SPHINX-II An AI-powered Pedagogy Helper Application

A Thesis Submitted

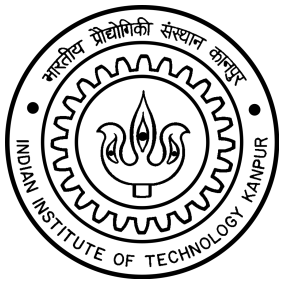
In Partial Fulfillment of the Requirements

For the Degree of Master of Technology

by

Shailesh Vishweshwar Nandkule

18111063



to the

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

June 2020

Page intentionally left blank

Declaration

This is to certify that the thesis titled **"SPHINX-II AN AI-POWERED PEDAGOGY HELPER APPLICATION"** has been authored by me. It presents the research conducted by me under the supervision of **PURUSHOTTAM KAR**.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgments, in line with established norms and practices.

Name: Shailesh Vishweshwar Nandkule (18111063)
Program: Master of Technology
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, Kanpur, 208016.
June 2020

Page intentionally left blank

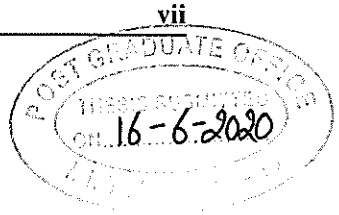
Declaration (To be submitted at the DOAA Office)

I hereby declare that

1. The research work presented in the thesis titled **"SPHINX-II AN AI-POWERED PEDAGOGY HELPER APPLICATION"** has been conducted by me under the guidance by my supervisor(s) **PURUSHOTTAM KAR**.
2. The thesis has been formatted as per Institute guidelines.
3. The content of the thesis (text, illustration, data, plots, pictures etc.) is original and is the outcome of my research work. Any relevant material taken from the open literature has been referred and cited, as per established ethical norms and practices.
4. All collaborations and critiques that have contributed to giving the thesis its final shape have been duly acknowledged and credited.
5. Care has been taken to give due credit to the state-of-the-art in the thesis research area.
6. I fully understand that in case the thesis is found to be unoriginal or plagiarized, the Institute reserves the right to withdraw the thesis from its archive and also revoke the associated degree conferred. Additionally, the Institute also reserves the right to apprise all concerned sections of society of the matter, for their information and necessary action (if any).

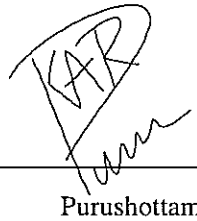
Name: Shailesh Vishweshwar Nandkule
Program: Master of Technology
Department of Computer Science and Engineering
Roll No.: 18111063
Indian Institute of Technology Kanpur, Kanpur, 208016.
June 2020

Page intentionally left blank



Certificate

It is certified that the work contained in the thesis titled **"SPHINX-II AN AI-POWERED PEDAGOGY HELPER APPLICATION"** by **SHAILESH VISHWESHWAR NANDKULE** has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



Purushottam Kar
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur, 208016.
June 2020

Page intentionally left blank

Abstract

Name of student: **Shailesh Vishweshwar Nandkule** Roll no: **18111063**
Degree for which submitted: **Master of Technology**
Department: **Department of Computer Science and Engineering**
Thesis title: **SPHINX-II An AI-powered Pedagogy Helper Application**

Name of thesis supervisor: **Purushottam Kar**
Month and year of thesis submission: **June 2020**

This thesis presents the design and fully deployed implementation of a secure, scalable and user-friendly pedagogy helper application. Building on previous work that designed the backend architecture of the SPHINX system, this thesis makes the following specific contributions:

Frontend Design. The thesis develops a user-friendly UI exposing all the various functionalities offered by the SPHINX backend is developed in the Angular framework. The UI pays close attention to proper information structuring, user experience, availability, scalability as well as visual design keeping in mind users with visual deficiencies.

Backend Augmentations. This thesis also makes important enhancements to the SPHINX backend. In particular, in order to improve the responsiveness of the backend, we implemented a set of customized middlewares to the existing backend architecture. The thesis also designs and develops a new set of APIs which aim to enrich the system by providing features related to assignment submission, performance evaluation and admin actions in the course.

Improved Roll-number Recognition System. This thesis improves the existing roll-number recognition system of SPHINX by substantially reworking the existing workflow of the system. In particular, on experiments with a large course with 200+ students, the proposed system offers absolute improvements in recognition accuracy of 10-15% as compared to the existing system and offering 90-95% recognition rates for most examinations.

AI-assisted Auto-grading. The thesis develops an end-to-end pipeline of AI-assisted tools fully integrated with the SPHINX platform. In particular, an image calibration tool designed to remove distortions from the scanned PDF documents is developed which greatly improves the performance of downstream AI applications. An AI-assisted auto-grading tool is also developed to grade true-false and MCQ-style questions. In particular, on experiments with a large course with 200+ students, our system offers autograding accuracy on true/false questions of 98.4%.

Page intentionally left blank

Acknowledgments

I would like to express my sincere gratitude to my supervisor Dr. Purushottam Kar, who encouraged me with his unparalleled guidance and support during all phases of this thesis. The door to his office was always open whenever I ran into any trouble or had a question about my work. Without his persistent help, the goal of this project would not have been realized.

I would like to thank Neeraj Kumar for sharing his knowledge of architecture of SPHINX system. His continuous availability to solve my doubts helps me to understand the system better without much difficulties. I would like to thank Amit Chandak for all the fruitful discussions and the valuable suggestions over the thesis work.

These two years are incomplete without friends, and here is cheerful thanks to every one of my friends, you have been of support in all ways, from academic help to just being there when I wanted to clear ideas.

Last but not the least, I would like to thank my parents and other family members for their unconditional support and love throughout my life. Without their support, it is impossible for me to complete my education seamlessly.

Apart from screenshots of the SPHINX system, diagrams depicting various modules and their components in various chapters were created using Google Drawings `https://docs.google.com/drawings/`.

This thesis was compiled using a template graciously made available by Olivier Commowick `http://olivier.commowick.org/thesis_template.php`. The template was suitably modified to adapt to the requirements of the Indian Institute of Technology Kanpur.

Shailesh Vishweshwar Nandkule
June 2020

Contents

List of Figures

List of Tables

Page intentionally left blank

CHAPTER 1

Introduction

Contents

The twin requirements of providing quality and scalable education over the internet have led to MOOCs (Massive Open Online Courses) increasing rapidly over past few years. The enhanced MOOC technologies enable us to deliver course content to thousands of students distributed over diverse geographical locations. As these platforms are accessible to vast and diverse audiences, such systems possess a huge potential of strengthening the way to educate people.

Although these systems are highly scalable by design, they present new challenges as well. The interaction between the students and the course administrators (instructors, tutors, graders) in such systems is seldom sufficient. The performance evaluation of the students is also challenging as grading the student work is hectic, laborious and time consuming task given the large number of students.

There have been several attempts made by various systems to solve the above mentioned issues. Examples of such systems are Moodle [15], GradeScope [20], Google Classroom [8], Canvas [5], and EasyClass [7]. Such systems mainly aim to handle course assignments and providing assistance in evaluating student performance in the course. As these platforms focus on different subsets of problems, users face other difficulties while using these services (possibly in combination) to manage courses effectively. Some of these challenges are listed out below,

1. The users of such a system may be enrolled in several courses simultanenously, and possess different sets of responsibilities in different courses. However, several existing systems do not provide efficient privilege separation among the users.

2. As many of the systems are not open sourced, several such platforms charge a relatively high subscription fee per student in order to enable the access to their service.

3. Assignments are an important source of data in a course. Several of these systems handle assignments at a superficial level as they merely provide an interface to upload assignments as PDF files without storing the details of the corresponding questions, which results into poor management of the course assignments.

4. Evaluation of assignments is another key factor in any course as it turns out to be a very burdensome and agonizing task. Several systems do not provide capabilities to perform online grading; they just allow a grader to upload the total graded marks by evaluating the student work offline first.

5. Another drawback is many of these systems have user interfaces that are not very well organized or structured because of which users may find it difficult to use various system features.

To resolve the above, we present the SPHINX pedagogy helper application. This highly scalable, flexible and secure platform provides numerous enhanced features which enable the instructor to manage large courses effectively. The latest version of SPHINX reported in this thesis provides an AI-assisted, rubric-based grading system that enables course administrators to grade thousands of student submissions in few minutes (for objective type questions) and evaluate the student performance in the course. As system contains several features, SPHINX offers a highly flexible role management system that allows the instructor to control user access to various non-admin actions at a very fine grained level by creating new customized roles such as 'Grader', 'Tutor', 'Student', 'Auditor' etc in the course. Using the well designed scalable architecture, the open source system of SPHINX is capable of providing services to numerous courses with thousands of enrolled students. SPHINX provides a rich set of features to the instructor to configure various activities in the course as *events* in the course with the great customization.

The development of the SPHINX was initiated in [16] as a master's thesis work at IIT Kanpur. [16] offered the design and development of a scalable core database and backend architecture of the system, as well as expose the system backend functionality using a set of restful APIs. However [16] does not offer an equally scalable and user-friendly frontend. The work instead offers a proof-of-concept frontend merely designed to establish the proper working of the backend functionalities.

As the existing SPHINX backend offers numerous course management capabilities through the exposed APIs, designing and developing an efficient and user-friendly frontend is highly essential for the system to be widely usable. There are several important considerations to be made while building the user interface of such a large system. Some of the key factors are mentioned below:

1. **Visual Design:** The user interface must be designed by considering students with visually impairments. The UI usability is improved by strategically utilizing the elements like fonts, color schemes, icons, images over the frontend, for instance, selecting color schemes appropriately such that the system will also be usable for those with color blindness.

2. **Information Organization:** As SPHINX produces large amounts of data related to a course, the architecture of the information needs to be well designed to make it easily accessible to the users to complete various tasks on the platform. This includes processing, organizing, storing the data hierarchically by the frontend.

3. **User Experience:** As the SPHINX backend offers a large number of features, these features must be organized and delivered into self-contained sections such that user finds them easily accessible over the platform. The system involves the large amount of user interaction in order to complete various tasks in a course. The user interface must anticipate and provide the various possible ways a user can interact with the system.

4. **Availability:** The user interface must be designed for various possible resolutions such that the system can be available over various electronic devices like desktops, laptops, tablets, mobile phones, etc, without facing any distortions.

5. **Scalability:** As the user requirements change over time, the user interface needs to be developed so that it can easily accommodate new features and design components on top of the existing system.

1.1 Related Works

As discussed above, the rising popularity of MOOCs (Massive Open Online Courses) increases the demand on online platforms to manage course assignments and evaluate student performance in the course. In recent years, several systems have come up to fulfill these requirements. Some of these systems are listed below:

1. Moodle [15] was developed by the Australian computer scientist and educator Martin Dougiamas as part of his doctoral thesis. Moodle is a free and open source learning management platform, and enables the user to create a customized site to manage and release course content. Even though platform provides good customization, it does not offer grading features natively and also faces scalability issues.

2. Canvas [5] is an open source system launched by the Instructure Inc. in order to provide learning management services. The system was initially developed for high school level audiences, and was later upgraded to offer services to graduate level courses as well. Although it offers numerous features to manage MOOCs, the system merely provides basic user roles which are insufficient to implement privilege separation in large courses. The platform also does not allow users to work on multiple assignments in parallel.

3. EasyClass [7] is an another Learning management system which allows users to manage course material & assignments, distribute grades to students, and manage class discussions in the course. The platform does not provide grading capabilities and as such, graders need to explicitly grade the student work offline and upload the grades manually to the platform.

4. GradeScope [20] is built by a group of teaching assistants and tutors previously enrolled at the CS department at the University of California, Berkeley. The rubric based grading approach of the platform allows the instructor to quickly (manually) grade hundreds of user submissions. Gradescope gained popularity over a short period of time as a result of this efficient performance evaluation mechanism. However, in order to use various AI-based features, instructors need to pay subscription charges per student to use the platform services which may be prohibitive for large classes in resource strapped educational institutions.

As mentioned earlier in the chapter, several such systems come up with their own set of new challenges. Many of them lack efficient grading capabilities while others require high subscription fees. The SPHINX platform seeks to rectify several of these challenges and offer numerous features to manage MOOCs effectively.

1.2 Our Contributions

In this thesis, we design and build a secure, scalable, efficient, and user-friendly frontend for the SPHINX pedagogy helper application. Specifically, this thesis makes the following contributions.

Frontend Design. The user interface is developed by taking into account important key motives like proper information structuring, visual design keeping in mind users with visual deficiencies, user experience, availability, scalability, etc. In order to offer an accessible UI, we build the frontend architecture and several set of APIs.

Backend Augmentations. This thesis also makes important enhancements to the system backend that was originally proposed in [16]. While building the frontend to the SPHINX system, we observed some necessary modifications to the existing system backend. In order to improve the responsiveness of the backend, we implemented a set of customized middlewares to the existing backend architecture. These middlewares mainly corresponding to CSRF Token management, User Authentication and User Authorization in a course. To enhance the existing backend capabilities, we also design and develop a new set of APIs which aim to enrich the system by providing features related to assignment submission, performance evaluation and admin actions in the course.

Improved Roll-number Recognition System. The work of [16] also proposed a roll number recognition system to automatically link student submissions in an exam to their accounts. In this thesis, we improve the accuracy of the existing roll number recognition system as well by completely reworking the existing workflow of the system. In particular, on experiments with a large course with 200+ students, our system offers absolute improvements in recognition accuracy of 10-15% as compared to the existing system and offering 90-95% recognition rates for most examinations.

AI-assisted Auto-grading. With the help of various machine learning algorithms, in this thesis we also develop image calibration and AI-assisted grading tools which are then fully integrated with the SPHINX platform. The image calibration tool is a critical part of the system and aims to remove distortions from the scanned PDF documents. Not doing so properly adversely affects the performance of any autograder. Our proposed AI-assisted grading system helps the instructor to grade the user work efficiently. In particular, on experiments with a large course with 200+ students, our system offers autograding accuracy on true/false questions of 98.4%.

In the remaining chapters of the thesis we explain the above mentioned contributions to the SPHINX system in detail. In chapter 2, we briefly explain the key components of the framework which are essential to understanding the system frontend. In the chapter 3 & 4 we present the proposed frontend architecture of SPHINX and explain the key components of major UI modules. The remaining chapters of the thesis describe various backend enhancements and AI apps built over the SPHINX platform in great detail.

CHAPTER 2

A Brief Introduction to Angular Application Framework

Contents

Abstract *In this chapter we give a brief overview over the main key components and the architecture of the Angular development framework used to build the* SPHINX *frontend. In the key terminologies section, we try to familiarize the reader with the key components by explaining their role in the framework. We provide a brief overview of the various features and responsibilities of the key components. In the architecture section, we explain how the various key components communicate and work in the system in order to provide system features to the end user.*

2.1 Introduction

As the SPHINX system seeks to offer a large number of features, it is necessary to provide a graphical user interface which is user friendly and easy to use. A graphical user interface provides an efficient way to use a system by understanding it without facing many complexities. To design and develop a graphical user interface with such features, SPHINX uses the Angular framework [1]. This chapter containing sections explaining the architecture overview and the basic terminologies of this design framework in detail.

2.2 Key Terminologies

In order to understand the application architecture, acquaintance with the set of main terminologies of Angular is important. A brief overview over some of the important terminologies of the Angular framework is given below,

2.2.1 Module

Modules are the main pillars of an Angular application. An Angular system is divided into multiple modules, such that each module provides a specific set of capabilities of the system. The module is the main building block of the Angular application which holds various files together like the set of components, the service injectables, and the data models. It also loads various dependencies like external libraries, components and provides the compilation context to the files for execution. In the Angular framework, the system contains one app module and multiple feature modules. The app module is used to load and initialize the application while the feature module represents the block of files that provides the set of correlated functionalities corresponding to a specific feature of the application. The feature modules are usually kept independent of each other so that developers can easily focus on the development and enhancement of features without much overhead of managing a large codebase. A module can import components, library functions provided by other modules and can also make available its module functionalities for other modules.

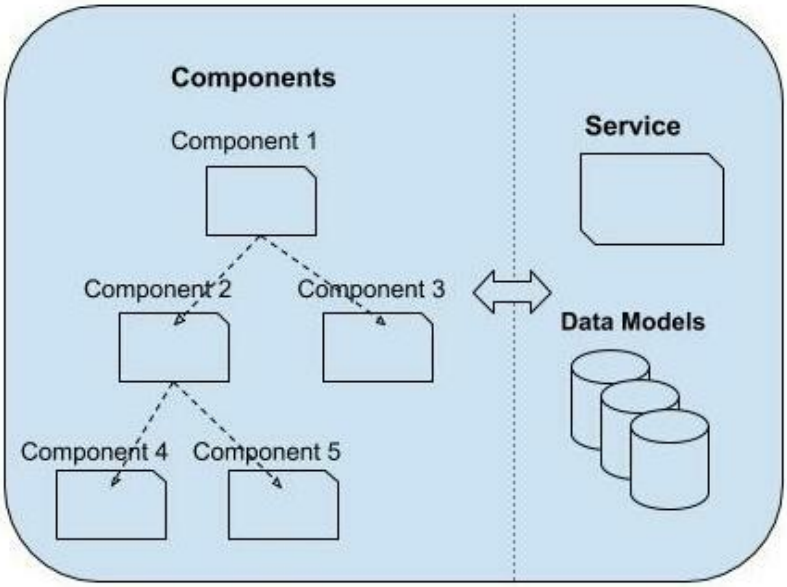


Figure 2.1: Module Overview

Each module has a set of metadata properties associated with it. Some of the properties are given below.
See the official Angular documentation of architecture modules [3] for more information.

2.2.2 Components

A component typically represents a page or a part of a page in the system using an associated template view. The template view is an HTML file that represents the design and view aspects of the page in the system. A component mainly contains the properties and various method APIs. The main responsibility of the method APIs of a component is to execute the necessary application logic to provide various capabilities of the corresponding application page / template view.

Property name	Description
Declarations	An array containing the list of components belonging to the module
Providers	An array containing the list of service injectable provided by the module
Import	An array containing the list of other external modules whose classes, components are used by the files in the module
Export	An array containing the list of components, directives, services which are allowed to be used by other modules

Table 2.1: Module Metadata Table

On the other hand, the properties hold the data values of the application which are processed by the component APIs and used by the template view to render the data on the page. The Angular framework provides a way to communicate data values between a view template and the component using the two-way data binding feature. Every component has a life cycle, such that Angular initializes the component when the template view gets rendered and destroys the component as the user navigates to other pages. The framework also provides various lifecycle hooks to perform necessary operations during various stages in the life cycle. A component may also contain various child components embedded in it such that they can be arranged in a hierarchy.
See the official Angular documentation of architecture components [2] for more information.

2.2.3 Service / Injectables

Injectables are classes that provide the functionalities common to a set of components in the module. The main responsibilities of the services are performing server communication, handling server response errors, processing server responses, etc. A service is made available to the components by injecting it during the loading of the application module. As services provide common functionalities and handle the server communication responsibility, they help the components focus on page specific functionalities and reduce code replication by avoiding redundancy.
See the official Angular documentation of architecture services [4] for more information.

2.2.4 Data Model

Model classes are mainly used for data handling in an application. The model represents the data objects used by the template views to render data on the pages. During server communication, often the server communicates large volumes of JSON data which is not present in a format readily interpretable by the module files. Thus, after fetching the application data from the server, the server response needs to be processed and stored in a format usable by the view templates and the components in the module. The data model provides the common entity format which is used by the module files to store and communicate application data between them.

2.3 Module Architecture

An application module mainly consists of a set of components, service APIs, data models, and external library packages. The module components follow an architecture pattern similar to MVC

architecture. The architecture mainly consists of three logical units, which are view, component, and service APIs.

- The view unit is responsible for providing a user-friendly graphical user interface which promises to deliver some application capability. The view renders the application data and provides user controls to perform some actions on the application page. Whenever the user performs any allowed action, the view triggers a call to the corresponding component API method. The view renders the data using the two-way binding, such that whenever the component updates the data properties on completion of the user action, the view data updates accordingly.

- The component unit mainly acts as a middleman between service APIs and the view unit and executes the necessary application logic. The major responsibilities of the component unit include handling user input, performing necessary service API calls, ensuring data availability for the view unit, server response data formatting, error handling, etc.

- The service unit contains API methods which mainly take care of all necessary server communication using RESTful API calls and return server responses to the component unit. The service unit ensures application data persistence at the server database for respective user actions by performing appropriate server API calls.

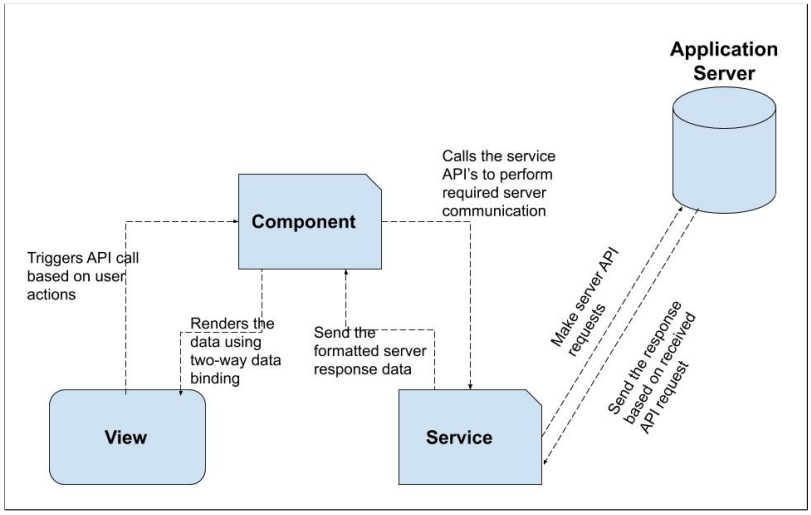


Figure 2.2: Component Workflow

CHAPTER 3

System Architecture

Contents

Abstract *In this chapter we present the frontend architecture of the* SPHINX *system in the detail. The architecture overview section provides the overall picture of the internal working of various logical units in the* SPHINX *frontend. In the next section, we offer brief explanation about each logical unit of the architecture. In the module subsections, we introduce various core feature modules of the architecture and explain various responsibilities of each module in the* SPHINX *system. The components and services subsections contain short introductions to the module containing important components & services, and also provide the actual view of the* SPHINX *system. In the application routing section, we explain the internal routing between various view pages provided by the* SPHINX *system.*

3.1 The Architecture Overview

The SPHINX frontend system is divided into several feature modules. Each feature module promises to deliver a set of capabilities corresponding to a specific feature of the system. The modules are developed in a modular fashion, such that all modules are independent of each other.

This modular approach provides scalability and maintainability to the system developer. Once the user enters a URL in the browser, the route guard validates the request and redirects it to the respective manager module. The corresponding module loads the application context in order to entertain the received request. Each module contains the application logic need to execute the user request. The modules use local storage to save user data corresponding to the system. The modules perform the necessary server communication using REST APIs to provide the system capabilities. The HTTP interceptor intercepts communication requests and handles server responses. The application server serves API requests by performing the necessary operations at the backend. This chapter explains the system modules and other logical units of the architecture in detail in various sections below.

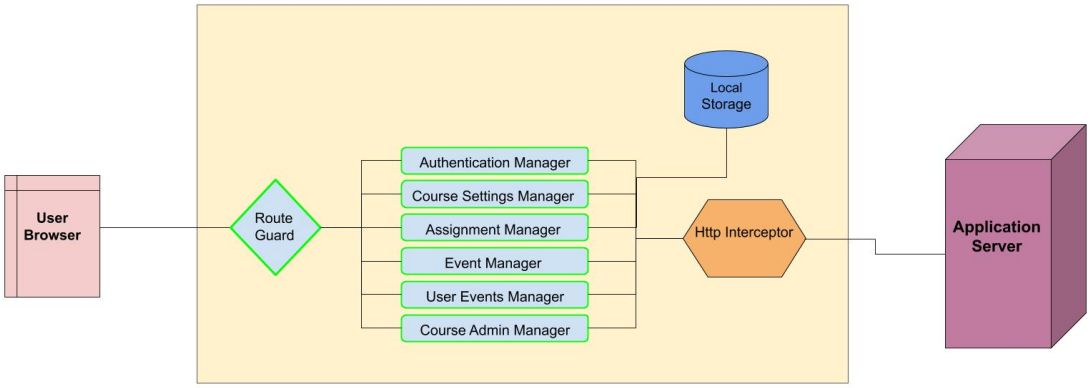


Figure 3.1: An overview of the The SPHINX Frontend Architecture

3.2 Logical Units of the Architecture

3.2.1 The Route Guard

The route guard controls the user access to the various modules provided by the system. Any user can move anytime to the routes corresponds to the authentication module, while other modules are allowed only to the logged-in user having required authorization. For every received route request, the guard validates user authentication using the local storage. If user authentication required, then the guard loads the login page of the system. If the user is logged-in, then the guard checks whether the user is authorized to load the requested route or not using the user assigned role. If the user is unauthorized, then it redirects the user to the course home page.

3.2.2 Local Storage

The system uses the local storage to store the application data to provide the various module-specific capabilities. The user session persistency is maintained with the help of the local storage. With the help of the local storage, the system reduces the server API calls and improves the user experience on the system.

3.2.3 The Http Interceptor

The interceptor acts as middleware between the system modules and the application server. It intercepts all the server API requests and modifies the request by adding the necessary request headers. The system uses the interceptor to provide CSRF token functionality and the user-authorization. The interceptor fetches the CSRF token value from the local storage and attaches it to outgoing API requests. The interceptor also handles the server error responses, It performs the authorization error handling and navigates the user to the course home page by showing an appropriate notification message.

3.3 Feature Modules

3.3.1 Authentication Manager Module

The user authentication is an essential aspect of the system as access to all other modules is restricted to only authenticated users. The feature module is mainly responsible for user authentication, session management, and the CSRF token management in the system. It contains the set of components, service injectable, and the models which together provides the capabilities like the user login - logout to the system, reset the user password using the system generated password reset token, the user session creation and deletion using the local storage, the CSRF token management required for the server communication. The module containing a set of components and service injectable are listed below,

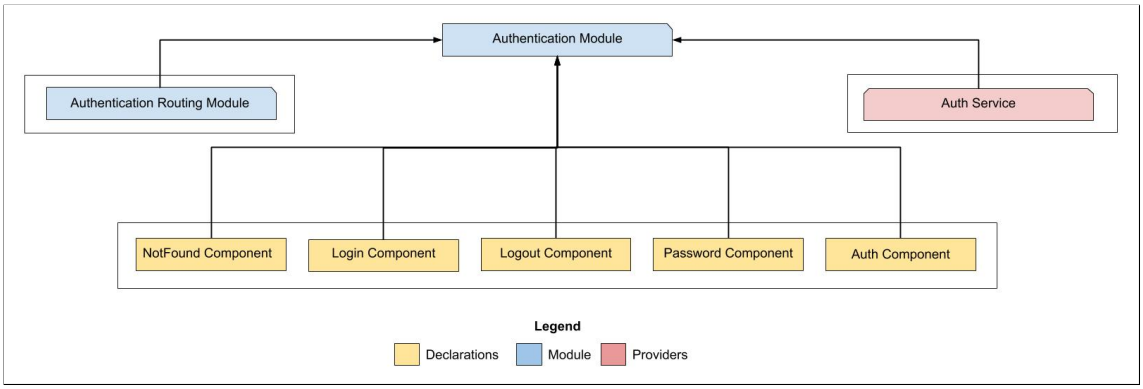


Figure 3.2: Authentication Module

3.3.1.1 Components and Services

The module containing a set of components and service injectable are listed below,

- **Authentication Routing Module:** The module contains the route configurations between the multiple pages of the authentication module
- **Authentication Service:** The injectable provided APIs handles all the server communication required to provide the authentication module capabilities

- **NotFound Component:** The component provides the view page to display when the application server is not reachable or down for the maintenance

- **Logout Component:** The component provided APIs make the user logout from the system and clears the session from local storage

- **Login Component:** The component corresponds to the login form view and the set of APIs required to perform the user login to the system

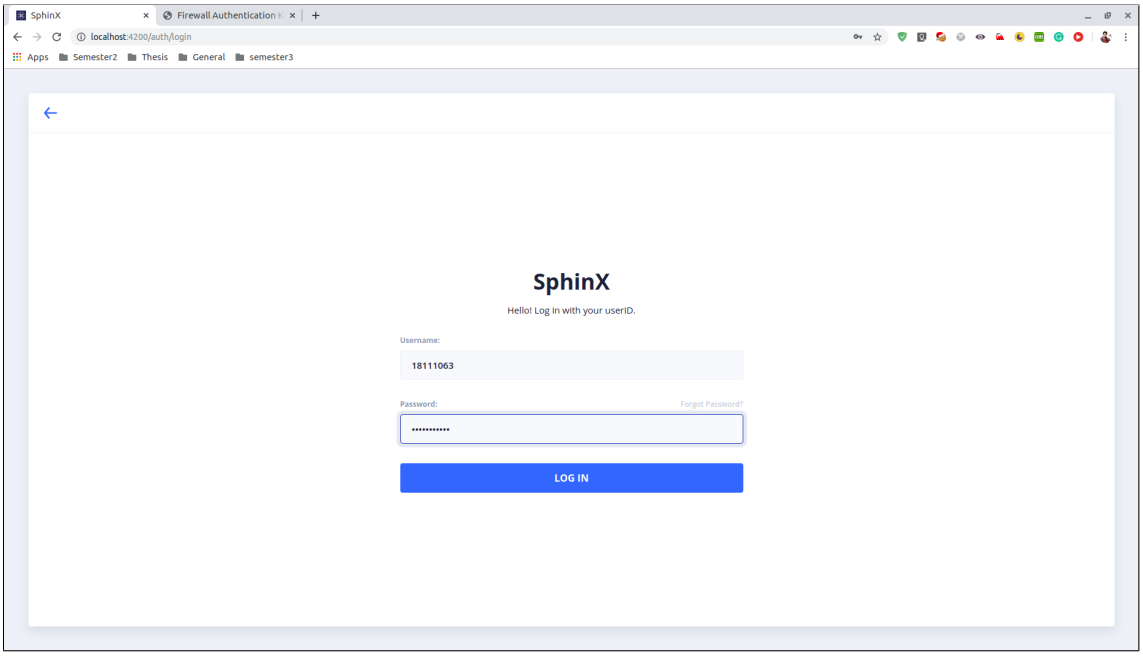


Figure 3.3: Login Component View

- **Password Component :** The component provides the forms and set of APIs to generate the password reset token and reset the password with the new password provided by the user

3.3.2 Course Manager Module

The SPHINX system provides a separate module to manage the various course-related settings. The module provides various capabilities to manage the course-related data like the course topics, the course sections, the user roles, the course enrollments, and the course details.

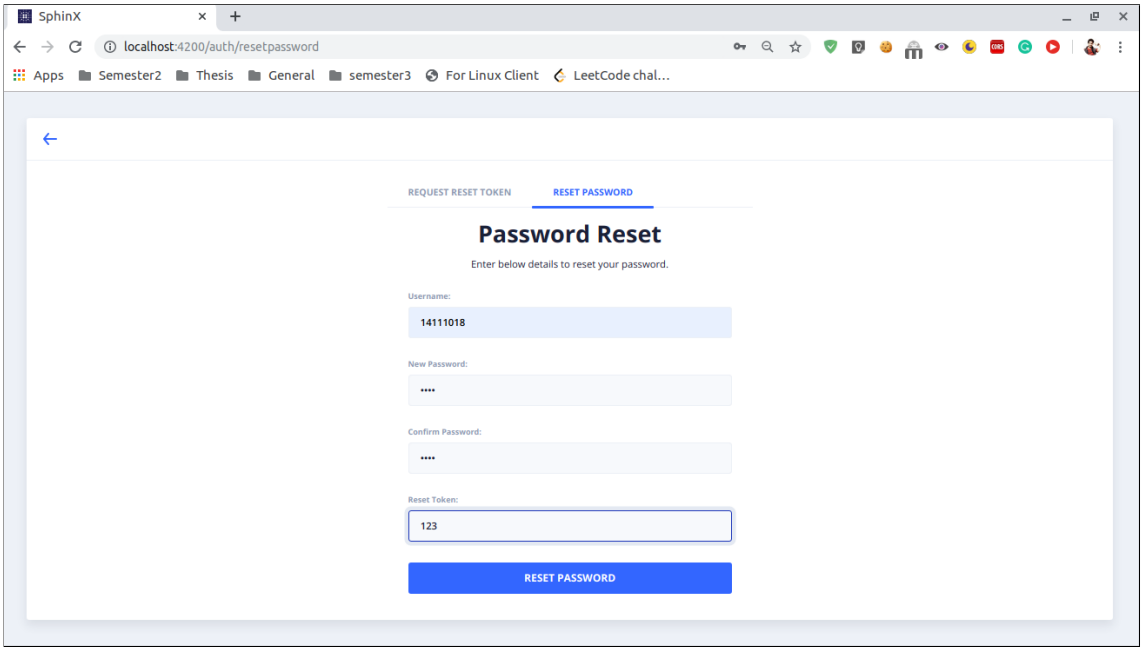


Figure 3.4: Password Component View

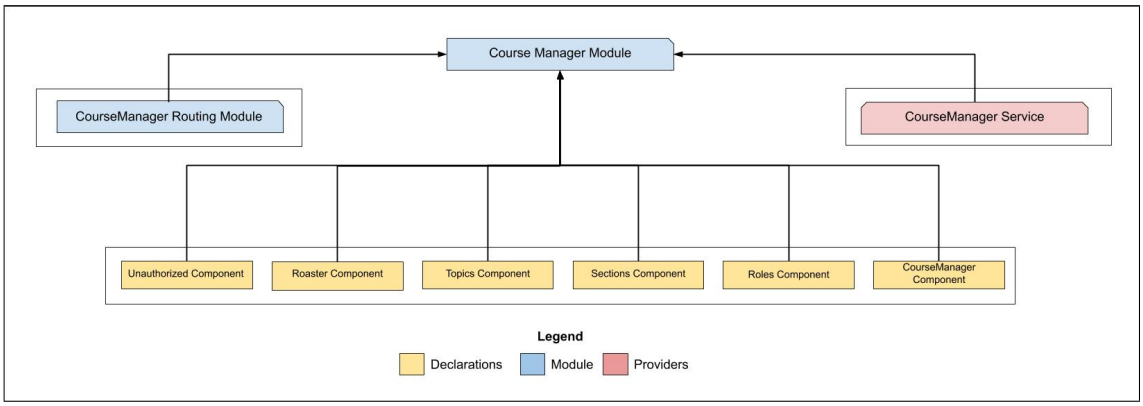


Figure 3.5: Course Manager Module

3.3.2.1 Components and Services

The module containing a set of components and the injectable as given below,

- **Course Manager Routing Module:** The module contains the route configurations between the multiple view pages of the course manager module
- **Course Manager Service:** The injectable provided APIs handles all the server communication required to provide the course manager module capabilities
- **Course Manager Component:** The component process the logged-in user data, then set the various navigation controls in the system using the user role and allowed actions
- **Unauthorized Component:** The component provides the view to display the error message to the user for the unauthorized action in the system

- **Roaster Component:** The component provides various capabilities using the view and the set of APIs to perform the course enrollment management. It allows us to add the list of users to course, edit the section and role of existing users, delete the existing users, etc.

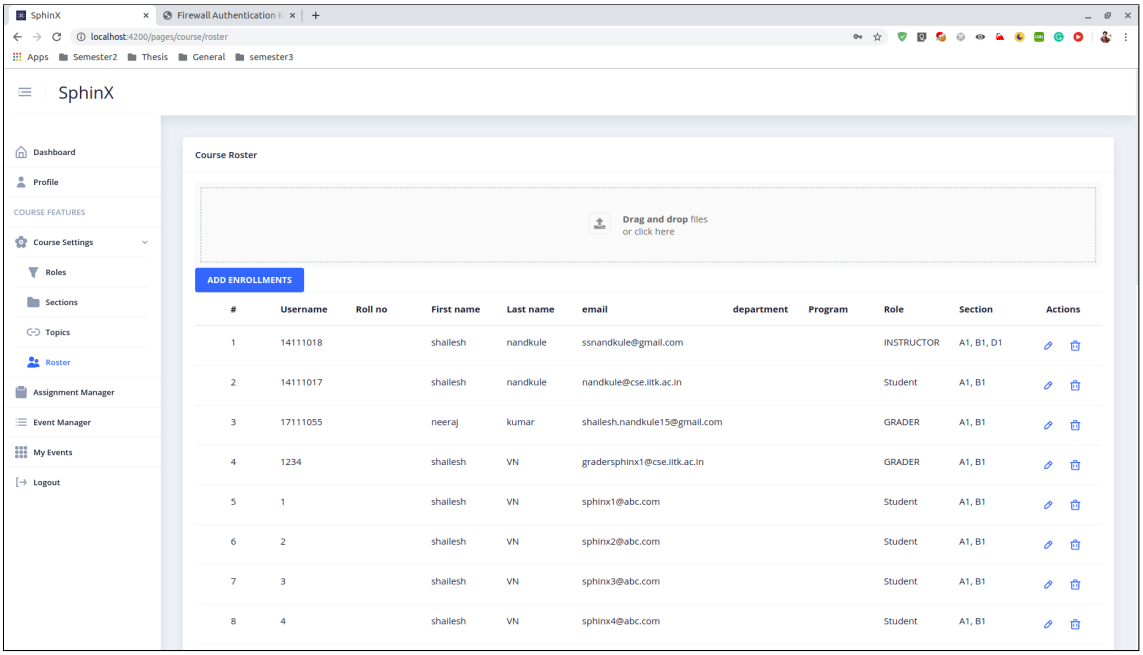


Figure 3.6: Course Roster Component View

- **Topic Component:** The component view displays the list of the course topics. The set of component API's provide capabilities to create the tree of topics by mentioning parent topic while adding the topics to the course. The component contains functionality to add, edit, and delete the course topics from the system

- **Section Component:** The component provides various capabilities using the view and the set of APIs to perform the course section management. It displays the list of existing sections and contains functionality to add the new section, edit / delete the existing course section from the system

- **Role Component:** The component list out the system provided various meta roles with the corresponding list of allowed user actions. The component view list out the existing course roles and provides functionalities to create/edit the role using various combination of the system provided meta roles, and delete the existing course roles from the system

3.3.3 Assignment Manager Module

The module is responsible for handling all the required operations to perform the assignment management in the course. It provides the view capabilities to view the course assignment data like the list of assignments, question sets, list of questions, and the set of rubrics correspond to each question in a well-organized manner. The module provides functionalities to create and manage the assignments and multiple sets of questions corresponding to them. It gives capabilities to create

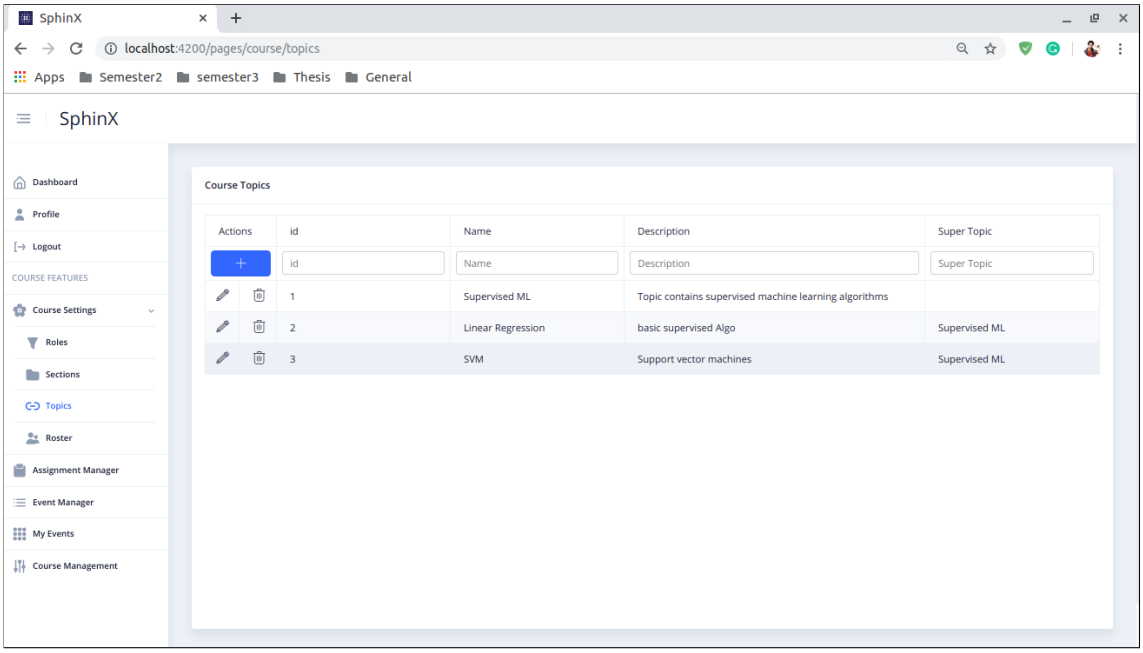


Figure 3.7: Course Topics Component View

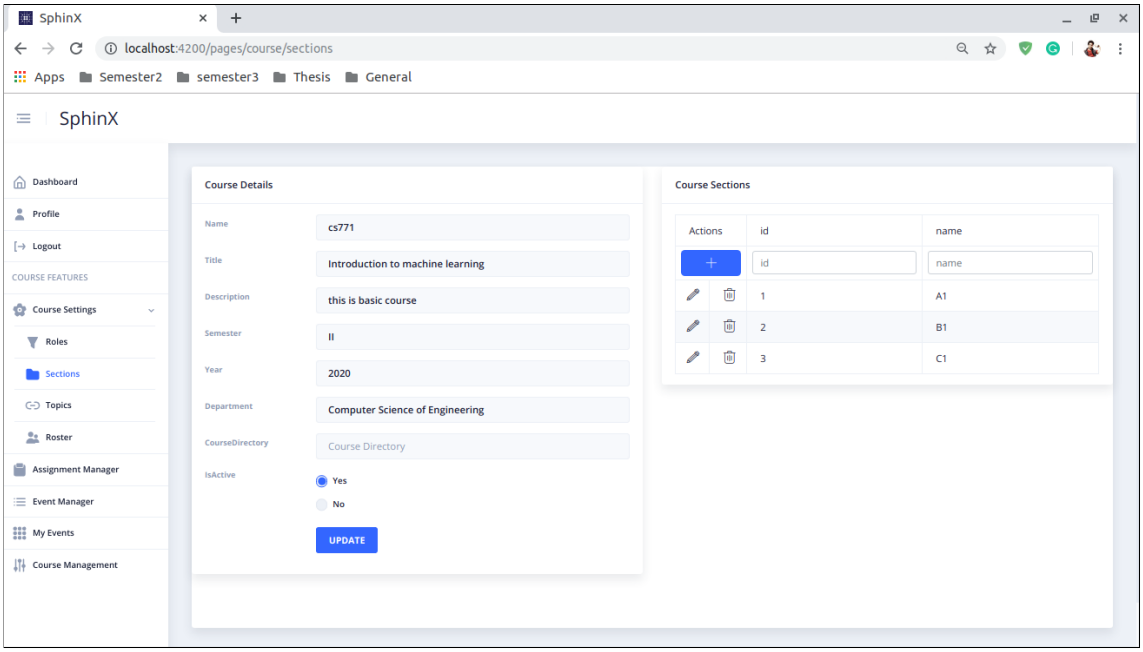


Figure 3.8: Section Manager Component View

and manage the list of rubrics used to grade the question for all submission copies corresponding to the assignment.

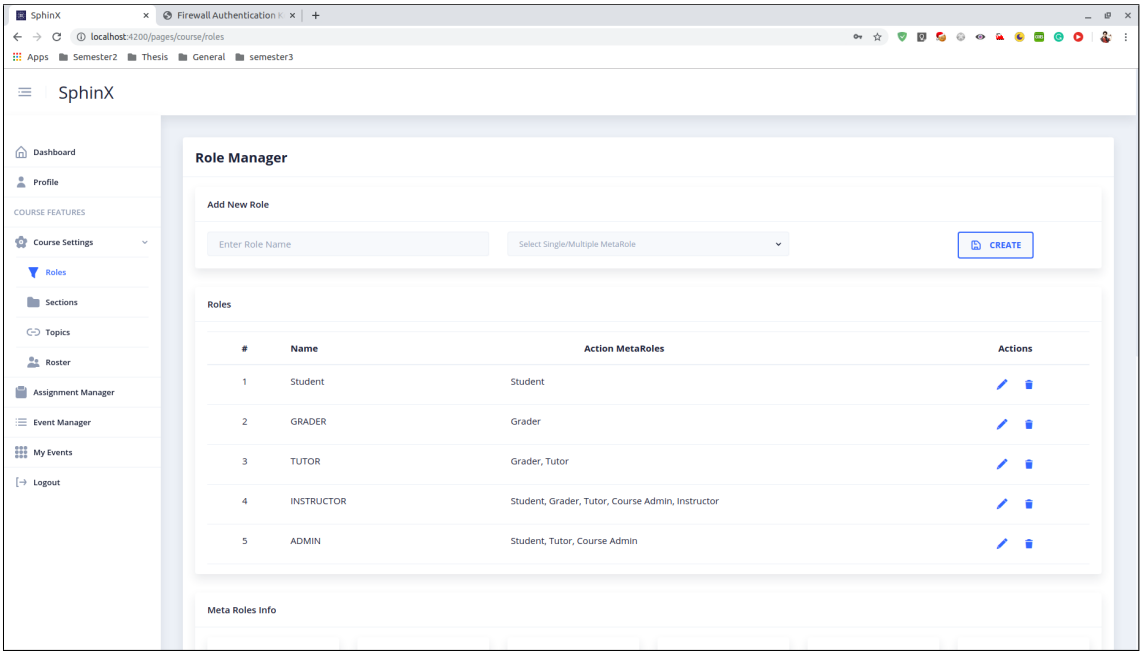


Figure 3.9: Role Manager Component View

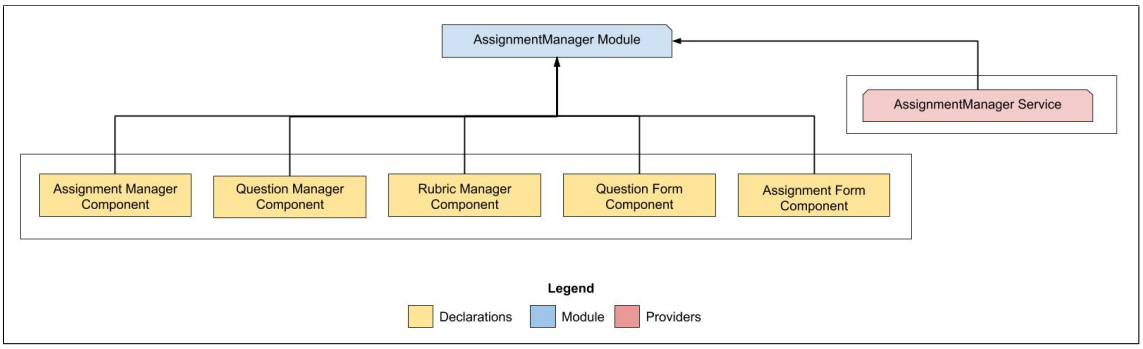


Figure 3.10: Assignment Manager Module

3.3.3.1 Components and Services

The module containing a set of components and service injectable are given below,

- **Assignment Manager Service:** The injectable provided APIs handles all the server communication required to provide the assignment manager module capabilities
- **Assignment Form Component:** The component provides the form and capabilities to the user to create/edit the Assignment in the course
- **Question Form Component:** The component provides the form and capabilities to the user to create/edit the question set within the course assignment
- **Assignment Manager Component:** The component contains the view and set APIs which provide capabilities like display all the course assignment and corresponding question sets in

a nested table, perform operations like create/edit/delete the assignment and corresponding question sets, navigation controls to the question manager and the rubric manager, etc.

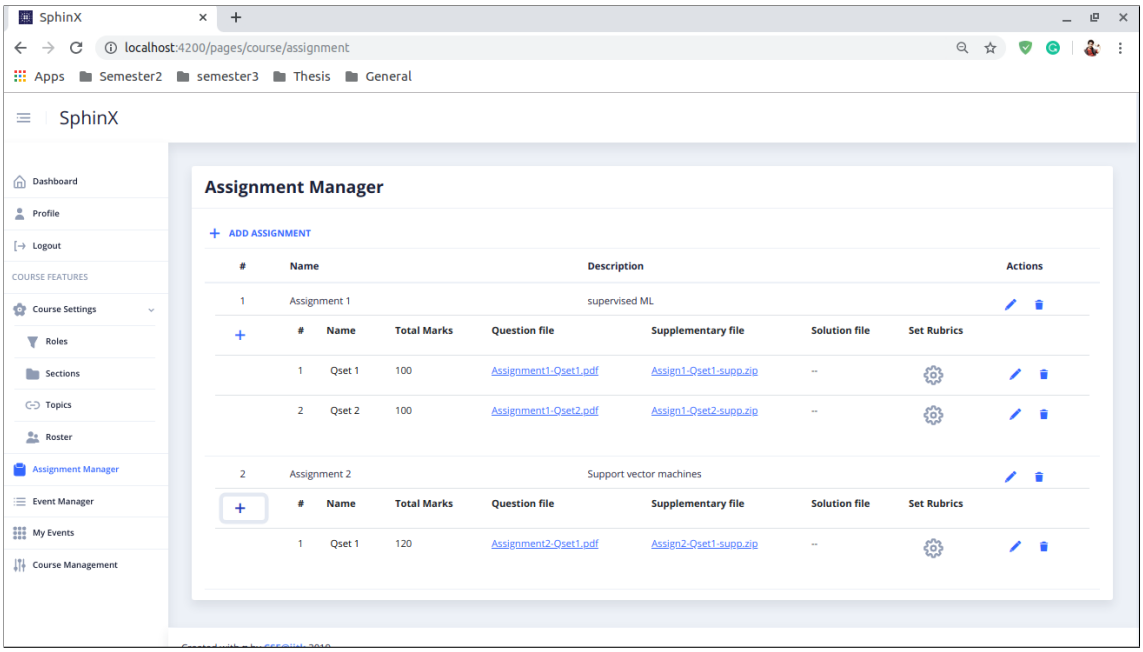


Figure 3.11: Assignment Manager Component View

- **Question Manager Component:** The component displays the question set file and provides capabilities to create the assignment questions in a hierarchical manner. The component provides functionalities to mark the box coordinates of the question and the roll number on the file pages, those coordinates later used by the system for the machine learning operations

- **Rubric Manager Component:** The component displays the question set file and provides the capabilities to manage the list of rubrics for each question in the question set

3.3.4 Event Manager Module

The module is mainly responsible for the event and subevent management in the course. The SPHINX separates the assignment related configuration from the main assignment data by providing the various types of events and subevents in the course. To release and manage the various configurations corresponding to various activities related to the assignment, the associated event & set of subevents need to be created in the course. The module provides various capabilities like view the existing list of events and corresponding subevents in the course, create multiple events and corresponding subevents of various types provided by the system, modify the time window to perform various activities related to the assignment, delete the existing events and corresponding subevents in the course, etc.

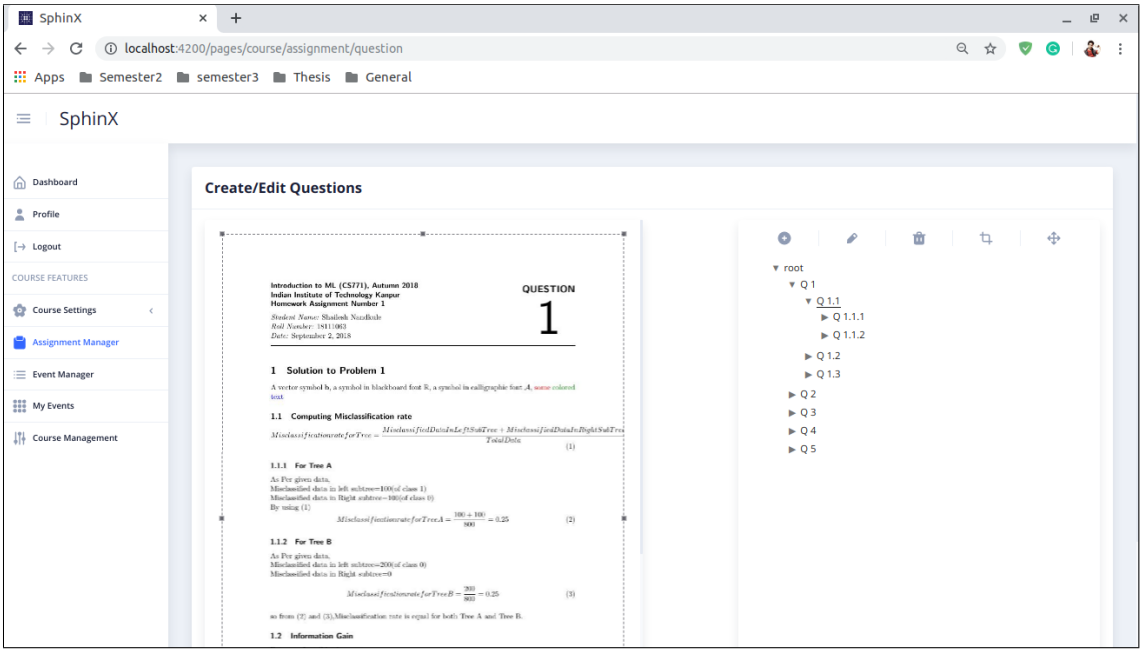


Figure 3.12: Question Manager Component View

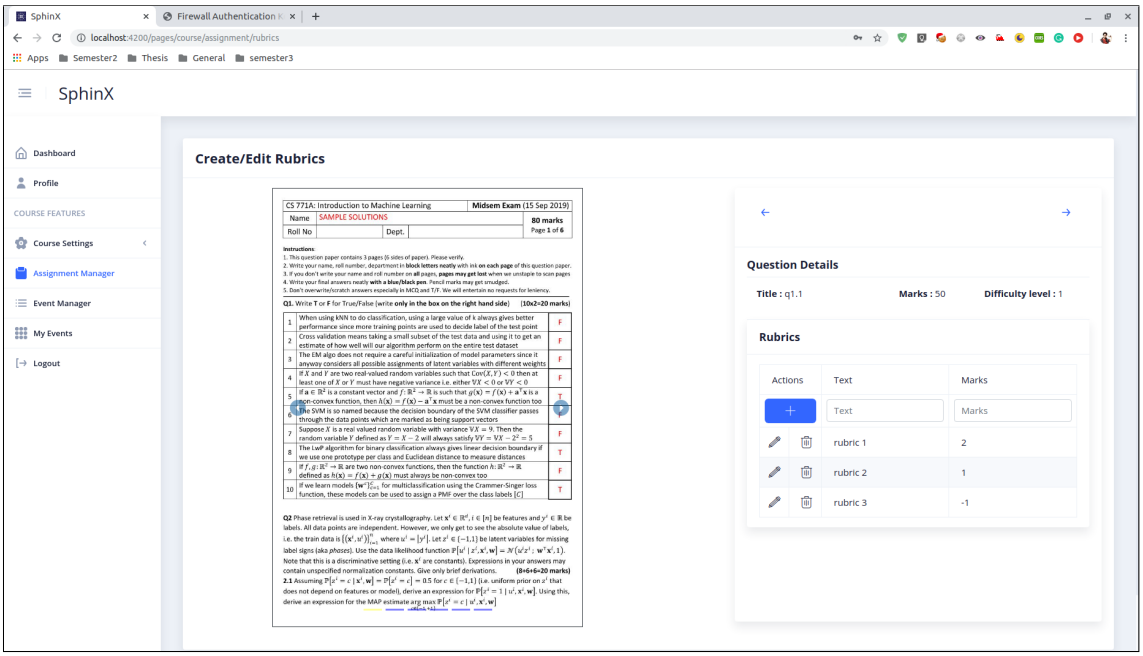
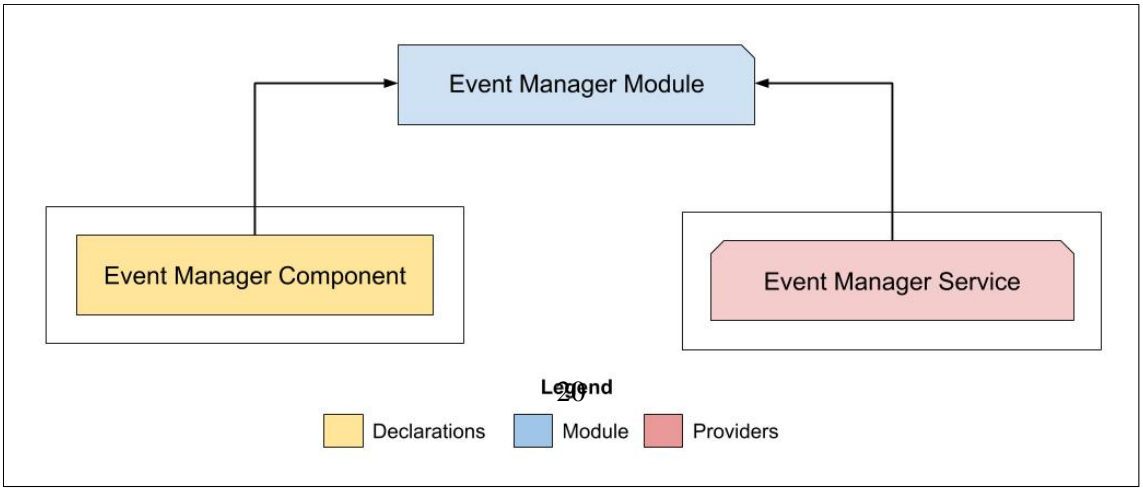


Figure 3.13: Rubric Component View



3.3.4.1 Components and Services

The module containing a set of components and service injectable are given below,

- **Event Manager Service:** The injectable containing API methods mainly performs the application server communication requires to provide the event management in the course

- **Event Manager Component:** The component associated view displays a list of events and corresponding subevents present in the course. It provides the view forms required to create/update the events & subevents in the course. The component contains the API methods to performs the necessary application logic required for the event management in the course

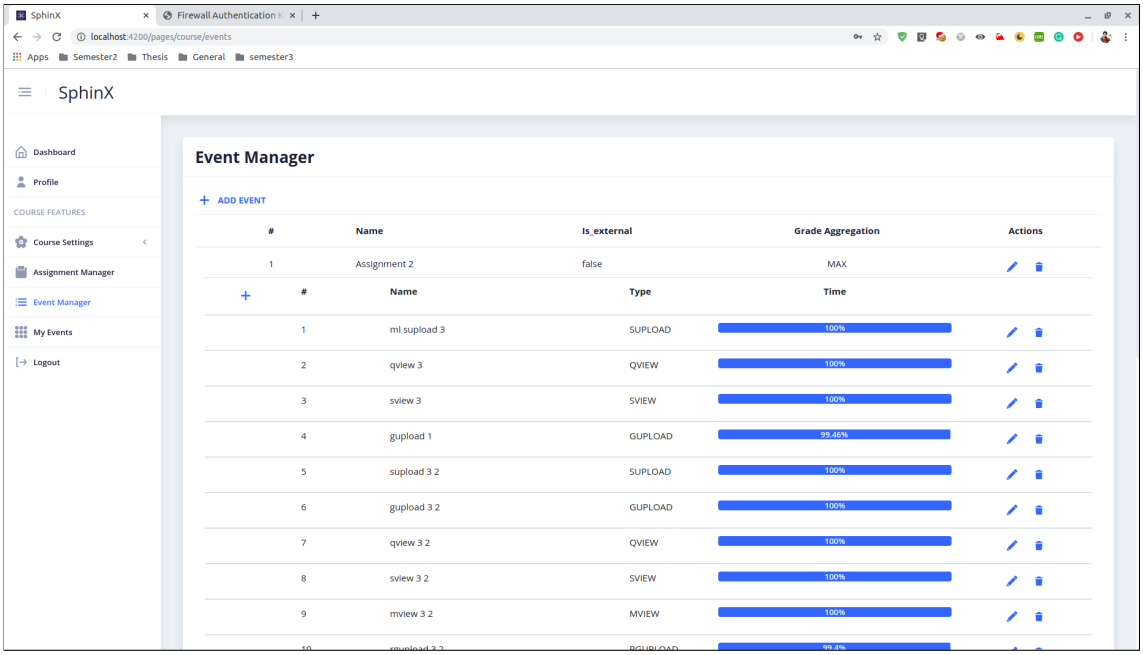


Figure 3.15: Event Manager Component View

3.3.5 MyEvent Manager Module

The module provides the MyEvent dashboard which displays all the user associated ongoing events in the course. The module takes care of the major assignment related activities like student assignment submission, grading of the submission copies, display the detailed graded submission copy with the user obtained grades, raise the regrade requests for any graded question in the assignment, review and regrade the allocated regrade requests, etc. The module takes care of the user flow corresponding to the above-mentioned activities in the system.

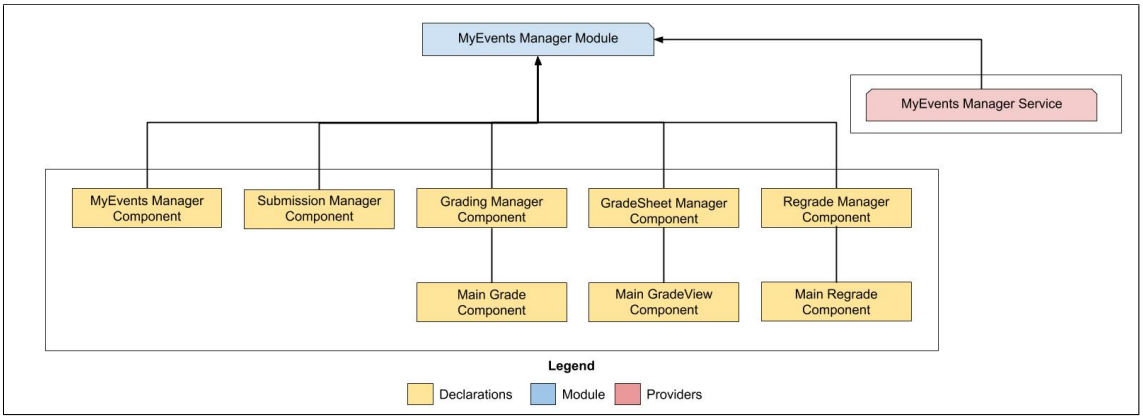


Figure 3.16: MyEvents Manager Module

3.3.5.1 Components and Services

The module containing a set of components and service injectable are given below,

- **MyEvents Manager Component:** The component displays all the ongoing events associated with the logged-in user on the MyEvents Dashboard of the system. It provides the navigation controls to perform various activities in the event

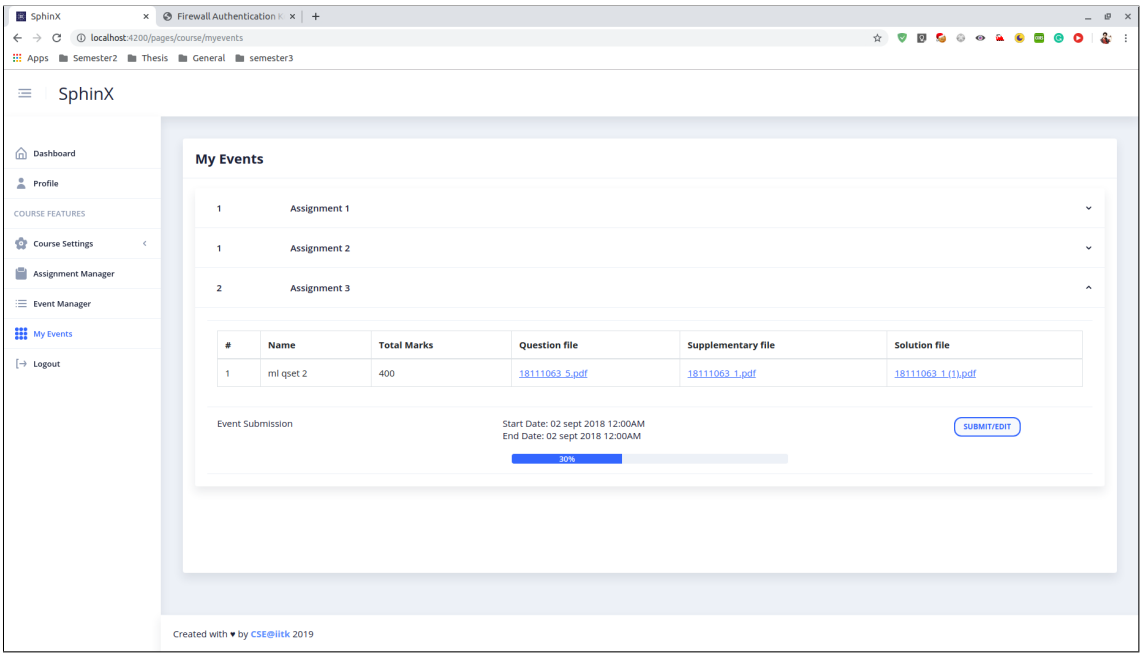


Figure 3.17: MyEvents Dashboard View(Student)

- **Submission Manager Component:** The component containing views and APIs provide the various capabilities to the user to perform the assignment submission in the event

- **Grading Manager Component:** The component loads the grading duties allocated to the user from the server and displays it with the corresponding grading details

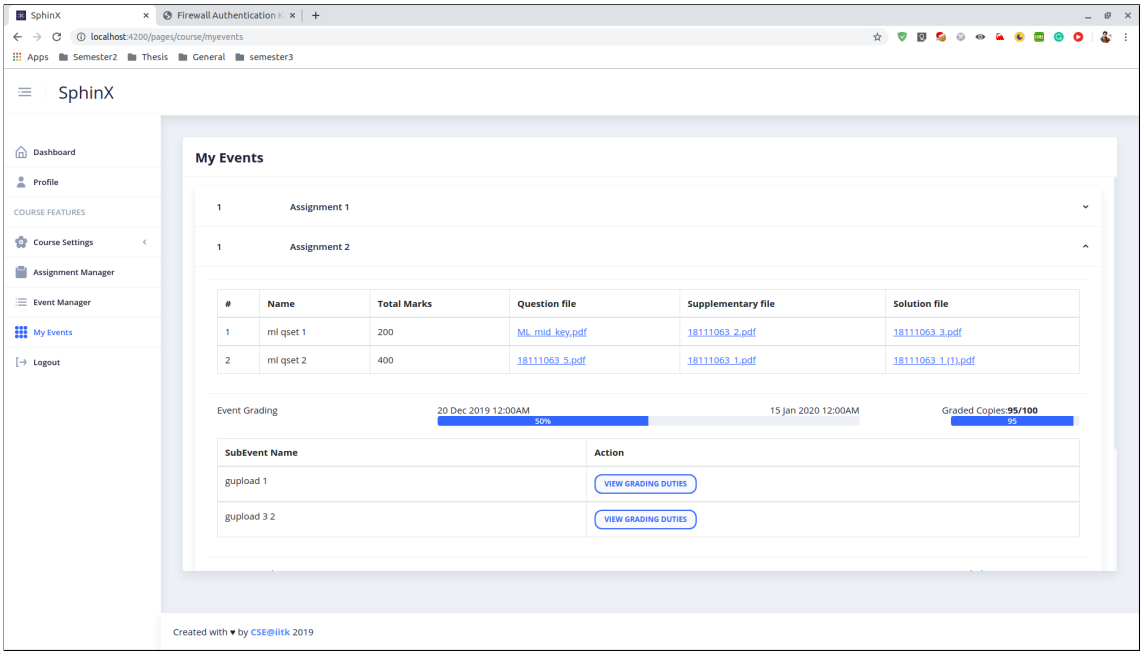


Figure 3.18: MyEvents Dashboard View(Grader)

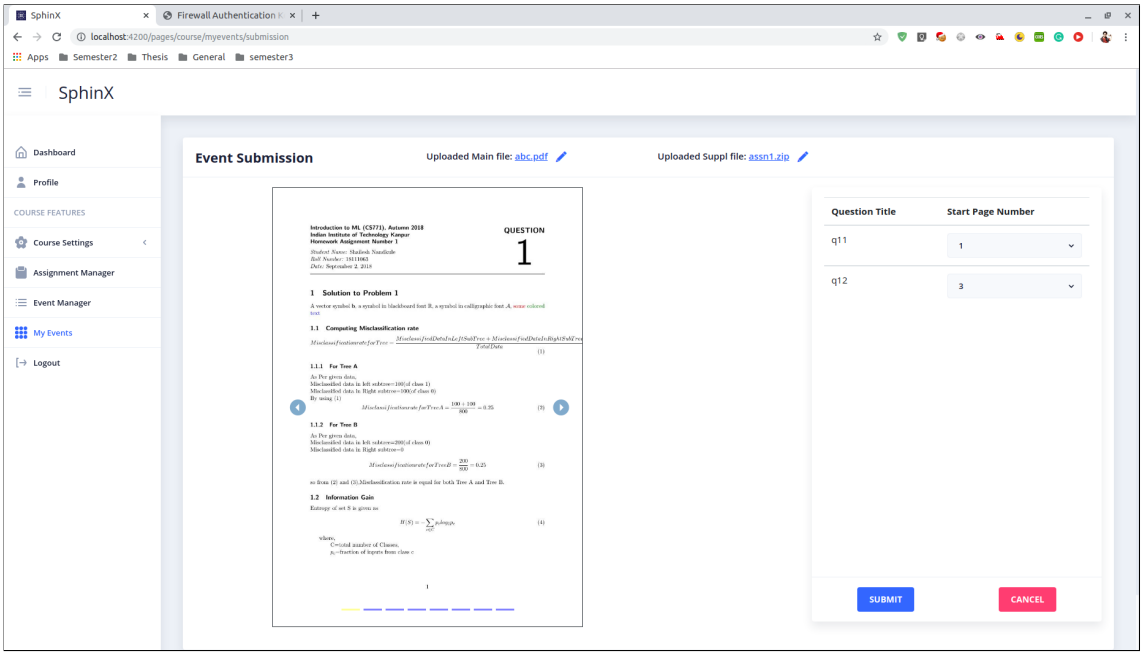


Figure 3.19: Submission Manager Component View

- **Main Grade Component:** The component containing view and APIs display the main submission file and provide capabilities to perform the rubric-based grading corresponding to any user-selected grading duty from the allocated grading duties

- **Grade Sheet Manager Component:** The component displays the assignment mark sheet which contains the grades given by the graders for all questions corresponding to the submission made by the user

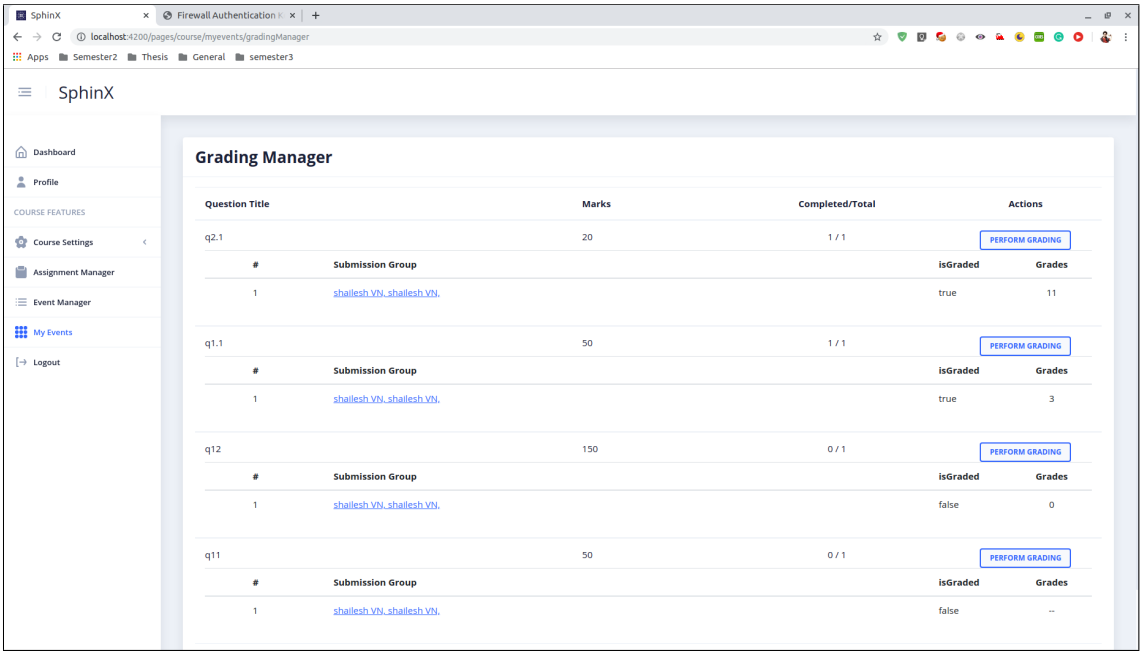


Figure 3.20: Grading Manager Component View

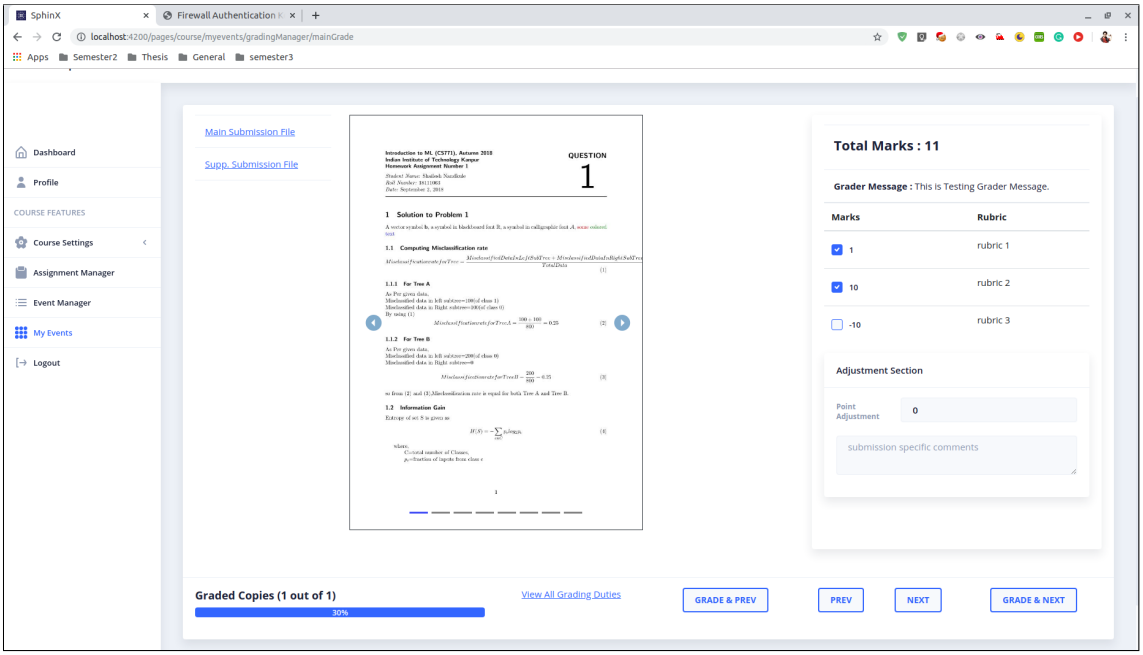


Figure 3.21: Main Grade Component View

- **Main GradeView Component:** The component displays the main submission file and the grading details corresponding to any user-selected question from the assignment mark sheet. The component provides the capability to raise the regrade request w.r.t. The corresponding question

- **Regrade Manager Component:** The component loads the regrade requests allocated to the user from the server and displays it with the corresponding review status

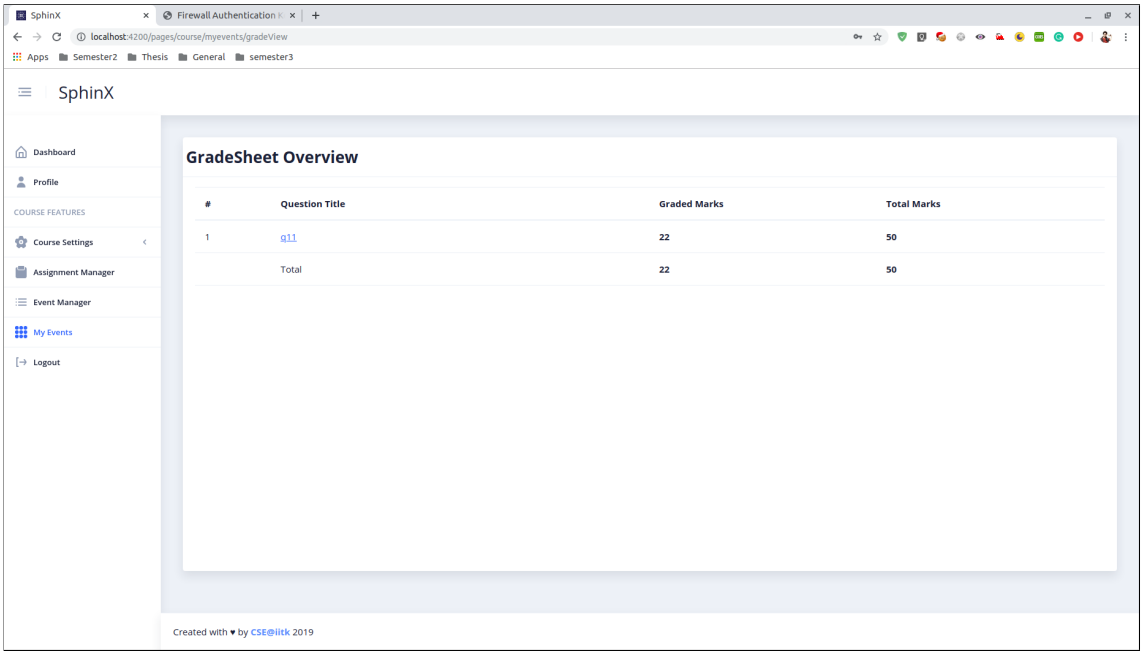


Figure 3.22: Grade Sheet Component View

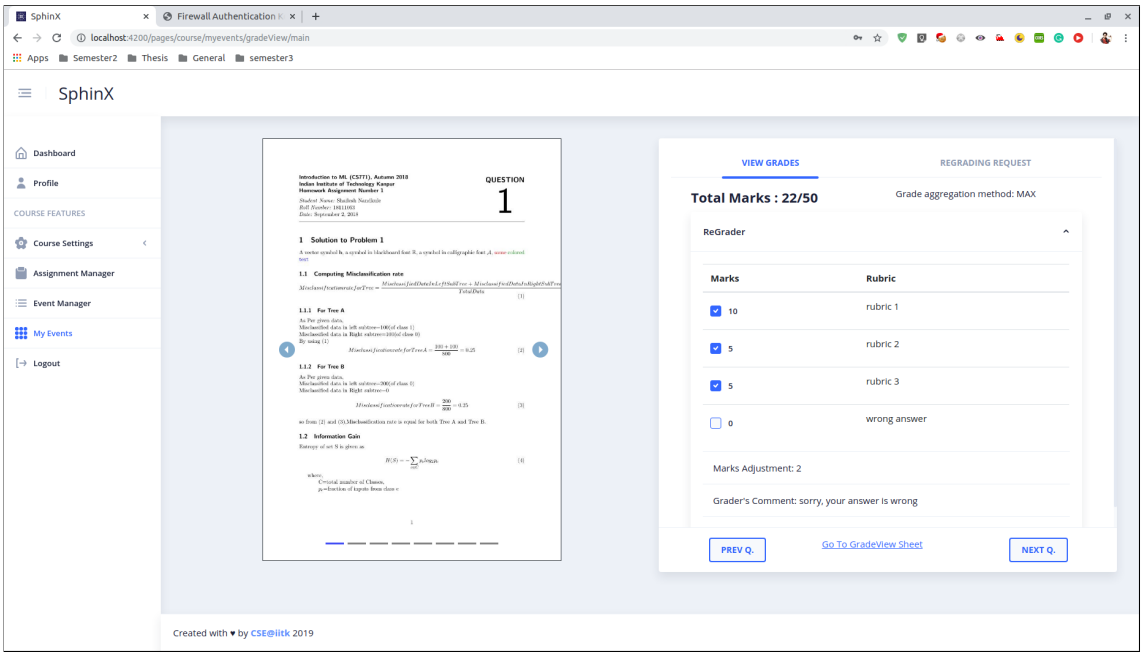


Figure 3.23: Question Grade View Component

- **Main Regrade Component:** The component displays the main submission file and old grading details associated with the user-selected regrade request for review on the regrading manager page. It provides the capabilities to review and complete the grading of the regrade request. The component provides the chatbox contains the regrade request communication between the student and the grader

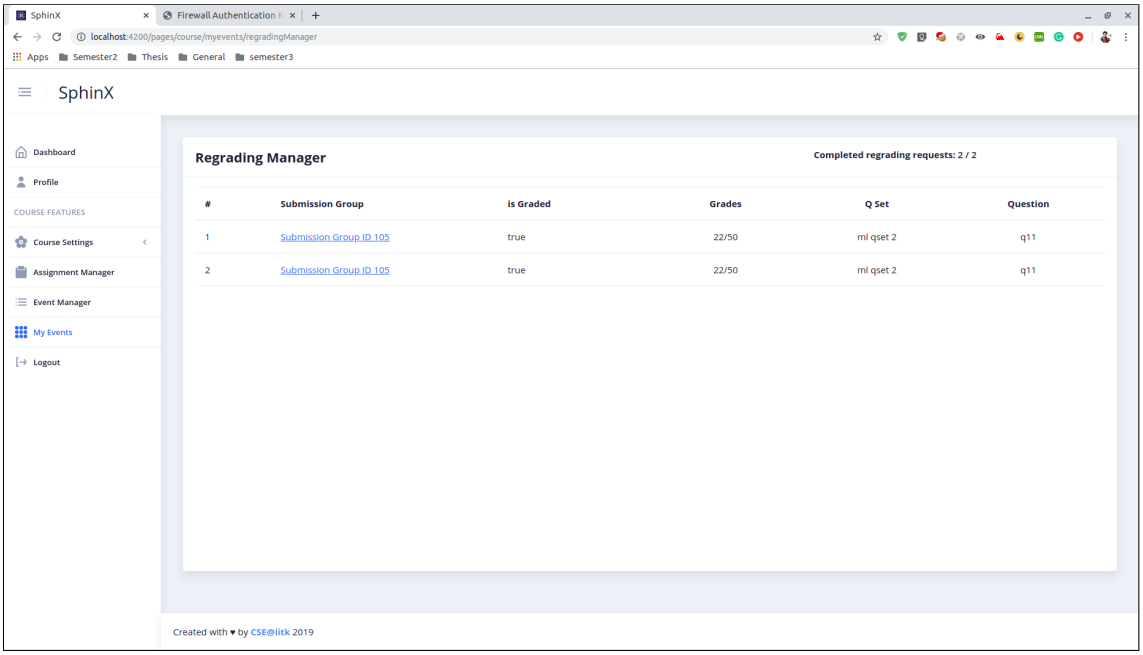


Figure 3.24: Regrading Manager Component View

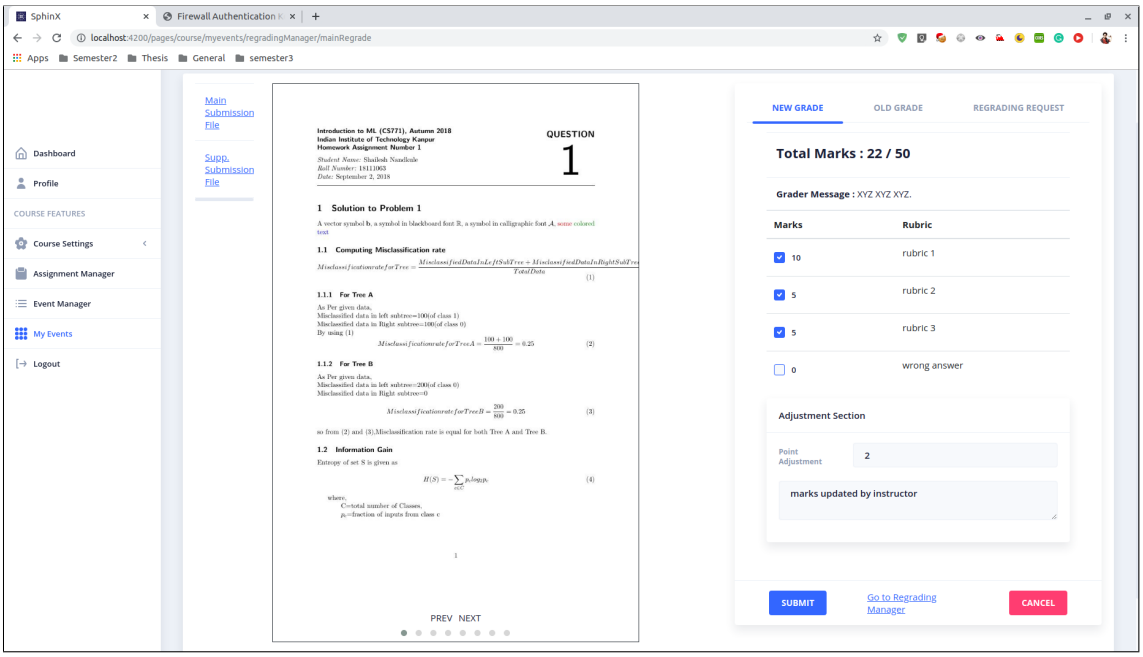


Figure 3.25: Main Regrade Component View(With New Grade Tab)

3.3.6 Course Admin Module

The module mainly handles the responsibility of the Admin-level features in the system. The various capabilities provided by the module are listed below,

- View the list of all submission groups with their corresponding group members for any event in the course
- View and update the submission made by any submission group in the course

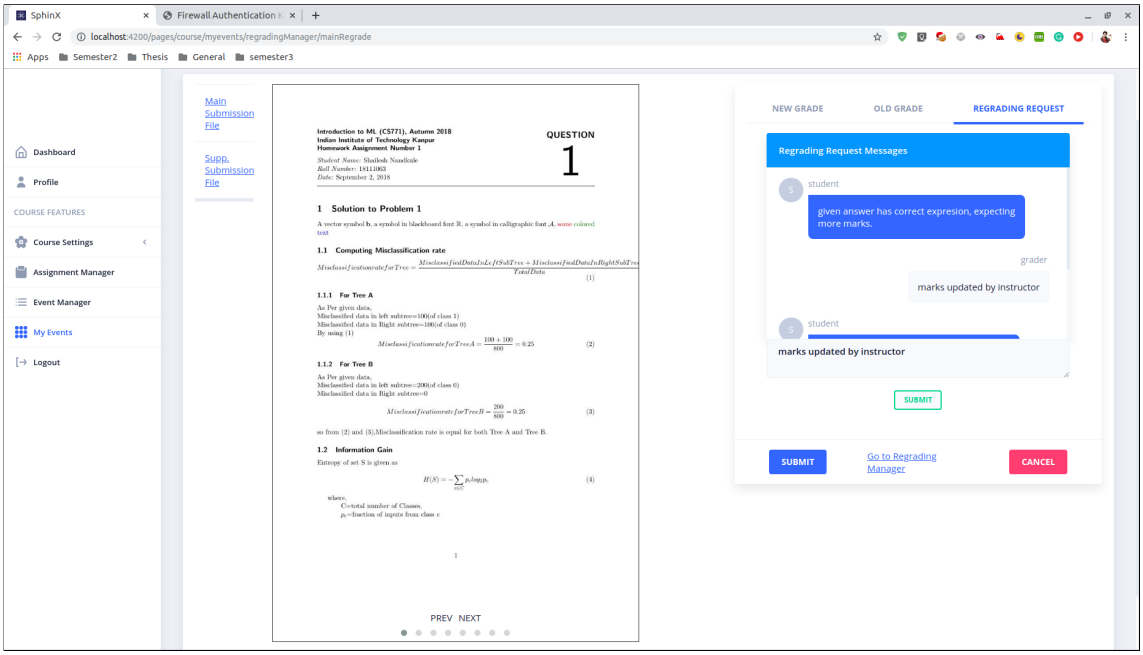


Figure 3.26: Main Regrade Component View(With Regrade Request Tab)

- Create the submission group by providing a list of group members in any event
- In case any student not able to find a submission group, Add the student to any existing submission group in the event
- In case any submission group not able to make submission till the deadline, perform the submission w.r.t. That submission group by uploading the supplementary file and the main submission file with pagination details
- View and download all the main submission files uploaded by the submission groups in the section for any event in the course
- Distribute and allocate the set of submission copies among the graders according to the grading scheme configured in the event
- View the graded copy and the given marks for all the submissions corresponding to any event in the course
- In case grader is not able to grade any submission copy, perform the rubric-based grading for that submission copy in the event
- View the list of regrade requests raised in the event with the corresponding review status
- In case if any grader not able to review any regrade request before the deadline, then review and grade the corresponding request

3.4 Application Routing

In order to make the system fast and improve the responsiveness, the system user interface is developed as a single page application. In this approach, the system delivers the application as a single page with multiple views corresponds to the different user-capabilities provided by the system. The application navigation flow between the multiple views in the system is set by configuring inter-component routes. As the SPHINX system provides a large set of capabilities, the navigation routes between the large set of views are configured such that it makes all system features easily accessible to the end-user without facing any difficulty.

As all module capabilities except the authentication module are required the user to be in the logged-in state, the routes are divided into the two sets /pages and /auth corresponding to the authenticated routes and non-authenticated routes. Whenever a non-logged-in user tries to access the authenticated routes, the auth guard redirects the user to the login view of the system. While the routes start with /auth are allowed to visit for all types of end-users. After the user logged-in successfully, the system redirects to the SPHINX dashboard where all the user-enrolled courses are displayed. Once the user selects the course from the dashboard, the system sets user course respective details and redirects the user to the course homepage. The system provides navigation capabilities to the various modules using the side menu links, such that all the major features like Course Manager, Assignment Manager, Events Manager, MyEvents Dashboard, Admin dashboard are one-click reachable to the user from any view in the course.

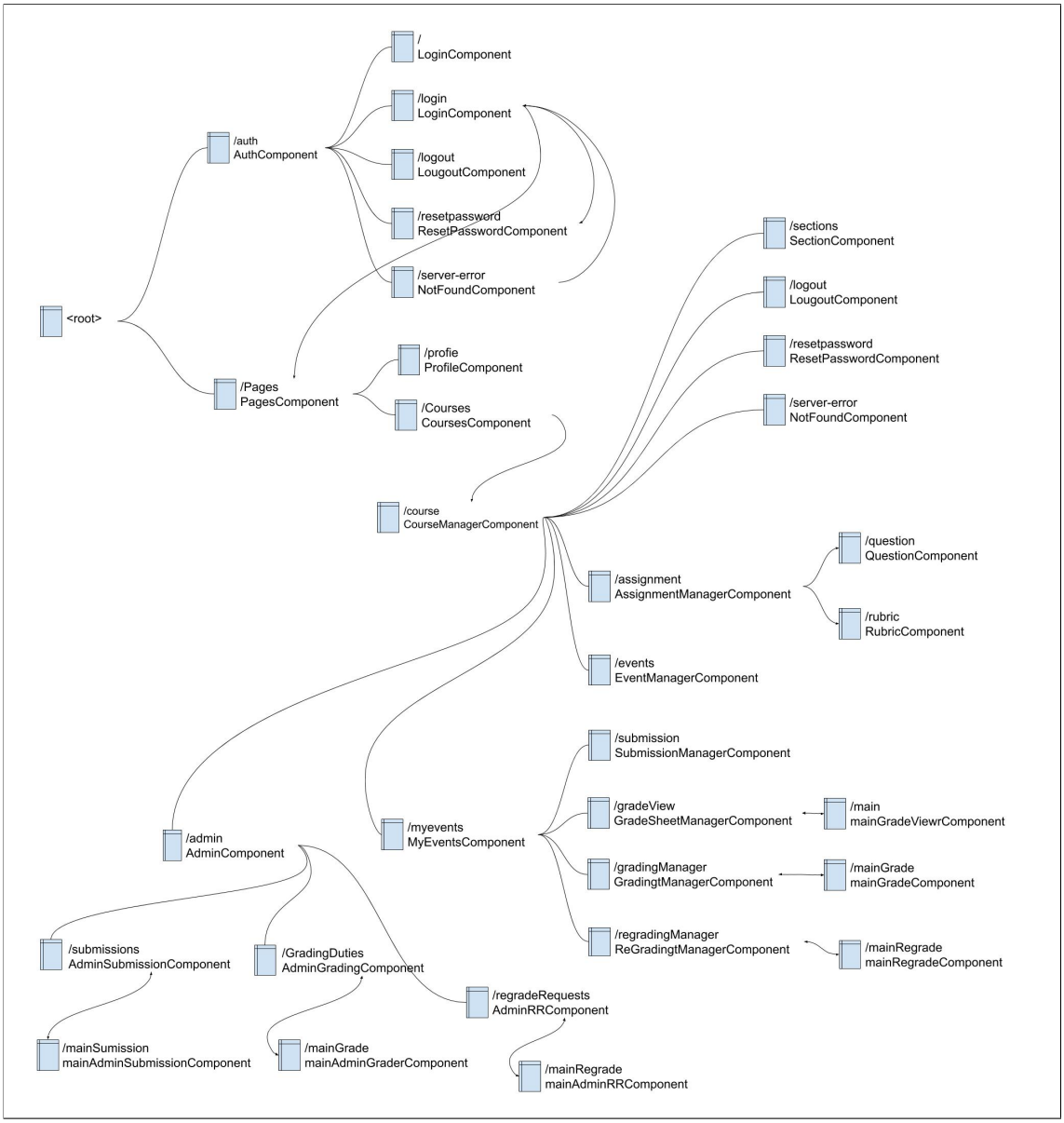


Figure 3.27: Application Navigation Routes

CHAPTER 4

Module Descriptions

Contents

Abstract *In this chapter present details of the major important modules of the* SPHINX *frontend architecture. We organized the numerous features provided by the platform into various sets of correlated capabilities, with each such set represented by a separate module. For each such module, the corresponding section explains the important design decisions taken related to information organization, platform hierarchy, and navigation options of the respective module. We enumerate the modules containing major services and components with a detailed explanation of the respective roles and responsibilities of the module. The list of key APIs of the frontend are provided with detailed description under the API sub-sections of corresponding service/component sections for various modules.*

4.1 Authentication Module

4.1.1 Overview

The feature module provides a set of functionalities like user authentication, user session management, CSRF token management. The module takes care of user authentication by mainly including login, logout, reset password capabilities. The user session management is implemented using browser local storage such that data is persisted until user clears browser cache. Authentication

Module also provides the route guard that's used to prevent an unauthenticated user from accessing restricted routes. The module containing Interceptor intercepts all the server API requests and adds necessary token headers before sending it to the application server. Module containing API methods of the components and the service provider are listed below in great detail.

4.1.2 Auth Manager Service

4.1.2.1 Overview

Authentication service provides server communication API's for CSRF Token Management, User Login Logout functionality, User Password reset functionality, and Fetch & store User account details to local storage

4.1.2.2 API Methods

Frontend API: confirmResetPassword	
Description	Method makes a put HTTP request to the backend server, which resets the password to provided value if no error occurred. If the user provides an incorrect username or password reset token, then the method returns an appropriate error message.
Server API URL	`/auth/confirm/reset/password/`
Parameters	1. username (string): User-provided username 2. Password1 (string): User-provided password 3. Password2 (string): User-provided confirmed password 4. Reset token (string): User-provided password reset token

Frontend API: getCSRFToken	
Description	Method fetches CSRF token from backend and handles backend API response. on successful return, stores CSRF token in the csrf data variable. If encounters error then navigates the user to the server-error page
Server API URL	`/auth/csrf\_token/`
Parameters	None

Frontend API: handleLoginError	
Description	This method displays appropriate error message based upon error details
Server API URL	`None`
Parameters	errorRes: HttpErrorResponse 1. HTTP error object consists of error description

Frontend API: login	
Description	Method makes login API post request to backend by including user-provided username and password. On successful return, 1. Method stores User details in local storage 2. Update and share User object asynchronously across the application using the currentUserSubject If the backend API call encounters an error, then call make to a handleLoginError method which handles error and shows an appropriate error message to the user
Server API URL	`/auth/login/`
Parameters	1. username (string) : user-provided username 2. password (string) : user-provided password

Frontend API: logout	
Description	Method makes logout post API request to the backend for currently logged-in user. On successful return, 1. Method removes User details from local storage 2. Update and share User object as null asynchronously across the application using the currentUserSubject
Server API URL	`/auth/logout`
Parameters	None

Frontend API: resetPassword	
Description	Reset password method makes a post backend API request to get reset password token for user-provided username. If post API request encounters an error, then an appropriate error message is shown based upon error details
Server API URL	`/auth/reset/password/`
Parameters	1. username (string) : user-provided username

Frontend API: sessionExpire	
Description	This method 1. Removes current logged-in user details from local storage 2. Update and share User object as null asynchronously across the application using the currentUserSubject 3. Navigate user to SPHINX login page
Server API URL	`None`
Parameters	None

Frontend API: setAccount	
Description	This method used to fetch user account details like 1. User information 2. List of objects representing user associated courses, where each object contains course information, Section and role-related information of the user After fetching user account details, method extracts, and stores information in local storage using User, Courses, RoleCourseMap variables. RoleCourseMap contains a key-value pair of roles and a list of user associated courses having the corresponding roles. RoleCourseMap is used to render the SPHINX dashboard view.
Server API URL	`/auth/account/`
Parameters	None

4.1.3 Auth Interceptor Service

4.1.3.1 Overview

An authentication interceptor intercepts API requests made by application to the backend server. Interceptor adds the CSRF token to POST / PUT / DELETE requests before they get sent to the server. It is implemented using the HttpInterceptor class provided by Angular Common module. The custom interceptor is created by the overriding intercept method of extended HttpInterceptor class. To use interceptor for every server API request, it added to the pipeline of providers section of the main app module

4.1.3.2 API Methods

Frontend API: intercept	
Description	Method intercepts the received Http Request, adds the CSRF Token to request Headers before sending it to the application server
Server API URL	`None`
Parameters	1. Request: The HttpRequest object 2. Next: The HttpRequest Handler object

4.1.4 Component Overview and API's

4.1.4.1 Auth Component

The component act as middleware before loading the authentication page components like login, logout and reset password. The middleware fetches CSRF token from server and makes it available for further authentication requests. Component Provides APIs for CSRF token management

Frontend API: ngOnInit	
Description	Fetches and update current CSRF token on component initialization
Parameters	None

Frontend API: updateCsrfToken	
Description	Fetches CSRF token from Backend using Authentication service and save token in variable csrfData such that it can be used for POST, PUT, DELETE API requests
Parameters	None

4.1.4.2 Login Component

The login component provides the view and component APIs to perform user login functionality. The login view uses external Nebular library [10] components like NbAlert, NbInput, and NbBut-

ton. The view contains forgot password link below the password input box which navigate to reset password page. The login component contains properties and method APIs to perform frontend application logic for Users login to the application. component API's provides functionalities which mainly includes fetching CSRF token on view initialization, User login to the application server, Store user account details on successful login.

Frontend API: forgotPassword	
Description	Method navigate the user to reset password page
Parameters	None

Frontend API: ngOnInit	
Description	As valid CSRF token required to send with every login server request, Component life cycle hook used to fetch CSRF token on login view initialization
Parameters	None

Frontend API: onLoginFormSubmit	
Description	The method takes login form data and makes a server API call to log in the user to the application server. On successful login, the method redirects the user to the application dashboard
Parameters	form (NgForm): Login form data contains username, password

Frontend API: setAccount	
Description	The method makes a service call to fetch and store user account details to local storage
Parameters	None

Frontend API: updateCSRF	
Description	Method fetch CSRF token from the server using getCSRFToken API of authentication service and store in service property for further availability to the interceptor
Parameters	None

4.1.4.3 Password Component

ResetPassword component provides a tabular view that contains two forms. The password reset token request form takes username input, if the username is valid then the corresponding user receives the password reset token on registered email. The view uses external Nebular library [10] components like tabs, alert, input fields, and loading spinner. The component provides properties and APIs to request reset token and submit a password reset request to the application server

Frontend API: onResetFormSubmit	
Description	Validate that user-provided new password and confirm password must be identical, else display an error message. On successful validation, the method makes a server API request to reset user password using confirmResetPassword API of service. On a successful password reset, the method redirects the user to the login page by displaying success notification. If the user-provided username or password reset token is invalid, component error property set appropriately using server error response
Parameters	form (NgForm): User-submitted form data for a password reset. Contains username, new password, confirm password and password reset token

Frontend API: onRequestResetToken	
Description	The method makes a password reset token request to the application server by providing a user-provided username. If a user-provided username is incorrect, component properties username and error are set appropriately using server error response
Parameters	form (NgForm): User-submitted form data for a password reset token

4.1.4.4 Logout Component

Logout component mainly calls a server logout API on component initialization, On receiving a successful response from the server, the component clears user state and session information from local storage and redirects the user to the application login page. The component does not provide any view as its main role is to provide logout functionality and navigate according to server response once the user clicks the logout link

4.2 Course Manager Module

4.2.1 Overview

Managing course-related things is an important aspect of an online course management platform. SPHINX provides various capabilities to manage course-related things through the Course Manager Module. The capabilities provided by SPHINX are listed below

1. View and edit Course information like course name, course title, course directory to store course data
2. Course Section management - allows various operation on course sections like create, view, edit, delete sections
3. Course Topic management - allows various operation on course topics like create, view, edit, delete topics, allows user to create tree hierarchy of topics by mentioning parent topic
4. Course Role management - allows user to create new role using various combination of meta roles, where each meta role provides set of functionalities to perform over platform. The module also provides capabilities to view, edit, delete existing roles

5. Course Roster - allows to enroll new users to course, also provides capabilities like view enrolled user list, change a user's enrolled section or role, delete existing enrolled users

4.2.2 Course Manager Service

4.2.2.1 Overview

The Service injectable provides function API's to perform backend API interaction for various capabilities like course detail management, topics, sections, role, and enrollment management in the course

4.2.2.2 API Methods

Frontend API: createRole	
Description	The method makes an backend API call to create new role in course at the application server
Server API URL	`/course/#courseid/roles/`
Parameters	• name (string): Name of new role • action_list (string): Action bit string of new role

Frontend API: createSection	
Description	The method makes an backend API call to create new section at the application server with provided name in course
Server API URL	`/course/#courseid/sections/`
Parameters	sectiondata : A structure contains the new section data

Frontend API: deleteRole	
Description	The method makes an backend API call to delete an existing role at the application server in the course
Server API URL	`/course/#courseid/role/#roleid/`
Parameters	roleId (number): database ID of the Existing CourseRole object to be deleted

Frontend API: deleteSection	
Description	The method makes an backend API call to delete an existing section at the application server in the course
Server API URL	`/course/#courseid/section/#sectionid/`
Parameters	sectiondata: A structure contains database ID of the existing section to be deleted

Frontend API: editCourseDetails	
Description	The method makes an backend API call to update the course details at the applicaion server
Server API URL	`/course/#courseid/`
Parameters	coursedetail (CourseDetail): The CourseDetail model object contains course information to be updated

Frontend API: editRole	
Description	The method makes an backend API call to update an existing role in course at the application server
Server API URL	`/course/#courseid/role/#roleid/`
Parameters	• role (CourseRole): An existing CourseRole object to be updated • allowed_action (string): A binary action bit string contains new set of allowed actions to be updated

Frontend API: editSection	
Description	The method makes an API call to backend to edit the existing section with provided name in the course
Server API URL	`/course/#courseid/section/#sectionid/`
Parameters	sectiondata: The structure contains new section name and database ID of the existing section

Frontend API: fetchCourseDetails	
Description	The method makes an backend API call to fetch course details. Whenever call made to this function, function extract the course details from API response and stores it in the local storage for further availability to application
Server API URL	`/course/#courseid/`
Parameters	None

Frontend API: fetchCourseRoles	
Description	The method makes an backend API call to fetch list of roles in course. after receiving response from the application server, method extract each role information and stores in the local storage for further availability to application. During information extraction, method process the action bit string of role and stores the meta-roles associated with role
Server API URL	`/course/#courseid/roles/`
Parameters	None

Frontend API: fetchSections	
Description	The method makes an backend API call to fetch list of existing sections in course. Whenever call made to this function, function extract the list of section information and stores it in the local storage for further availability to application
Server API URL	`/course/#courseid/sections/`
Parameters	None

Frontend API: handleAPIError	
Description	The method process the error response received during the server communication for various API's of the service and displays appropriate error message
Server API URL	`None`
Parameters	errorRes: An error response object received from the application server

Frontend API: addEnrollments	
Description	The addEnrollments method receives CSV file and calls the server API to create new enrollments in the course
Server API URL	`/course/#courseid/enrollments/`
Parameters	file (File): The CSV file contains list of users with corresponding details to be enrolled

Frontend API: delEnrollment	
Description	The method makes an server API call to delete an existing enrollment corresponding to provided enrollment id in the course
Server API URL	`/course/#courseid/enrollment/#enrollmentid/`
Parameters	enrollid (number): The user enrollment ID

Frontend API: putEnrollment	
Description	The method makes an server API request to edit the existing enrollment with provided role and section list in course
Server API URL	`/course/#courseid/enrollment/#enrollmentid/`
Parameters	enroll (Enrollment): The Enrollment model object contains user's course related information to be updated

Frontend API: createTopic	
Description	The method makes an API call to backend to create new topic at the application server with provided details in the course
Server API URL	`/course/#courseid/topics/`
Parameters	topicdata: Topic model object contains new topic data like topic name, description, super topic id

Frontend API: deleteTopic	
Description	The method makes an API call to backend to delete an existing topic from the application server in the course
Server API URL	`/course/#courseid/topic/#topicid/`
Parameters	topicdata: Topic model object contains topic details to be deleted

Frontend API: editTopic	
Description	The method makes an API call to backend to update an existing topic with provided details in course
Server API URL	`/course/#courseid/topic/#topicid/`
Parameters	topicdata: Topic model object contains topic details to be updated like topic name, description, super topic id

Frontend API: fetchTopics	
Description	The method makes an backend API call to fetch list of existing topics in the course. Whenever call made to this function, function extract the list of topic information and stores in the local storage for further availability to the application
Server API URL	`/course/#courseid/topics/`
Parameters	None

4.2.3 Component Overview and API's

4.2.3.1 Course Manager Component

Whenever the user selects the course from the SPHINX dashboard, Course Manager component acts as middleware before displaying a Course-specific view to user. The main responsibility of the middleware is filtering out action item links from the side menu. So it filters action item links based on the role assigned to the user in the corresponding selected course

Frontend API: onComponentInit	
Description	The component life-cycle hook gets called every time on component initialization. This method initializes user permission map for component view, makes a call to set the user permissions and update side menu action items for the corresponding user-selected course
Parameters	None

Frontend API: onComponentDestroy	
Description	The component life-cycle hook gets called before destroying the component. This method makes the call to remove user assigned permissions in the component. Then method update the side-menu action items in a view for the corresponding user-selected course
Parameters	None

Frontend API: removeCourseFeatures	
Description	The method removes the course-specific action items from the side menu and updates the user application view
Parameters	None

Frontend API: setCourseFeatures	
Description	The setCourseFeatures method adds the course-specific action items to the side menu using the user permission map, then updates the user application view
Parameters	None

Frontend API: setCoursePermissions	
Description	The setCoursePermissions method process the allowed action list of the selected course then sets the user permission map for frontend capabilities. The allowed action list is the binary string representing the user allowed server APIs in the corresponding course. After setting the user permission map, Method makes a call to update side menu action items in a user application view
Parameters	None

4.2.3.2 Roster Component

The Roster Component provides all necessary capabilities to perform Course Roster management. The Roster View mainly contains file drop-zone at the top of the page and after that course enrollment view is displayed. It provides the way to add new enrollments to the course using the CSV file. To provide the file upload capability to the user, The component uses the ngx-file-upload library [11]. Roster Component contains the necessary properties and API methods to perform all roster management operations like view/add/edit/delete course enrollments. On Edit mode, The component gives way to edit user assigned role and section. The API methods mainly handle the user-triggered events, process the corresponding enrollment data and make calls to appropriate methods of course manager service

Frontend API: AddEnrollments	
Description	The AddEnrollments method takes user input CSV file from view using property uploadedFiles, validates the file and shows an appropriate error message if an error occurs. If the input file is valid, the method calls the service method to add enrollments to the course database. On receiving a success response from the server, the method updates the course enrollment list in the view
Parameters	None

Frontend API: fetchEnrollments	
Description	The method fetches existing course enrollment data from the server using the service API fetchEnrollments. On receiving a successful response from the server, method loads enrollment data from local storage to the component property enrollments
Parameters	None

Frontend API: fetchCourseRoles	
Description	The method fetches existing course role data using service API fetchCourseRoles. On receiving a successful response from the server, the method sets component property course_roles and makes list of roles available to edit the existing enrollment in the view
Parameters	None

Frontend API: fetchCourseSections	
Description	The method fetches existing course section data using service API fetchSections. On successful retrieval of data, method sets component property course_sections and makes section list available to edit the existing enrollment
Parameters	None

Frontend API: onDeleteEnroll	
Description	The roaster view triggers call to the onDeleteEnroll method, once the user clicks delete for some enrollment. The method makes a call to service API delEnrollment which deletes selected enrollment from the server. On receiving a successful response from Server, the method updates the user list in the view
Parameters	enrollIdx (number): The list index of user selected enrollment

Frontend API: onEditEnroll	
Description	The method receives the list index of user-selected enrollment from the course enrollment list. method fetches course role and section data to show in edit form dropdowns, Then method shows the edit form with selected enrollment data
Parameters	enrollIdx (number): The list index of user selected enrollment

Frontend API: onEditEnrollSubmit	
Description	The roaster view triggers call to the onEditEnrollSubmit method, once the user clicks edit for some enrollment. the method makes a call to update selected enrollment using Service API putEnrollment. On receiving a successful response from the server, the method updates the list of existing course enrollments in the view
Parameters	None

Frontend API: toggleEditEnrollForm	
Description	The method toggles the enrollment edit form in view by inverting value of the component property openEditForm
Parameters	None

4.2.3.3 Topic Component

The Topic component provides all capabilities to perform the Topic management of Course. The view uses the ng2-smart-table library [13] which represents topic data in the smart table. The smart table provides different features like sorting, searching, pagination, inline add/edit/delete capabil-

ities. The ng2-smart-table triggers event on user actions like add/edit/delete topic which gives the way to perform application logic on such user actions. The SPHINX gives the capability to create a tree of topics by providing super-topic detail. For the super-topic column, The customized configuration is provided in component as a drop-down list needs to be used for edit/create mode. The component provides function API's to fetch course topics data from the server, facilitate user performed actions like adding a new topic to course, edit/delete existing topics in the course

Frontend API: onComponentInit	
Description	The component lifecycle hook method used to fetch course topics data from the server. The method sets the following things on receiving a successful server response 1. Topic table data to be displayed in the view 2. Topic list data object which used to show on the drop-down in the edit form 3. The super-topic name for each retrieved topic using the component property topicMap
Parameters	None

Frontend API: onCreateTopicConfirm	
Description	The method makes the server request to add a new topic using the service API method. On receiving a successful response from the server, the method updates table data and shows an appropriate notification message
Parameters	event: The event triggered by topic smart table once user clicks the add confirm button

Frontend API: OnDelete	
Description	The method shows a confirmation window for topic deletion. If the user clicks yes, the method triggers a call to the OnDeleteConfirm method
Parameters	event: The event triggered by topic smart table once user clicks the delete button

Frontend API: onDeleteConfirm	
Description	The method makes the server request to delete the user-selected topic using the service API method. On receiving a successful response from the server, the method updates table data and shows an appropriate notification message
Parameters	event: The event triggered by topic smart table once user clicks the delete button

Frontend API: onEditConfirm	
Description	The method calls the service API method to edit the existing topic by sending the required topic data. On receiving a successful response from the server, the method updates topic table data and shows the appropriate notification message
Parameters	event: The event triggered by topic smart table once user clicks the edit confirm button

4.2.3.4 Section Component

The Section component provides capabilities to perform Course information management and Section management of course. The component associated view contains two parts that correspond to Course detail management and Course section management respectively. The course detail management view displays all course details in editable input boxes. The view provides a way to update the course directory path, which stores user submission files and other course data. The Section management view uses the ng2-smart-table library [13] same as the topic component to represents section data in the smart table. The Section component provides the settings required for the section smart table. The setting contains customized settings for add/edit/delete buttons and column details of the section table. The component provides function API's to fetch course and section related data from server using Course manager service. The function API's also facilitating user performed actions like adding new sections to course, edit/delete existing sections in the course and update the course details in the application server

Frontend API: onCreateConfirm	
Description	The onCreateConfirm method calls the service API method to create new section by passing required section data. On successful response, method updates table data and shows appropriate notification message
Parameters	event: Event triggered by section smart table once user clicks the add confirm button

Frontend API: onDeleteConfirm	
Description	The method calls the service API method to delete an existing section by passing required section data. On successful response, method updates table data and shows appropriate notification message
Parameters	event: Event triggered by section smart table once user clicks the delete confirm button

Frontend API: onDelete	
Description	The method shows a confirmation window for section deletion. If the user clicks YES then method triggers a call to delete the section, else reject the received event
Parameters	event: Event triggered by section smart table once user clicks delete

Frontend API: onEditConfirm	
Description	The method calls the service API method to edit existing section details. On receiving a successful response, the method updates section table data and shows the success notification message
Parameters	event: Event triggered by section smart table once user clicks the edit confirm button

Frontend API: onUpdateCourse	
Description	method calls the service API method to update the course details. On receiving a successful response, the method shows an appropriate notification message to the user
Parameters	None

4.2.3.5 Role Manager Component

The Role Manager component provides the capabilities to perform all Role management operations in the user-selected Course. In application server, the corresponding allowed actions for the role are stored and used as an action list. The action list is the string consisting of 0's and 1's for each server API. As the user does not understand the binary string of action list, The component takes care of string processing and provides user understandable application meta roles. It provides method API's to perform functionalities on user actions like Add/Edit/Fetch/Delete course roles. The API's mainly manipulates user actions, processes the corresponding role data, extracts the action string and makes calls to the application server using course manager service. The component associated view lists out all existing roles in the course. The view provides two forms to perform add/edit role actions using an application provided meta roles. The role edit form initially in a hidden state, will be toggled once the user clicks on the edit button of any role. The view also lists out all meta roles and there respective allowed capabilities in detail.

Frontend API: addRole	
Description	Once the user clicks Add new role button, view triggers the call to addRole method of the component. The method takes the data values of the form using component property new_role. Service API createRole does not understand the term of meta roles and requires an action list of a new role as a binary bit string. So addRole method extract and process the list of user-selected meta roles for a new role. Then the method generates the appropriate action string for the corresponding new role. After generating an action string of the new role, the method sends a server request to add a new role using Service API createRole. On receiving a successful response from Server, method updates the list of course roles in the view
Parameters	None

Frontend API: fetchRoles	
Description	The method fetches existing Course Role data using service API fetchCourseRoles. On successful retrieval of data, the method sets component property course_roles and makes list of existing roles available to component view
Parameters	None

Frontend API: onDeleteRole	
Description	The method makes a server request to delete a user-selected role using a service API method deleteRole. On receiving a successful response from the server, method refresh course role list in role view using property course_roles
Parameters	None

Frontend API: onEditSubmit	
Description	Once the user clicks the submit button on edit role form, view triggers call to onEditSubmit method of the component. The onEditSubmit method takes the data values of the form using component property edit_role. The application server does not understand the term of meta roles and requires an updated action list as a binary bit string. So the method process the list of user-selected meta roles then generates the appropriate binary bit string for the role. After generating an action list, the method makes the server API call to update the selected role using the service method editRole. On receiving a successful server response, method updates the course role view
Parameters	None

Frontend API: replaceSubString	
Description	The method replaces the substring of main string object by other string from the given index position
Parameters	• str (String): Main string to be updated • index(Number): start index from which substring to be replaced • replacement (String): New substring which should be replaced in main string

4.3 Assignment Manager Module

4.3.1 Overview

The assignment manager module mainly handle the responsibility of all the operations related to the assignment data in the course. Using the module, SPHINX provides various features related to assignment like managing the multiple question-sets within the assignment, managing the question tree & rubrics within each question set, managing question-set files, question-topic linking, page coordinate selection for the AI related features, etc. As there are many capabilities, the module provides all the capabilities using three view pages corresponding to the assignment manager,

the question manager and the rubric manager. It also provides the navigation capabilities between them such that user can easily move in any question set to manage the corresponding questions and rubrics in the course. The detailed explanation of the important service injectable and components is given in the remaining sections of the module

4.3.2 Assignment Manager Service

4.3.2.1 Overview

In the assignment manager injectable we mainly handle the server communication required to perform various activities in assignment creation at the application server. The service responsible for receiving and entertaining all the server communication requests from all the components in the assignment manager module.It provides necessary function API's to fetch the course assignment data from the application server like list of the existing course assignments, question sets, question set associated files, questions, rubrics, etc. The service also handles assignment data creation/modification/deletion requests by gathering and formatting required parameters of various server API requests

4.3.2.2 API Methods

Frontend API: addAssignment	
Description	The addAssignment method makes the server post API request to add a new assignment to the course at the application server with provided assignment data
Server API URL	`/course/#courseid/assignments`
Parameters	assign_data (Assignment): The Assignment model object contains user supplied details to be created

Frontend API: addQuestion	
Description	The method makes the server post API request to add a new question to the given question set with provided question data
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/ #questionsetid/questions/`
Parameters	question_data: Question structure contains the new Question properties

Frontend API: addQuestionSet	
Description	The method makes the server post API request to add a new question set corresponding to the given assignment with provided question set data
Server API URL	`/course/#courseid/assignment/#assignmentid/questionsets`
Parameters	• assign_id(number): The request corresponding Assignment ID in which question set to be created • qset(QuestionSet): The Question Set object data to be created

Frontend API: createRubric	
Description	The method makes the server post API request to add a new rubric to the course database for provided question
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/question/#questionid/rubrics`
Parameters	• rubric: Rubric data object to be created • question_id(number): the database ID of the question to which new rubric should be added

Frontend API: deleteAssignment	
Description	The method makes the server API request to delete an existing assignment with provided assignment id in the course
Server API URL	`/course/#courseid/assignment/#assignmentid`
Parameters	data(Assignment): Assignment model object contains details of the assignment to be deleted

Frontend API: deleteQuestion	
Description	The method makes the server API request to delete the requested question of given question set in the Course Assignment
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/question/#questionid`
Parameters	question_id: Database id of the user requested question

Frontend API: deleteQuestionSet	
Description	The method makes the server API request to delete the existing question set correspond to given assignment with provided question set ID
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid`
Parameters	• assign_id(number): Question set corresponding Assignment ID • qset_data(QuestionSet): Question Set object data to be deleted

Frontend API: deleteRubric	
Description	The method makes the server API request to delete the existing rubric associated to the provided question
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/question/#questionid/rubric/#rubricid`
Parameters	rubric(Rubric): Rubric model data object to be deleted

Frontend API: downloadFile	
Description	The method takes the file object and downloads the same with provided filename in user local machine
Server API URL	`None`
Parameters	• file: The file object to be download • filename: Filename to be assigned for downloaded file

Frontend API: editAssignment	
Description	The method makes the server put API request to edit the existing assignment with new provided assignment details at the application server
Server API URL	`/course/#courseid/assignment/#assignmentid`
Parameters	assign_data(Assignment): Assignment model object contains new assignment name, description, existing assignment id

Frontend API: editQuestion	
Description	The method makes the server put API request to edit an existing question of given question set with provided question data
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/question/#questionid`
Parameters	question_data: Question structure contains the new Question properties to be updated

Frontend API: editQuestionFile	
Description	The method makes the server API request to update the main Question Set file of provided question set correspond to given assignment in Course
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/questionFile`
Parameters	• assign_id(number): question set corresponding Assignment ID • qset_id(number): file corresponding Question set ID • file(File): file to be updated as new Question set file

Frontend API: editQuestionSet	
Description	The method makes the server API request to edit the existing question set correspond to given assignment with provided question set data
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid`
Parameters	• assign_id(number): Question set corresponding Assignment ID • qset(QuestionSet): Question Set object data to be updated

Frontend API: editRubric	
Description	The method makes the server API request to edit the existing rubric of the provided question
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/question/#questionid/rubric/#rubricid`
Parameters	rubric(Rubric): Rubric data object to be updated

Frontend API: fetchAssignment	
Description	fetchAssignment method makes an server get API request to fetch list of existing assignments in course. Whenever call made to this function, function extract the list of assignment information and stores in local storage for further availability to application
Server API URL	`/course/#courseid/assignments`
Parameters	None

Frontend API: fetchQuestionFile	
Description	The method makes the server API request to fetch the main Question Set file of provided question set correspond to given assignment in Course
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/questionFile`
Parameters	• assign_id(number): Question set corresponding Assignment ID • qset_id(number): The requested file corresponding Question set ID

Frontend API: fetchQuestions	
Description	Main responsibility of method is to • Fetch the all questions data from server database for user selected assignment, question set • Process the server response and set the list of questions as Question Tree Model So method makes the server API get request to fetch all question data correspond to requested question set. After receiving success response from server, method process the received list of questions and form the tree of Questions using parent property of question. On successful Tree formation, TreeModel is saved in local storage which makes Tree data available to Question View
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/questions`
Parameters	None

Frontend API: fetchQuestionset	
Description	fetchQuestionset method makes multiple server API requests to fetch all question sets corresponds to all assignments in course. Method collects server responses of all get API requests, format the response data into list of Question set model objects then save the data into local storage for further availability to application method uses RxJS operator concatMap to make multiple server requests synchronously
Server API URL	`/course/#courseid/assignment/#assignmentid/questionsets`
Parameters	None

Frontend API: fetchQuestionSetImages	
Description	The method makes the server API get request to fetch Question set file as array of jpeg images. Method pass the user selected assignment id and question set id as a API request parameters necessary to the server in order to entertain the request
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/questionFile/images`
Parameters	None

Frontend API: fetchRubrics	
Description	The method makes an server API get request to fetch list of existing rubrics for provided question ID. Whenever call made to this function, function extract the list of rubric information and stores in local storage for further availability to application view
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/question/#questionid/rubrics`
Parameters	question_id: The database ID of the request corresponding question

Frontend API: fetchSolutionFile	
Description	The method makes the server API request to fetch the Solution file of provided question set correspond to given assignment in the Course
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/solutionFile`
Parameters	• assign_id number: Question set corresponding Assignment ID • qset_id(number): The requested file corresponding Question set ID

Frontend API: fetchSupplementaryFile	
Description	The method makes the server API request to fetch the Supplementary Question Set file of provided question set correspond to given assignment in Course
Server API URL	`/course/#courseid/assignment/#assignmentid/questionset/#questionsetid/supplementaryFile`
Parameters	• assign_id number: Question set corresponding Assignment ID • qset_id(number): The requested file corresponding Question set ID

Frontend API: handleAPIError	
Description	The method process the received error response and displays appropriate error message
Server API URL	`None`
Parameters	errorRes: An error received from the application server

Frontend API: getSingleQuestionSet	
Description	The method makes the server API request to fetch all question sets corresponds to an assignment with provided assignment id in the course
Server API URL	
Parameters	assign_id(number): Question set corresponding Assignment ID

4.3.3 Component Overview and API's

4.3.3.1 Assignment Manager Component

Assignment Manager component provides the capabilities to perform all assignment and question set related operations in the Course. Assignment manager view provides user interface for Assignment and Question Set management in the Course. View list out all the assignments in the course and provides necessary control buttons to perform assignment specific operations. Each list item contains the assignment details and action buttons like edit and delete. The list of question set corresponding to each assignment is displayed nested below to respective assignment. To add the Assignment / Question Set view provides button control at the top of respective section. By clicking on Question set name / Rubric action item user can navigate to respective Question manager / Rubric manager of corresponding Question set. Assignment manager component contains properties and method API's to perform front-end application logic for Assignment and Question Set management in the Course. method API's of component provide functionalities like fetch assignment/question set data, add/edit/delete assignment/question set to course, navigate to rubric/ question manager by storing required details to the local storage

Frontend API: addAssignment	
Description	The method create a popup window containing the form to create a new assignment in the course
Parameters	None

Frontend API: addQuestionSet	
Description	Whenever user clicks the add questionset button under any assignment, view triggers call to this method with respective assignment to add new question set. Method takes the user selected assignment and create a popup window contains form to create new question set for the respective Assignment
Parameters	assignId(number): The list index of user selected assignment to add a question set

Frontend API: deleteAssignment	
Description	The method takes user selected assignment and makes the service API call to delete the user selected assignment and the assignment corresponding data like the questions sets, question files, questions, rubrics ,etc.
Parameters	assignId(number): The list index of user selected assignment

Frontend API: deleteQuestionSet	
Description	Whenever user clicks the delete questionset button for any question set, view triggers call to this method with respective question set and assignment to delete the question set. Method takes user selected question set data and make the service API call to delete the question set for the respective Assignment
Parameters	• assignIdx(number): The list index of the question set corresponding assignment • qsetIdx(number): The list index of user selected question set to be deleted

Frontend API: editAssignment	
Description	The method takes user selected assignment and shows the popup window containing form to edit selected assignment data
Parameters	assignId(number): The list index of user selected assignment

Frontend API: editQuestionSet	
Description	Whenever user clicks the edit questionset button for any question set, view triggers call to this method with respective question set and assignment to edit the question set Method takes user selected question set data and create a popup window contains form to edit the question set for the respective Assignment
Parameters	• assignIdx(number): The list index of the question set corresponding assignment • qsetIdx(number): The list index of user selected question set to be deleted

Frontend API: getQuestionSetFile	
Description	Whenever user clicks the Main Question set filename for any question set, view triggers call to this method with respective question set and assignment to download the main Question set file Method takes user selected question set data and make the service API call to download the main question set file for the respective question set
Parameters	• assignIdx(number): The list index of the question set corresponding assignment • qsetIdx(number): The list index of user selected question set to download the corresponding main file

Frontend API: getSolutionFile	
Description	Whenever user clicks the Solution filename for any question set, view triggers call to this method with respective question set and assignment to download the Solution file Method takes user selected question set data and make the service API call to download the Solution file for the respective question set
Parameters	• assignIdx(number): The list index of the question set corresponding assignment • qsetIdx(number): The list index of user selected question set to download the corresponding Solution file

Frontend API: gotoQuestionSet	
Description	Whenever user clicks the question set name link for any question set, view triggers call to this method with respective question set and assignment to navigate to Question Manager Method takes user selected question set and assignment data, store it to local storage and navigate to Question Manager View
Parameters	• assignIdx(number): The list index of the question set corresponding assignment • qsetIdx(number): The list index of user selected question set for Question manager navigation

Frontend API: onComponentInit	
Description	The component life-cycle hook method used to fetch existing Course Assignments on Component initialization Method fetch list of question set for each assignment on successful retrieval of assignment data
Parameters	None

Frontend API: goToRubricManager	
Description	Whenever user clicks the rubric action item for any question set, view triggers call to this method with respective question set and assignment to navigate to Rubric Manager Method takes user selected question set and assignment data, store it to the local storage and navigate to Rubric Manager View
Parameters	None

4.3.3.2 Question Manager Component

Question Manager component provides the capabilities to perform all Question related operations in the Course. Question Manager view mainly divides into two parts Question File view and Question Tree view. Question file view display the main question set file with area select box for each page of file. Question Tree view shows all questions of corresponding question set using the tree representation with action buttons at the top of view. Control action buttons provide user accessibility to add new question, edit/delete existing question, select name/roll number/question area coordinates on the Question file page. In order to perform add question operation user need to select parent question from tree and then click on the add button, after that view adds new node in the tree rooted at the selected question and provides input text box to enter new question title. User need to press enter key by providing non empty question title in order to add Question successfully. If user provided empty title then newly added question node is removed from tree and appropriate error message displayed. For adding / editing question details user must select the question and click on edit button. After that form will be shown with corresponding question fields and appropriate action buttons. For setting question area coordinates on file page, user must select the question, then click on set Question coords action button. After that as soon as user updates box position, view triggers call to update coordinates at server database. For setting the name/roll number coordinates, user must click on respective action button. After that as soon as user updates box position, view triggers call to update coordinates at server database. Question manager component contains properties and method API's to perform front-end application logic for Question management of corresponding Question set. Method API's of component provide functionalities like fetch Question data, add/edit/delete question to the Question set

Frontend API: deleteQuestion	
Description	The method checks that whether user selects any question or not before clicking delete button, shows error message and returns if no question selected by user. Check that selected question must not be non leaf node as user not allowed to delete question having child questions. If all validations passed, then make an server API call to delete the selected question from application database and update the View accordingly
Parameters	None

Frontend API: fetchQuestions	
Description	Method fetch the list of Questions corresponding to selected question set using service API fetchQuestions. On successful retrieval of Question list, set the question tree object which used by Question view
Parameters	None

Frontend API: fetchTopics	
Description	etch the list of Course Topics from application server and store into component property topics for availability to question view
Parameters	None

Frontend API: imageCropped	
Description	The Question Manager View calls this method once area box position updated by user on question file page. The imageCropped method find the area update mode(Question page area, roll number area, user name area) and call the respective component API method to update area coordinates at application database
Parameters	None

Frontend API: onComponentInit	
Description	The component life-cycle hook method used to fetch existing list of Questions, Main Question set file corresponding to user selected QuestionSet on component initialization
Parameters	None

Frontend API: onNodeCreated	
Description	On receiving triggered event from view, method sets the question fields parent id, sub-part no, is_actual_question, filepage for Question to be added. After setting new Question fields, method makes an server API request to add new Question to application database. If server responds error, then method removes newly added question from Tree view and shows appropriate error notification
Parameters	None

Frontend API: updateQCoords	
Description	The method makes an server API request to update question coordinates and page no for user selected question at the application database
Parameters	• coords (string): The area coordinates of question on question set file page • page_no (number): The question corresponding file page number

Frontend API: updateQuestionDetails	
Description	The method retrieves the Question edit form data and makes call to update question details at application server using service API. On receiving server response, method shows appropriate error/success notification message
Parameters	None

Frontend API: updateQSetNameCoords	
Description	The method makes an server API request to update user name coordinates for corresponding question set at application database
Parameters	coords (string): The area coordinates of user name on question set file page

Frontend API: updateQSetRollCoords	
Description	The method makes an server API request to update user roll number coordinates for corresponding question set at application database
Parameters	coords (string): The area coordinates of user roll number on question set file page

4.3.3.3 Rubric Manager Component

Rubric Manager component provides the capabilities to perform all Rubric related operations in the Course. Rubric Manager view mainly divides into two parts Question File view and Rubric manager view. Question file view display the main question set file with area select box for each page of file. rubric manager view shows Question with some details and corresponding list of rubrics in Smart table. Smart table provides features to add/edit/delete rubric, searching, sorting and pagination. View stores list of questions in a queue , such that user can navigate through all questions using arrow buttons provided at the top of view Rubric manager component contains properties and method API's to perform frontend application logic for Rubric management of corresponding Question set. method API's of component provide functionalities like fetch Rubric data, add/edit/delete rubric to Question set

Frontend API: loadRubrics	
Description	The method fetch the list of Rubrics corresponding to question of selected QuestionSet using service API fetchRubrics
Parameters	None

Frontend API: nextQuestion	
Description	The component view calls this method once user clicks the forward arrow button to navigate next question method update the current question index and calls the method to refresh page details like current question details, rubric table data
Parameters	None

Frontend API: onComponentInit	
Description	The component life-cycle hook method used to fetch existing list of Questions, Rubrics and Main Question set file corresponding to user selected QuestionSet on Component initialization
Parameters	None

Frontend API: onCreateConfirm	
Description	The method calls the service API method to create new rubric by passing required rubric data. On successful response, method updates table data and shows appropriate notification message
Parameters	event: The event triggered by rubric smart table once user clicks the add confirm button

Frontend API: OnDelete	
Description	The method calls the service API method to delete an existing rubric by passing required rubric data. On successful response, method updates table data and shows appropriate notification message
Parameters	event: The event triggered by rubric smart table once user clicks the delete confirm button

Frontend API: onEditConfirm	
Description	The method calls the service API method to edit existing rubric by passing required rubric data. On successful response, method updates table data and shows appropriate notification message
Parameters	None

Frontend API: prevQuestion	
Description	The view calls this method once user clicks the backward arrow button to navigate prev question method update the current question index and calls the method to refresh page details like current question details, rubric table data
Parameters	None

Frontend API: refreshPageDetails	
Description	The method refresh the page details using current question index property of component. The page details includes Question details, page no of question and corresponding list of rubrics
Parameters	None

4.4 Event Manager Module

4.4.1 Overview

Using the Events and Sub-Events, The SPHINX system provides a nice way to perform detailed customization of many possible activities related to the assignment or the exam. In order to release the assignment, the user needs to create an event associated with the assignment. Users can set the time window, only during that period, the assignment is active and visible to corresponding users. The various types of sub-events provides a way to set the customized configuration for various things related to the event. The sub-event basically represents the type of activities possible in the event. Such that users can set the activity related setting using the sub-events. we explained the various configuration parameters provided by each sub-event in detailed in the API documentation

of the SPHINX front-end. The Event Manager module mainly provides all the capabilities required to perform the event management in the course. Some of the major capabilities are,

- Display the list of all the existing events and corresponding sub-events in the course
- Add new events and sub-events of various types in the course to the application server
- Update the various details of the existing event and the sub-events in the course
- Delete any existing event or sub-event in the course from the application server

4.4.2 Event Manager Service

4.4.2.1 Overview

The Event Manager service contains the method API's to perform all necessary server communication required for the Event Management in the Course. It provides method API's mainly for view, create, edit, and delete operations related to events and sub-events As the event and sub-event related server API's required large number of parameters, The service methods performs necessary data extraction and type conversions of various data properties of received model objects. The service performs the server error response handling and returns appropriate error message

4.4.2.2 API Methods

Frontend API: addEvent	
Description	The method makes the server post API request to add the new event in the course at the application server
Server API URL	`/course/#courseid/events`
Parameters	event_data(EventClass): The EventClass model object contains the event to be created. Model holds event information like name, corresponding assignment ID, grade aggregation method and the is_external flag

Frontend API: addSubEvent	
Description	The method makes a server post API request to create new subevent in the event at the application database. The server API for subevent creation requires large number of parameters representing various subevent properties. Using the received SubEvent model object, method set the required parameters of server API in the angular formData object
Server API URL	`/course/#courseid/event/#eventid/subevents`
Parameters	subevent_data(SubEvent): The object of SubEvent Model class, contains user provided subevent details

Frontend API: deleteEvent	
Description	The method makes server API request to delete the existing event from the course in the application server
Server API URL	`/course/#courseid/event/#eventid`
Parameters	event_data(EventClass): The EventClass model object contains the event details to be deleted

Frontend API: deleteSubEvent	
Description	he method makes a server API call to delete the existing subevent from the application server
Server API URL	`/course/#courseid/event/#eventid/subevent/#subeventid`
Parameters	subevent_data(SubEvent): The object of SubEvent Model class to be deleted from the application server

Frontend API: editEvent	
Description	The method makes server put API request to update the existing event details with new provided data at the application server
Server API URL	`/course/#courseid/event/#eventid`
Parameters	event_data(EventClass): The EventClass model object contains the event details to be updated Model holds event information like event id, name, corresponding assignment id, grade aggregation method and the is_external flag

Frontend API: editSubEvent	
Description	The method makes a server API call to edit the existing subevent at the application server. While making request, method sending the multiple required parameters as the angular FormData object. For the date related parameters, method converts the received subevent properties into ISO string formats
Server API URL	`/course/#courseid/event/#eventid/subevent/#subeventid`
Parameters	subevent_data(SubEvent): The object of SubEvent Model class, used to send server API request parameters

Frontend API: fetchEvents	
Description	the method fetches all the events present in the course. It makes the get API request to application server using http object. On receiving successful response from server, method taps the server response and performs data formatting in order to store it. The server response mainly consist of list of events and nested list of corresponding subevents in json format. During data formatting, the method mainly create the list of EventClass model objects using response data. Where each EventClass model object contains event information and corresponding list of SubEvents. Then method stores the extracted event list in localstorage to make it available to application components
Server API URL	`/course/#courseid/events`
Parameters	None

4.4.3 Event Manager Component

4.4.3.1 Overview

In order to provide the module capabilities, the component view is designed as follows. The view mainly consists of 5 parts,

1. The data display view is designed as the 2-layered nested table. The outer Table created using the list component provided by the external nebular library. While the inner table is wrapped inside the list item of the outer table
2. Form to create a new event, initially in a hidden mode
3. Form to edit the existing event details, initially in a hidden mode. As the user click's edit button for the event, the view hides the corresponding event list item and displays the edit form by changing the hidden mode
4. The view consists of the Form to create a new subevent, initially in a hidden mode. The subevent edit form is designed using the nebular stepper Angular UI component. As subevent creation requires a large number of parameters, the form is divided into two steps. On the first step, the form takes the user input for the information common to all types of the subevent. While The second step collects the subevent type-specific information, so based on the user-selected subevent type, the corresponding form displayed on the second step. When the user clicks the Add new subevent, the component displays the form view by hiding the data display view
5. The form to edit the existing subevent details, initially in a hidden mode. When the user clicks the edit button for any subevent, the component displays the subevent edit form view by hiding the data display view

On the EventManager view initialization, The component fetches the Event and Subevent data and set the component properties. Using those properties, view displays the data in a nested tabular view. Whenever the user clicks the Add/Edit event button, The component toggle the corresponding hidden form and binds the component property to access the form data for further server communication. To add/edit the subevent, once the user clicks the Add/Edit subevent button, the component hides the actual tabular data view and shows the corresponding form view

4.4.3.2 API Methods

Frontend API: addEvent	
Description	The method makes an server API call to add a new event using event manager service API addEvent. On receiving successful response from server, method toggles the add event form, update the data display view with updated data list and shows appropriate notification message
Parameters	None

Frontend API: cancelAddSubeventForm	
Description	The method used to hide the subevent create form view by showing the hidden main data table view. It resets the form data object with empty data values
Parameters	None

Frontend API: cancelEditSubeventForm	
Description	The method used to hide the subevent edit form view by showing the hidden main data table view. It resets the form data object with empty data values
Parameters	None

Frontend API: fetchAssignments	
Description	The method fetches the list of course assignments from the application database using service API. On receiving successful response from server, method sets the assignments property of the component and makes data available to component view and other API's
Parameters	None

Frontend API: getEvents	
Description	The method fetch the Event and Subevent data using service API fetchEvents. On receiving successful response from server, method loads the processed server response from local storage and set the component property events and makes data available to component view and other API's
Parameters	None

Frontend API: onComponentInit	
Description	The component life-cycle hook, get called on every component initialization. It used to make required API calls to fetch all event and subevent data, initialize the properties which binds to view forms present for the event and subevent
Parameters	None

Frontend API: onDeleteEvent	
Description	The method calls the server API request to delete the event from the course using the service API deleteEvent. On receiving success response from the server, method updates the events property of component in order to refresh the view. It also shows appropriate toastr notification based on the received server response
Parameters	event_idx (number): The list index of user-selected event for deletion

Frontend API: onDeleteSubEvent	
Description	The method calls the server API request to delete the subevent from the course using the service API deleteSubEvent. On receiving success response from the server, method updates the events property of component in order to refresh the view. Apropriate toastr notification is shown based on the received server response
Parameters	• event_idx (number): The list index of the event corresponding to the user-selected subevent for deletion • subevent_idx (number): The list index of user-selected subevent for deletion

Frontend API: onEditEvent	
Description	Once the user clicks the edit button for any event, the view triggers call to the onEditEvent method. The method loads the edit form with the user-selected event data, then display the edit form by calling the toggleEventEditForm API of the component
Parameters	event_idx (number): The list index of the user-selected event

Frontend API: onSelectRDS	
Description	The method used to dynamically update the form fields of subevent related form based on user action. For the subevent of type RGUPLOAD, if user selects QRN as regrading duty scheme, method displays the participant's CSV file input field by setting the participants_spec property of the subevent to 0
Parameters	None

Frontend API: openAddSubeventForm	
Description	The method used to display the subevent create form view by hiding the main data table view. It fetches the required existing subevent lists based on type for subevent creation, like list of existing SUPLOAD's, list of existing GUPLOAD's, list of existing RGUPLOAD's in the same corresponding event
Parameters	event_idx (number): The list index of the event corresponds to the new subevent reques

Frontend API: openEditSubeventForm	
Description	The method used to display the subevent edit form view by hiding the main data table view. It loads the user-selected subevent details into the edit form. The details mainly includes time range values like start time, end time, display end time, late end time, displate late end time, and etc.
Parameters	• event_idx (number): The list index of the event corresponds to the user-selected subevent • subevent_idx (number): The list index of the user-selected subevent

Frontend API: submitAddSubeventForm	
Description	The method validate the data values of the add subevent form. If the subevent type is SUPLOAD and submission mode is not OLI, then participant's CSV file must provided else method shows an error notification and returns. If the subevent type is AVIEW/QVIEW, then participants of subevent must be same as participants of gen_subevent list. If the subevent type is RGUPLOAD and regrading duty scheme is RQN, then participant's CSV file must provided else method shows an error notification and returns. The method makes server API request to add the new subevent to the event using the service provided addSubEvent method. On receiving success response from the server, it updates the list of subevents in the view. If server responds error, then method performs error processing and shows the toastr notification with error message received from the server
Parameters	None

Frontend API: submitEditEvent	
Description	The method makes an server API request to update the event details using service provided editEvent method. On receiving success response from the server, method hide the event edit form and update the event list view with updated data. If server returns an error response, method handles the error and shows the toastr notification with the error message received from the server
Parameters	None

Frontend API: submitEditSubeventForm	
Description	The method validate the user-provided display end time must greater than end time, else display the toastr notification with error message and returns. The method makes an server API request to update the subevent details using service provided editSubEvent method. On receiving success response from the server, method hide the subevent edit form and update the subevent list view with updated data. If server returns an error response, method handles the error and shows the toastr notification with the error message received from the server
Parameters	None

Frontend API: toggleEventCreateForm	
Description	The method toggle the display mode of the Event creation form by inverting value of the property isOpenCreateForm
Parameters	None

Frontend API: toggleEventEditForm	
Description	The method toggle the display mode of the edit Event form by inverting value of the property isOpenEditForm
Parameters	None

Frontend API: toggleSubEventCreateForm	
Description	The method toggle the display mode of the SubEvent creation form by inverting value of the property isOpenSubeventCreateForm
Parameters	None

Frontend API: toggleSubEventEditForm	
Description	The method toggle the display mode of the edit SubEvent form by inverting value of the property isOpenSubeventEditForm
Parameters	None

4.5 User Events Manager Module

4.5.1 Overview

The module provided major Capabilities are listed below,

1. View all ongoing events corresponding to the logged-in user in the course
2. For each event, provide the user allowed capabilities based on subevents configuration
   (a) View and download event associated files like the Main Question file, the supplementary file, and the solution file
   (b) Perform main event submission and other submission related activities
   (c) Perform Grading of event submission copies
   (d) View the Grades and the graded copy
   (e) Submit Regrading requests
   (f) Review the regrading requests and regrade the copies

In order to provide the above capabilities in detail, the module is divided into five parts,

1. **MyEvents Dashboard:**

   The dashboard shows a list of all ongoing events that correspond to the user in the course. Each event item from the list contains the action items allowed to the user, for example, make a submission, view the grading duties, view the regrading duties, etc. The action items provide a way to navigate to the respective activity page. Once a user clicks any action item, the required details for the corresponding activity page are stored in the local storage and redirect the user to the corresponding activity page. If the user corresponding QVIEW / AVIEW subevent present in the event, then the dashboard shows the table of questions sets and file links associated with the event, the user can download files using the file links. For more information, see the MyEvents Manager component sub-section

2. **The Submission Manager:**

   Once the user clicks the submit button on the event dashboard, the myEvents Manager component stores the corresponding SUPLOAD subevent and the event information to the local storage and redirects the user to the submission manager. The submission manager provides the user interface to upload the submission related files and submit pagination for each question in the assignment. The Pagination form takes the start page index of the answer corresponding to all questions as the user input. The submission manager loads the SUPLOAD subevent info from the local storage, and provide the necessary UI features based upon subevent configuration parameters. The UI features mainly contain create/join the submission group, access code verification, change the submission associated question set, and download uploaded submission files. For more details, see the submission manager component subsection

3. **The Grading Manager:**

   The grading manager consists of two main components, the grading manager dashboard, and the main grade page. Once the user clicks the view grading duties button on the event dashboard, the myEvents Manager component stores the corresponding GUPLOAD subevent & the event information to the local storage then redirects the user to the grading duty dashboard.

   The dashboard loads the subevent & event information from the local storage and fetches the list of grading duties associated with the GUPLOAD subevent from the server. The dashboard displays all the user allocated grading duties according to the grading scheme of the GUPLOAD subevent. It shows the multiple lists of grading duties, where each list corresponds to the question in the event associated assignment. Where each list item contains grading duty details like submission group id, grading status, graded marks. To perform the grading, the user can select any submission group id or click the perform grading button. Once the user selects the submission group id, the dashboard saves the grading duty details in local storage and redirects the user to the main grade page.

   The main grade page loads the user submission and grading duty details and provides access controls to perform the rubric-based grading. The navigation link to the grading dashboard is provided in the view such that the user can navigate to the dashboard in between grading. For more details, see the mainGrade component sub-section

4. **The grade view Manager:**

   The grade view manager mainly provides capabilities to view the grades with the graded copy and make regrade requests. The manager mainly consists of two components, the grade sheet view component and the main grade view component.

   The grade sheet view lists all the answered questions with corresponding graded marks and total marks. To view the graded copy, the user can navigate to the main grade view by clicking on the respective questions. The component saves the required details like submission id, associated grading duties before navigating to the main grade view.

   The main grade view shows the main submission copy and the grading duties associated with the user-selected question. Each grading duty displays all the rubrics with the applied one by the grader in the checked state. Also, the main grade view provides a regrading box to the user. The regrading box contains the previous regrading requests and grader response messages. Users can submit a new regrade request using the provided regrade request window. For more details, see the main grade view component sub-section

5. **The Regrading Manager:**

   The regrading manager provides the capabilities to view all allocated regrade requests, review and perform regrading for each re-grade request. The regrading manager mainly consists of two components, the regrading duty dashboard, and the main re-grade page. Once the user clicks the view regrading duties button on the myevents dashboard, the component saves the corresponding RGUPLOAD subevent and event details in the local storage, then navigates to the regrading duty dashboard.

The regrading duty dashboard displays the list of all regrading requests allocated to the user using the corresponding RGUPLOAD subevent. Each list item represents the regrade request details like submission id, review status, and graded marks. By clicking on the regrade request, the user can navigate to the main regrade page to review the regrade request.

The main regrade page is mostly similar to the main grade page, in addition, it provides the regrade request box and previous grading duties associated with the submission. The regrading box contains the previous regrading requests and grader response messages correspond to the submission. It provides a way to submit the review comment of the regrade request. For more details, see README of main regrade component

4.5.2 User Events Manager Service

4.5.2.1 Overview

The MyEventsService is the important injectable provided by the MyEvents module, the service contains a large number of APIs as it performs all the necessary server communication for the MyEvents Dashboard, the submission manager, the grading manager, the grade sheet view manager, and the regrading manager. The service provided API's mainly perform server communication for below activities(grouped by the corresponding manager)

- MyEvents Dashboard
  - Fetch the event and subevent list corresponding to the user
  - Fetch event associated files like the main question file, the supplementary file, and the solution file
- The Submission Manager
  - Fetch the user submission details in the event
  - create/ join the submission group in the event
  - Perform pre-submission activities like access code verification, change the submission associated question set
  - Fetch/upload the submission related files, like the main submission file and supplementary submission file
  - Fetch/update/delete the pagination for the submitted main file
- The Grading Manager, The Regrading manager
  - Fetch all user associated grading duties in the event
  - Update grading duty details at the server - it covers the total grade submission, review regrade requests
  - Fetch/add/delete grading duty - rubric association
- The Grade Sheet view manager
  - Fetch user grade sheet in the event
  - Fetch/create the regrading requests for any question in the grade sheet

4.5.2.2 API Methods

Frontend API: createSubmissionGroup	
Description	The method makes an server API request to create the submission group for the logged-in user in the given event
Server API URL	`/course/#courseid/event/#eventid/mySubmissions/`
Parameters	myEvent (MyEventClass): The request corresponding event object of MyEventClass model

Frontend API: delResponsePagination	
Description	The method makes an server API request to delete the question pagination link submitted by the user in the given event
Server API URL	`/course/#courseid/event/#eventid/myResponses/#question_id/#pageno/`
Parameters	• event_idx (number): The request corresponding event ID • question_idx (number): The request corresponding question ID • page_no (number): The user-selected page number

Frontend API: delRubricGradingDutyLink	
Description	The method makes an server API request to remove the applied rubric to the requested grading duty ID in the given subevent & event
Server API URL	`/course/#courseid/event/#eventid/myGrading/#subeventid/gradingDuty/#gradingdutyid/#rubricId`
Parameters	• event_idx (number): The request corresponding event ID • subevent_idx (number): The request corresponding subevent ID • gd_idx (number): The request corresponding grading duty ID • rubric_idx (number): The request corresponding rubric ID

Frontend API: fetchMyEvents	
Description	The method calls the server API to get all the events and corresponding subevents associated with the logged-in user. On receiving successful response from the server, method iterate through JSON list and extract the event properties. It stores the extracted data as list of MyEventClass model objects, where each object stores the event properties and corresponding subevents as list of MySubevent model objects. For each event, method also sets the event flags(for each subevent,showing whether subevent is present or not), progress bar values for the subevent timeline, active subevent flags, etc. the method stores the extracted list of MyEventClass model objects in the local storage, such that extracted data is available to the all components
Server API URL	`/course/#courseid/myEvents`
Parameters	None

Frontend API: fetchMySubmissions	
Description	The method makes an server API request to fetch the user-submission details in the received event
Server API URL	`/course/#courseid/event/#eventid/mySubmissions/`
Parameters	my_event (MyEventClass): The request corresponding event object of MyEventClass model

Frontend API: fetchQuestionSets	
Description	The method makes the server API request to fetch the user-allowed questions sets and corresponding files in the requested event.There must QVIEW subevent present coresond to the logged-in user in the event, else server returns an empty response
Server API URL	`/course/#courseid/event/#eventid/myQuestions`
Parameters	my_event (MyEventClass): The event object of which question sets data to be fetched

Frontend API: fetchSolutionSets	
Description	The method makes the server API request to fetch the gold solution of questions sets and corresponding files in the requested event.There must AVIEW subevent present coresond to the logged-in user in the event, else server returns an empty response
Server API URL	`/course/#courseid/event/#eventid/mySolutions`
Parameters	my_event (MyEventClass): The request corresponding event object of MyEventClass model

Frontend API: getGradingDutyDetails	
Description	The method makes an server API request to fetch the grading duty details of the requested grading duty ID in the given sub-event & event
Server API URL	`/course/#courseid/event/#eventid/myGrading/#subeventid/gradingDuty/#gradingdutyid`
Parameters	• event_idx (number): The request corresponding event ID • subevent_idx (number): The request corresponding subevent ID • gd_idx (number): The request corresponding grading duty ID

Frontend API: getMainSubmissionFile	
Description	The method makes an server API request to get the main submission file related to the requested grading duty ID in the given subevent & event
Server API URL	`/course/#courseid/event/#eventid/myGrading/#subeventid/gradingDuty/#gradingdutyid/main`
Parameters	• event_idx (number): The request corresponding event ID • subevent_idx (number): The request corresponding subevent ID • gd_idx (number): The request corresponding grading duty ID

Frontend API: getMyGrades	
Description	The method makes an server API request to fetch the grade sheet of the submission associated to the given event
Server API URL	`/course/#courseid/event/#eventid/myMarks`
Parameters	event_id (number): The request corresponding event ID

Frontend API: getMyGradingDuties	
Description	The method makes an server API request to fetch the list of grading duties allocated to the user in the given subevent & event
Server API URL	`/course/#courseid/event/#eventid/myGrading/#subeventid/`
Parameters	• event_idx (number): The request corresponding event ID • subevent_idx (number): The request corresponding subevent ID

Frontend API: getMyMainSubmission	
Description	The method makes an server API request to fetch the main submission file uploaded by the user in the given event
Server API URL	`/course/#courseid/event/#eventid/mySubmissions/main`
Parameters	event_idx (number): The request corresponding event ID

Frontend API: getMySupplSubmission	
Description	The method makes an server API request to fetch the supplementary submission file uploaded by the user in the given event
Server API URL	`/course/#courseid/event/#eventid/mySubmissions/supplementary`
Parameters	event_idx (number): The request corresponding event ID

Frontend API: getResponsePagination	
Description	The method makes an server API request to fetch the question pagination for all questions submitted by the user during the main submission in the given event
Server API URL	`/course/#courseid/event/#eventid/myResponses/`
Parameters	event (MyEventClass): The request corresponding event object of MyEventClass model

Frontend API: getSuppSubmissionFile	
Description	The method makes an server API request to get the supplementary submission file related to the requested grading duty ID in the given subevent & event
Server API URL	`/course/#courseid/event/#eventid/myGrading/#subeventid/gradingDuty/#gradingdutyid/supplementaryy`
Parameters	• event_idx (number): The request corresponding event ID • subevent_idx (number): The request corresponding subevent ID • gd_idx (number): The request corresponding grading duty ID

Frontend API: isSubeventActiveAtMoment	
Description	The method checks whether the received event is active at the moment by using start and end time, then returns the Boolean value accordingly
Server API URL	`None`
Parameters	subevent (SubEvent): The request corresponding subevent object

Frontend API: joinSubmissionGroup	
Description	The method makes an server API request to join the user-provided submission group for the logged-in user in the given event. It uses the user-selected submission group id from the received event model object to make the server request
Server API URL	`/course/#courseid/event/#eventid/mySubmissions/#submission_group_id/join`
Parameters	event (MyEventClass): The request corresponding event object of MyEventClass model

Frontend API: setResponsePagination	
Description	The method makes an server API request to add the question pagination link updated by the user in the given event
Server API URL	`/course/#courseid/event/#eventid/myResponses/#question_id/#pageno/`
Parameters	• event_idx (number): The request corresponding event ID • question_idx (number): The request corresponding question ID • page_no (number): The user-selected page number

Frontend API: setRubricGradingDutyLink	
Description	The method makes an server API request to apply the rubric to the requested grading duty ID in the given subevent & event
Server API URL	`/course/#courseid/event/#eventid/myGrading/#subeventid/gradingDuty/#gradingdutyid/#rubricId`
Parameters	The database ID's of the request corresponding event, sub-event, grading duty, and rubric

Frontend API: submitRegradeRequest	
Description	The method makes an server API request to raise a regrade request for the user selected question in the grade sheet, the grade sheet and question must exist in the provided subevent and the event, else server returns the error
Server API URL	`/course/#courseid/event/#eventid/subevent/#subeventid/response/#responseid/regrading`
Parameters	• event_idx (number): The request corresponding event ID • subevent_idx (number): The request corresponding subevent ID • response_idx (number): The request corresponding response ID • stud_comment (string): The request corresponding student comment

Frontend API: updateGradingDutyDetails	
Description	The method makes an server API request to update the grading duty details of the requested grading duty ID in the given subevent & event
Server API URL	`/course/#courseid/event/#eventid/myGrading/#subeventid/gradingDuty/#gradingdutyid`
Parameters	• event_idx (number): The request corresponding event ID • subevent_idx (number): The request corresponding subevent ID • gd (MyGradingDuty): The request corresponding Grading duty object

Frontend API: updateMyPreSubmission	
Description	The method makes an server API request to update the submission group details like associated question set, the access code in the given event. It uses the submission_ group_form property of the received event to retrieve the required parameters for API request
Server API URL	`/course/#courseid/event/#eventid/mySubmissions/`
Parameters	event (MyEventClass): The request corresponding event object of MyEventClass model

Frontend API: updateReGradingDutyDetails	
Description	The method makes an server API request to update the grading duty details of the requested regrading duty ID in the given subevent & event
Server API URL	`/course/#courseid/event/#eventid/myGrading/#subeventid/gradingDuty/#gradingdutyid`
Parameters	• event_idx (number): The request corresponding event ID • subevent_idx (number): The request corresponding subevent ID • rgd (MyReGradingDuty): the request corresponding Re-Grading duty object

Frontend API: uploadMainSubmission	
Description	The method makes an server API request to upload the main submission file submitted by the user in the given event. the method sends the file and file hash to the server, So the server verify that no data tampering happen during the communication
Server API URL	`/course/#courseid/event/#eventid/mySubmissions/main`
Parameters	• event (MyEventClass): The request corresponding event object of MyEventClass model • file (File): The user submitted file • hash (string): The Sha1 hash of the file

Frontend API: uploadSuppSubmission	
Description	The method makes an server API request to upload the supplementary submission file submitted by the user in the given event. the method sends the file and file hash to the server, So the server verify that no data tampering happen during the communication
Server API URL	`/course/#courseid/event/#eventid/mySubmissions/supplementary`
Parameters	• event (MyEventClass): The request corresponding event object of MyEventClass model • file (File): The user submitted file • hash (string): The Sha1 hash of the file

Frontend API: verifyMyNAC	
Description	The method makes an server API request to verify the access code for the submission in the given event. It uses the submission_group_form property of the received event to retrieve the required parameters for API request
Server API URL	`/course/#courseid/event/#eventid/mySubmission/verifyNAC/`
Parameters	event (MyEventClass): The request corresponding event object of MyEventClass model

4.5.3 Component Overview and API's

4.5.3.1 My Events Dashboard Component

The MyEvents Dashboard view displays the list of user corresponding events. For each list item, the view process the list of subevents associated with the respective event and provide the action items to the user. If the subevent is active(start time<= current time<= end time) then based on the subevent type, the view designed as below

- **QVIEW:** The view list out the event associated question sets and their corresponding files in the table format. Each row contains the question set name, main question file name, and supplementary file name. The view provides file names as hyperlinks, which user can use to download the files on the local system

- **SUPLOAD/ SVIEW:** The view displays the timeline progress bar for the subevent. If the SUPLOAD or SVIEW subevent active, then the view provides a navigation button to the submission manager. Once the user clicks the submit button, the component saves the corresponding event object & the SUPLOAD subevent object in the local storage before navigating to the submission manager

- **MVIEW:** The view displays the timeline progress bar for the subevent. If the MVIEW subevent active, then the view provides a navigation button to the gradeview manager. Once the user clicks the view grades button, the component saves the corresponding event object & the MVIEW subevent object in the local storage before navigating to the gradeview manager

- **GUPLOAD/GVIEW:** As the user may have associated with multiple GUPLOAD subevents in the event, the view displays all the GUPLOAD subevents in table format. If the GUPLOAD / GVIEW is active, then the view provides a navigation button to the grading manager. Once the user clicks the view grading duty button, the component saves the corresponding event object & the GUPLOAD subevent object in the local storage before navigating to the grading manager

- **RGUPLOAD/RGVIEW:** As the user may have associated with multiple RGUPLOAD subevents in the event, the view displays all the RGUPLOAD subevents in table format. If the RGUPLOAD / RGVIEW is active, then the view provides a navigation button to the re-grading manager. Once the user clicks the view regrading duty button, the component saves the corresponding event object & the RGUPLOAD subevent object in the local storage before navigating to the re-grading manager

On initialization, the component fetches the events data from the server and performs data formatting, so data is available to view in the required format. During data formatting, component iterate through the event list and set the event properties like subevent flags, progress bar values which help to render the view correctly. The component provides the navigation method APIs which handle user-triggered requests and redirects the user to respective view by saving required information into the local storage. It also provides the APIs, which download the user-requested

files from the server to the local system. See the below mentioned component API's for more detailed information

Frontend API: fetchMyGradingDuties	
Description	The method calls the server API to fetch the grading duties allocated to the user using the service API fetchMyGradingDuties
Parameters	• event_idx (number): The list index of event associated with the user's action on the view • subevent_idx (number): The list index of subevent associated with the user's action on the view

Frontend API: getMyEvents	
Description	The method fetch the user corresponding Event and Subevent data using the service API fetchMyEvents. On receiving successful response from the server, method loads the processed response data from local storage and set the component property my_events. For each event in the event list, the method fetch the event associated question set information like the question set details, the question set files, the solution file, and the list of questions belongs to the question set
Parameters	None

Frontend API: getQuestionFile	
Description	The method download the user requested main question file of the question set in the local system
Parameters	• event_idx (number): The event ID corresponding to the user-selected file • qset_idx (number): The question set ID corresponding to the user-selected file

Frontend API: getSolutionFile	
Description	The method download the user requested solution file of the question set in the local system
Parameters	• event_idx (number): The event ID corresponding to the user-selected file • qset_idx (number): The question set ID corresponding to the user-selected file

Frontend API: getSupplementaryFile	
Description	The method download the user requested supplementary file of the question set in the local system
Parameters	• event_idx (number): The event ID corresponding to the user-selected file • qset_idx (number): The question set ID corresponding to the user-selected file

Frontend API: goToGradeViewManager	
Description	The method checks whether user corresponding MVIEW subevent is going on at the moment, if not then method shows error notification and returns. It stores the event object corresponding to received ID in the local storage and redirects the user to the Grade sheet view manager
Parameters	event_idx (number): The list index of event associated with the user's action on the view

Frontend API: gotoGradingManager	
Description	The method stores the event object and database id's of the event & subevent corresponding to the user selected grading subevent in the local storage and redirects the user to the Grading manager
Parameters	• event_idx (number): The list index of event associated with the user's action on the view • subevent_idx (number): The list index of subevent associated with the user's action on the view

Frontend API: gotoReGradingManager	
Description	The method stores the event object and database id's of the event & subevent corresponding to the user selected re-grading subevent in the local storage and redirects the user to the Re-Grading manager
Parameters	• event_idx (number): The list index of event associated with the user's action on the view • subevent_idx (number): The list index of subevent associated with the user's action on the view

Frontend API: gotoSubmissionManager	
Description	The method stores the event object corresponding to received ID in the local storage and redirects the user to the Submission manager
Parameters	event_idx (number): The list index of event associated with the user's action on the view

Frontend API: onComponentInit	
Description	The component lifecycle hook, get called on every component initialization. It used to make required API calls to fetch all user-corresponding event and subevent data, initialize the properties which binds to view forms present for the event and subevent
Parameters	None

4.5.3.2 Submission Manager Component

Capabilities:

The component provides the below-mentioned capabilities through the view and the component APIs,

- Create/Join the submission group
- Access code verification required to perform the submission
- Selection of the question set corresponding to the main submission
- Upload the main submission file, the supplementary submission file and download the respective uploaded files
- View the uploaded main submission file in an image carousel
- Provide the pagination(start page number of the answer in the main submission file) for each question in the question set and submit the assignment

The Component View design:

In order to provide the above capabilities, the view contains various form components and data display panels as listed below,

1. The Submission Group form: The form provides toggle choice input to create or join the submission group in the event. On selecting the choice, the form displays input fields accordingly to create/ join the submission group
2. The Access code verification form
3. The question set selection form: The form provides the list of question sets corresponding to the event in a dropdown, so the user can select and set the question set for the submission
4. The main submission file upload form: The form provides the file dropzone, such that user can select the file to upload using file directory or by dropping the file into the dropzone
5. The supplementary file upload form: The form structure is similar to the main submission file upload form
6. The action panel: The panel displays the submission associated question set name, the main submission file name, and the supplementary file name with the edit control button to open the respective form to edit the already selected question set or the uploaded files
7. The file view panel: The panel displays the user uploaded the main submission file in the image carousel such that user can view the file for providing the pagination for the assignment questions
8. The Submission panel: The panel provides a list of questions with a dropdown to select the pagination for each question of the submission associated question set. The dropdown contains the list of page numbers of the main submission fie. The panel contains control buttons to submit the uploaded file or cancel the submission. It also displays the submission group details like group ID, and list of group members

The component functionality:

The component provides the above-mentioned functionalities in three stages in a pipelined manner,the stages are the submission group creation, access code verification, and the other submission related activities. On the component initialization, component fetches below details in order to implement pipelined action flow,

- The submission corresponding SUPLOAD subevent and the event from the local storage
- The submission related details from the server, it mainly includes the selected question set, access code verification status, submission group details (if the submission group is already created)
- The list of question sets in the event from the local storage
- The main submission files from the server (if uploaded any)
- The pagination details for the submission (if uploaded any)

Using the SUPLOAD subevent configuration parameters and the server responses, the component takes decides the user-submission stage. If the submission group scheme is individual or fixed or the submission group is already created then the component moves the user to the second stage. Else the component displays the submission group form, Once the user creates/joins the submission group then component moves the user to the second stage.

If the access code verification flag is false in the SUPLOAD configuration, then the component moves the user to the third stage. Else the component displays the access code verification form, Once the user verifies the required access code then component moves the user to the last stage.

The above two stages are kind of the pre-submission stages, which need to be done for a single time if required. In the last stage, the component allows the user to submit the submission related files, provide the question-pagination, change the question set(if allowed), and make the final submission. If the question set scheme is open, then the user can change the question set associated with the submission multiple times. Once the user changes the question set, the respective list of questions is updated in the question-pagination panel view. The component allows the user to upload the submission files any number of times. Once the submission file uploaded, the component refreshes the file view panel data and resets the question pagination for all the questions in the submission panel. For uploading the submission file at the application server, the component generates the hash of the uploaded file using sha1 API of the external CryptoJS library [6]. The generated hash is sent with the server API request, such that the server verifies the file integrity before storing it in the application server. For the question pagination, whenever the user selects the page number for the question, the component makes an independent server API request to the server. Such that user submission state is persistent even user close the application in between submission. Once the user clicks the submit button, the component checks whether pagination for each question is provided or not. If no error occurred then It displays the success notification and redirects the user to the myEvents dashboard.

See the below mentioned component API's for detailed information

Frontend API: deleteOldPagination	
Description	The method make the multiple server API requests to delete all the existing question - page number links for the submission at the application server
Parameters	None

Frontend API: fetchMainSubmission	
Description	The method makes the server API request to fetch the main submission file uploaded by the user. On receiving success response from the server, method sets the pages array for the file view panel and fetch the pagination data using component API fetchPagination(if question set is selected for the submission)
Parameters	isQsetPresent (boolean): The flag representing whether the submission associated question set is selected/not

Frontend API: fetchMySubmissionGroup	
Description	The method make the server API request to fetch the submission group details in the received event. On receiving successful response from the server, method stores the details like group database ID, selected question set, access code submitted, and the list of group members
Parameters	None

Frontend API: fetchPagination	
Description	The method make server API request to fetch the pagination data of the submission, On receiving success response from the server, method sets the component properties and the drop-down data for the pagination panel view
Parameters	totalPages (number): Total number of pages in the main submission file

Frontend API: fetchSelectedQSet	
Description	The method make the server API request to fetch the question set associated to the submission from the application server. On receiving success response from the server, the method fetch the list of questions corresponds to the question set using component API getActualQuestions with the onload flag
Parameters	None

Frontend API: fetchSuppSubmission	
Description	The method makes the server API request to fetch the supplementary submission file uploaded by the user. On receiving success response from the server, method sets the component properties suppFile and isSuppSubmissionPresent
Parameters	None

Frontend API: getActualQuestions	
Description	The method sets the component property myQuestions for the pagination using the user-selected question set for the submission. The method is used in two circumstances, the data load at the component initialization and another is once user update the question set corresponding to the submission. On data load mode, the method calls the component API fetchMainSubmission to fetch the main submission file and the pagination data from the server. On second mode, the method initializes pagination for the updated question list in the view using the component API initPagination
Parameters	flag (string): The string flag representing the usage mode of API, for example data load mode(onload) or the question set update mode(init)

Frontend API: getMainSubmFile	
Description	The method downloads the user-submitted main submission file into the local system, it internally uses the component API downloadFile to download the file
Parameters	None

Frontend API: getSupplFile	
Description	The method downloads the user-submitted supplementary file into the local system, it internally uses the component API downloadSuppFile to download the file
Parameters	None

Frontend API: gotoMyEvents	
Description	The method redirects the user to the MyEvents Dashboard of the application
Parameters	None

Frontend API: getSha1Hash	
Description	The method computes the hash of the received file using the SHA1 API of CryptoJS library [6] and returns the computed hash
Parameters	• fileToHandle (File): The file to be processed for the hash computation • callbackFunc: The function callback to return the hash

Frontend API: initPagination	
Description	The method initialize the pagination data to zero for all the questions in the submission and sets the drop-down data for the pagination panel view
Parameters	None

Frontend API: isSubeventActiveAtMoment	
Description	The method checks whether the received subevent is active at the moment or not, then returns true/false accordingly
Parameters	subevent (MySubEvent): The request corresponding subevent object

Frontend API: isSuploadActive	
Description	The method checks whether there is the subevent of type SUPLOAD is active at the moment or not , returns true/false accordingly
Parameters	subevents (MySubEvent[]): The list of subevents in the event

Frontend API: onComponentInit	
Description	The component lifecycle hook, get called on every component initialization. The method fetches the submission associated event information from the local storage, then store it in the mySubmissionEvent property. The method makes the server API request to fetch the submission related information, On receiving successful response from the server, The method performs below actions 1. Store the submission group details like user selected question set, user-submitted access code, and list of group members 2. Store the necessary flags for submission flow like submission group is created/not, access code verification required/not, the question set selection allowed/not, etc 3. Store the the subevent configuration for the submission like submission group scheme, question set scheme, supplementary submission allowed or not, etc. 4. Fetches the user submitted pagination and main submission file from the server using component API getActualQuestions 5. If supplementary submission allowed, then fetches the user-submitted supplementary submission file from the server using component API fetchSuppSubmission
Parameters	None

Frontend API: onFinalSubmit	
Description	The method checks whether user uploaded main submission file or not. If the file is not uploaded then method shows the appropriate error message and returns. The method checks whether user provides the pagination for all the questions or not. If the pagination is missing for any question then method shows the appropriate error message and returns. Else it shows the success notification and redirects user to the MyEvents dashboard
Parameters	None

Frontend API: pageSelectionUpdate	
Description	The method make the server API request to set the question - page number link for the submission at the application server
Parameters	None

Frontend API: setPagesArray	
Description	The method sets the dropdown data as the sequence of integers from 1 to total number of pages for the pagination panel view
Parameters	length (number): Total number of pages in the main submission file

Frontend API: setQuestionPagination	
Description	The method make the server API request to set the question - page number link for the submission at the application server
Parameters	question (MyQuestion): The request corresponding question object

Frontend API: submitCreateJoinForm	
Description	The method checks the user-selected choice on the submission group form whether it is to create or join the submission group, based on the choice method makes the server API request to CREATE/JOIN the submission group. On receiving the success response from the server, method fetches the submission group details and group member from the server using the component API fetchMySubmissionGroup
Parameters	None

Frontend API: submitSubmissionFileForm	
Description	The method computes the file sha1 hash using component API hashFile. Then method calls the service API uploadMainSubmission to upload the file at the application server. On receiving success response from the server, 1. Update the main submission file data used by file view panel 2. Delete the pagination corresponding to old submission file(if uploaded any) 3. Initialize the pagination panel using component API initPagination 4. Display the success notification and hide the main submission file upload form using component API toggleFileForm
Parameters	None

Frontend API: submitPreSubmissionForm	
Description	The method makes the server API request to update the pre-submission details like submission corresponding question set and access code. On receving success response from the server, the method reset the question pagination panel data using the questions of the updated question set
Parameters	None

Frontend API: submitQSetForm	
Description	The method make server API request to update the user-selected question set for the submission. On receiving success response from the server, the method update the pagination panel view with the updated question set data by calling the component API getActualQuestions(in init flag mode)
Parameters	None

Frontend API: submitSuppFileForm	
Description	The method computes the file sha1 hash using component API hashFile. Then method calls the service API uploadSuppSubmission to upload the file at the application server. On receiving success response from the server, the method set the isSuppSubmissionPresent as true, stores the supplementary file, display the success notification, and hide the supplementary submission file upload form using component API toggleSuppFileForm
Parameters	None

Frontend API: toggleSubmissionFileForm	
Description	The method toggle the view mode of the main submission file form using the component property isOpenFileForm
Parameters	None

Frontend API: toggleQSetForm	
Description	The method toggle the view mode of the question set form using the component property isOpenQSetForm
Parameters	None

Frontend API: toggleSuppFileForm	
Description	The method toggle the view mode of the supplementary submission file form using the component property isOpenSuppFileForm
Parameters	None

Frontend API: verifyNACForm	
Description	The method makes server API request to verify the user-submitted access code. On receiving successful response from the server, the method sets the access code verification flag and move the user-submission to the submission stage
Parameters	None

4.5.3.3 Grading Manager Component

Capabilities:

The grading manager component provides the below capabilities to the user,

- View the Grading duties allocated to the user in the user-selected GUPLOAD subevent on the MyEvents dashboard
- View the grading statistics data like Total number of graded submission copies and the total rewarded grades of each submission copy
- Navigate to the Main Grading page for any grading duty to grade the submission copy

The Component View design:

The view is the nested two-level table representation of the user grading duty data in the GUPLOAD subevent. In the upper-level table, each row corresponds to the question for which the grading duties are allocated. Each row contains the question title, total marks, the number of graded copies, and an action button to perform grading for ungraded copies in this question. The table view is created using the list component provided by the nebular theme library.

While the child table represents the list of allocated copies for which corresponding question to be graded. Each row contains the submission group ID/group member names, the grading status, and the graded marks. Each inner table is wrapped into a nebular list item in order to provide the nested list view

The component functionality:

On component Initialization, the component fetches the grading duties data and format the data into the grading duties queues and store with the corresponding question data. While formatting the server response, the component computes the total number of graded copies for each question and sets the grading status for each grading duty. Once the user clicks the perform grading button then the component iterates through the corresponding grading queue, finds the ungraded copy and redirects the user to the main grading page for the ungraded copy. Before navigation, the component stores the required data like the queue index of ungraded copy and corresponding grading duty queue into local storage. The grading duty queues are used on the main grading page to provide the navigation capability to the prev/next grading duties in the same question. See the below mentioned API's for detailed information

Frontend API: fetchGradingDuties	
Description	The method makes an server API request to fetch the user-allocated grading duties in the corresponding GUPLOAD subevent & event. On receiving the success response from the server, the method process the response data and store the data into the component property myGradingDuties. While processing the method retrieves the list of questions for which grading duties are allocated, the corresponding list of grading duties, the group member names for each grading duty, the total number of graded copies, etc.
Parameters	None

Frontend API: findNextUnGraded	
Description	The method iterates through the list of grading duties allocated for the received question and returns the index of first grading duty for which grading is not completed. If no such grading duty found, then method returns -1
Parameters	qset_idx(number): The request corresponding list index of question

Frontend API: goToMainGrading	
Description	The method redirects the user to the Main Grading page to perform the grading
Parameters	None

Frontend API: gotoMyEvents	
Description	The method redirects the user to the MyEvents dashboard of the application
Parameters	None

Frontend API: OnComponentInit	
Description	The component life-cycle hook, get called on every component initialization. The method calls the fetchGradingDuties API of component to fetch the grading duties allocated to the user in the user-selected GUPLOAD subevent on the myevents dashboard
Parameters	None

Frontend API: performGrading	
Description	The method find the first ungraded copy in the grading duty list of the question using the component API findNextUnGraded. If no such copy found, then method shows notification message for grading completion and returns. Else method stores the ungraded grading duty, corresponding grading duty list index, and the grading duty list of question in the local storage. Then redirects the user to the main grading page to perform grading
Parameters	qset_idx(number): The request corresponding ID of question

Frontend API: performSelectedGrading	
Description	The method find the grading duty corresponding to the received grading duty list index of the received question. Then method stores the grading duty, corresponding grading duty list index, and the grading duty list of question in the local storage. After storing required data, The method redirects the user to the main grading page to perform/view grading
Parameters	• qset_idx(number): The request corresponding ID of question • gd_idx(number): The request corresponding list index of grading duty

4.5.3.4 Main Grade Component

Capabilities

The main grading component provides the below capabilities to the user,

- Grade the allocated submission copy by using the rubric-based grading system
- View and download the submission copy related files .i.e. the main submission file and the supplementary submission file
- Perform the grading efficiently using two ways, the keyboard shortcuts or using the mouse controls on SPHINX User Interface
- The grader can navigate to the next/prev ungraded submission and the next/prev submission in the allocated grading duties of the same question using the keyboard shortcuts or the navigation control buttons
- View the progress of the total number of graded submission copies

The Component View design

The view contains the following view parts,

- The submission copy view: The view displays the submission copy to be graded in the nice image carousel, the start page of carousel set as the page containing answer for the corresponding question using user-provided pagination
- The grading panel: The grading panel contains the Instructor message for graders, total graded marks, rubrics section, and the adjustment section. In the rubric section, all the rubrics corresponding to question are listed with the corresponding check-boxes. The adjustment section contains the input areas for the point adjustment and the specific comments by the grader
- The submission files panel: The panel contains the links to download the main submission file and the supplementary submission file
- The bottom panel: The bottom panel contains the grading progress bar, navigation link to the grading manager, and the queue navigation section. The queue navigation section contains the buttons which load the next/prev ungraded submission copy, the next/prev submission copy in the grading duty queue corresponding to the same question

The component functionality

On component Initialization, the component takes the following actions,

1. Load and store the data from local storage corresponding to the user-selected grading duty on the grading manager. The data contains the grading duty, the corresponding queue index, and the grading duty queue
2. Fetch the main submission file and the supplementary submission file corresponding to the grading duty from the application server

3. Fetch and store the existing grading details like the list of applied rubrics, the total graded marks from the application server
4. Create the keyboard shortcuts to apply the rubric for grading as per the number of rubrics available to perform grading
5. Load the submission file in the submission copy view panel

The component provides the keyboard shortcuts to the grader using the HotkeysService of the angular2-hotkeys library. The component provides keyboard shortcuts as below,

- For each rubric in the rubric list, 1-9 keys corresponding as per the list position of the rubric
- $[A, S, D, F]$ keys corresponding to grade & prev, prev, next, and grade & next buttons respectively

Whenever the grader check/uncheck the checkbox for any rubric, the component creates/delete the rubric-grading duty link at the application server to apply the respective changes for grading. The navigation buttons and corresponding component action as listed below,

- Grade & Prev: The component submits the grades onto the server and loads the view with next ungraded submission in the opposite direction from the queue
- Prev: The component save the grades onto the server and loads the view with the previous submission from the grading duty queue
- Next: The component save the grades onto the server and loads the view with the next submission from the grading duty queue
- Grade & Next: The component submits the grades onto the server and loads the view with next ungraded submission from the queue

The component provides API's to provide the above-mentioned functionalities such as apply/remove the rubric for the grading, download submission copies, navigation to other submission copies, etc. See the below mentioned API's for more detailed information

Frontend API: deleteKeyboardShortcuts	
Description	The method disable and deletes all the keyboard shortcuts available for the grading
Parameters	None

Frontend API: delRubricGDLink	
Description	The method makes the server API request to delete the rubric-grading duty link on the application server. On receiving success response from the server, the method updates the aggregated graded marks accordingly
Parameters	ribric_idx (number): The request corresponding rubric ID

Frontend API: fetchGradingDutyDetails	
Description	The method makes the server API request to fetch the grading duty details using service method getGradingDutyDetails. On receiving the success response from the server, the method stores the data like the main submission file, graded marks, grader comment, start page index of the answer, the mark adjustment, and the applied rubrics corresponding to the current grading duty. The method sets the image array for carousel using submission file and sets the rubric checkbox state for the list of applied rubrics using the component API setRubrics
Parameters	None

Frontend API: fetchNextDuty	
Description	The method returns the index position of the next grading duty from the queue w.r.t. the current grading duty
Parameters	None

Frontend API: fetchNextUnGradedDuty	
Description	The method iterate through the grading duty queue in circular manner with the start position as current grading duty index and returns the index position of the next first ungraded duty from the queue. If no such grading duty found, method simply returns -1
Parameters	None

Frontend API: fetchPrevDuty	
Description	The method returns the index position of the previous grading duty from the queue w.r.t. the current grading duty
Parameters	None

Frontend API: fetchPrevUnGradedDuty	
Description	The method iterate through the grading duty queue in circular backward manner with the start position as current grading duty index and returns the index position of the first ungraded duty from the queue. If no such grading duty found, method simply returns -1
Parameters	None

Frontend API: getMainSubmFile	
Description	The method makes the server API request to fetch the main submission file associated to the grading duty. On receiving success response from the server, it downloads the file into the local system using the service API downloadFile
Parameters	None

Frontend API: getSuppSubmFile	
Description	The method makes the server API request to fetch the supplementary submission file associated to the grading duty. On receiving success response from the server, it downloads the file into the local system using the service API downloadFile
Parameters	None

Frontend API: gotoGradingManager	
Description	The method redirects the user to the Grading manager of the course
Parameters	None

Frontend API: gotoMyEvents	
Description	The method redirects the user to the MyEvents Dashboard of the course
Parameters	None

Frontend API: gradeSubmission	
Description	The method make the server API request to upload the grading on the application server with is_completed = true. On receiving the success response from the server, the method performs below mentioned actions, 1. Update the grading completion count according to the old status of grading duty 2. Shows success notification to the user 3. Fetch next ungraded grading duty from the CurrGDs queue, the looking direction is decided using the received move. For example +1 for forward and -1 for backward direction. Uses the component API's fetchNextUnGradedDuty and fetchPrevUnGradedDuty 4. Make call to the component API takeNextMove which fetch the grading duty details, set rubric data, the submission view pane data for the next ungraded grading duty
Parameters	move (number): The direction move, For example +1 for forward and -1 for backward direction

Frontend API: initializeKeyBoardShortcuts	
Description	The method create and initializes the keyboard shortcuts using the hotkeysService, the shortcuts are the number keys 1-9 ,each key assigned to the rubric of the list of question-rubrics as per the list index and $[A, S, D, F]$ keys corresponding to grade & prev, prev, next, and grade & next buttons respectively
Parameters	None

Frontend API: performPointsAdjustment	
Description	The method takes the value of the point adjustment input box from the adjustment section, then add the value to the total graded marks and update the aggregated marks of the current grading duty
Parameters	None

Frontend API: onComponentDestroy	
Description	The component life-cycle hook gets called before destroying the component. The method deletes all the keyboard shortcuts created to perform grading using the component API deleteKeyboardShortcuts
Parameters	None

Frontend API: onComponentInit	
Description	The component lifecycle hook, get called on every component initialization. The method fetches the current grading duty associated information from the local storage, then set the component properties currGDs, currGDIdx, currGD, eventId, subEventId, and isGUPLOAD flag. The method fetches the current grading duty details like the applied rubrics , the main submission copy from the server using the component API fetchGradingDutyDetails . Method sets the grading progress bar value w.r.t. the grading queue using the component API setProgressBarValue
Parameters	None

Frontend API: onGradeAndNext	
Description	The method submits the grading for current grading duty on the server with is_completed flag = true and load the next ungraded grading duty from the CurrGDs queue. The method uses the component API gradeSubmission and performs below mentioned actions, 1. Call the server API to update grading for the current grading duty 2. Update the completed grading count of current grading duty question 3. Fetch next ungraded grading duty from the CurrGDs queue 4. Make call to the fetchGradingdutydetails method which calls -> setImages, setRubrics -> initialize keyboard shortcuts
Parameters	None

Frontend API: onGradeAndPrev	
Description	The method submits the grading for current grading duty on the server with is_completed flag = true and load the next ungraded grading duty from the CurrGDs queue in the backward direction. The method uses the component API gradeSubmission and performs below mentioned actions, 1. Call the server API to update grading for the current grading duty 2. Update the completed grading count of current grading duty question 3. Fetch next ungraded grading duty from the CurrGDs queue in backward direction 4. Make call to the fetchGradingdutydetails method which calls -> setImages, setRubrics -> initialize keyboard shortcuts
Parameters	None

Frontend API: onNext	
Description	The method submits the grading for current grading duty on the server with is_completed flag = false and load the next grading duty from the CurrGDs queue. The method internally uses the component API submitSubmission
Parameters	None

Frontend API: onPrev	
Description	The method submits the grading for current grading duty on the server with is_completed flag = false and load the next grading duty from the CurrGDs queue in backward direction. The method internally uses the component API submitSubmission
Parameters	None

Frontend API: rubricSelected	
Description	The method check the state for the requested rubric and make the respective server API request to apply/delete the rubric-grading duty link accordingly on the application server. On receiving success response from the server, the method updates the aggregated graded marks accordingly
Parameters	rubirc_idx (number): The request corresponding rubric ID

Frontend API: setProgressBarValue	
Description	The method calculate the percentage of grading duty completion w.r.t. the total number of grading duties in the queue and set the percentage value as the progress bar value
Parameters	None

Frontend API: setRubricGDLink	
Description	The method makes the server API request to apply the rubric-grading duty link on the application server. On receiving success response from the server, the method updates the aggregated graded marks accordingly
Parameters	None

Frontend API: setRubrics	
Description	The method sets the rubric list for the grading panel using the received list of applied rubrics and the list of all question rubrics. It initializes the keyboard shortcuts corresponding to the rubrics using the component API initializeKeyBoardShortcuts
Parameters	• question_rubrics (any[]): The list of rubrics corresponding to the question • applied_rubrics (any[]): The list of applied rubrics

Frontend API: submitSubmission	
Description	The method make the server API request to upload the grading on the application server with is_completed = false. On receiving the success response from the server, the method performs below mentioned actions, 1. Shows success notification to the user 2. Fetches the next grading duty from the CurrGDs queue, the looking direction is decided using the received move. For example +1 for forward and -1 for backward direction. Uses the component API's fetchNextDuty and fetchPrevDuty 3. Make call to the component API takeNextMove which fetch the grading duty details , set rubric data, the submission view panel data for the next grading duty
Parameters	move (number): The direction move, For example +1 for forward and -1 for backward direction

Frontend API: takeNextMove	
Description	The method validate the received list index of grading duty, if -1 then it shows the grading completion notification for current question and redirects the user to the grading manager page. If received list index is valid, then method takes below actions 1. Updates the component properties currGDIdx as the received list index & currGD with the respective grading duty object 2. Fetches the grading duty details and set the rubric data, the submission view panel data for the next grading duty
Parameters	next_id (number): The list index corresponding to the next grading duty

4.5.3.5 Grade Sheet Manager Component

Capabilities

The Grade Sheet manager component provides the below capabilities to the user,

- View the graded marks for each answered question in the submission and total graded marks of the submission
- Navigate to the Main GradeView page for any question on the grade sheet dashboard to view the submission copy with the grading details like the applied rubrics, grader's comment, etc.

The Component View design

The Grade Sheet view contains the list of questions answered by the student during the submission in the event. Each list item contains the question title, the graded marks, and the total marks of the corresponding question. The view displays the total obtained marks as the last row item of the list. Each list item displays the question title as the hyperlink, which navigates the user to the Main Grade view page for the corresponding question. The view uses the Nebular List component of the Nebular theme library [10] to show the data in the list format

The component functionality

On component Initialization, the component fetches the list of graded questions and grading details corresponding to the submission made by the user in the event. The component fetches the below-mentioned grading details of the graded submission,

- The list of grading duties in case the question is graded by multiple graders. Each grading duty contains the applied rubrics, point adjustment, the grader's comment, and the total graded marks
- The main submission file corresponding to the submission in the event
- The communication messages corresponding to the regrade requests
- The regrading duty contains the applied rubrics, point adjustment, the re-grader comment, and the total re-graded marks

The component stores the above details and the list of questions using the model class MyMarksQuestion, and the MyMarksGradingDuty class. While formatting the above details from the server response, The component computes the total graded marks and set the corresponding component properties. Once the user clicks on any question title the component redirects the user to the main grade view page for the respective question. Before navigation, the component stores the required data like the list index of the respective graded question, the list of questions, and corresponding grading details in the local storage

Frontend API: calculateAndSetMarks	
Description	The method calculates the total aggregated graded marks of the question using the received grading duties(multiple entries present in case question is graded by multiple graders). The total marks are aggregated using the received grade aggregation method, in case question is graded by multiple graders
Parameters	1. question_gds (any[]): The list of grading duties corresponding to the question 2. aggregation_method (string): The grade aggregation method. For example MIN, MAX, AVG

Frontend API: createAndSetGD	
Description	The method process and format the received grading duty data and store it into the MyMarksGradingDuty model object. The method sets the rubrics applied by the grader using the received parameters q_rubric_data and the gdhr_data. It return the structure containing the MyMarksGradingDuty model object, and the database ID of associate Response and Subevent table entry
Parameters	• q_rubric_data (any[]): The list of rubrics corresponding to the question • gd_data (any): The grading duty data • gdhr_data (any[]): The grading duty has rubrics data, The list containing multiple entries the grading duty id and the applied rubric id • curr_GD (number): The list index of grading duty

Frontend API: createAndSetRGDMessages	
Description	The method receive the list of messages of regrading requests and format it into the list of the RegradingMessage model objects. Where each RegradingMessage object contains the message sequence index, message text, and the sender of the message
Parameters	regrading_messages (any[]): The list of regrading request messages of the question

Frontend API: goToViewMainGrading	
Description	The method redirects the user to the Main grade view page to view the graded copy
Parameters	None

Frontend API: onComponentInit	
Description	The component lifecycle hook, get called on every component initialization. The method calls the getMyGrades API of the service to fetch the gradesheet data from the server. On receiving the success response from the server, the method process the response data, sets the list of answered questions in the submission using the MyMarksQuestion model objects, and set the below details for each question in the list, 1. Extract the grading duty list and store using the MyMarksGradingDuty model and the createAndSetGD API of the component 2. Extract the list of regrade request messages and store using the createAndSetRGDMessages API of the component 3. Compute and set the total graded marks using the calculateAndSetMarks API of the component 4. Store the database ID of corresponding entries of the response and subevent table , used to raise the new regrade request at application server The method store the above formatted data in the my_marks property of the component
Parameters	None

Frontend API: viewMainGrading	
Description	The method stores the gradesheet data and the list index of user-selected question in the local storage and redirects the user to the main grade view page
Parameters	question_idx (number): The list index of the user-selected question

4.5.3.6 Main GradeView Component

Capabilities

The main Grade View component provides the below capabilities to the user,

- View the main submission file corresponding to the submission in the event
- View the total graded marks and the grade aggregation method in case the copy is graded by multiple graders
- View the list of grading duties(in case the answer copy is graded by multiple graders) and regrading duty corresponding to the question, Each grading duty contains grading details like the list of applied rubrics, the point adjustment, and the grader's comment submitted by the respective grader
- View the communication messages regarding existing re-grading requests and Submit the new regrade request

The Component View design

The view contains the following view parts,

- The submission copy view: The view displays the submission copy to be graded in the nice image carousel, the start page of carousel set as the page containing answer for the corresponding question using user-provided pagination
- The Question-Grade Sheet panel: The panel consists of two tabs representing the grades panel and the regrading request chatbox. The grades panel provides the list of grading duties(in case the answer copy is graded by multiple graders) and regrading duty corresponding to the question, Each grading duty contains grading details like the list of applied rubrics, the point adjustment, and the grader's comment submitted by the respective grader. The chat-box tab list out all the communication messages between grader and student. The tab provides text-area to submit the new regrade request for the student
- The bottom panel: The bottom panel contains the control buttons to navigate to the other graded questions in the grade sheet. The panel provides the navigation link to the Grade Sheet dashboard page

The component functionality

On component Initialization, the component takes the following actions,

1. Load and store the list index of the user-selected question and the list of the questions of grade-sheet from the local storage
2. Using the user-selected question and corresponding grading details, the component sets the properties for the Question-Grade Sheet panel like the list of grading duties, the chat-box messages of regrading requests, etc.
3. Load the submission file in the submission copy view panel

The component provides the navigation controls to the user such that the user can move to the other questions to view the respective grading details. The component uses the list of MyMarksQuestion model objects and the list index of the current question to provide the navigation capability. Whenever the user clicks the right/left arrow on the view, component moves the current question index to forward/backward in the question queue and loads the view with the respective MyMarksQuestion object data at the updated index.

The component provides the API's to raise the new regrade request using the chat-box provided in the view, so once the user clicks the submit button by entering the regrade request message, the component calls the necessary server API to raise the new regrade request at the application server. See the below mentioned API's for more detailed information

Frontend API: addRegradeRequest	
Description	The method checks is there currently active RGREQ subevent is going on or not. If not going, then method shows the error notification and returns. The method makes the server API request to submit the regrade request on the application server. On receiving success response from the server, the method add the regrade request message to the chat-box using the sendMessage API of the component and redirects the user to the grade sheet manager page
Parameters	None

Frontend API: gotoGradeViewManager	
Description	The method redirects the user to the GradeSheet manager of the event
Parameters	None

Frontend API: loadCurrQ	
Description	The method sets the The Question-GradeSheet panel data using the value of curr_q_idx and my_marks property of the component. Then method load the regrade request messages of the chatbox using the setChatboxMessages API of the component
Parameters	None

Frontend API: onComponentInit	
Description	The component lifecycle hook, get called on every component initialization. The method fetches the user-selected graded question associated information from the local storage, and sets the component properties ,which used to render the data in the view. The method set the submission file carousel images using the setSlides API. The method sets the The Question-GradeSheet panel data corresponding to the user-selected question using the loadCurrQ API.
Parameters	None

Frontend API: onNext	
Description	The method moves the current question pointer to forward direction and load the grading details of the next question in the queue using the loadCurrQ API of the component
Parameters	None

Frontend API: onPrev	
Description	The method moves the current question pointer to backward direction and load the grading details of the previous question in the queue using the loadCurrQ API of the component
Parameters	None

Frontend API: sendMessage	
Description	The method receive the message structure and set the message in the required structure format by Nebular chat-box component. After formatting the message, method add it in the message queue represented by the component property messages
Parameters	• message (string): The message text • user_name (string): The sender of the message, used to display in the chat-box • reply (Boolean): The Boolean flag representing whether the message is sent by the student or not

Frontend API: setChatboxMessages	
Description	The method sets the list of communication messages of regrade requests using the received server response data. The method sets the each message in the required format by Nebular Chat-box component using the component API sendMessage
Parameters	regrading_msg_data (any[]): The server response containing the list of message structures, Each structure contains the message text and sender of the message i.e. grader or student

4.5.3.7 Re-Grading Manager Component

Capabilities

The re-grading manager component provides the below capabilities to the user,

- View the Re-Grading requests allocated to the user in the user-selected RGUPLOAD subevent on the MyEvents dashboard
- View the regrading statistics data like Total number of graded re-grade requests and the total rewarded grades of each submission copy
- Navigate to the Main ReGrading page for any grading request to re-grade the submission

copy

The Component View design

he view list out all the re-grade requests allocated to the grader in the corresponding RGUPLOAD subevent. each list item contains the regrade request details like the submission group ID/group member names, the review status, the graded marks, the corresponding question, and the question set. The view uses the Nebular List component of the Nebular theme library to show the data in the list format

The component functionality

On component Initialization, the component fetches the list of re-grading duties data corresponding to the user-selected RGUPLOAD subevent. It processes the server response data and stored the formatted list of allocated re-grading requests as an array of MyReGradingDuty model objects. While formatting the server response, the component computes the total number of graded regrade requests sets the grading status for each re-grading duty. Once the user clicks on any regrade request the component redirects the user to the main re-grading page for the respective regrade request. Before navigation, the component stores the required data like the list index of the respective regrade request and corresponding grading duty queue into local storage

Frontend API: fetchReGradingDuties	
Description	The method makes an server API request to fetch the user-allocated re-grading duties in the corresponding RGUPLOAD subevent. On receiving the success response from the server, the method process the response data and store the data into the component property myGradingDuties. Each regrading duty represents the regrading request made by student. While processing the method computes the total number of closed regrading requests and update the component properties completed_cnt & tot_cnt accordingly
Parameters	None

Frontend API: goToMainRegrading	
Description	The method redirects the user to the Main Re-grading page to perform grading
Parameters	None

Frontend API: onComponentInit	
Description	The component lifecycle hook, get called on every component initialization. The method calls the fetchReGradingDuties API of component to fetch the re-grading duties allocated to the user in the user-selected RGUPLOAD subevent on the myevents dashboard
Parameters	None

Frontend API: performSelectedReGrading	
Description	The method find the re-grading duty corresponding to the received re-grading duty list index. Then method stores the re-grading duty, corresponding list index, and the re-grading duty list in the local storage. After storing required data, The method redirects the user to the main re-grading page to review the regrade request
Parameters	rgd_idx (number): The request corresponding list index of the re-grading duty

4.5.3.8 Main Re-Grade Component

Capabilities

The main regrading component provides the below capabilities to the user,

- Grade the allocated regrade request by using the rubric-based grading system
- View and download the submission copy related files .i.e. the main submission file and the supplementary submission file
- Perform the grading efficiently using two ways, the keyboard shortcuts or using the mouse controls on UI
- View the old grading duties(applied rubrics and graded marks) corresponding to the regrade request
- View the communication messages regarding re-grading requests and Submit the grader's review comment

The Component View design

The view contains the following view parts,

- The submission copy view: The view displays the submission copy to be graded in the nice image carousel, the start page of carousel set as the page containing answer for the corresponding question using user-provided pagination
- The grading panel: The grading panel consists of three tabs representing the new re-grading panel, Old grade panel, and the regrading request chat-box. The new grading panel provides the list of rubrics having a checkbox associated with each rubric and adjustment section to perform the re-grading. The old grade tab contains the list of old grading duties corresponding to the regrade request, where each list item contains the list of applied rubrics and the corresponding adjustment section. The chat-box tab list out all the communication messages between grader and student. The tab provides text-area to submit the grader review message for the regrade request
- The submission files panel: The panel contains the links to download the main submission file and the supplementary submission file
- The bottom panel: The bottom panel contains the control buttons to submit the grades for the regrade request or cancel the regrading and returns to the regrading manager

The component functionality

On component Initialization, the component takes the following actions,

1. Load and store the re-grading duty from local storage corresponding to the user-selected re-grading request on the grading manager
2. Fetch the main submission file and the supplementary submission file corresponding to the regrade request from the application server
3. Fetch and store the existing grading details like the list of applied rubrics, the total graded marks, list of old grading duties from the application server
4. Create the keyboard shortcuts to apply the rubric for grading as per the number of rubrics available to perform grading
5. Load the submission files in the submission copy view panel

The component provides the keyboard shortcuts to the grader using the HotkeysService of the angular2-hotkeys library. The component provides keyboard shortcuts for each rubric in the rubric list, 1-9 numeric keys corresponding as per the list position of the rubric. The component API's provides similar functionalities to the Main-Grade component API's for rubric-based grading. In addition, the component provides the API's to

- Fetch the old grading duties and the list of regrading request communication messages from the application server
- Submit the review comment for the corresponding regrade request to the server

See the below mentioned API's for more detailed information

Frontend API: fetchRegradingDuty	
Description	The method make the server API request to fetch the regrading duty details using service method getGradingDutyDetails. On receiving success response from the server, the method takes below actions to extract the details and stored in the component properties, 1. Call the component API setPrevGradingDuties to store the list of old grading duties 2. Call the component API setChatboxMessages to store the communication messages corresponding to regrade requests 3. Call the component API setCurrGDDetails to store the regrading duty details like the list of applied rubrics, Point adjustment data, graded marks, etc. 4. Call the component API setSlides to store the pages of submission file as array of images
Parameters	None

Frontend API: gotoReGradingManager	
Description	The method deletes all the keyboard shortcuts created to perform re-grading using the component API deleteKeyboardShortcuts and redirects the user to the ReGrading manager of the course
Parameters	None

Frontend API: gradeSubmission	
Description	The method make the server API request to upload the grading on the application server with is_completed = true. On receiving the success response from the server, the method shows success notification to the user and redirects the user to the Regrading Manager
Parameters	None

Frontend API: onComponentInit	
Description	The component lifecycle hook, get called on every component initialization. The method fetches the current re-grading duty associated information from the local storage, and sets the component properties myReGDs, currReGDIdx, currReGD, eventId, subEventId, and isRGUPLOAD flag. The method fetches the current re-grading duty details like the applied rubrics , the main submission copy, old grading duties and the regrade request messages from the server using the component API fetchRegradingDuty
Parameters	None

Frontend API: onCancel	
Description	The method save the grading for current grading duty on the server with is_completed flag = false and then redirects the user to the re-grading manager. The method uses the submitSubmission API of the component to save the grades
Parameters	None

Frontend API: onChatboxSubmit	
Description	The method make the server API request to save the grader's review message on the application server. On receiving the success response from the server, the method add the grader's review message to the chat-box message array using the sendMessage API of the component
Parameters	None

Frontend API: onSubmit	
Description	The method submits the grading for current grading duty on the server with is_completed flag = true and then redirects the user to the re-grading manager. The method uses the gradeSubmission API of the component
Parameters	None

Frontend API: sendMessage	
Description	The method receive the message structure and set the message in the required structure format by Nebular Chat-box component. After formatting the message, method add it in the message queue represented by the component property messages
Parameters	• message (string): The message text • user_name (string): The sender of the message, used to display in the chat-box • message (Boolean): The Boolean flag representing whether the message is sent by the grader or not

Frontend API: setChatboxMessages	
Description	The method sets the list of communication messages of regrade requests using the received server response data. The method sets the each message in the required format by Nebular Chat-box component using the component API sendMessage
Parameters	regrading_msg_data (any[]): The server response containing the list of message structures, Each structure contains the message text and sender of the message i.e. grader or student

Frontend API: setPrevGradingDuties	
Description	The method sets the list of old grading duties corresponding to the regrading request. The method process the received grading duty data from the server response, and sets below details for each list item, 1. Grading duty details like graded marks, point adjustment, grader comment 2. The list of applied rubric
Parameters	• prev_gd_data (any[]): The list of old grading duties in the server response format • question_rubrics (any[]):The list of rubrics corresponding to the questions

CHAPTER 5

Backend Augmentations

Contents

Abstract *This chapter contains detailed explanations of the augmentations made to the existing backend of the* SPHINX *system. The section on application middleware describes three application middleware designed to streamline the backend processing and avoid code redundancy. The process description section lists out a set of newly exposed APIs by the backend with the corresponding detailed descriptions. These APIs are critical to the proper functioning of some novel functionalities offered by* SPHINX*, above and beyond those proposed in [16].*

The backend of the SPHINX system provides various functionalities by exposing a set of RESTful APIs. As showing in the figure below, application middlewares are architecture hooks that enable execution of common application logic before/after processing API requests. In order to improve the system performance we propose three new custom middlewares to the backend architecture.

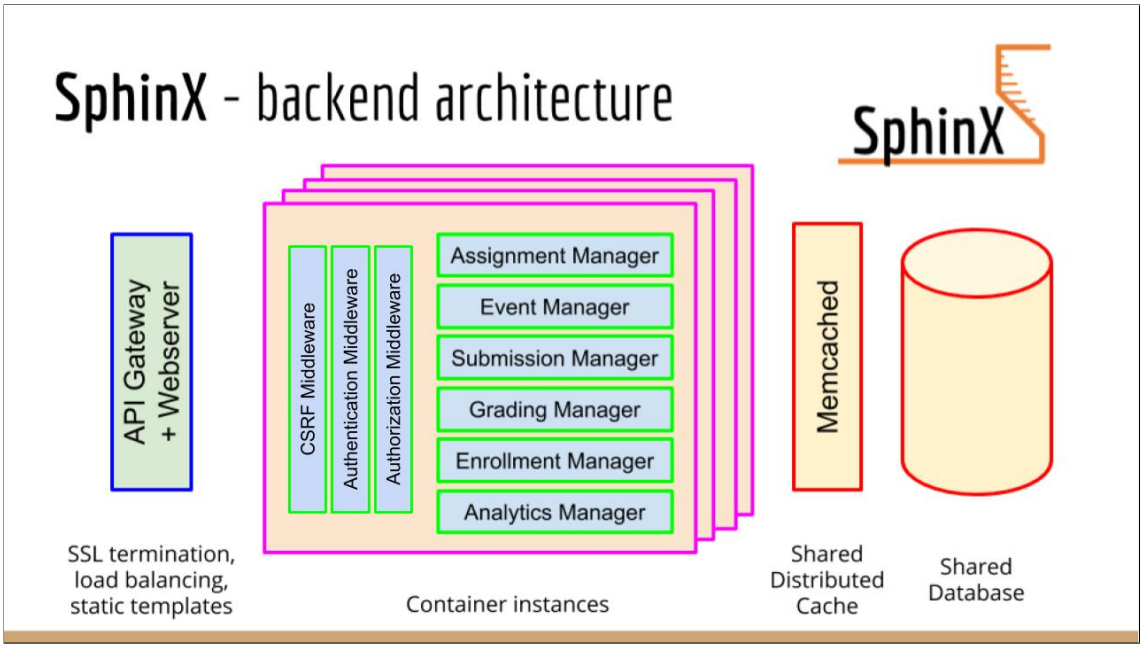


Figure 5.1: An Overview of the SPHINX Architecture. Figure taken from [16].

5.1 Application Middleware

Each received API request at the server is intially need to progress through the middleware pipeline. The custom middlewares arranged in the pipeline as the CSRF Middleware -> Authentication Middleware -> Authorization Middleware.

5.1.1 CSRF Middleware

On recieving the request, the middleware first check the type of received API request. If the type is **POST/PUT/DELETE**, then the middleware fetch the true value of CSRF token from the user session information in the **Memcached**. After comparing the received CSRF token value with the expected true value of the token, The middleware pass the request to the next middleware in pipeline if it matches successfully, else returns the **NOCSRF** error. The Middlware also refreshes the token value and updated in the **Memchached** after the fixed length expiration period.

5.1.2 Authentication Middleware

If the received request is correspond to any other manager than authentication manager, the middleware checks whether the request corresponding user is logged in or not using the session information from the **memchached**. If User is in logged-in state, then the request is progressed to next middleware, else send-back the **NOLOGIN** error and rejects the request.

5.1.3 Authorization Middleware

In order to provide the role separation in the SPHINX, we check the user authority for every incoming request except the authentication manager requests. Using the user-allowed action list from the **Memcached** and the URL of received API request, the middleware checks whether the user is allowed/ not to perform the requested operation at the backend. If the user have the respective permission then middleware pass the the request for further processing, else returns the **NOACCESS** error.

5.2 Process Description

We have added the set of new processes to the core back-end of the SPHINX system. The URL and detail description of each new exposed API is given below,

Backend API: Fetch Solution Files	
URL	`/course/$courseid/event/$eventid/mySolutions`
HTTP Method	**GET**
Pre-Validations	If there does not exist even one **AVIEW** 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 1. Identify all rows in **users_have_subevents** that are of type **AVIEW** 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. 2. If there are subevents in the GEN parameter of the **AVIEW** subevent 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** (note that only submission_groups related to this SUPLOAD subevent are to be considered), then find out the question set associated with that submission group from the table **submission_groups** and add that question_set to the pile if it is not already in the pile. 3. If there are subevents in the GEN parameter of the **AVIEW** subevent of type **GUPLOAD** or **RGUPLOAD**, find all grading_duty rows linked to that subevent which are assigned to this user, then for all response_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 solution_file of all question_sets in the pile**

Backend API: Post Question Pagination	
URL	`/course/$courseid/event/$eventid/myResponses/$question_id/$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 not 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 submission group, then throw a **NOVAL** error and finish (only actual questions need to be linked to pages). If the $pageno parameter is less than 1 or else if the page number is more than the number of pages in the main file uploaded by this submission group, then throw a **NOVAL** error and finish. Allow negative page no for option "Do not answer question", also do not consider responses with negative page for grader assignment
Action	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 **responses** row with this new pageno. Otherwise, create a new **responses** row for this submission group linking this question to this pageno. Thus, note that this POST URL serves as a PUT URL too
Return Data	**The page-question link that was created/updated so that the frontend may display the same**

Backend API: Delete Question Pagination	
URL	`/course/$courseid/event/$eventid/myResponses/$question_id/$pageno/`
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
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 row and all grading_duty_has_rubrics that correspond to those grading_duty_rows. Note that these cascading deletes should happen automatically due to foreign key correspondences
Return Data	**None**

Backend API: Get Grading Duty Main File	
URL	`/course/$courseid/event/$eventid/myGrading/$subeventid/gradingDuty/$gradingdutyid/main`
HTTP Method	**GET**

Pre-Validations	Verify that this subevent is indeed one of type **GUPLOAD** or **RGUPLOAD** or else a **GVIEW** or **RGVIEW** subevent linked to this subevent is going on, else throw a **NOACCESS** error and finish. Also ensure that this grading duty row is a valid one and is indeed linked to this very 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 subeventid or else a **GVIEW** or **RGVIEW**-type subevent otherwise throw a **NOACCESS** error and finish. Also ensure that this subevent is still going on (possibly in the late period) otherwise throw a **NOEVENTsss** error and finish
Action	None
Return Data	**Return the main file (not just images but the contents of the file itself) submitted by the submission group linked to this grading duty. Sometimes student answers can span multiple pages and so it is good to show the grader the entire file if they wish to access it**

Backend API: Get Grading Duty Supplementary File	
URL	`/course/$courseid/event/$eventid/myGrading/$subeventid/gradingDuty/$gradingdutyid/supplementary`
HTTP Method	**GET**
Pre-Validations	Verify that this subevent is indeed one of type **GUPLOAD** or **RGUPLOAD** or else a **GVIEW** or **RGVIEW** subevent linked to this subevent is going on, else throw a **NOACCESS** error and finish. Also ensure that this grading duty row is a valid one and is indeed linked to this very 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 subeventid or else a **GVIEW** or **RGVIEW**-type subevent 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 the supplementary file (not just images but the contents of the file itself) submitted by the submission group linked to this grading duty. Since supplementary file contents are not linked to any particular question, we have to show the grader the entire supplementary file**

Backend API: Get User Mark-Sheet	
URL	`/course/$courseid/event/$eventid/myMarks`
HTTP Method	**POST**
Pre-Validations	If there does not exist even one **MVIEW** or **RMVIEW** 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	If in any of the grading duty rows linked to the submission group of which this user is a part, the is_aggregate_marks_dirty flag is set to 1, recompute aggregate marks for those grading duty rows and set the flag to 0
Return Data	**Consider the submission group of this user (recall that in any event, a single user can be a part of at most one submission group). Also consider the SUPLOAD subevent of which this user was a part (recall that in any event, a single user can be a part of at most one SUPLOAD subevent). For each GUPLOAD subevent for which the SUPLOAD event is a generating subevent (return the following as a package)** **1. If an MVIEW event is going on, return all grading_duty rows (except grader_id, is_late_grading, is_aggregate_marks_dirty) that are linked to responses of this submission group and linked to that GUPLOAD subevent for which is_completed = 1. Also return all grading_duty_has_rubrics rows related to those (completed) grading_duty rows as well as all rubrics rows related to that question (so that the frontend may also show the student which all rubrics were not applied to their response). If there are grading_duty rows which have is_completed = 0, do not send their related information and simply send a blank collection so that frontend may alert the student that grading was not complete for that grading duty** **2. If an RMVIEW event is going on, return all grading_duty rows (except grader_id, is_late_grading, is_aggregate_marks_dirty) that are linked to responses of this submission group and linked to any RGREQ subevent for which this GUPLOAD is specified as the TED param for which is_completed = 1. Also return all grading_duty_has_rubrics rows related to those grading_duty rows as well as all rubrics rows related to that question (so that the frontend may also show the student which all rubrics were not applied to their response). If there are grading_duty rows which have is_completed = 0, do not send their related information and simply send a blank collection so that frontend may alert the student that grading was not complete for that regrading duty** **Note that if both an MVIEW and RMVIEW event is going on then both the above kinds of rows will have to be returned. Also Return list of all regrading discussion messages in format sender,message,sequenceId**

Backend API: Post Regrade Request	
URL	`/course/$courseid/event/$eventid/subevent/$subeventid/response/$responseid/regrading`
HTTP Method	**POST**
Pre-Validations	Make sure that 1. The user has a row in the table **enrollment_has_subevents** with respect to an **RGREQ** subevent of this event. Also make sure that the RGREQ event is currently going on 2. That RGREQ event has $subeventid as its TED parameter 3. Make sure that $subeventid is a **GUPLOAD**-type subevent of this event Also make sure that **responseid** is a response such that there is a **grading_duty** row for that response with $subeventid as **grading_duty**.subeventid (i.e. that this is a response that was graded during the GUPLOAD subevent $subeventid) 4. Check if there are any **grading_duty** rows for this $responseid with the currently ongoing RGREQ subevent as the **grading_duty**.request_subevent_id and grading_duty.is_completed = 0. If so, throw a **NOACCESS** error and finish (a student cannot raise a new regrading request for the same question unless the previous regrading request for this question has been completed) 5. Check if there are any grading_duty rows for this $responseid with event_id of subevent as the grading_duty.subevent.event and grading_duty.is_completed = 0. If so, throw a **NOACCESS** error and finish (such that user cannot raise regrading request if any uncompleted grading duty row (includes all prev grading and regrading requests) for this response within this event exists.)

Action	Select a regrader for this regrading request by choosing a regrader as per the regrading policy for the **RGUPLOAD** subevent that is the GEN parameter for the **RGREQ** subevent that is currently going on. If that regrader does not have a row in the table **user_has_subevents** with respect to that **RGUPLOAD** subevent, then create a new **user_has_subevents** row permitting the regrader to perform the regrading task. If there are **grading_duty** rows for this $responseid with the currently ongoing **RGREQ** subevent as **grading_duty**.request_subevent_id, then find the latest regrading request - call the id of this row as prev_id and do the following 1. If the prev_id grading duty row has is_aggregate_marks_dirty = 1, recalculate the marks, store it in aggregate marks and set the dirty flag to 0 2. Create a new **grading_duty** row for this grading duty request with parameters with parameters as following, student_comment set to whatever comment the student sent along with this POST request, prev_grading_duty set to prev_id, subevent_id set to the RGUPLOAD subevent that is the GEN parameter for the RGREQ subevent that is currently going on, grader_id set to the regrader we identified in step 1, request_subevent_id set to the RGREQ subevent that is currently going on, marks_adjustment, aggregate_marks, is_aggregate_marks_dirty copied from the grading_duty row prev_id, is_completed = 0, grader_comments empty, is_late_regrading = 0, Create new **grading_duty_has_rubrics** rows that are copies of those that were associated with the row prev_id.The above is done so that when the regrader sees a repeated regrading request, they see the applied rubrics, marks adjustment etc exactly as they last set thems Otherwise, choose any grading_duty request for this $response that is linked to the GUPLOAD subevent $subeventid that is present in the URL and call the id of that row as prev_id and do the following 1. If the prev_id grading duty row has is_aggregate_marks_dirty = 1, recalculate the marks, store it in aggregate marks and set the dirty flag to 0 2. Create a new **grading_duty** row for this grading duty request with parameters as following, student_comment set to whatever comment the student sent along with this POST request, prev_grading_duty set to prev_id, subevent_id set to the RGUPLOAD subevent that is the GEN parameter for the RGREQ subevent that is currently going on, grader_id set to the regrader we identified in step 1, request_subevent_id set to the RGREQ subevent that is currently going on, rest of the fields blank

Return Data	**Return the student_message that the student wrote so that the frontend may display it as well**

Backend API: Get Section Submissions	
URL	`/course/$courseid/event/$eventid/mySectionSubmissions/`
HTTP Method	**GET**
Pre-Validations	None (Warning: this is supposed to be an instructor/tutor 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 such submission group, return the list of students linked to that submission group. Also, for each SUPLOAD subevent, return a list of students (if submission mode is OG) or submission groups (if submission mode is IN or FG) who were given permission to make submissions (i.e. they had rows created in the table users_have_subevents), but have not (yet) made any uploads**

Backend API: Post Section Submission Group	
URL	`/course/$courseid/event/$eventid/mySectionSubmissionGroups/`
HTTP Method	**POST**
Pre-Validations	None (Warning: this is supposed to be an instructor/tutor level URL)
Action	Given a list of students, a question set, try to create a submission group out of them by performing the following checks. First of all, ensure that at least one of these students is in the same section as the user making this query. Next, make sure that all these students had **user_has_subevent** rows with respect to the same **SUPLOAD** subevent else throw a **NOVAL** error (cannot create a submission group out of students in different **SUPLOAD** subevents). Next, make sure that none of these students already belong to a submission group. If even one student already belongs to a submission group, throw a **NOVAL** error (a student cannot belong to more than one submission group). If the submission mode of the **SUPLOAD** event is not OG, throw a **NOACCESS** error (cannot create a submission group unless it was an open group policy). If the number of students exceeds the $max value in the OG parameter, throw a **NOVAL** error and finish. If the question set specified does not belong to the assignment linked to this event, throw a **NOVAL** error
Return Data	**Return the submissiongroups id of the newly created submission group as well as the list of students in the newly created group so that the front end may display them**

Backend API: Verify Submission Code	
URL	`/course/$courseid/event/$eventid/mySubmission/verifyNAC/`
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 this **SUPLOAD** subevent has parameter **SGS** = OG, then NAC is not required so just return
Action	Allow edits to submission_groups.access_code_submitted if this **SUPLOAD** subevent requires an access code i.e. the parameter NAC = 1 (for example if the user had entered an incorrect code earlier, they may want to correct the access code). verify the submitted access code is equal to access_code_gold , if equal then return flag as true else return flag as false
Return Data	**is_NAC_correct flag**

CHAPTER 6

AI-based Apps on SPHINX

Contents

Abstract *In this chapter we describe a few apps we developed on top of the* SPHINX *system using various machine learning and visual recognition techniques. The image calibration section will describe each stage of the working pipeline of the system that takes in scanned copies of student attempts in written examinations or quizzes and calibrates them to remove shear and rotation effects introduced by the scanning process. The section also contains details of the various experiments we conducted with the corresponding results. In the third section, we explain improvements we propose to the existing roll number recognition system in* SPHINX *by modifying its basic workflow. The last section describes the core workflow of an AI-assisted auto-grading system for objective-style questions.*

6.1 Introduction

In the SPHINX system, course data is an important source of information. As SPHINX is a course management platform, it produces large amounts of data for several courses, course assignments / examinations, questions of various types, graders, graded user submissions, user obtained marks, and much more. The system database is designed and developed to captures and store all course data in an organized and formatted manner at the application server. This enables the extraction of large-sized data sets and development of various AI apps by feeding these extracted data-sets to several machine learning and visual recognition techniques. In this chapter we explain a few such AI tools like an image calibration tool, roll number recognition tool, and an auto-grading system.

The image calibration tool transforms scanned copies of student submissions which may contain distortions (shear, rotation etc) into a calibrated set of images as per the target orientation. As various AI tools in the system uses scanned copies uploaded by users, the calibration tool plays an important role in improving overall accuracy of these tools by removing distortions in the scans. The roll number recognition tool mainly aims to reduce the manual effort of mapping the user uploaded submission copies to the respective user accounts in the course. We enhance the accuracy and performance of the existing roll number recognition tool in the system by modifying the existing pipeline. In the last section we explain the auto-grading tool, which aims to eliminate the manual grading of some type of questions and increase the scalability of the core grading feature of the SPHINX system.

6.2 Image Calibration

The SPHINX system allows instructors to scan and upload thousands of exam / assignment copies of students as PDF/image files on the platform to perform grading. As SPHINX allows the instructor to specify the page coordinates, using a skeleton file, the user roll number, the questions and corresponding answers on the question paper, the uploaded images are a major source of information for many AI based tools in the system. However, while scanning the hard copies, often the pages may get rotated or else the scanned image may experience shear due to effects of the scanner device which directly results the distortion in the uploaded images.

As many AI based tools require coordinate based cropped regions of the uploaded copies as a dataset, the machine learning / visual recognition techniques may experience catastrophic failure because of these distortions in the uploaded copies. Thus, this is an important issue that needs to be resolved in order for the AI techniques to perform well. We develop an image calibration tool which takes the source & target (instructor supplied skeleton) images as input, performs feature based multi-dimensional alignment, and returns the set of transformed images as the output. The various stages of the calibration pipeline are explained below in the great detail,

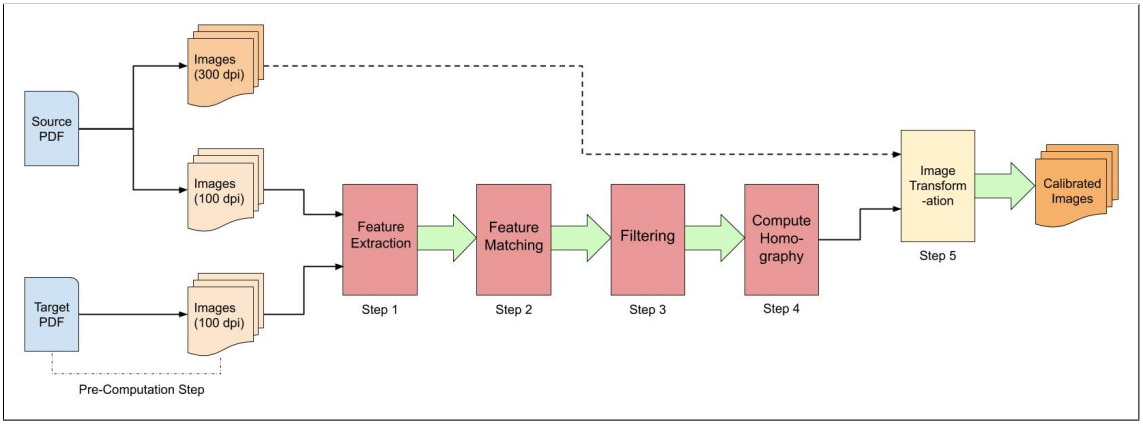


Figure 6.1: The Calibration Pipeline

6.2.1 Pre-Computation stage

As the later stages of the calibration pipeline requires imput as a set of images, we convert any uploaded PDF files into images in the pre-computation stage. Various machine learning / visual recognition techniques perform very well when the data set contains the high resolution images. However, aligning high resolution images using calibration is very expensive as feature extraction stage takes too long to compute on high quality images. In order to solve this issue we compute respective calibration stage using the low quality versions of the source & target files, then transform the higher quality images of the source file using the obtained transformation parameters. The detailed explanation of transformation is provided in the last stage of the calibration. Due to this we convert the source PDF files into two sets of different quality images, higher resolution (300 dpi) images and lower resolution (100 dpi) images respectively. The target source file is converted into only lower resolution images as only that is required to compute the homography matrix [9] for transformation. The sets of low resolution images of source and target image files are given as input to the feature extraction step.

6.2.2 Feature extraction

To align the images, we need to find stable points on the images, also known as the features or key points of the image. There are various feature extraction algorithms like ORB [19], SIFT [18], and SURF [14] that are implemented in the OpenCV library [12]. SIFT applies the DoG (difference of gaussian) function to a series of images at different scales and finds out scale invariant stable features in the image. SURF is inspired by SIFT but it uses a box filter approach to compute the operators which makes it faster than SIFT. ORB (Oriented FAST and Rotated BRIEF) is developed by OpenCV using the FAST algorithm for keypoint detection and the BRIEF algorithm for keypoint descriptor. ORB uses the muliscale pyramid approach, in which the image is scaled at many resolutions, and computes the features at each level and then aggregates the features to make it partially scale invariant.

We compute and compare above three approaches using OpenCV APIs and various configuration parameters. We use student examination data of the 2019-20-I offering of the course CS771 as the dataset. We found SIFT to be robust and powerful but computationally expensive than others. While SURF gives similar performance to SIFT with faster computation speed, as ORB is lightweight, it significantly outperforms the other two approaches and detects good features for the further steps. As SURF and SIFT are computationally expensive in case of large number of files, we use the ORB feature extraction algorithm in implementation of the calibration tool.

Thus, we extract the feature points and their descriptors from the received source and target images using the detectAndCompute method of ORB provided by OpenCV [12]. Then the extracted features points are given as input for the next step in the calibration pipeline.

Algorithm	Computation time
SIFT (keypoints = 5000)	1243.60 Seconds
SIFT (keypoints = 1000)	483.96 Seconds
SURF (HessianThreshold = 3000)	890.41 Seconds
SURF (HessianThreshold = 5000)	412.63 Seconds
ORB	80.45 Seconds

Table 6.1: Feature Extraction Algorithm comparison for set of 900 images

6.2.3 Feature matching and Filtering

In this step, we match the feature points of the source image to the feature points of the target image and form the respective matched pairs which are used to identify the transformation between the source and the target images. There are two approaches that can be used to match the features, a brute force matcher and FLANN based matcher. A brute force matcher computes the distance between each point to all points in another set to match the respective features between two sets while theFLANN based matcher provides a collection of algorithms to optimize searching of the nearest neighbour for high dimensional features in order to match the two sets. In most of the cases FLANN based approach outperforms the brute force matcher. We use the FLANN based matcher to match the source image features to target image features. After finding matches, we filter the matches and find out the best matches. To find the best matches, we compare the first nearest feature with the second nearest feature; if the distance of these two features from the source image feature is more than a chosen constant then the match is considered as a good match for further step.

6.2.4 Finding Homography matrix

A homography is a 3×3 dimensional transformation matrix which maps points from source image to points in target image. To compute the homography matrix, the minimal requirement is that of at-least four matched pairs of features. In this step, we compute the homography matrix using the findHomography API [1] provided by OpenCV [12]. The obtained matrix is given as input to the last stage of calibration pipeline.

6.2.5 Transformation of image using the learnt homography

In this stage we calibrate and map the pages of the source PDF to the pages of the target PDF. The received homography is applied to each pixel of the higher resolution images in order to remove the distortion in them. However, since the homography matrix was computed on lower resolution images, applying it directly on the higher resolution images does not give correct results. Thus, to align the source images successfully, we perform the following three steps for each pixel of the image in the below given order,

[1] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findhomography#findhomography

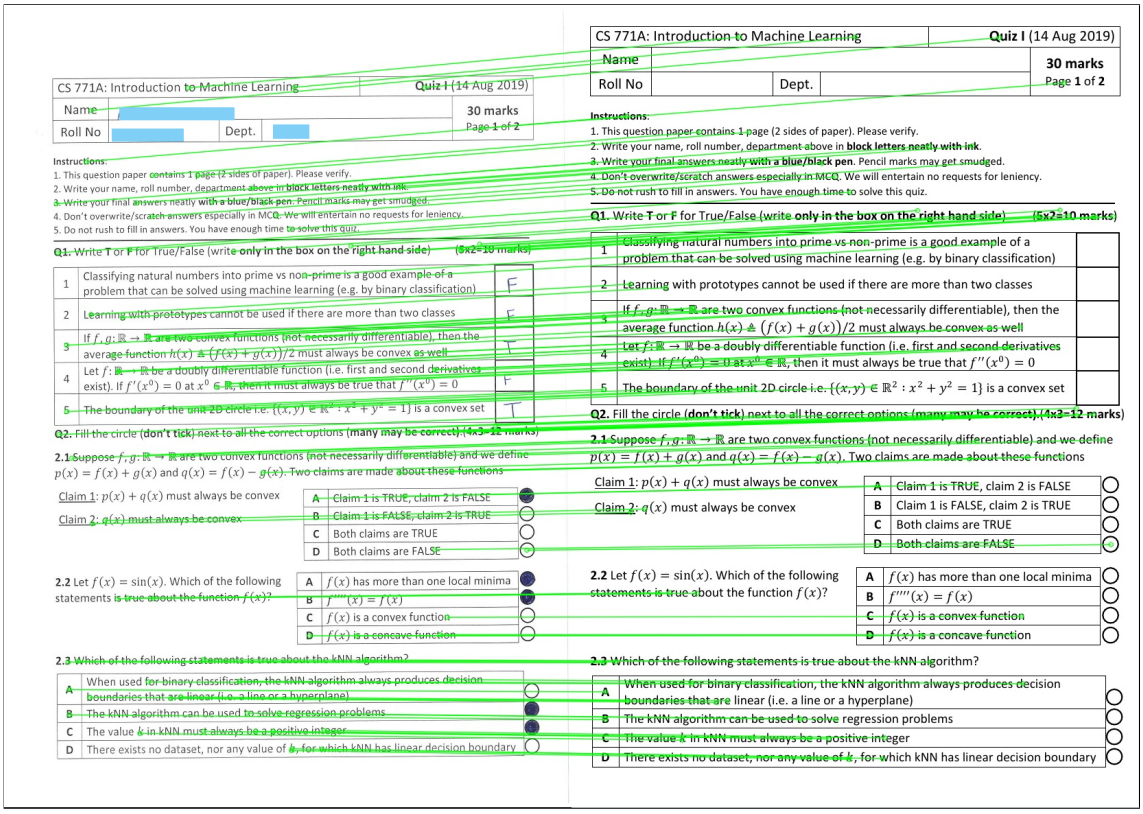


Figure 6.2: The matched features between the Source PDF Page and Target PDF Page

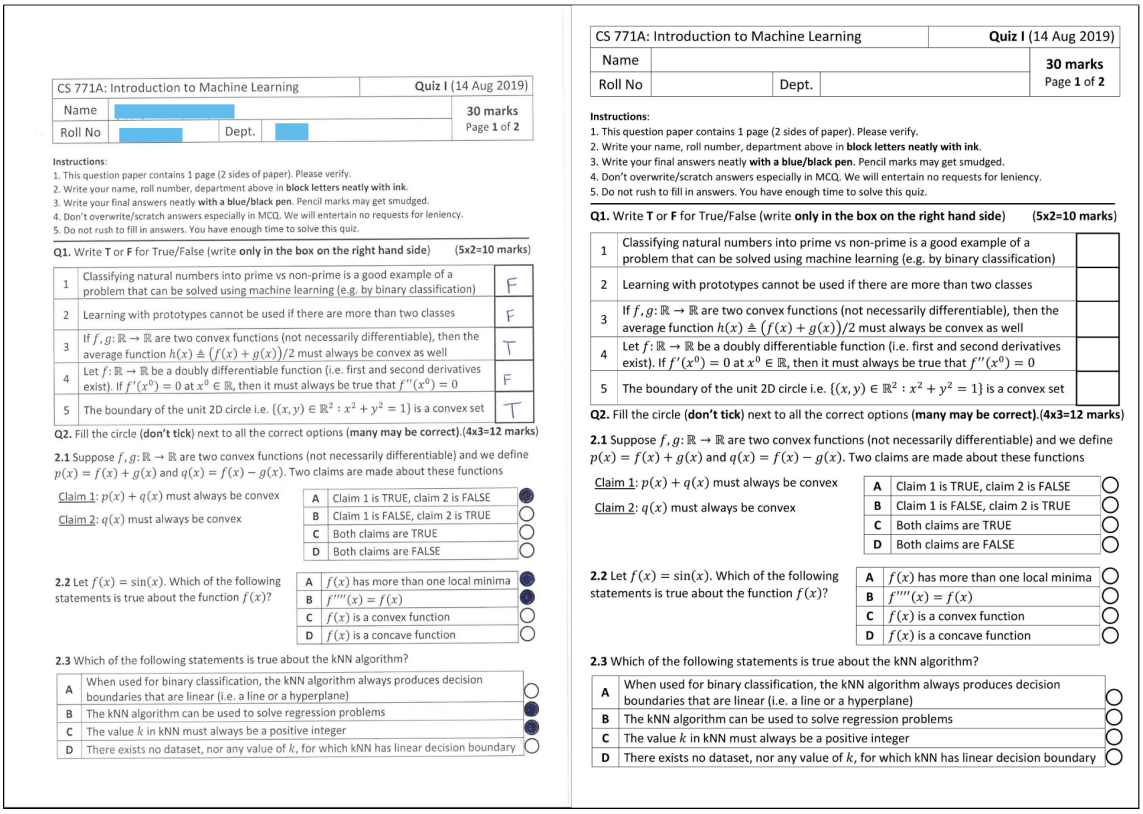
1. Downscale the pixel from higher resolution to the lower resolution.
2. Map the distorted pixel to its desired position by applying the homography on the pixel using the warpPerspective method [2] of OpenCV [12]
3. Upscale the transformed pixel from lower resolution to the higher resolution.

To downscale and upscale the pixels, we achieve the required transformation parameters using getPerspectiveTransform API [3] of OpenCV [12]. After performing the above transformation we obtain well calibrated images of the source.

6.3 Roll Number Recognition

As SPHINX allows instructors to scan and upload examination copies of the students as the one large PDF file on the platform, it raise another challenge to map those copies to corresponding user accounts of the student. It is time consuming for the instructor to manually map each student to their respective exam copy as many times there are hundreds of students are present in the course. To solve this problem [16] proposed a roll number recognition system. The system takes the user handwritten roll number in image format, performs image processing, and returns the recognized roll number of the user. The existing roll number recognition system performs the steps shown in the below figure in order to obtain the output.

[2] https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#warpperspective
[3] https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#getperspectivetransform



CS 771A: Introduction to Machine Learning — Quiz I (14 Aug 2019)

Name [redacted]

Roll No [redacted] Dept. [redacted]

30 marks

Page 1 of 2

Instructions:
1. This question paper contains 1 page (2 sides of paper). Please verify.
2. Write your name, roll number, department above in **block letters neatly with ink**.
3. Write your final answers neatly **with a blue/black pen**. Pencil marks may get smudged.
4. Don't overwrite/scratch answers especially in MCQ. We will entertain no requests for leniency.
5. Do not rush to fill in answers. You have enough time to solve this quiz.

Q1. Write **T** or **F** for True/False (write **only in the box on the right hand side**) **(5x2=10 marks)**

1	Classifying natural numbers into prime vs non-prime is a good example of a problem that can be solved using machine learning (e.g. by binary classification)	F
2	Learning with prototypes cannot be used if there are more than two classes	F
3	If $f, g: \mathbb{R} \to \mathbb{R}$ are two convex functions (not necessarily differentiable), then the average function $h(x) \triangleq \big(f(x) + g(x)\big)/2$ must always be convex as well	T
4	Let $f: \mathbb{R} \to \mathbb{R}$ be a doubly differentiable function (i.e. first and second derivatives exist). If $f'(x^0) = 0$ at $x^0 \in \mathbb{R}$, then it must always be true that $f''(x^0) = 0$	F
5	The boundary of the unit 2D circle i.e. $\{(x, y) \in \mathbb{R}^2 : x^2 + y^2 = 1\}$ is a convex set	T

Q2. Fill the circle (**don't tick**) next to all the correct options (**many may be correct**).**(4x3=12 marks)**

2.1 Suppose $f, g: \mathbb{R} \to \mathbb{R}$ are two convex functions (not necessarily differentiable) and we define $p(x) = f(x) + g(x)$ and $q(x) = f(x) - g(x)$. Two claims are made about these functions

Claim 1: $p(x) + q(x)$ must always be convex

Claim 2: $q(x)$ must always be convex

A	Claim 1 is TRUE, claim 2 is FALSE
B	Claim 1 is FALSE, claim 2 is TRUE
C	Both claims are TRUE
D	Both claims are FALSE

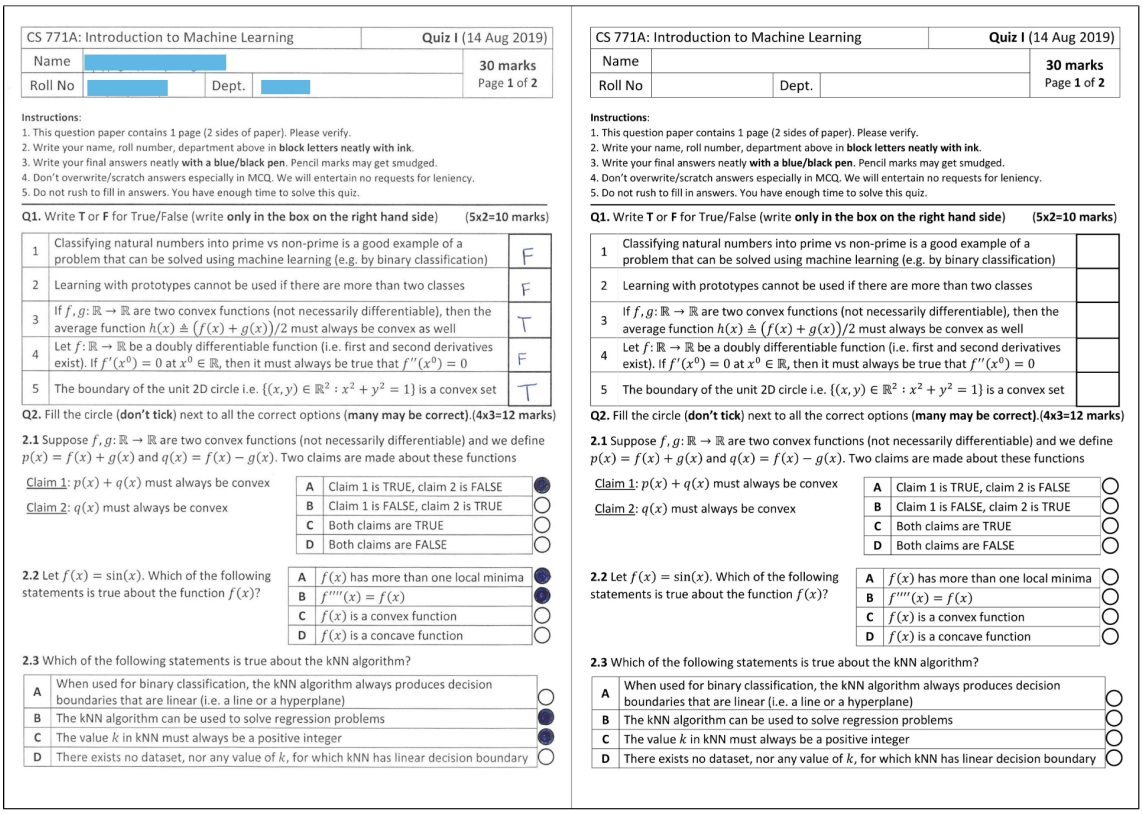
2.2 Let $f(x) = \sin(x)$. Which of the following statements is true about the function $f(x)$?

A	$f(x)$ has more than one local minima
B	$f''''(x) = f(x)$
C	$f(x)$ is a convex function
D	$f(x)$ is a concave function

2.3 Which of the following statements is true about the kNN algorithm?

A	When used for binary classification, the kNN algorithm always produces decision boundaries that are linear (i.e. a line or a hyperplane)
B	The kNN algorithm can be used to solve regression problems
C	The value k in kNN must always be a positive integer
D	There exists no dataset, nor any value of k, for which kNN has linear decision boundary

Figure 6.3: The Source PDF Page and Target PDF Page before the Calibration



CS 771A: Introduction to Machine Learning — Quiz I (14 Aug 2019)

Name [redacted]

Roll No [redacted] Dept. [redacted]

30 marks

Page 1 of 2

Instructions:
1. This question paper contains 1 page (2 sides of paper). Please verify.
2. Write your name, roll number, department above in **block letters neatly with ink**.
3. Write your final answers neatly **with a blue/black pen**. Pencil marks may get smudged.
4. Don't overwrite/scratch answers especially in MCQ. We will entertain no requests for leniency.
5. Do not rush to fill in answers. You have enough time to solve this quiz.

Q1. Write **T** or **F** for True/False (write **only in the box on the right hand side**) **(5x2=10 marks)**

1	Classifying natural numbers into prime vs non-prime is a good example of a problem that can be solved using machine learning (e.g. by binary classification)	F
2	Learning with prototypes cannot be used if there are more than two classes	F
3	If $f, g: \mathbb{R} \to \mathbb{R}$ are two convex functions (not necessarily differentiable), then the average function $h(x) \triangleq \big(f(x) + g(x)\big)/2$ must always be convex as well	T
4	Let $f: \mathbb{R} \to \mathbb{R}$ be a doubly differentiable function (i.e. first and second derivatives exist). If $f'(x^0) = 0$ at $x^0 \in \mathbb{R}$, then it must always be true that $f''(x^0) = 0$	F
5	The boundary of the unit 2D circle i.e. $\{(x, y) \in \mathbb{R}^2 : x^2 + y^2 = 1\}$ is a convex set	T

Q2. Fill the circle (**don't tick**) next to all the correct options (**many may be correct**).**(4x3=12 marks)**

2.1 Suppose $f, g: \mathbb{R} \to \mathbb{R}$ are two convex functions (not necessarily differentiable) and we define $p(x) = f(x) + g(x)$ and $q(x) = f(x) - g(x)$. Two claims are made about these functions

Claim 1: $p(x) + q(x)$ must always be convex

Claim 2: $q(x)$ must always be convex

A	Claim 1 is TRUE, claim 2 is FALSE
B	Claim 1 is FALSE, claim 2 is TRUE
C	Both claims are TRUE
D	Both claims are FALSE

2.2 Let $f(x) = \sin(x)$. Which of the following statements is true about the function $f(x)$?

A	$f(x)$ has more than one local minima
B	$f''''(x) = f(x)$
C	$f(x)$ is a convex function
D	$f(x)$ is a concave function

2.3 Which of the following statements is true about the kNN algorithm?

A	When used for binary classification, the kNN algorithm always produces decision boundaries that are linear (i.e. a line or a hyperplane)
B	The kNN algorithm can be used to solve regression problems
C	The value k in kNN must always be a positive integer
D	There exists no dataset, nor any value of k, for which kNN has linear decision boundary

Figure 6.4: The Source PDF Page and Target PDF Page After the Calibration

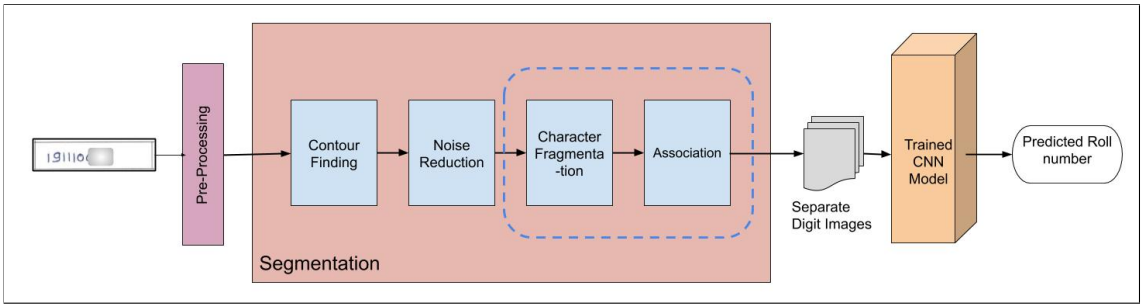


Figure 6.5: The Existing Roll Number Recognition System

At the pre-processing stage, the existing system performs image processing operations like converting from RGB to grey-scale, thresholding and morphological operations to make the image ready for the segmentation. The segmentation stages identify contours from the image, remove noisy contours, and perform contour cleaning in order to identify each digit separately. After obtaining the digits as a set of separate images, they are fed to a learnt CNN model. The model is trained using the standard MNIST data-set [17], which contains around 70000 handwritten digit images. The system returns the roll number using the digits predicted by the CNN model.

Although the existing system gives acceptable accuracy, it has some drawbacks. After the noise reduction step, if the segmentation steps fails to identify each character separately from the received roll number image, then the system terminates the process without executing further steps and returns an error for the corresponding request. The existing system also does not make any use of existing enrollment data in the course database like list of enrolled students in the course, and their respective roll numbers to predict the handwritten roll number in case the segmentation procedure is unable to identify or the CNN model fails to predict any one of the roll number digits correctly. The existing system also requires fine tuning of the segmentation steps based on the number of identified contours and the length of the roll number. During segmentation, the system executes character fragmentation and association steps for all the cases of input, However this results in incorrect prediction in some of the cases.

6.3.1 Improvements

In order to improve the accuracy and enhance the performance of the existing roll number recognition system, we implement the changes mentioned in following list,

- Image Segmentation: After the noise reduction, we handle the remaining steps of segmentation based on the number of identified contours and the length of the roll number
  - If number of identified contours = the length of the roll number, then we skip character fragmentation and association steps and forward the separated digit images to the remaining steps.
  - If number of identified contours < the length of the roll number, only then we perform character fragmentation as it split the contours which contain more than one digit
  - If number of identified contours > the length of the roll number, only in this case we perform association as it combines smaller contours which belong to a single digit of

the roll number

- Even though in some cases the segmentation steps are not able to separate all characters correctly from the received roll number image, we try to predict as many possible digits from the roll number using the CNN model by handling various raised exceptions appropriately. The remaining digits are predicted in the last step using the confidence score obtained from the CNN model and the existing course enrollment data in the course database of the SPHINX system.

- In order to the improve the accuracy of the roll number recognition, We add a digit cleaning step at the end of the existing procedure. This step receives the list of predicted roll numbers and corresponding confidence scores obtained from the CNN model. In this step, we use the list consisting true roll numbers of the students enrolled in the course from the application database. The step takes the following actions in given sequence,

  - Map each roll number from the predicted list to the list of true roll numbers. In case two predicted numbers match the same entry, resolve the conflict by preferring the number with higher confidence score.
  - After completion of mapping, filter out the mapped pair of roll numbers from both the lists and decrease the confidence score of each remaining roll number from the predicted list by 25%.
  - After filtering out the mapped roll numbers, the predicted list of roll numbers contains the numbers in which one or more digits are predicted incorrectly. To identify and replace those digits we use the popular longest common sub-sequence algorithm (LCS). The algorithm processes two received strings and returns the length of longest common sub-sequence. We perform LCS for each roll number from the predicted list with all the remaining true roll numbers and replace the predicted roll number with the corresponding true number having maximum length of common sub-sequence.

After the last step, the system returns a sorted list of roll numbers with corresponding confidence scores.

6.3.2 Experiments

We tested the updated roll number recognition system using the a dataset curated from the 2019-20-I offering of the CS771 course. The dataset contains submission copies of all the students for mid-sem exam, end-sem exam, and four quizzes conducted during the semester. We run three incremental versions of the roll number recognition system for comparison, these are the old version of the system, the system after updating the segmentation process, and the system with the complete update (updated segmentation + digit cleaning using LCS etc). The obtained accuracies with these three versions for all the exams are given below.

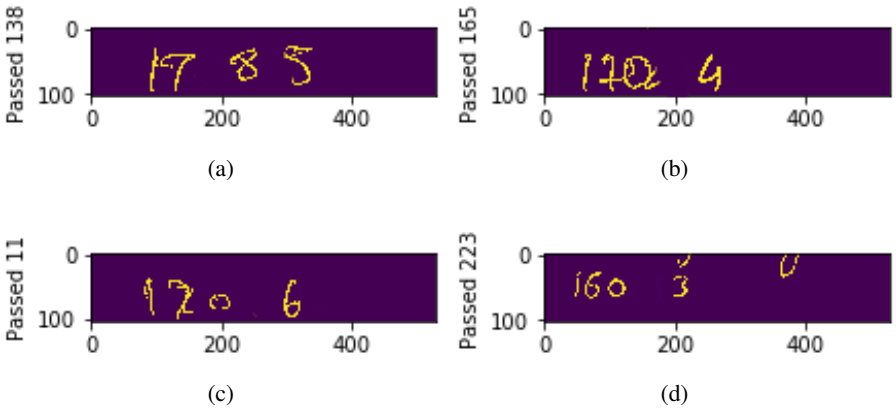


Figure 6.6: Examples where the updated segmentation method Passed. Some digits of each roll number are obfuscated to protect the identity of the student.

Exam Name (Total number of appeared students)	Old version	Improved Segm.	Improved Seg. + Digit Cleaning
End-sem (227)	82.58	88.39	91.96
Mid-sem (238)	77.96	83.05	92.37
Quiz 1 (240)	81.61	87.81	92.85
Quiz 2 (234)	81.03	88.36	96.12
Quiz 3 (225)	81.61	86.09	92.85
Quiz 4 (219)	75.11	82.94	87

Table 6.2: A Performance Comparison Table

6.4 An AI-assisted Auto-grading System

The number of enrolled students in online courses, owing to their good quality and accessibility, is increasing rapidly. In such large courses, grading student submissions of exams or quizzes becomes very time consuming for the instructor, even with the help of few graders. Even manual grading of short-answer questions may require several manual hours for hundreds of copies. In the SPHINX system, we try to solve this problem for short answer type questions by implementing an AI-assisted auto-grading system.

The SPHINX system allows instructors to upload and store PDF files of exam paper and specify the page coordinates for each question and respective answer boxes. SPHINX also provides capabilities to the instructor or the student themself to upload the scanned student submission copies as a PDF file. Our task is to use this information and provide the capabilities to grade the thousands of similar submissions at the same time using AI algorithms.

6.4.1 System description

We developed a system which provides assistance in the task of evaluating short answer questions like True/False and multiple choice questions. The tool receives two parameters, the question type and the list of images, where each image represents the cropped region of question answer.The above problem is solved as below,

1. **Pre-processing:** Each image from the received array of images is passed through the preprocessing step. In this step various image processing operations are performed like converting image from RGB to Gray-scale, thresholding, and some required morphological operations in order to remove the noise from image, are done.

2. **Contour Detection:** In many cases the question contains several sub-questions for example the true/false type question usually contains more than one sub-questions. In this step, we search for identical geometrical shapes like circles or squares which corresponding to answers of the sub-questions. To identify the boxes we use the contour-finding API provided by OpenCV. Then we separate those sub-question answer boxes by assigning a unique ID to each contour and forming the data-set to use in next step

3. **AI Computation:** After receiving the data-set, this step takes the following action based on question type.

    - If the question type is True/False, we use a Convolution Neural Network (CNN) model to recognize the handwritten character from the received answer boxes. The CNN model processes the images and predicts the cluster label among three possibilities. The three clusters representing the user given answer as True, False, and Other. The architecture details, dataset and the results of the CNN model are explained below separately in detail.
    - If the question type is MCQ (fill in the circles etc), we compute the sum of each pixel intensity of the image. The computed sum is compared with the threshold constant. If sum is greater or equal to threshold constant then we predict the image state as having been marked by the student as a correct option, else predict the image as unmarked.

4. **Prediction Formatting:** After receiving the results of the above computations, we format and return the results based on the question type as below,

    - If the question type is True/False, we form the three clusters for sub-questions using the unique assigned ID and the predicted cluster ID. After formatting the result we return three sets of identical images with their cluster information for each sub-question, so that a human grader can grade the three clusters instead of grading each user submission separately.
    - If the question type is MCQ, we form a list for each image using the assigned unique ID and predicted image state (marked/unmarked). The list contains the predicted marked options by the user in corresponding answer. After formatting all the data, we return the array of lists, where each list correspond to each image of the user submission

SPHINX can then autograde the MCQs using the predicted list and the expected gold answer of the question for each submission.

6.4.2 The CNN model

We create our own dataset from student submissions of the 2019-2020 - I offering of the course CS771. The dataset contains the 7200 training images and 2332 testing images of handwritten character **T**, **F** and others. Each image from the dataset is 88×88 dimensional and labeled as 0/1/2 corresponding to True, False and Others respectively. We trained the below given architecture of the convolutional neural network over the training set for 10 epochs to obtained the reasonable performance.

CNN Architecture								
Input layer	**conv**	**conv**	**maxpooling**	**conv**	**conv**	**maxpooling**	**Fully connected**	**output**
88x88x1	3x3x32	3x3	3x3	3x3x64	3x3x64	2x2	256	3

Table 6.3: The CNN architecture used by SPHINX for Handwritten Character classification

We tested the model over the test set containing 2332 images obtained from the mid-sem examination data of the course. We observed that our approach works reasonably well and gives an accuracy over 98%. The model is not able to classify certain images where students try to re-answer the question by crossing out the old answer. Some of the failed results are given in the below figure.

Character Recognition			
# Total Examples	#Correctly Recognized	# Incorrectly Recognized	% Accuracy
2332	2297	35	98.41

Table 6.4: Character Classification Results on our T-F Dataset

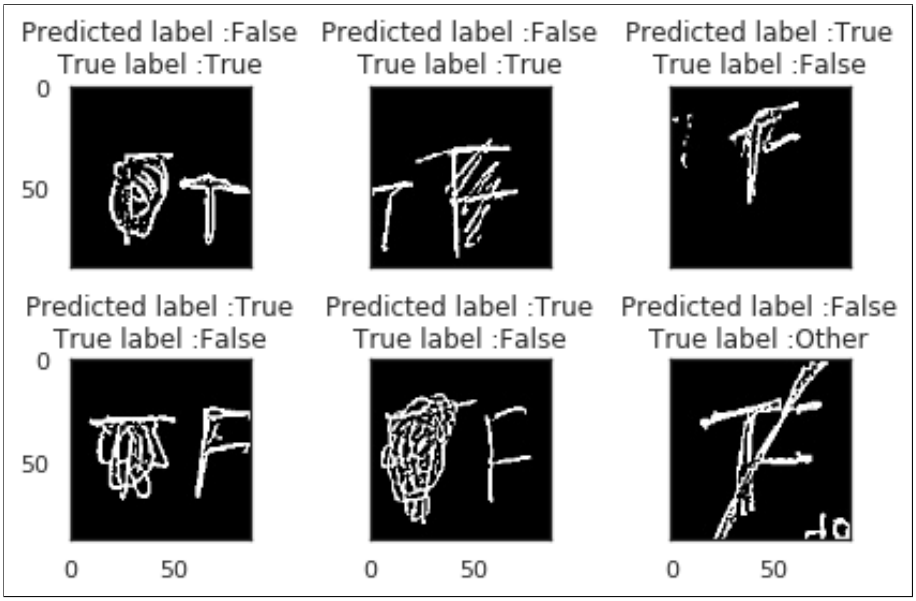


Figure 6.7: The examples where our model failed to predict the label correctly

CHAPTER 7

Conclusion

This thesis offers and efficient and user friendly front-end system and augments the existing back-end server architecture of the SPHINX Pedagogy Helper Application. In our work, we reported a highly efficient and well designed user interface for the SPHINX system. The user interface provides a rich set of features to make the SPHINX system easily accessible to the end user. Using machine learning techniques, we built various apps like image calibration, an improved roll number recognition app, and an autograder app on top of the SPHINX system, which reduces the course management efforts required by the instructor significantly. We augmented the back-end of the SPHINX system by implementing a set of new features in the existing system. The new SPHINX system offers plenty of scope for building the new useful applications. Some of the possible advancements in current system are given below,

1. Enable live online examinations on the platform so that instructor can set the exam paper and the students can appear for the exam by submitting answers online during a given time window of the exam.

2. Develop a course analytics layer that it analyses various activities performed by the users on the platform

3. Analyze various course data available at the application database like type of questions, user submitted responses, user obtained grades, etc and provide the useful statistical information to the instructor like user performance in various type of questions, the list of topics in which students facing difficulties, possible predicted grades based on current performance of the students, etc.

4. Build AI-enabled recommendation systems for the students based on their performance in the course. The system can use the student obtained grades in various questions, and the list of course topics associated those questions by the instructor and provide the student specific guidance on which topics to revise better or focus more.

5. Provide a platform to the users to discuss various technical doubts raised during the course. Each doubt can be tagged with course topics or keywords which can provide possibilities for further AI enabled applications.

6. Increase support of the AI-assisted grading system to other type of questions like one word answer, single sentence answer questions, etc.

Bibliography

[1] Angular. https://angular.io/.

[2] Angular Components. https://angular.io/guide/architecture-components.

[3] Angular Modules. https://angular.io/guide/architecture-modules.

[4] Angular Services. https://angular.io/guide/architecture-services.

[5] Canvas. https://www.instructure.com/canvas/higher-education/platform/products/canvas-lms.

[6] Cryptography Standards Library. https://www.npmjs.com/package/crypto-js.

[7] EasyClass. https://www.easyclass.com/.

[8] Google Classroom. https://classroom.google.com.

[9] Homography in Computer Vision. https://en.wikipedia.org/wiki/Homography_(computer_vision).

[10] Nebular the customizable Angular UI Library. https://akveo.github.io/nebular/.

[11] Ngx File Upload Library. https://www.npmjs.com/package/@iplab/ngx-file-upload.

[12] Open-CV. https://opencv.org/.

[13] Smart Table Library. https://akveo.github.io/ng2-smart-table/.

[14] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[15] Martin Dougiamas and Peter Taylor. Moodle: Using learning communities to create an open source course management system. In *EdMedia+ Innovate Learning*, pages 171–178. Association for the Advancement of Computing in Education (AACE), 2003.

[16] Neeraj Kumar. A Scalable and Flexible Pedagogy Helper Application. Master's thesis, Indian Institute of Technology, Kanpur, 2019.

[17] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[18] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[19] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *International Conference on Computer Vision*, pages 2564–2571, 2011.

[20] 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 ACM Conference on Learning @ Scale*, L@S '17, page 81–88, New York, NY, USA, 2017. Association for Computing Machinery.