **Login:** Authentication is mandated for every user requesting access to resources on the platform. SphinX maintains a session for each user logged into the system and every user

[1] `https://www.owasp.org`

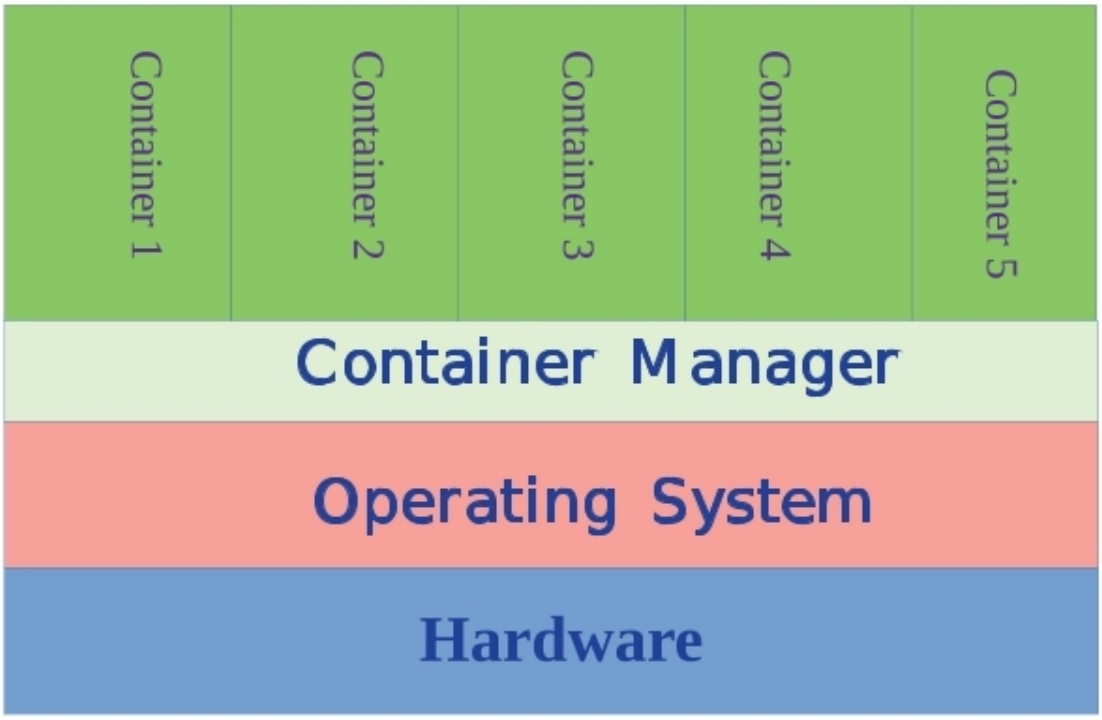


Figure 6.2: Container Overview

must send the session cookie to access resources on the server. SphinX relies on the same-origin-policy implemented in modern browsers to safeguard against false GET requests being used to exfiltrate user data from the system.

2. **Cross Site Scripting (XSS):** This is a security vulnerability commonly found in web applications that accept user input that may then be displayed to other users. This potentially allows malicious scripts to be injected into benign and trusted websites. SphinX adopts aggressive input sanitization and output encoding to safeguard users against such attacks. All data received or sent by the backend is encoded in the URL-safe base 64 scheme which prevents it from being accidentally interpreted as code. The same origin policy in modern browsers also prevents malicious scripts from one domain getting access to the sensitive data from other domains.

3. **Cross-site Request Forgery (CSRF):** This attack can force the victim to unknowingly change the state of the server and in case of SphinX, can cause fake regrading requests to be raised, or in case of an instructor-level account being compromised, can even lead to question papers getting leaked before an examination, or compromise the entire class data. SphinX prevents this by requiring CSRF access tokens with all HTTP methods that change the state of the server i.e. POST, PUT, and DELETE.

   The frontend is required to request a CSRF token before making a POST, PUT, DELETE

request (this is not needed for GET requests since the same origin-policy protects data sent as a result of a malicious GET request from being actually transferred to a malicious script operating on another domain). Now, it is commonly advised that CSRF tokens be refreshed. However, if we refresh the token upon each request then this may open the system to a denial-of-service attack wherein an attacker can keep asking for CSRF tokens and invalidating the old ones thus making it impossible for the real user to make any POST requests at all.

SphinX prevents this by using a timeout for CSRF tokens. Once a session makes a CSRF token request, it is stored in the backend distributed store along with the time of issue of the token. When the session asks for a token again before the timeout, the same token is returned. If asked after the timeout, a new token is generated, updated in the store, and returned. This method may still suffer from the token invalidation attack but not badly if the timeout is reasonably large.

4. **Clickjacking:** When an attacker uses transparent or opaque layers to trap a user into clicking a link or button, unintentionally leads to revealing secret data or changing the state of the server. SphinX avoids this attack by adding "X-Frame-Options" in the response header which will prevent clickjacking in most modern browsers.

5. **SQL Injection:** This is a type of attack where an attacker exploits an opportunity to provide input to the system that is stored in a database, to execute arbitrary SQL code on the database. SphinX backend prevents SQL injection by sanitizing all user parameters used in an SQL query. Database drivers supplied alongwith modern development frameworks like Django are utilized for escaping user-controlled parameters automatically.

Access Control. SphinX maintains a list of roles and permissions that have been specified by the instructor for the course. Every user in the course has restricted access to the course material as well as course actions depending on their role. SphinX denies all but system administrators from creating instructor level roles i.e. an instructor cannot create another instructor. All these safeguards try to ensure that even if an account is compromised, the system prevents them from performing any actions above their clearance level as specified by their role.

Performance. The SphinX backend offers only a minimalist API and offloads view composition, which can be time consuming, especially if done for dozens of requests per second, to the front-end. This brings down the response time of the system. It is advised that SphinX be installed on a local server, thereby reducing the network overheads. The SphinX front-end is built on top of an MVC framework that calls services at the back-end and receives JSON data as responses.

Availability. The SphinX platform has no single point of failure. The API Gateway, web application , memcached store, and database all run in seperate dockers, whereas the application and core business logic of the system resides in a cluster of containers, the cluster being as large as demanded by horizontal scaling. If any container node stops sending responses, the loadbalancer automatically starts redirecting requests to other nodes in the cluster, before attempting to bring up the non-responsive node.

Maintainability This denotes the ability of the system to support code and structural changes. The SphinX platform exposes more than hundred API calls to perform user's tasks. These API routines are modularized into various managers namely Authentication Manager, Course Manager, Assignment Manager, Event Manager, Submission Manager and Gradesheet manager. The code is written in a modular manner and is easy to extend and modify. We will discuss this further in the application component view section.

Usability. Since SphinX exposes a largely REST-ful API to the frontend, users can integrate their own front-end with the SphinX platform. The frontend can be developed using any language or framework independent of the back-end and supports well established conventions in designing uniform resource identifiers for all resources on the platform, making it easy for users to compose requests at the frontend. SphinX also offers its own user-friendly Graphical User Interface (GUI) frontend to end users. SphinX can be used in two modes; a local mode where a user can have SphinX installed on their personal local system, and a server mode where SphinX can be installed on a more powerful server and be used by many instructors simultaneously.

6.2 Development View

Web Services End Point. The following describes some general development principles adopted by SphinX

- Model View Template (MVT) architecture is used to implement services at the back-end. Controllers are placed in view files. The routing mappings are declared in a designated file called urls.py file at the root folder. The global configuration, including database, cache, are placed in a separate file called setting.py file placed in the root folder.

- Every Manager package includes a serializer, view, model, and util python files or packages. Views are the controller in the application that take an HTTP request and return a response to the client. Inside Views various **GET, PUT, POST** and **DELETE** methods are declared. The view is analogous to the controller in MVC terminology.

- Serializers allow complex database querysets and models to be converted into native python datatypes that can be efficiently rendered into JSON for communication to the frontend. This also supports deserialization of data where parsed data is converted back into model instances after validating the input data. Serializers are used to create new models or to update models. Field validations are also done in serializer classes.

Middleware. The following logic is implemented in the middleware or validation layer. These checks are performed before any request is processed.

- If a URL does not have a method, for example, GET, defined over it, calls to that method over that URL result in a **NOACCESS** error.
- For all URLs except /, **/login/**, **/password/reset/**, and **/password/reset/confirm/$userid-token**, if this session is not logged in, return a **NOLOGIN** error and finish (a user must be logged in if they wish to access **/logout/**).
- For all non-GET actions, the system must perform a CSRF check and if CSRF does not match, return a **NOCSRF** error and finish. For all URLs that are of the form /some/$some_id (e.g., some = course, topic, role, etc.), if a row with some_id does not exist (or if it exists but has been soft-deleted), return a **NOEXIST** error and finish. For all URLs that contain /course/$courseid, do the following checks
  - If this user is not enrolled in that course, return a **NOACCESS** error and finish.
  - Before processing all non-GET requests (i.e., PUT, POST, DELETE), check if courses.is_active is true or not. If not active, return a **NOEVENT** error and finish.
  - If this user is enrolled and the course is active, but the user is not permitted to perform this action (**GET, POST, PUT, DELETE**) on this URL, return a **NOACCESS** error and finish.
  - For all URLs that contain /event/$eventid, if this user has a blocking sub-event going on, deny all (**GET, POST, PUT, DELETE**) requests to any other event or any other subevent other than the blocking subevent. To do so, simply return a **BLOCK** error and finish.

Database. SphinX utilizes a MySQL database as its persistent store.

- SphinX utilizes two different schema: the **Global Schema** that is common to all the courses. This stores user accounts and course details. On the other hand, the **Course Schema** stores all the course related data. One of the main challenges faced in this structure is maintaining foreign key references between two different databases, i.e., Course and Global. **MySQL**

support foreign key reference between different database if they are hosted on the same SQL server.

- SphinX stores audit information for every row of the database, including the creation time, the userid for the user who created this row, the last update time, and the user id for the user who last updated this row.
- The schema has separate tables for logs. Course-related logs are logged into the local log table, and there is a global log table that logs overall application related requests, application details, status code, error code, request payload, etc.
- **PUT/POST** requests are never allowed to specify the primary key (id) or else creation/update time or created/updated by fields of any table. They are also never allowed to specify or else update foreign key values in the table (except rare exceptions). All of these are automatically updated/created by the back-end scripts. If a malicious **POST/PUT** request tries to, for example, supply the creation_time of a row, the backend logic simply ignores it while processing that **POST** request. Similarly, a malicious request to update a foreign key is ignored as well.

DB Routing and Object Relation Model(ORM) . SphinX utilizes an ORM for safe and effective backend development.

- SphinX uses the Django ORM for mapping complex data from the database to the model classes defined in the project environment. This ORM comes with an automatic SQL generator engine, that dynamically generates SQL queries as well.
- Every model in the application is associated with at least one manager that is used to query object related database tables. Each course maintains a separate schema in the database. Course resource requests need to be routed to the requested course schema. We use a database router that routes the database requests based on the course id present in the **HTTP** request parameter. All course related requests must contain the course id.

Application Logs. Application logs are used to identify any issue and monitor application health status in production. The various types of response status codes are given below.

Application Logging Structure	
Status Code	**Description**
SUCC	Success - no problems in processing request
NOUSER	Username is wrong or user does not exist
NOACTIVE	Account is **deactivated**
NOKEY	The **key/password** supplied is **wrong**
NODUP	An attempt was made to create a **duplicate entry** (for example, two roles with same name, two topics with same name or two question papers for the same **question_set**, or two submissions from the same student) which is not allowed (since SphinX supports soft deletes, the UNIQUE constraint in MySQL could not be utilized to do this automatically).
NOLOGIN	Requested operation requires **login**
NOCSRF	Requested operation requires a **valid access token**
NOEXIST	The resource being requested or modified either does **not exist** or has been **deleted**
NOACCESS	No **permission** to perform the requested operation
NOEVENT	No **event** going on at the moment that permits the requested action (e.g. student attempting submission after deadline)
BLOCK	Due to a **blocking examination event** going on at the moment, access to other events has been temporarily blocked. Access to these events will be restored after this blocking event is over
NOVAL	The data entered represents **illegal values** (to handle manufactured POST requests e.g. student assigning page negative 2 to a question)
NODEL	The deletion operation requested is not allowed as it will create **inconsistencies** in the database

Table 6.1: Application Logs Management

6.3 Framework, Libraries and Tools

SphinX utilizes the following technologies to implement its various components

1. Web Server and Gateway: NGINX 1.13.6.2 web server + openresty/1.13.6.2(LuaJIT)
2. Business Logic and API: Django 2.1.5, Django Rest Framework 3.9.1, Gunicorn 19.9.0 application server
3. Backend and Distributed in-memory store: MySQL and Memcached
4. Apps: Keras, TensorFlow
5. Containers: Docker
6. Frontend: Angular JS, JQuery, Bootstrap
7. Frontend apps: DropzoneJS, imageviewer, imageAreaSelect

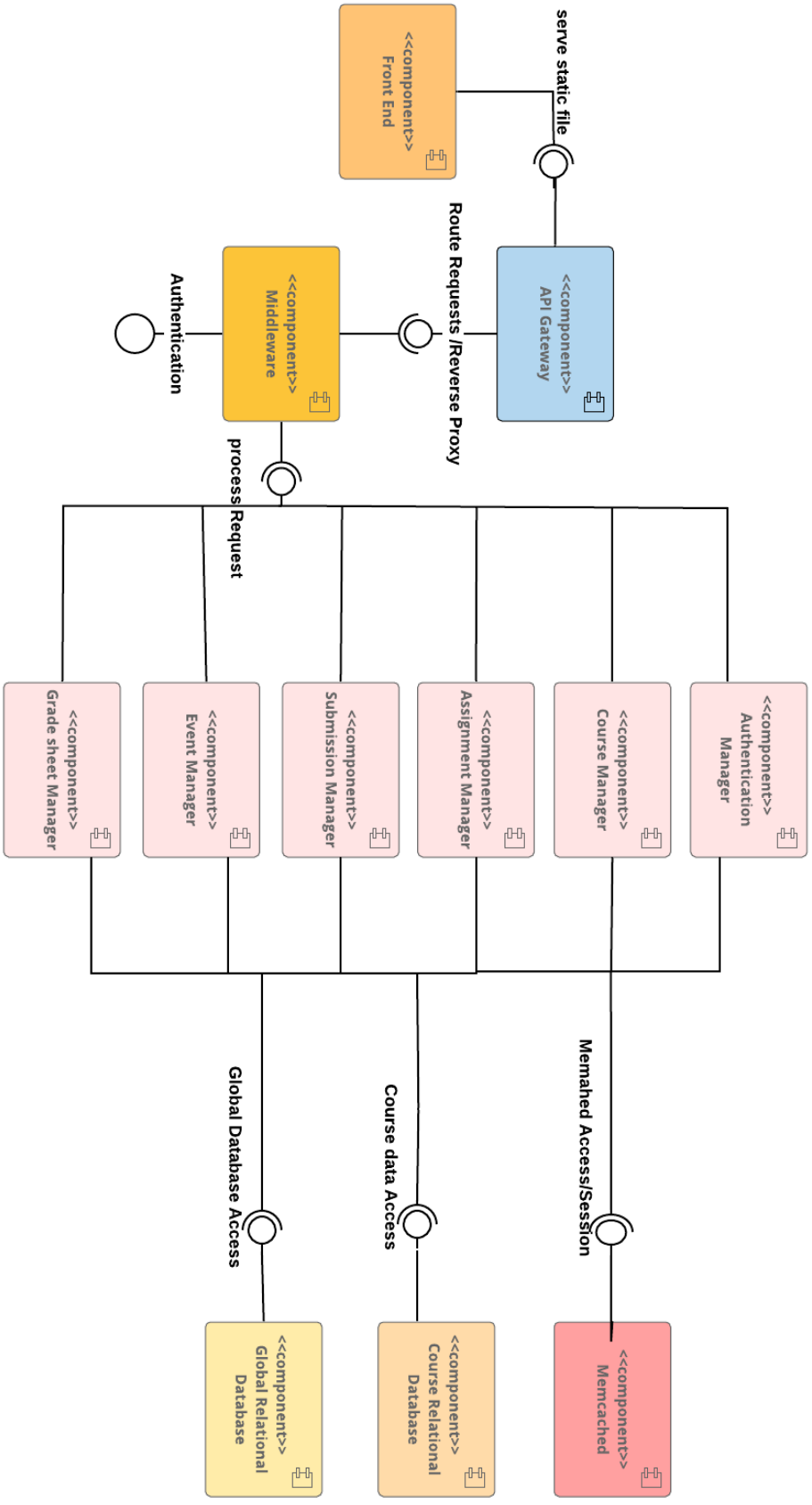


Figure 6.3: SphinX Component View

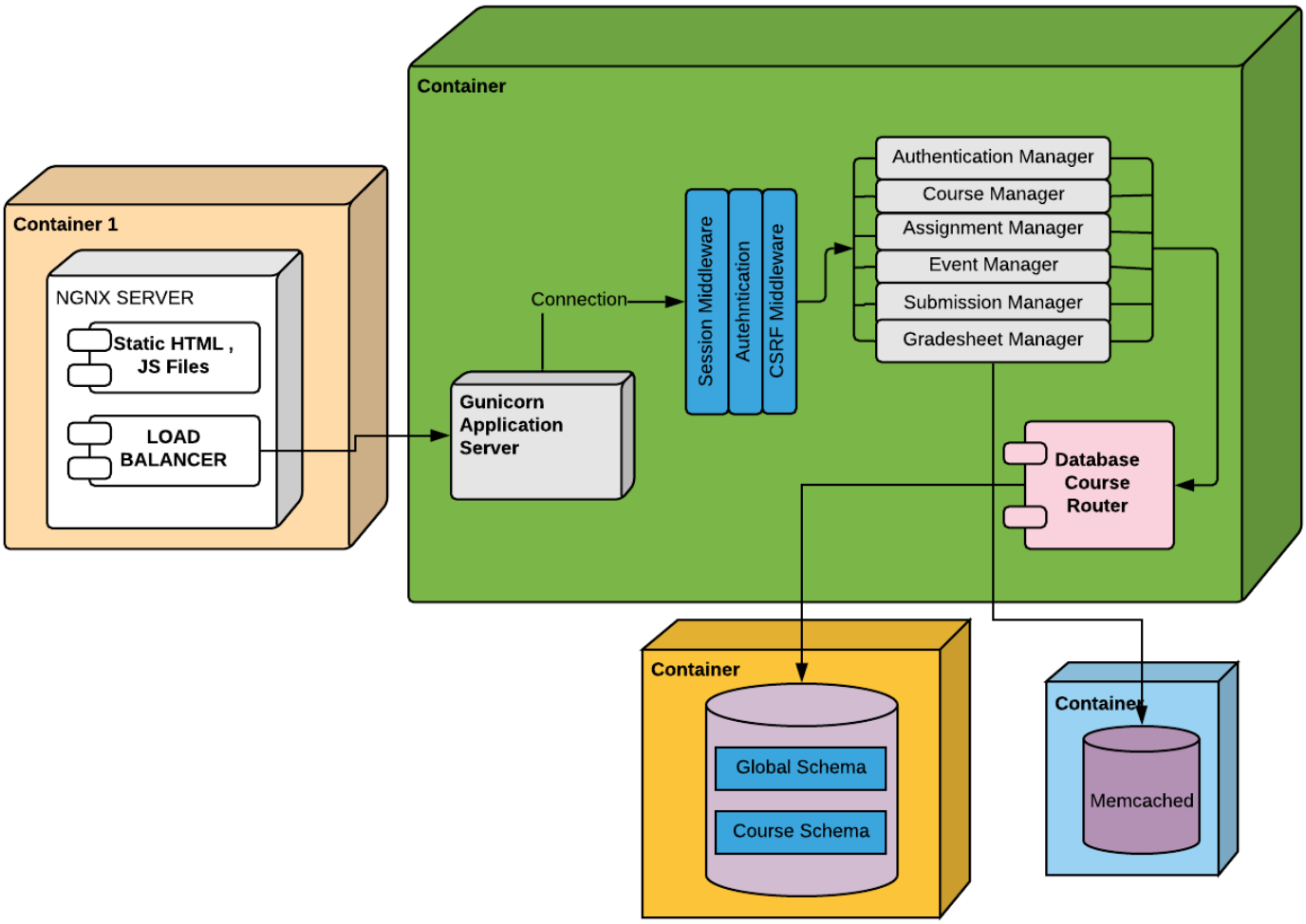


Figure 6.4: SphinX Deployment Diagram

Chapter 7

Conclusion

In this work we reported SphinX, a scalable and flexible system for managing and executing grading tasks for large courses. There is plenty of scope for building applications using advanced machine learning techniques that increase users throughput by simplifying various tasks and preventing users from making mistakes.

In particular, some of the future directions of work being pursued at the moment are

1. Enable onsite assignments i.e. assignments where students enter solutions to the questions on the website itself using a browser.

2. Build applications for AI-based grading or assistive grading, and AI-based plagiarism detector.

3. Use multi-model inputs to improve the accuracy of the submission mapper described in Chapter 5.

4. Introduce an analytics layer into the SphinX back-end to not only perform system monitoring and anomaly detection, but also generate helpful analytics for instructors and students based on their interactions with the system.

5. Introduce support for third-party apps to be written for SphinX in a secure manner.

6. Develop a mobile front-end for SphinX for the Android and/or iOS platforms.

Bibliography

[1] Blackboard, https://www.blackboard.com.

[2] Coursera, https://www.coursera.org/.

[3] Crowdmark, https://crowdmark.com/.

[4] Google Classroom, https://classroom.google.com.

[5] Grad-it, https://www.edx.org/.

[6] mooKIT, https://www.mookit.co.

[7] NPTEL, https://nptel.ac.in/.

[8] Michel Cesar and Rajjan Shinghal. An algorithm for segmenting handwritten postal codes. *International Journal of Man-Machine Studies*, 33(1):63–80, 1990.

[9] Rajdeep Das, Umair Z Ahmed, Amey Karkare, and Sumit Gulwani. Prutor: A system for tutoring cs1 and collecting student programs for analysis. *arXiv preprint arXiv:1608.03828*, 2016.

[10] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000.

[11] Andre G Hochuli, Luiz S Oliveira, AS Britto Jr, and Robert Sabourin. Handwritten digit segmentation: Is it still necessary? *Pattern Recognition*, 78:1–11, 2018.

[12] Jatin Jindal and Swaprava Nath. Truthful peer grading with limited effort from teaching staff. *arXiv preprint arXiv:1807.11657*, 2018.

[13] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[14] Sagar Parihar, Ziyaan Dadachanji, Praveen Kumar Singh, Rajdeep Das, Amey Karkare, and Arnab Bhattacharya. Automatic grading and feedback using program repair for introductory programming courses. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pages 92–97. ACM, 2017.

[15] Arjun Singh, Sergey Karayev, Kevin Gutowski, and Pieter Abbeel. Gradescope: a fast, flexible, and fair system for scalable assessment of handwritten work. In *Proceedings of the 4th (2017) ACM Conference on Learning@Scale*, pages 81–88. ACM, 2017.

[16] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.

[17] Keith Topping. Peer assessment between students in colleges and universities. *Review of educational Research*, 68(3):249–276, 1998.