



**BSc (Hons) in Information Technology Specializing in Software  
Engineering  
Year 3 - 2022**

**SE3040 – Application Frameworks  
Group Project: Technical Report**

Group Name:

Group Member Details

Student ID number	Name	SLIIT Email
IT20025076	Ranasinghe R D T D	<a href="mailto:it20025076@my.sliit.lk">it20025076@my.sliit.lk</a>
IT20019372	Peiris W.O.A.	<a href="mailto:it20019372@my.sliit.lk">it20019372@my.sliit.lk</a>
IT20003678	Pitawela P. C. Y.	<a href="mailto:it20003678@my.sliit.lk">it20003678@my.sliit.lk</a>
IT20018832	Abeykoon A.M.U.S.B.	<a href="mailto:it20018832@my.sliit.lk">it20018832@my.sliit.lk</a>

Git Repository:

Group Presentation Video:

Project Start Date:

Project End Date:

**Project Declaration:**

I the undersigned solemnly declare that the project technical based on our group members own work carried out during the course of our study under the supervision of Mr. Thusithanjana Thilakarathna.

---

Group Leader: Ranasinghe R. D. T. D

## Marking Rubric

Date	Weight	Marks (out of 5)			
Description		5	5	5	5
<b>1. SRS Document -</b>	<b>10</b>				
N/A	5				
N/A	5				
<b>2. Frontend Development</b>	<b>25</b>				
a. Viva	10				
b. demo					
1. Implemented all the components [2]					
2. UI/UX [08]					
3. Components [5]	15				
<b>3. Backend Development</b>	<b>25</b>				
a. Viva	10				
b. Demo					
1. Routes [5]					
2. DAL [5]					
3. DAO [5]	15				
<b>4. Apps are hosted</b>	<b>10</b>				
a. Frontend	5				
b. Backend	5				
<b>5. Unit Tests</b>	<b>10</b>				
a. Frontend	5				
b. Backend	5				
<b>6. Test Cases</b>	<b>5</b>				
<b>7. Technical Report</b>					
a. Screenshots of the UI [1]					
b. Screenshots of the MongoDB [2]					
c. Description [2]	5				
<b>Total</b>	100				
<b>Comments</b>					

# **Table of Contents**

	Page No
1. Component Description	
1.1. IT20025076 (User login, Sign-up, and Authentication)	04
1.2. IT20019372 (Student group registration, Topic Registration, and document update)	04
1.3. IT20018832 (Topic Approval and Document Reviewing)	04
1.4. IT20003678 (Administration and Student-staff real-time communication)	05
2. Screenshots of the Frontend	
2.1. IT20025076	06 - 07
2.2. IT20019372	08 - 12
2.3. IT20018832	12 - 14
2.4. IT20003678	15 - 18
3. Screenshots/ Code Snippets of the Backend	
3.1. Routes	
3.1.1. Index.js	
3.2. Controllers	
3.2.1. IT20025076	
3.2.2. IT20019372	
3.2.3. IT20018832	
3.2.4. IT20003678	
3.3. Models	
3.3.1. IT20025076	
3.3.2. IT20019372	
3.3.3. IT20018832	
3.3.4. IT20003678	
4. Screenshots of the MongoDB Schemas	
4.1. IT20025076	
4.2. IT20019372	
4.3. IT20018832	
4.4. IT20003678	

## 5. Proof of Hosting

### 1. Component Description

#### 1.1. IT20025076

##### User login, Sign-up, and Authentication

Research management application users can either log in to their existing account or sign up as a new user. The application consists of mainly 2 parts students and staff. Staff can sign up as either admin or general staff members. There is only one login page for both staff members and students, users will be directed to the relevant interface according to their user roles. User roles will be differentiated by their credentials during the login process.

#### 1.2. IT20019372

##### Student group registration, Topic Registration, and document Upload

After a student enters valid credentials and clicks on the login button, they will be directed to the student welcome page. There are options to register groups, register topics, etc. But students can't navigate to topic registration or any other page except for the notices page without registering a group. A research group must have 4 members and students must mention a supervisor and a co-supervisor in the topic registration. And after creating the project and topics other members of the group can log in to their accounts and see the registration of topic and group also. However, any student can navigate to the notices page to view notices published by the staff. Students can upload the submissions using the template provided by the admin.

#### 1.3. IT20018832

##### Topic Approval and Document Reviewing

Staff –After Student groups register a topic, the status of the topic will be “waiting for Approval” state. A Supervisor/Co-Supervisor is required to review Topics of assigned student groups and approve or reject the topics. After that Assigned student's group will upload the required documents regarding each topic. A Supervisor/Co-Supervisor can download the uploaded documents and refer them. After those two steps, the project forum will be available to students and their relevant supervisors. The supervisor/Co-Supervisor is required to do the discussions as the student requires. A Supervisor/Co-Supervisor also create notices, upload schedules, and publish them in the special notices interface

#### 1.4. IT20003678

##### Administration and Student-staff real-time communication

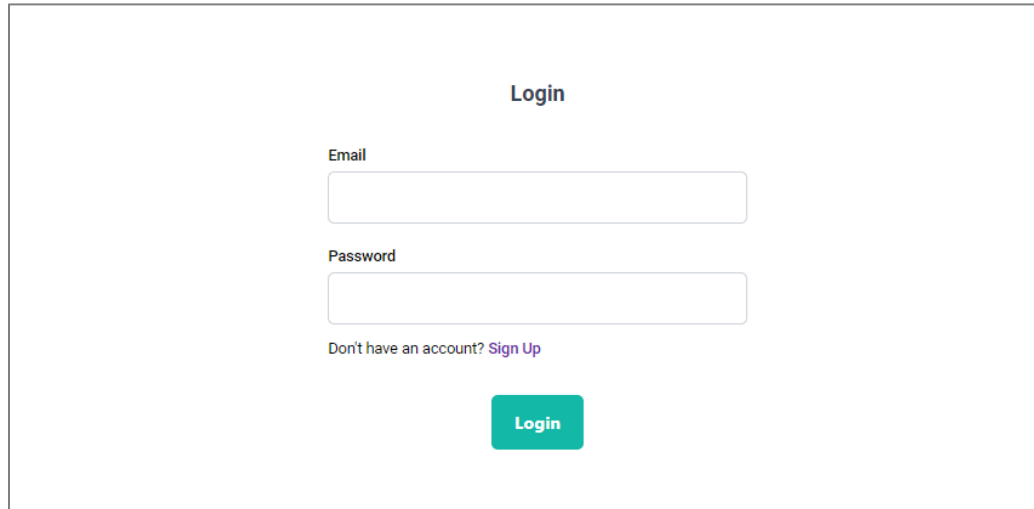
There are System admins for each Faculty of SLIIT. Admins can simply log in to the system using valid credentials. They are responsible for deleting and updating users of the system, allocating Panel members to each student group, creating marking schemes for the projects (year wise/subject wise), and uploading the document/presentation templates. These templates and marking schemes are allowed to be downloaded by the students and staff to do the tasks of their end. Admin can view the roles of the users to manage them properly. Admin can publish notices to inform students and supervisors about upcoming research submissions.

A Chat platform has been developed to provide better communication between the students and their supervisor/co-supervisor. One chat thread is only allowed for the relevant student group and their supervisors. Outsiders are not allowed to view others' chat threads. Students and supervisors can communicate their issues through this chat thread.

## 2. Screenshots of the Frontends

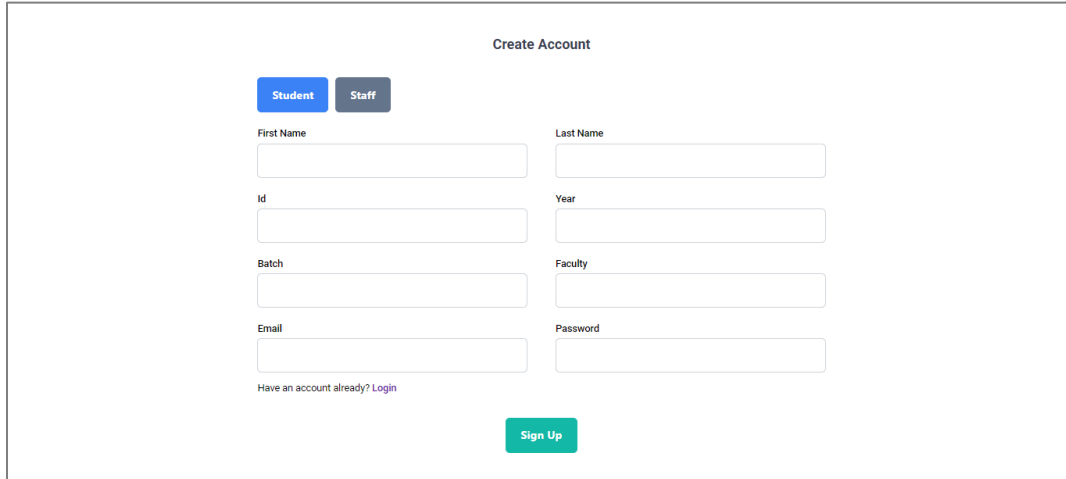
### 2.1. IT20025076

- This is the initial login page. Users can either use their credentials to log in or signup as a new user.



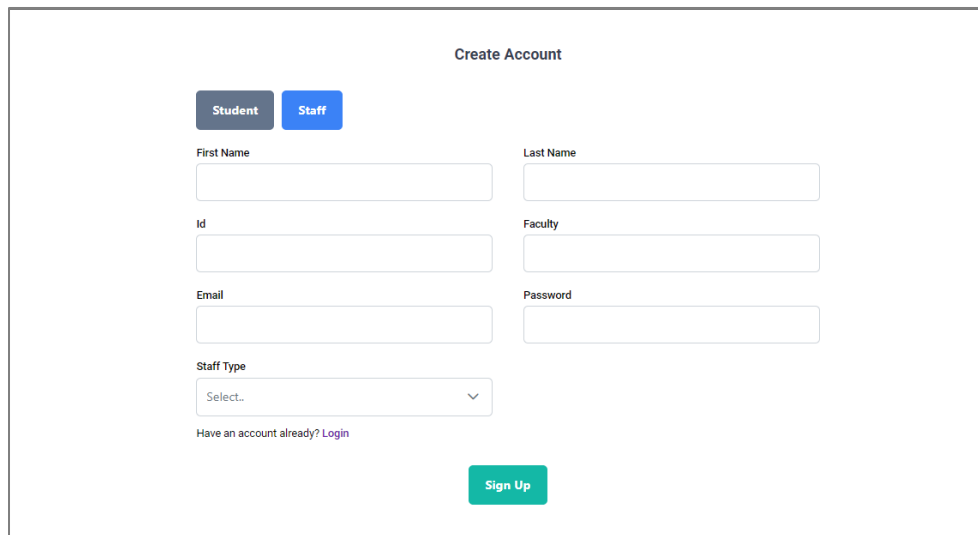
The screenshot shows a login page with a light gray background. At the top center, the word "Login" is displayed in a bold, dark gray font. Below it, there are two input fields: "Email" and "Password", both with light gray borders and no text inside. Under the "Password" field, there is a link that says "Don't have an account? Sign Up" in a small, purple font. At the bottom center, there is a green button with the word "Login" in white text.

- The student signup page requires users to fill out the necessary information to create a new student account.



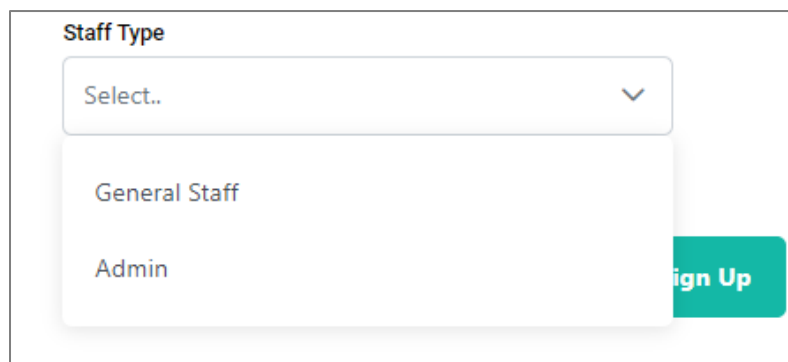
The screenshot shows a "Create Account" page. At the top center, the text "Create Account" is displayed in a small, dark gray font. Below it, there are two buttons: "Student" (blue with white text) and "Staff" (gray with white text). Below these buttons, there are two columns of input fields. The left column has fields for "First Name", "Id", "Batch", and "Email". The right column has fields for "Last Name", "Year", "Faculty", and "Password". All input fields have light gray borders. At the bottom left, there is a link that says "Have an account already? Login" in a small, purple font. At the bottom center, there is a green button with the text "Sign Up" in white.

- The staff signup page also requires users to fill out their information to create a new staff account.



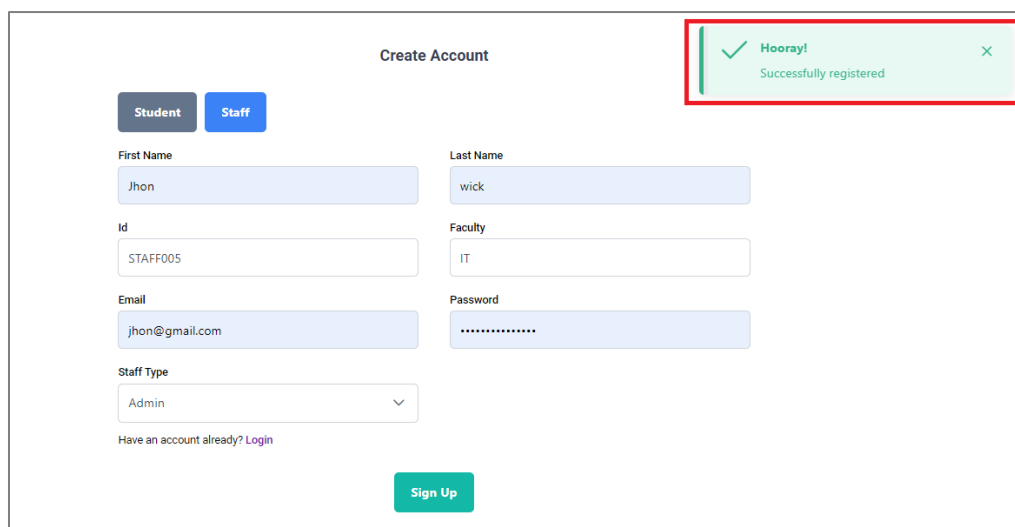
The image shows a 'Create Account' form with two tabs: 'Student' and 'Staff'. The 'Staff' tab is selected. The form contains the following fields: First Name, Last Name, Id, Faculty, Email, Password, and a Staff Type dropdown menu. Below the dropdown is a link 'Have an account already? Login'. A green 'Sign Up' button is at the bottom right.

- There is a staff type selection dropdown on the staff signup page. Staff can create either admin or general staff accounts. Accessibility differs according to the staff type.



The image shows a close-up of the 'Staff Type' dropdown menu. The dropdown is open, showing two options: 'General Staff' and 'Admin'. A green 'Sign Up' button is visible to the right of the dropdown.

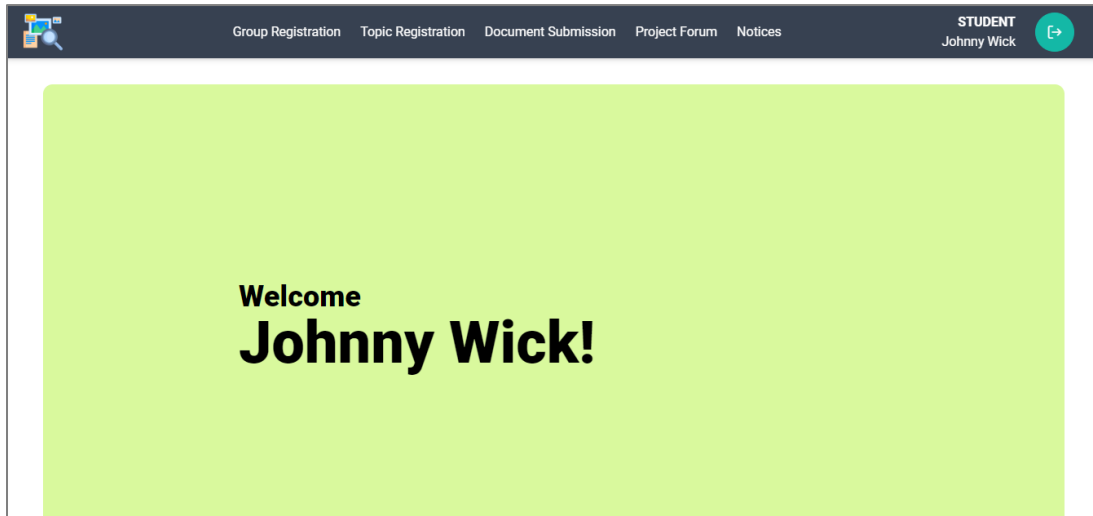
- After pressing the sign-up button, both staff and student pages display either a success message or an error message.



The image shows the 'Create Account' form after a successful registration. A green success message box is displayed at the top right, containing a checkmark, the text 'Hooray! Successfully registered', and a close button (X). The form fields are filled with the following data: First Name: Jhon, Last Name: wick, Id: STAFF005, Faculty: IT, Email: jhon@gmail.com, Password: ..... The 'Staff Type' dropdown is set to 'Admin'. A green 'Sign Up' button is at the bottom right.

## 2.2. IT20019372

- This is the first page when a student gets logged in and sees. In the navigation bar, only the group registration and notices are enabled other tabs are disabled.



- This page is related to registering the student group. Each group should have 4 members only. The leader is responsible for writing the groups which is why the ID entering is disabled in the first column.

A screenshot of a web application interface showing a form titled 'Student Group Registration'. The form is located below a dark navigation bar that is identical to the one in the previous image. The form contains four input fields, each preceded by a label: 'Group Leader ID:', '2nd Member ID:', '3rd Member ID:', and '4th Member ID:'. The first input field contains the text 'STD007'. At the bottom right of the form is a blue button labeled 'Create'.

- If the registration is successful, this appears on the screen, and we can continue to the topic registration.



The screenshot shows the 'Student Group Registration' page. At the top, there is a navigation bar with links: Group Registration, Topic Registration, Document Submission, Project Forum, and Notices. On the right, it says 'STUDENT' and 'STD007 STD007' with a green circular icon containing a white arrow. A green box with a white checkmark and the text 'Successfully registered' is highlighted in the top right corner. Below this, the form fields are: Group Leader ID (STD007), 2nd Member ID (STD001), 3rd Member ID (STD006), and 4th Member ID (STD003).

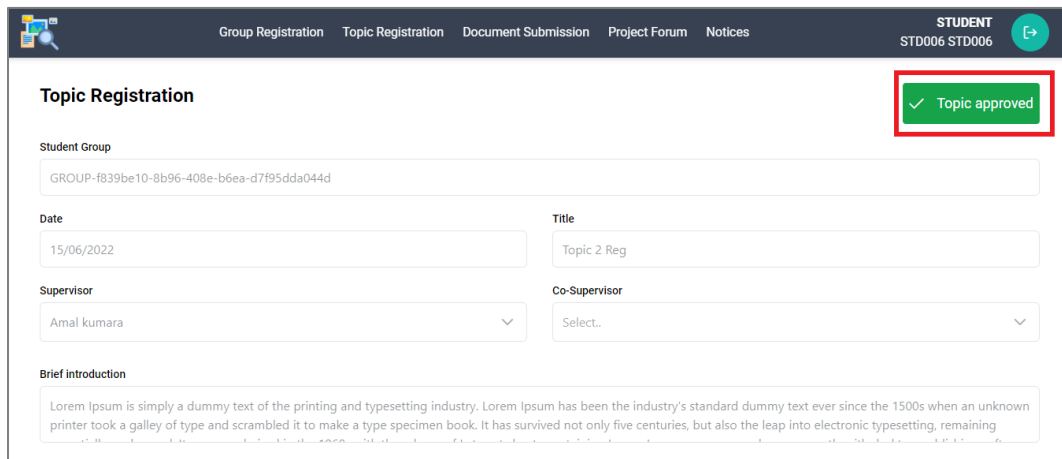
- This is the page for registering the topic.

The screenshot shows the 'Topic Registration' page. It has a form with the following fields: Student Group (GROUP-f839be10-8b96-408e-b6ea-d7f95dda044d), Date (empty), Title (empty), Supervisor (Select..), Co-Supervisor (Select..), and Brief introduction (empty). A blue 'Create' button is located at the bottom right.

- This is what the topic registration interface looks like when the topic is submitted. Until the supervisor Accepts/ Rejects the topic it will show as 'Topic Registration Pending'.

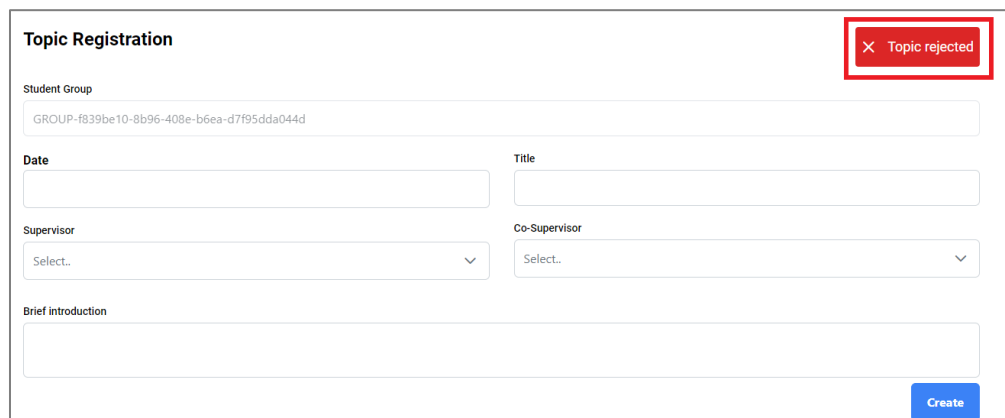
The screenshot shows the 'Topic Registration' page after submission. A yellow box with a red border and the text '... Topic approval pending' is highlighted in the top right corner. The form fields are: Student Group (GROUP-155a3cf2-7d29-4a58-b4d2-5c20ecd8af23), Date (13/06/2022), Title (Topic 1), Supervisor (Amal kumara), Co-Supervisor (Poornima Peiris), and Brief introduction (Lorem Ipsum is simply a dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum).

- If the topic gets approved, we can go to the document submission page. And it shows as below when the topic is approved.



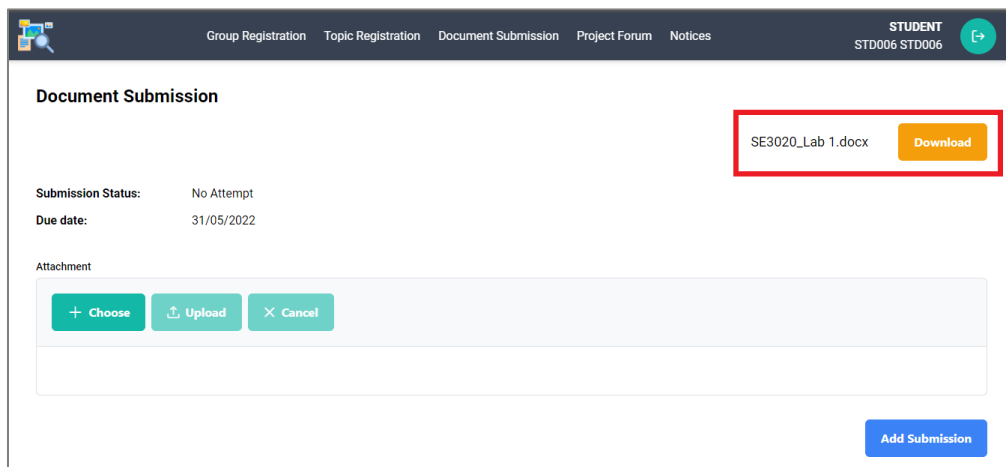
The screenshot shows the 'Topic Registration' page in a web application. At the top, there is a navigation bar with links: 'Group Registration', 'Topic Registration', 'Document Submission', 'Project Forum', and 'Notices'. On the right, it says 'STUDENT STD006 STD006' with a green circular icon. The main content area is titled 'Topic Registration'. In the top right corner of this area, a green box with a checkmark and the text 'Topic approved' is highlighted with a red border. Below this, the form contains the following fields: 'Student Group' (GROUP-f839be10-8b96-408e-b6ea-d7f95dda044d), 'Date' (15/06/2022), 'Title' (Topic 2 Reg), 'Supervisor' (Amal kumara), and 'Co-Supervisor' (Select..). At the bottom, there is a 'Brief introduction' section with placeholder text.

- If the topic gets rejected, we need to submit another topic to continue the other tabs.



The screenshot shows the 'Topic Registration' page. In the top right corner, a red box with an 'X' icon and the text 'Topic rejected' is highlighted with a red border. The form fields are identical to the previous screenshot, but the 'Create' button at the bottom right is visible.

- Then using the mentioned template, we can submit the relevant document for our group submission.



The screenshot shows the 'Document Submission' page. At the top, the navigation bar is the same. The main content area is titled 'Document Submission'. In the top right corner, a red box highlights the text 'SE3020\_Lab 1.docx' and a 'Download' button. Below this, the 'Submission Status' is 'No Attempt' and the 'Due date' is '31/05/2022'. Under the 'Attachment' section, there are three buttons: '+ Choose', 'Upload', and 'Cancel'. At the bottom right, there is a blue 'Add Submission' button.

- Clicking the below button, we can submit the document.

The screenshot shows the 'Document Submission' interface. At the top, there's a navigation bar with links: Group Registration, Topic Registration, Document Submission, Project Forum, and Notices. On the right, it says 'STUDENT STD006 STD006' with a user icon. The main content area has a title 'Document Submission'. Below it, on the right, is a file name 'SE3020\_Lab 1.docx' and a 'Download' button. On the left, it shows 'Submission Status: No Attempt' and 'Due date: 31/05/2022'. Under the 'Attachment' section, there are three buttons: '+ Choose' (highlighted with a red box), 'Upload', and 'Cancel'. At the bottom right, there is an 'Add Submission' button.

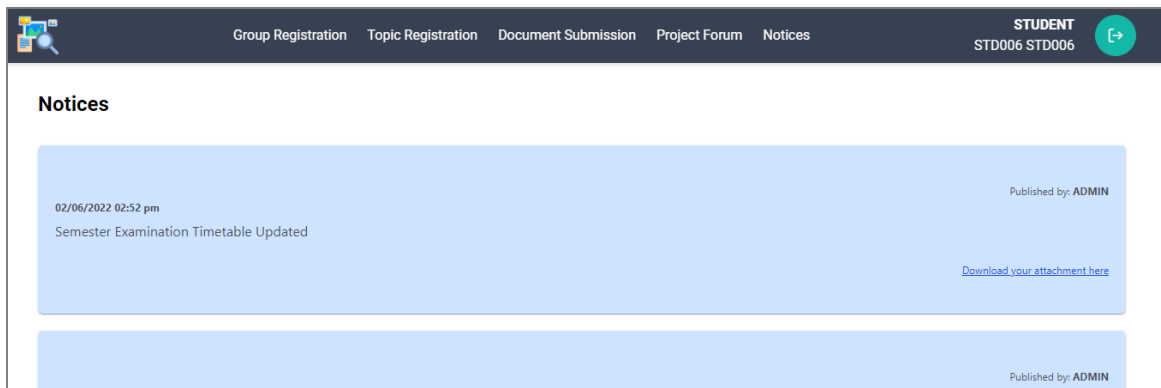
- After adding the document first, you need to click the upload button to upload the document then when clicking the ‘Add Submission’ button the relevant document will get uploaded to the cloud storage.

This screenshot shows the document upload process. The 'Attachment' section now displays a file: 'SE3040-2021-S1-Assesment02-Group-Project.pdf' with a size of '228.259 KB'. A red box labeled '1' is around the 'Upload' button. Another red box labeled '2' is around the 'Add Submission' button at the bottom right. The 'Submission Status' remains 'No Attempt' and the 'Due date' is '31/05/2022'.

- This will display after the document submission is a success. By clicking the “View Submission” can view the uploaded document

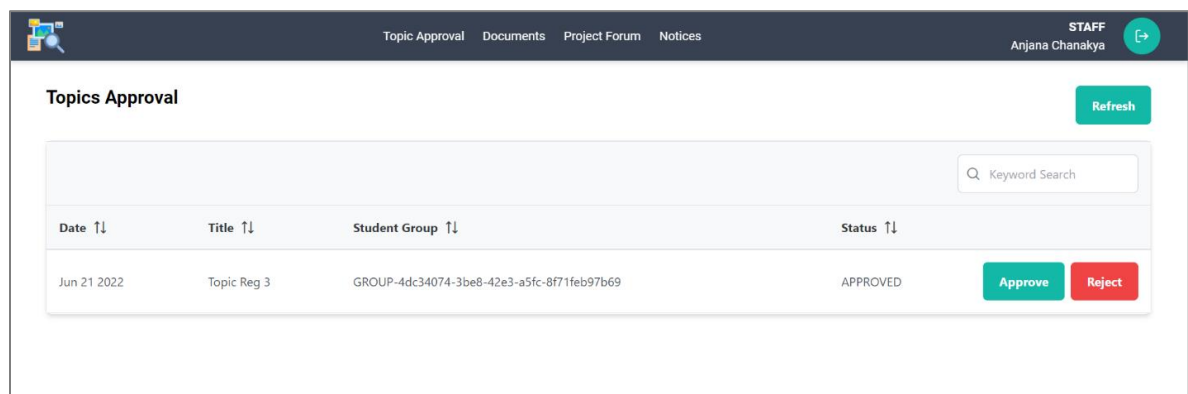
The screenshot shows the successful submission status. The 'Submission Status' is now 'Submitted'. The 'Due date' remains '31/05/2022'. Under 'Submitted file:', there is a blue link 'View submission' which is pointed to by a red arrow. The navigation bar and student information at the top are the same as in previous screenshots.

- These are the notices that are published by the staff and the admin. We can see the notices and can download the relevant documentation.



### 2.3. IT20018832

- Topic Approval – After a supervisor or Co-Supervisor logs in using credentials, can view all the approval requests that student groups that were assigned sent. The supervisor/Co-Supervisor can sort the requests and search with the search box. And then after reviewing the topics Supervisor/Co-Supervisor can either approve or reject the Topics.



- Before Approval



- After Approval



- After Rejected

Date ↑↓	Title ↑↓	Student Group ↑↓	Status ↑↓	
Jun 21 2022	Topic Reg 3	GROUP-4dc34074-3be8-42e3-a5fc-8f71feb97b69	REJECTED	<a href="#">Approve</a> <a href="#">Reject</a>

- Document Download – Then Supervisor/Co-Supervisor can download the research document submission done by the assigned student group. The documents will be downloaded to the local machine after clicking the download button. Supervisor/Co-Supervisor can search specific document by the search box. And there is a refresh button for refresh the submission log.

Topic Approval Documents Project Forum Notices

STAFF  
Anjana Chanakya

Document Download

Refresh

Date ↑↓	Title ↑↓	Student Group ↑↓	
Jun 21 2022	Topic Reg 3	GROUP-4dc34074-3be8-42e3-a5fc-8f71feb97b69	<a href="#">Download</a>

- Project Forum -Staff - Then Supervisor/Co-Supervisor can view all the discussion threads sent by assigned student groups.

Topic Approval Documents Project Forum Notices

STAFF  
Anjana Chanakya

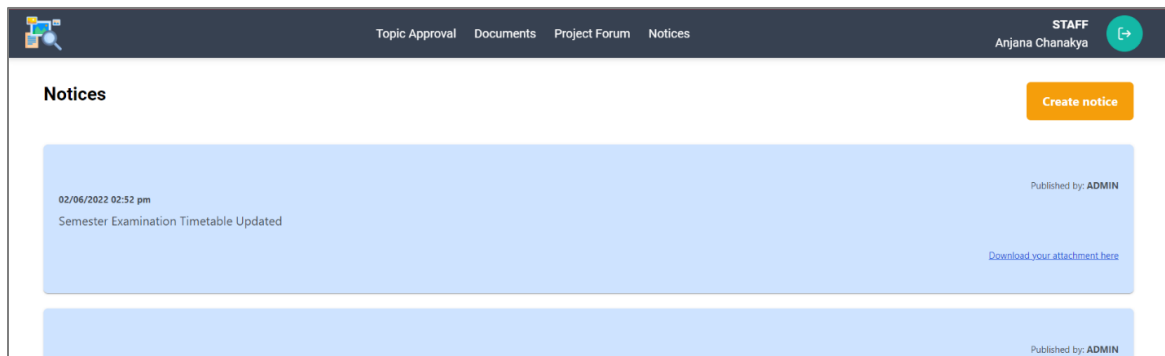
Project Forum

Group Id ↑↓

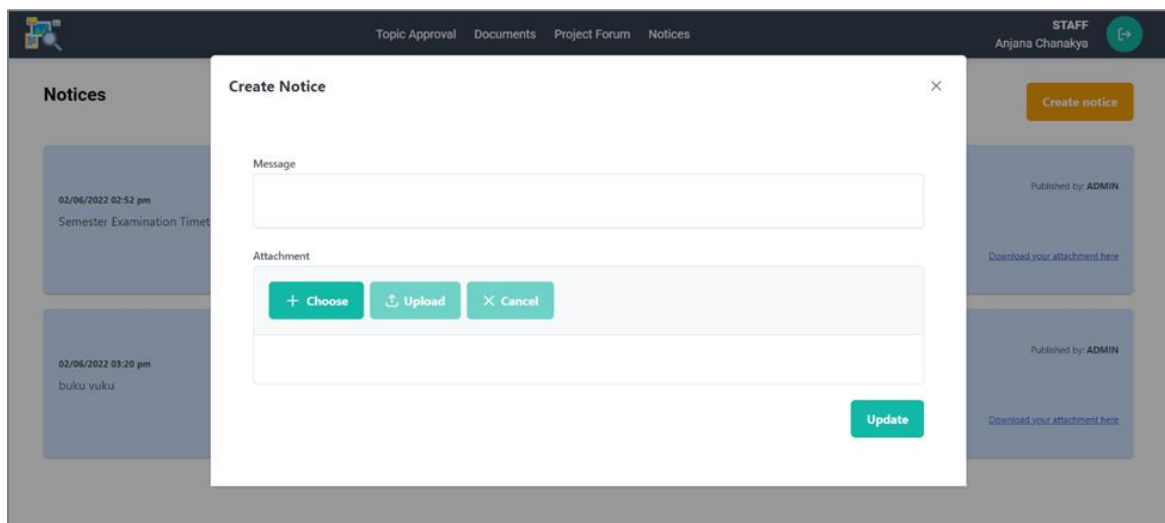
GROUP-4dc34074-3be8-42e3-a5fc-8f71feb97b69

[View](#)

- Notice Interface -Staff - After topic evaluation, approving, and referring documents, Supervisor/Co-Supervisor can create notices for the assigned student groups. Viva schedules, Additional documents, and special notices can be published in the notice section.

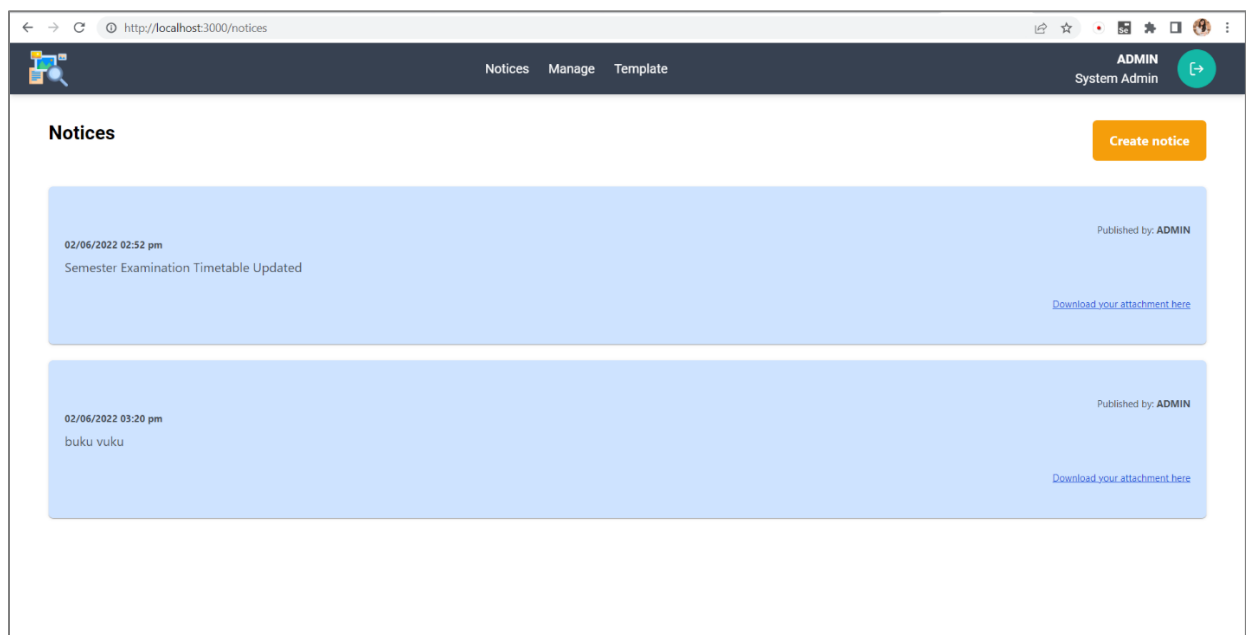
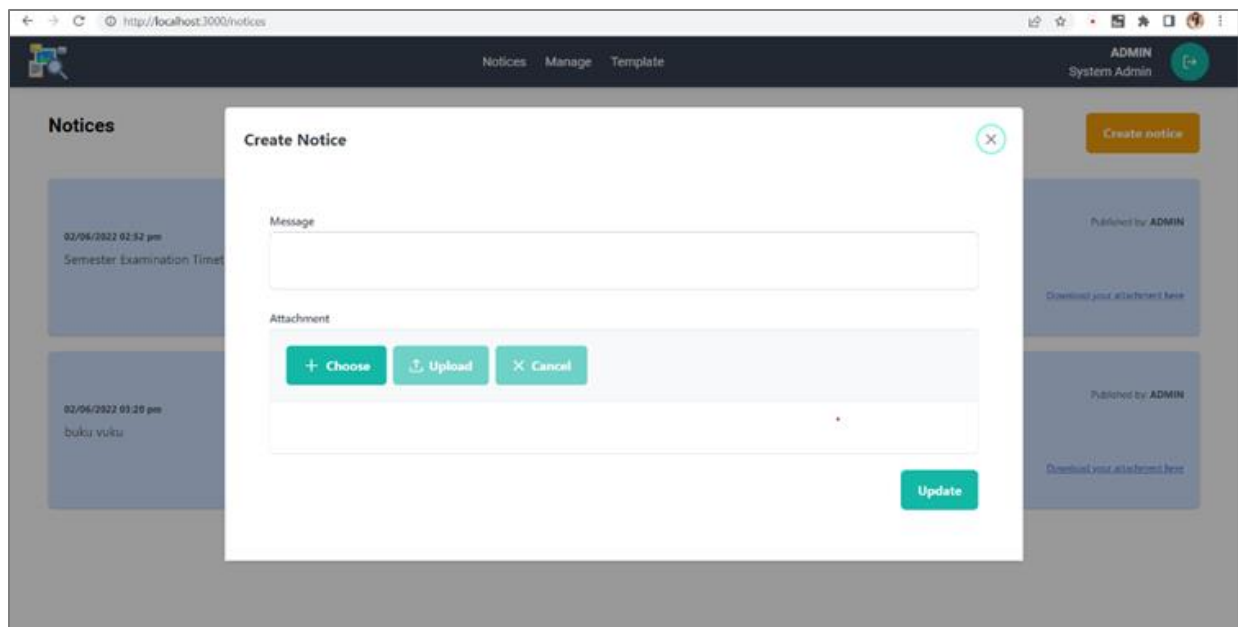


- After clicking Create, the notice creation interface pops up and Supervisor/Co-Supervisor can write text and upload important documents in a notice.

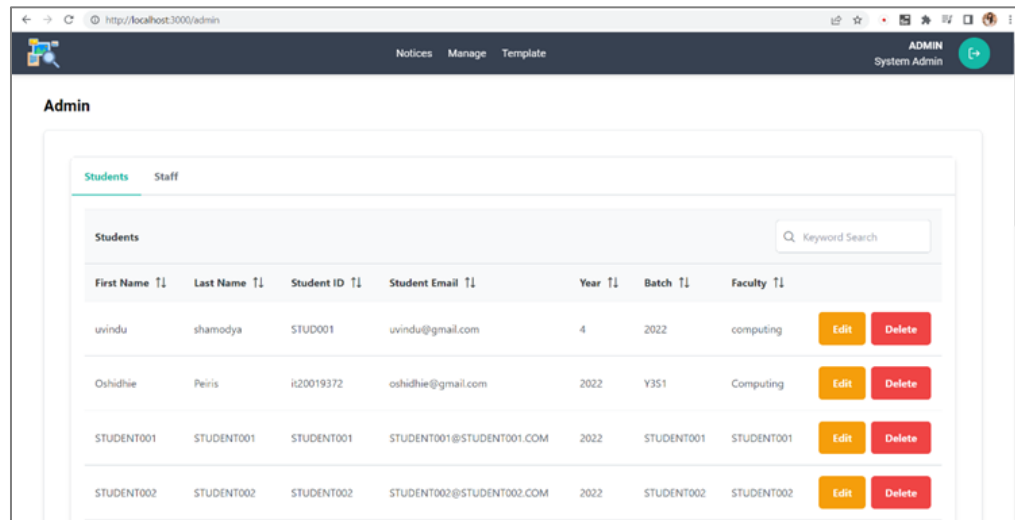


## 2.4. IT20003678

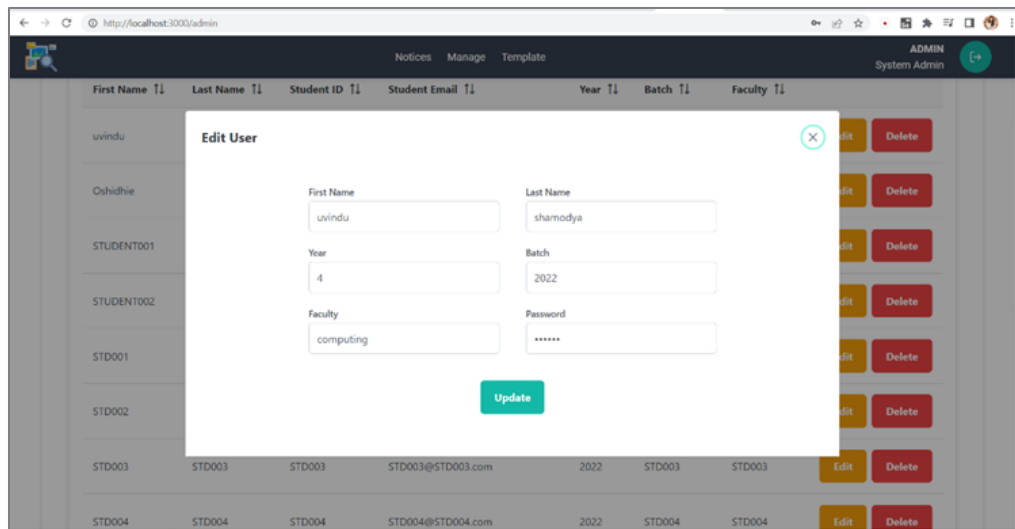
- Publish notices



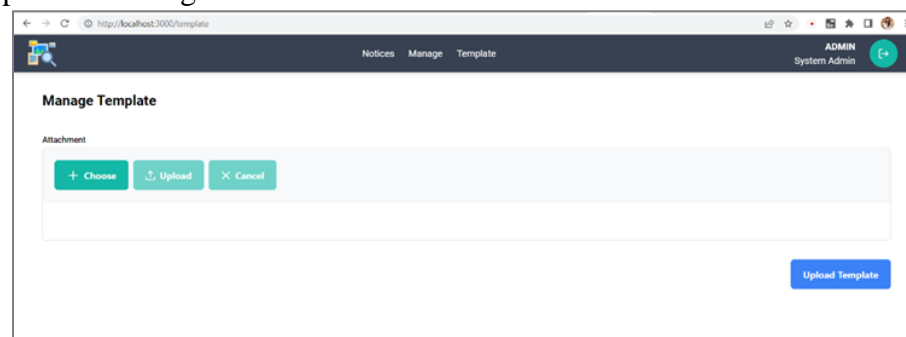
- Manage Users



- Update Students/ Staff Update

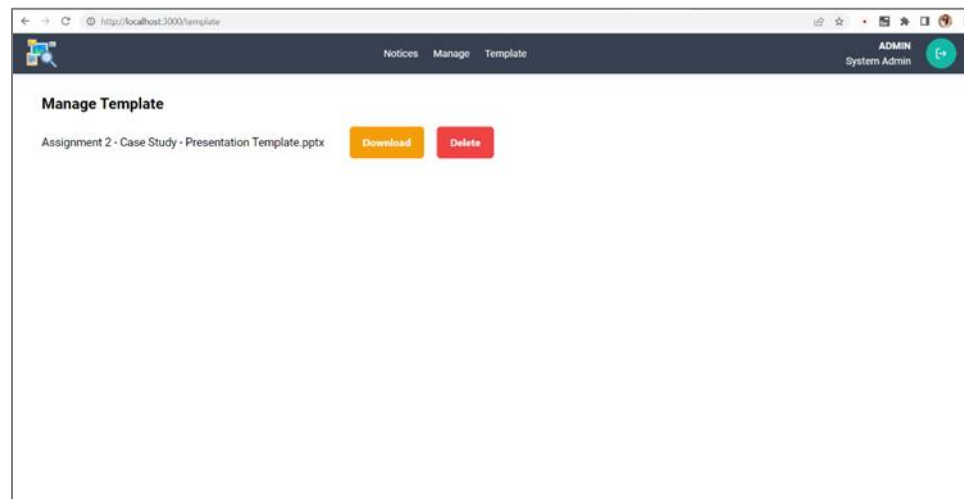


- Upload Templates/Marking Schemes





- Uploaded Templates and Marking Schemes with delete and download options.



- Adding messages and documents to the Chat thread.

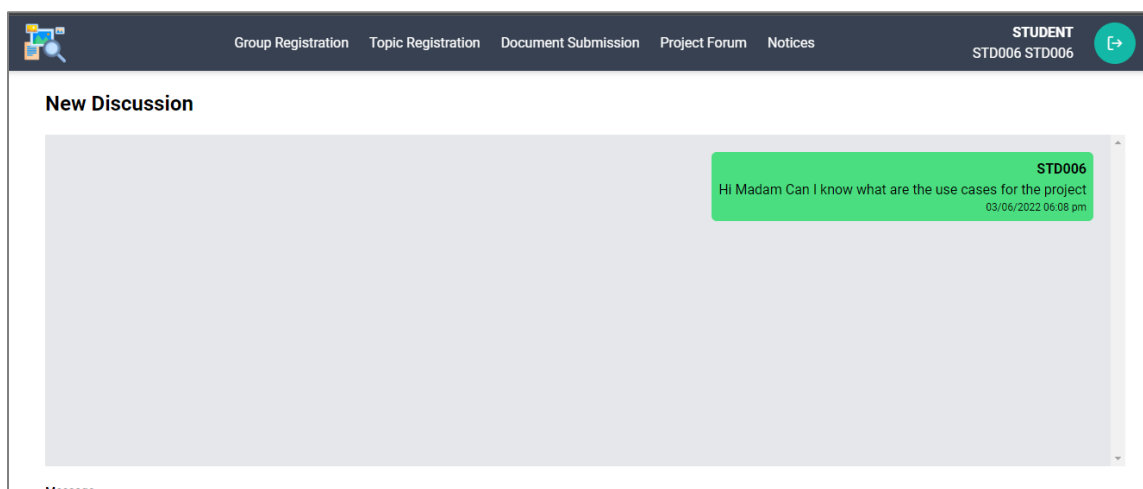
Message

Attachment

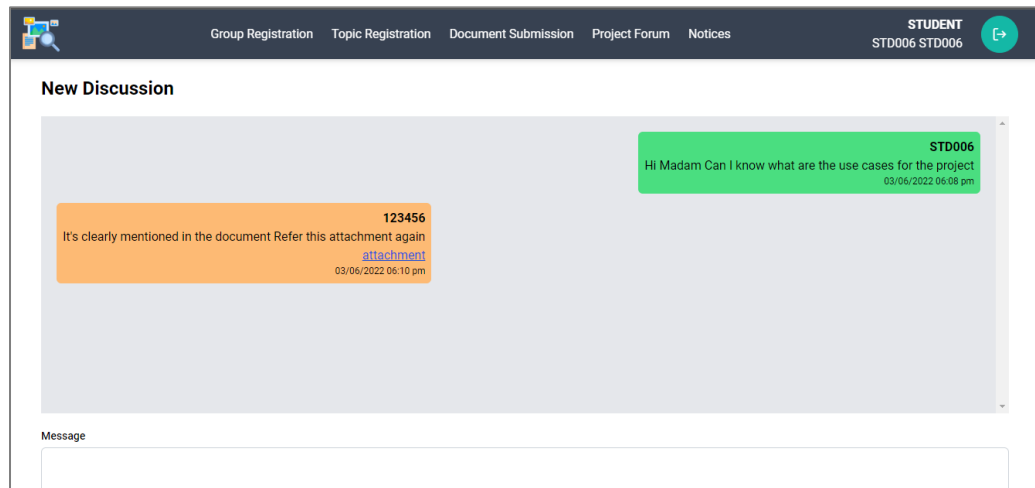
+ Choose Upload Cancel

Back Submit

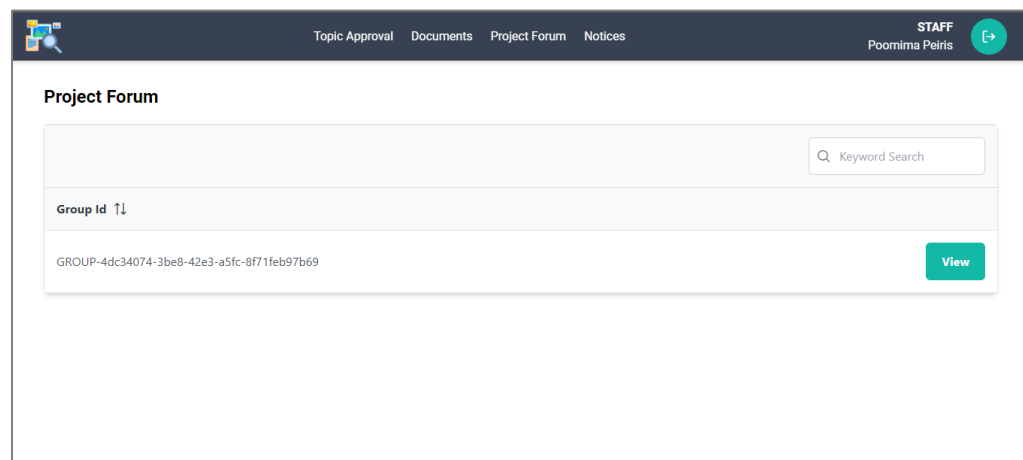
- Chat Thread (Student Side)



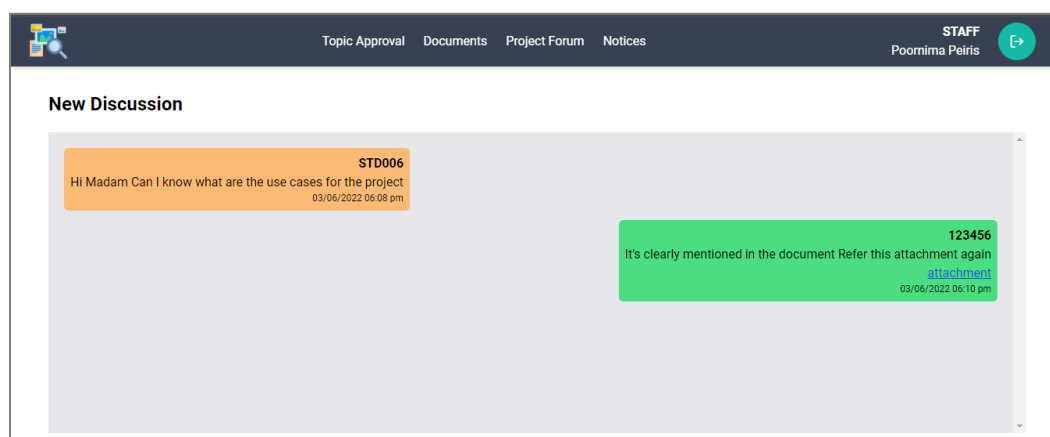
- Supervisor replied to student side chat thread



- Chat thread for Supervisor/co-supervisor side
- By clicking this view button, we can see the relevant groups chat thread for Supervisor/co-supervisor side



- This is the forum where the student's message is displayed, and the staff member can also chat with the student.



### 3. Screenshots/ Code Snippets of the Backend

#### 3.1. Routes

##### 3.1.1. Index.js

```
const express = require("express");
const router = express.Router();

const userController = require("../controllers/user.controller");
const groupController = require("../controllers/group.controller");
const topicController = require("../controllers/topic.controller");
const discussionController = require("../controllers/discussion.controller");
const submissionController = require("../controllers/submission.controller");
const noticeController = require("../controllers/notice.controller");
const panelController = require("../controllers/panel.controller");
const templateController = require("../controllers/template.controller");

router.get("/", (req, res) => {
  res.send("server is up and running");
});

router.route("/login").post(userController.login);

router.route("/createUser").post(userController.createUser);
router.route("/users").get(userController.getUsers);
router.route("/userById").post(userController.getUserById);
router.route("/updateUser").post(userController.updateUser);
router.route("/deleteUser").post(userController.deleteUser);

router.route("/createGroup").post(groupController.createGroup);
router.route("/groups").get(groupController.getGroups);
router.route("/groupById").post(groupController.getGroupById);
router.route("/updateGroup").post(groupController.updateGroup);
router.route("/deleteGroup").post(groupController.deleteGroup);

router.route("/createTopic").post(topicController.createTopic);
router.route("/topics").get(topicController.getTopics);
router.route("/approveTopic").post(topicController.approveTopic);

router.route("/createSubmission").post(submissionController.createSubmission);
router.route("/submissions").get(submissionController.getSubmissions);

router.route("/createChat").post(discussionController.createChat);
router.route("/chats").get(discussionController.getChats);
router.route("/chatsByGroupId").post(discussionController.getChatByGroupId);
```

```

router.route("/createNotice").post(noticeController.createNotice);
router.route("/notices").get(noticeController.getNotices);

router.route("/createPanel").post(panelController.createPanel);
router.route("/panels").get(panelController.getPanels);
router.route("/updatePanel").post(panelController.updatePanel);

router.route("/createTemplate").post(templateController.createTemplate);
router.route("/templates").get(templateController.getTemplate);
router.route("/deleteTemplate").post(templateController.deleteTemplate);

module.exports = router;

```

## 3.2. Controllers

### 3.2.1. IT20025076

- User.controller.js

```

const { User } = require("../models/user.model");

const createUser = async (req, res) => {
  User.init()
    .then(async () => {
      const user = new User();

      user.id = req.body.id;
      user.fname = req.body.fname;
      user.lname = req.body.lname;
      user.email = req.body.email;
      user.year = req.body.year;
      user.batch = req.body.batch;
      user.faculty = req.body.faculty;
      user.password = req.body.password;
      user.role = req.body.role;
      user.activated = true;

      const response = await user.save();
      return res.json({ data: response });
    })
    .catch((err) => {
      return res.json(err.message);
    });
};

```

```

});

const getUsers = async (req, res) => {
  User.find({}, function (err, users) {
    if (err) return res.json(err);
    return res.json(users);
  });
};

const getUserById = async (req, res) => {
  const userId = req.body.id;
  User.findOne({ id: userId }, function (err, users) {
    if (err) return res.json(err);
    return res.json(users);
  });
};

const updateUser = async (req, res) => {
  User.updateOne(
    { id: req.body.id },
    {
      $set: {
        id: req.body.id,
        fname: req.body.fname,
        lname: req.body.lname,
        email: req.body.email,
        year: req.body.year,
        batch: req.body.batch,
        faculty: req.body.faculty,
        password: req.body.password,
        role: req.body.role,
        activated: req.body.activated,
      },
    },
    function (err, resp) {
      if (err) return res.json(err);
      return res.json(resp);
    }
  );
};

const deleteUser = async (req, res) => {
  User.deleteOne({ id: req.body.id }, function (err, resp) {
    if (err) return res.json(err);
    return res.json(resp);
  });
};

```

```

});

const login = async (req, res) => {
  User.findOne({ email: req.body.email }, function (err, resp) {
    if (err) return res.json(err);
    if (resp && req.body.password === resp.password && resp.activated) {
      res.json(resp);
    } else {
      res.json({ exists: false });
    }
  });
});

module.exports = {
  createUser,
  getUsers,
  getUserById,
  updateUser,
  deleteUser,
  login,
};

```

- **Panel.controller.js**

```

const { v4: uuidv4 } = require("uuid");
const { Panel } = require("../models/panel.model");

const createPanel = async (req, res) => {
  Panel.init()
    .then(async () => {
      const panel = new Panel();

      panel.id = "PANEL-" + uuidv4();
      panel.name = req.body.name;
      panel.panelist = req.body.panelist;

      const response = await panel.save();
      return res.json({ data: response });
    })
    .catch((err) => {
      return res.json(err.message);
    });
});

```

```

const getPanels = async (req, res) => {
  Panel.find({}, function (err, panels) {
    if (err) return res.json(err);
    return res.json(panels);
  });
};

const updatePanel = async (req, res) => {
  Panel.updateOne(
    { id: req.body.id },
    {
      $push: {
        panelist: req.body.staffId,
      },
    },
    function (err, response) {
      if (err) return res.json(err);
      Panel.findOne({ id: req.body.id }, function (err, panel) {
        return res.json(panel);
      });
    }
  );
};

module.exports = {
  createPanel,
  getPanels,
  updatePanel,
};

```

### 3.2.2. IT20019372

- **group.controller.js**

```

const { v4: uuidv4 } = require("uuid");
const { Group } = require("../models/group.model");
const { User } = require("../models/user.model");

const createGroup = async (req, res) => {
  Group.init()
    .then(async () => {
      const group = new Group();

      group.id = "GROUP-" + uuidv4();

```

```

    group.groupLeader = req.body.groupLeader;
    group.secondMember = req.body.secondMember;
    group.thirdMember = req.body.thirdMember;
    group.fourthMember = req.body.fourthMember;

    const response = await group.save();

    const studentArray = [
      "groupLeader",
      "secondMember",
      "thirdMember",
      "fourthMember",
    ];

    studentArray.forEach((member) => {
      User.updateOne(
        { id: req.body[member] },
        {
          $set: {
            studentGroup: group,
          },
        },
        function (err, resp) {
          if (err) return res.json(err);
        }
      );
    });

    return res.json({ data: response });
  })
  .catch((err) => {
    return res.json(err.message);
  });
});

const getGroups = async (req, res) => {
  Group.find({}, function (err, groups) {
    if (err) return res.json(err);
    return res.json(groups);
  });
};

const getGroupById = async (req, res) => {
  const groupId = req.body.id;
  Group.findOne({ id: groupId }, function (err, groups) {
    if (err) return res.json(err);
  });
};

```



```

        return res.json(groups);
    });
};

const updateGroup = async (req, res) => {
    Group.updateOne(
        { id: req.body.id },
        {
            $set: {
                id: req.body.id,
                panelId: req.body.panelId,
                groupLeader: req.body.groupLeader,
                secondMember: req.body.secondMember,
                thirdMember: req.body.thirdMember,
                fourthMember: req.body.fourthMember,
                supervisor: req.body.supervisor,
                coSupervisor: req.body.coSupervisor,
            },
        },
        function (err, resp) {
            if (err) return res.json(err);
            return res.json(resp);
        }
    );
};

const deleteGroup = async (req, res) => {
    Group.deleteOne({ id: req.body.id }, function (err, resp) {
        if (err) return res.json(err);
        return res.json(resp);
    });
};

module.exports = {
    createGroup,
    getGroups,
    getGroupById,
    updateGroup,
    deleteGroup,
};

```

- submission.controller.js

```
const { v4: uuidv4 } = require("uuid");
```

```

const { Submission } = require("../models/submission.model");
const { User } = require("../models/user.model");
const { Group } = require("../models/group.model");

const createSubmission = async (req, res) => {
  Submission.init()
    .then(async () => {
      const submission = new Submission();

      submission.id = "SUBMISSION-" + uuidv4();
      submission.fileHash = req.body.fileHash;

      const response = await submission.save();

      Group.updateOne(
        { id: req.body.studentGroupId },
        {
          $set: {
            submission: submission,
          },
        },
        function (err, groupResp) {
          if (err) return res.json(err);

          Group.findOne({ id: req.body.studentGroupId }, function (err, group) {
            if (err) return res.json(err);

            const studentArray = [
              group.groupLeader,
              group.secondMember,
              group.thirdMember,
              group.fourthMember,
            ];
            group.submission = submission;

            studentArray.forEach((member) => {
              User.updateOne(
                { id: member },
                {
                  $set: {
                    studentGroup: group,
                  },
                },
                function (err, studentResp) {
                  if (err) return res.json(err);
                }
              );
            });
          });
        }
      );
    })
    .catch((err) => {
      res.json(err);
    });
};

```

```

    }
  );
});

const supervisorArray = [group.supervisor, group.coSupervisor];

supervisorArray.forEach((staff) => {
  User.findOne({ id: staff }, function (err, staffResponse) {
    if (staffResponse.staffGroups.length > 0) {
      staffResponse.staffGroups.forEach((element) => {
        if (element?.id === req.body.studentGroupId) {
          User.updateOne(
            { id: staff },
            {
              $set: {
                staffGroups: group,
              },
            },
            function (err, staffResp) {
              if (err) return res.json(err);
            }
          );
        } else {
          User.updateOne(
            { id: staff },
            {
              $push: {
                staffGroups: group,
              },
            },
            function (err, staffResp) {
              if (err) return res.json(err);
            }
          );
        }
      });
    } else {
      User.updateOne(
        { id: staff },
        {
          $set: {
            staffGroups: group,
          },
        },
        function (err, staffResp) {
          if (err) return res.json(err);
        }
      );
    }
  });
});

```

```

        }
      );
    }
  });
});
}
);

    return res.json({ data: response });
  })
  .catch((err) => {
    return res.json(err.message);
  });
};

const getSubmissions = async (req, res) => {
  Submission.find({}, function (err, submissions) {
    if (err) return res.json(err);
    return res.json(submissions);
  });
};

module.exports = {
  createSubmission,
  getSubmissions,
};

```

### 3.2.3. IT20018832

- topic.Controller.js

```

const getTopics = async (req, res) => {
  Topic.find({}, function (err, topics) {
    if (err) return res.json(err);
    return res.json(topics);
  });
};

const approveTopic = async (req, res) => {
  Topic.init()
    .then(async () => {
      Topic.updateOne(

```

```

{ id: req.body.topicId },
{
  $set: {
    accepted: req.body.accepted,
  },
},
function (err, topicResponse) {
  Topic.findOne({ id: req.body.topicId }, function (err, topic) {
    Group.updateOne(
      { id: topic.studentGroupId },
      {
        $set: {
          topic: topic,
        },
      },
      function (err, groupResp) {
        if (err) return res.json(err);

        Group.findOne(
          { id: topic.studentGroupId },
          function (err, group) {
            if (err) return res.json(err);

            const studentArray = [
              group.groupLeader,
              group.secondMember,
              group.thirdMember,
              group.fourthMember,
            ];

            studentArray.forEach((member) => {
              User.updateOne(
                { id: member },
                {
                  $set: {
                    studentGroup: group,
                  },
                },
                function (err, studentResp) {
                  if (err) return res.json(err);
                }
              );
            });

            const supervisorArray = [
              group.supervisor,

```



```

discussion.userId = req.body.userId;
discussion.message = req.body.message;
discussion.attachment = req.body.attachment;
discussion.groupId = req.body.groupId;
const response = await discussion.save();

// update group object with the discussion object
Group.updateOne(
  { id: req.body.groupId },
  {
    $push: {
      discussion: discussion,
    },
  },
  function (err, groupResp) {
    if (err) return res.json(err);

    Group.findOne({ id: req.body.groupId }, function (err, group) {
      if (err) return res.json(err);

      const studentArray = [
        group.groupLeader,
        group.secondMember,
        group.thirdMember,
        group.fourthMember,
      ];

      studentArray.forEach((member) => {
        User.updateOne(
          { id: member },
          {
            $set: {
              studentGroup: group,
            },
          },
          function (err, studentResp) {
            if (err) return res.json(err);
          }
        );
      });

      const supervisorArray = [group.supervisor, group.coSupervisor];

      supervisorArray.forEach((staff) => {
        User.findOne({ id: staff }, function (err, staffResponse) {

```

```

        if (staffResponse.staffGroups.length > 0) {
            staffResponse.staffGroups.forEach((element) => {
                if (element?.id === req.body.groupId) {
                    User.updateOne(
                        { id: staff },
                        {
                            $set: {
                                staffGroups: group,
                            },
                        },
                        function (err, staffResp) {
                            if (err) return res.json(err);
                        }
                    );
                } else {
                    User.updateOne(
                        { id: staff },
                        {
                            $push: {
                                staffGroups: group,
                            },
                        },
                        function (err, staffResp) {
                            if (err) return res.json(err);
                        }
                    );
                }
            });
        } else {
            User.updateOne(
                { id: staff },
                {
                    $set: {
                        staffGroups: group,
                    },
                },
                function (err, staffResp) {
                    if (err) return res.json(err);
                }
            );
        }
    });
}
});
});
}
);

```



```

        return res.json({ data: response });
    })
    .catch((err) => {
        return res.json(err.message);
    });
};

const getChats = async (req, res) => {
    Discussion.find({}, function (err, chats) {
        if (err) return res.json(err);
        return res.json(chats);
    });
};

const getChatByGroupId = async (req, res) => {
    Group.findOne({ id: req.body.groupId }, function (err, group) {
        if (err) return res.json(err);
        return res.json(group);
    });
};

module.exports = {
    createChat,
    getChats,
    getChatByGroupId,
};

```

- notice.controller.js

```

const { Notice } = require("../models/notice.model");

const createNotice = async (req, res) => {
    Notice.init()
        .then(async () => {
            const notice = new Notice();

            notice.userRole = req.body.userRole;
            notice.message = req.body.message;
            notice.attachment = req.body.attachment;

            const response = await notice.save();
            return res.json({ data: response });
        })
        .catch((err) => {

```

```

        return res.json(err.message);
    });
};

const getNotices = async (req, res) => {
    Notice.find({}, function (err, notices) {
        if (err) return res.json(err);
        return res.json(notices);
    });
};

module.exports = {
    createNotice,
    getNotices,
};

```

- template.controller.js

```

const { v4: uuidv4 } = require("uuid");
const { Template } = require("../models/template.model");

const createTemplate = async (req, res) => {
    Template.init()
        .then(async () => {
            const template = new Template();

            template.id = "TEMPLATE-" + uuidv4();
            template.fileName = req.body.fileName;
            template.fileHash = req.body.fileHash;

            const response = await template.save();
            return res.json({ data: response });
        })
        .catch((err) => {
            return res.json(err.message);
        });
};

const getTemplate = async (req, res) => {
    Template.find({}, function (err, template) {
        if (err) return res.json(err);
        return res.json(template);
    });
};

```

```

const deleteTemplate = async (req, res) => {
  Template.deleteOne({ id: req.body.id }, function (err, resp) {
    if (err) return res.json(err);
    return res.json(resp);
  });
};

module.exports = {
  createTemplate,
  getTemplate,
  deleteTemplate,
};

```

### template.controller.js

```

const { v4: uuidv4 } = require("uuid");
const { Template } = require("../models/template.model");

const createTemplate = async (req, res) => {
  Template.init()
    .then(async () => {
      const template = new Template();

      template.id = "TEMPLATE-" + uuidv4();
      template.fileName = req.body.fileName;
      template.fileHash = req.body.fileHash;

      const response = await template.save();
      return res.json({ data: response });
    })
    .catch((err) => {
      return res.json(err.message);
    });
};

const getTemplate = async (req, res) => {
  Template.find({}, function (err, template) {
    if (err) return res.json(err);
    return res.json(template);
  });
};

const deleteTemplate = async (req, res) => {
  Template.deleteOne({ id: req.body.id }, function (err, resp) {
    if (err) return res.json(err);
    return res.json(resp);
  });
};

```

```

    });
};

module.exports = {
  createTemplate,
  getTemplate,
  deleteTemplate,
};

```

- **user.controller.js (User update and delete)**

```

41
42 const updateUser = async (req, res) => {
43   User.updateOne(
44     { id: req.body.id },
45     {
46       $set: {
47         id: req.body.id,
48         fname: req.body.fname,
49         lname: req.body.lname,
50         email: req.body.email,
51         year: req.body.year,
52         batch: req.body.batch,
53         faculty: req.body.faculty,
54         password: req.body.password,
55         role: req.body.role,
56         activated: req.body.activated,
57       },
58     },
59     function (err, resp) {
60       if (err) return res.json(err);
61       return res.json(resp);
62     }
63   );
64 };
65
66 const deleteUser = async (req, res) => {
67   User.deleteOne({ id: req.body.id }, function (err, resp) {
68     if (err) return res.json(err);
69     return res.json(resp);
70   });
71 };
72

```

### 3.3. Models

#### 3.3.1. IT20025076

- **user.model.js**

```
var mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");
const { groupSchema } = require("../models/group.model");

const userSchema = mongoose.Schema({
  id: {
    type: String,
    unique: true,
  },
  fname: {
    type: String,
  },
  lname: {
    type: String,
  },
  studentGroup: groupSchema,
  staffGroups: [groupSchema],
  email: {
    type: String,
    unique: true,
  },
  year: {
    type: String,
  },
  batch: {
    type: String,
  },
  faculty: {
    type: String,
  },
  password: {
    type: String,
  },
  role: {
    type: String,
  },
  activated: {
    type: Boolean,
```

```

    },
    createdAt: {
      type: Date,
      default: Date.now,
    },
  });

userSchema.plugin(uniqueValidator);

const User = mongoose.model("User", userSchema);
module.exports = { User, userSchema };

```

- **panel.models.js**

```

var mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const panelSchema = mongoose.Schema({
  createdAt: {
    type: Date,
    default: Date.now,
  },
  id: {
    type: String,
  },
  name: {
    type: String,
  },
  panelist: [],
});

panelSchema.plugin(uniqueValidator);

const Panel = mongoose.model("Panel", panelSchema);
module.exports = { Panel, panelSchema };

```

### 3.3.2. IT20019372

- **group.models.js**

```
var mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const { topicSchema } = require("../models/topic.model");
const { discussionSchema } = require("../models/discussion.model");
const { submissionSchema } = require("../models/submission.model");

const groupSchema = mongoose.Schema({
  id: {
    type: String,
  },
  panelId: {
    type: String,
  },
  groupLeader: {
    type: String,
  },
  secondMember: {
    type: String,
  },
  thirdMember: {
    type: String,
  },
  fourthMember: {
    type: String,
  },
  supervisor: {
    type: String,
  },
  coSupervisor: {
    type: String,
  },
  topic: topicSchema,
  discussion: [discussionSchema],
  submission: submissionSchema,
});

groupSchema.plugin(uniqueValidator);

const Group = mongoose.model("Group", groupSchema);
```

```
module.exports = { Group, groupSchema };
```

- **submission.models.js**

```
var mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const submissionSchema = mongoose.Schema({
  id: {
    type: String,
  },
  fileHash: {
    type: String,
  },
});

submissionSchema.plugin(uniqueValidator);

const Submission = mongoose.model("Submission", submissionSchema);
module.exports = { Submission, submissionSchema };
```

- **topic.models.js**

```
var mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const topicSchema = mongoose.Schema({
  id: {
    type: String,
  },
  date: {
    type: String,
  },
  title: {
    type: String,
  },
  studentGroupId: {
    type: String,
  },
  intro: {
    type: String,
  },
  accepted: {
    type: String,
  },
});
```



```
});

topicSchema.plugin(uniqueValidator);

const Topic = mongoose.model("Topic", topicSchema);
module.exports = { Topic, topicSchema };
```

### 3.3.3. IT20018832

- **topic.model.js**

```
var mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const topicSchema = mongoose.Schema({
  id: {
    type: String,
  },
  date: {
    type: String,
  },
  title: {
    type: String,
  },
  studentGroupId: {
    type: String,
  },
  intro: {
    type: String,
  },
  accepted: {
    type: String,
  },
});

topicSchema.plugin(uniqueValidator);

const Topic = mongoose.model("Topic", topicSchema);
module.exports = { Topic, topicSchema };
```

### 3.3.4. IT20003678

- discussion.model.js

```
var mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const discussionSchema = mongoose.Schema({
  createdAt: {
    type: Date,
    default: Date.now,
  },
  userId: {
    type: String,
  },
  message: {
    type: String,
  },
  attachment: {
    type: String,
  },
  groupId: {
    type: String,
  },
});

discussionSchema.plugin(uniqueValidator);

const Discussion = mongoose.model("Discussion", discussionSchema);
module.exports = { Discussion, discussionSchema };
```

- notice.model.js

```
var mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const noticeSchema = mongoose.Schema({
  createdAt: {
    type: Date,
    default: Date.now,
  },
  userRole: {
    type: String,
  },
});
```

```

    message: {
      type: String,
    },
    attachment: {
      type: String,
    },
  });

noticeSchema.plugin(uniqueValidator);

const Notice = mongoose.model("Notice", noticeSchema);
module.exports = { Notice, noticeSchema };

```

- **template.model.js**

```

var mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const templateSchema = mongoose.Schema({
  id: {
    type: String,
  },
  fileName: {
    type: String,
  },
  fileHash: {
    type: String,
  },
});

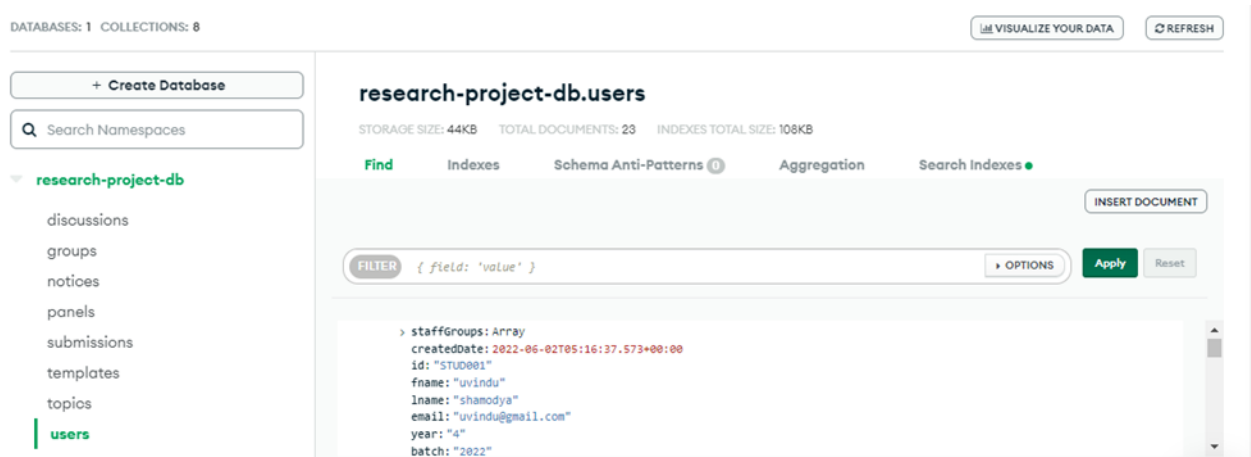
templateSchema.plugin(uniqueValidator);

const Template = mongoose.model("Template", templateSchema);
module.exports = { Template, templateSchema };

```

## 4. Screenshots of the MongoDB Schemas

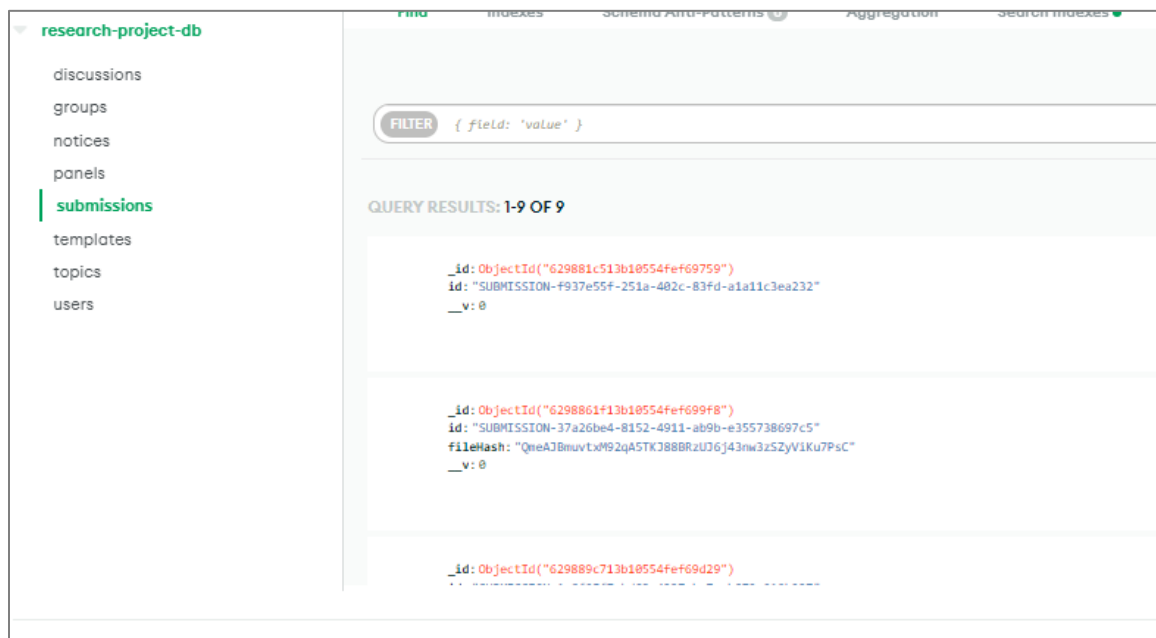
### 4.1. IT20025076



- When new user's signup for the application, an object will be generated in the "users" collection with all the relevant information to uniquely identify the user. This data is being used during the login process to direct each user to the correct interfaces.

### 4.2. IT20019372

- This is the database table for submitting the document for the group project



- This is the database table for topic submission for separate groups

research-project-db

- discussions
- groups
- notices
- panels
- submissions
- templates
- topics**
- users

FILTER { field: 'value' }

QUERY RESULTS: 1-8 OF 8

```

_id: ObjectId("629880b613b10554fef6964d")
id: "TOPIC-e81ab939-be47-419f-8edc-556cbc17413f"
date: "2022-06-20T18:30:00.000Z"
title: "Topic 1"
studentGroupId: "GROUP-a6037511-e58c-44df-a93a-f42c4b6aa1e3"
intro: "sdhaosdohasdhoa"
accepted: "APPROVED"
__v: 0

_id: ObjectId("6298848513b10554fef69960")
id: "TOPIC-ec3b752-c944-4e00-b707-2deee7a3946f"
date: "2022-06-05T18:30:00.000Z"
title: "STD004"
studentGroupId: "GROUP-41372778-c91f-4626-92e8-03baf9304fb9"

```

- This is the database table for group registration by the leader.

research-project-db

- discussions
- groups**
- notices
- panels
- submissions
- templates
- topics
- users

FILTER { field: 'value' }

QUERY RESULTS: 1-5 OF 5

```

_id: ObjectId("62987fec13b10554fef695f8")
discussion: Array
id: "GROUP-a6037511-e58c-44df-a93a-f42c4b6aa1e3"
groupLeader: "STUDENT002"
secondMember: "STUDENT001"
thirdMember: "123456"
fourthMember: "IT20019372"
__v: 0
coSupervisor: "STAFF001"
supervisor: "123456"
topic: Object
submission: Object

_id: ObjectId("6298845613b10554fef69929")

```

System Status: All Good

```

    _id: ObjectId("62987fec13b10554fef695f8")
  > discussion: Array
    id: "GROUP-a6037511-e58c-44df-a93a-f42c4b6aa1e3"
    groupLeader: "STUDENT002"
    secondMember: "STUDENT001"
    thirdMember: "123456"
    fourthMember: "IT20019372"
    __v: 0
    coSupervisor: "STAFF001"
    supervisor: "123456"
  > topic: Object
    id: "TOPIC-e01ab939-be47-419f-8edc-556cbc17413f"
    date: "2022-06-20T18:30:00.000Z"
    title: "Topic 1"
    studentGroupId: "GROUP-a6037511-e58c-44df-a93a-f42c4b6aa1e3"
    intro: "sdhaosdohasdhoa"
    accepted: "APPROVED"
    _id: ObjectId("629880b613b10554fef6964d")
    __v: 0
  > submission: Object
    id: "SUBMISSION-f937e55f-251a-402c-83fd-a1a11c3ea232"
    _id: ObjectId("629881c513b10554fef69759")
    __v: 0

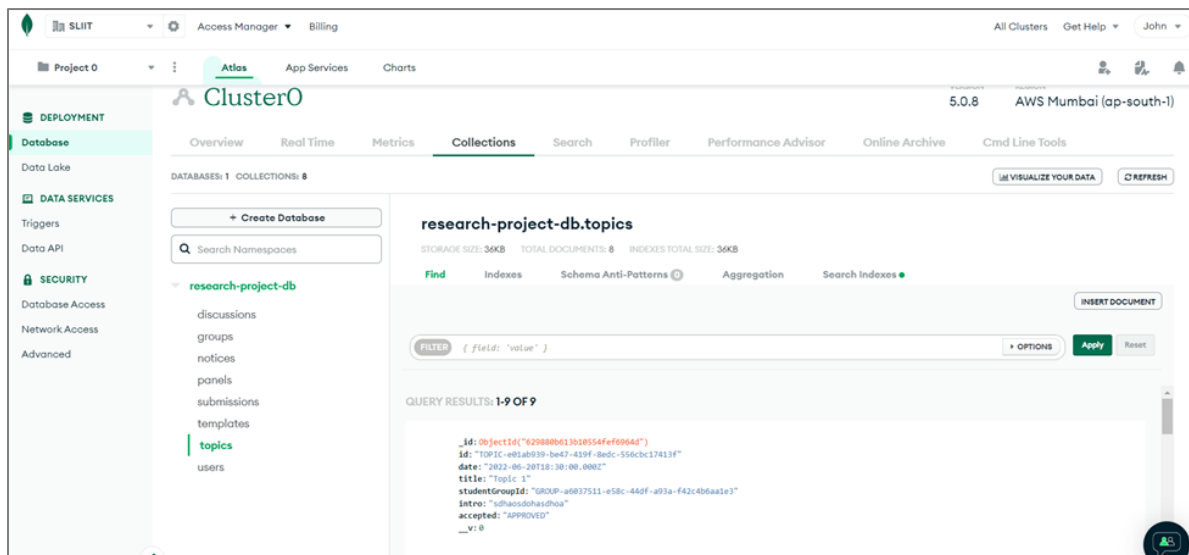
```

This is an example output for divided the code into relevant schemas

(Output: Group Registration table)

### 4.3. IT20018832

- Topic Schema



- Submission download schema

The screenshot shows the MongoDB Atlas interface for a database named 'research-project-db'. The 'submissions' collection is selected in the left sidebar. The main panel displays the 'research-project-db.submissions' collection with a storage size of 36KB and 9 total documents. The 'Find' tab is active, showing a filter bar with the text '{ field: 'value' }'. Below the filter, the query results show 1-9 of 9 documents. Two document snippets are visible, each containing fields: '\_id' (ObjectId), 'id' (string), 'filehash' (string), and '\_\_\_v' (0).

- Notice Schema

The screenshot shows the MongoDB Atlas interface for a database named 'research-project-db'. The 'notices' collection is selected in the left sidebar. The main panel displays the 'research-project-db.notices' collection with a storage size of 36KB and 2 total documents. The 'Find' tab is active, showing a filter bar with the text '{ field: 'value' }'. Below the filter, the query results show 1-2 of 2 documents. Two document snippets are visible, each containing fields: '\_id' (ObjectId), 'createdAt' (timestamp), 'userRole' (string), 'message' (string), and 'attachment' (string).

## 4.4. IT20003678

- Discussion

research-project-db

discussions

groups

notices

panels

submissions

templates

topics

users

Find

Indexes

Schema Anti-Patterns ⓘ

Aggregation

Search Indexes ●

FILTER { field: 'value' }

QUERY RESULTS: 1-4 OF 4

\_id: ObjectId("629a291fece9645ec989746e")

createdAt: 2022-06-03T15:30:39.486+00:00

userId: "STUD004"

message: "Hello Madam!"

groupId: "GROUP-6412a9f9-6083-447e-88e7-c90f1a86f7a1"

\_\_v: 0

\_id: ObjectId("629a292ee27674c7e282605b")

createdAt: 2022-06-03T15:30:54.415+00:00

userId: "STAFF003"

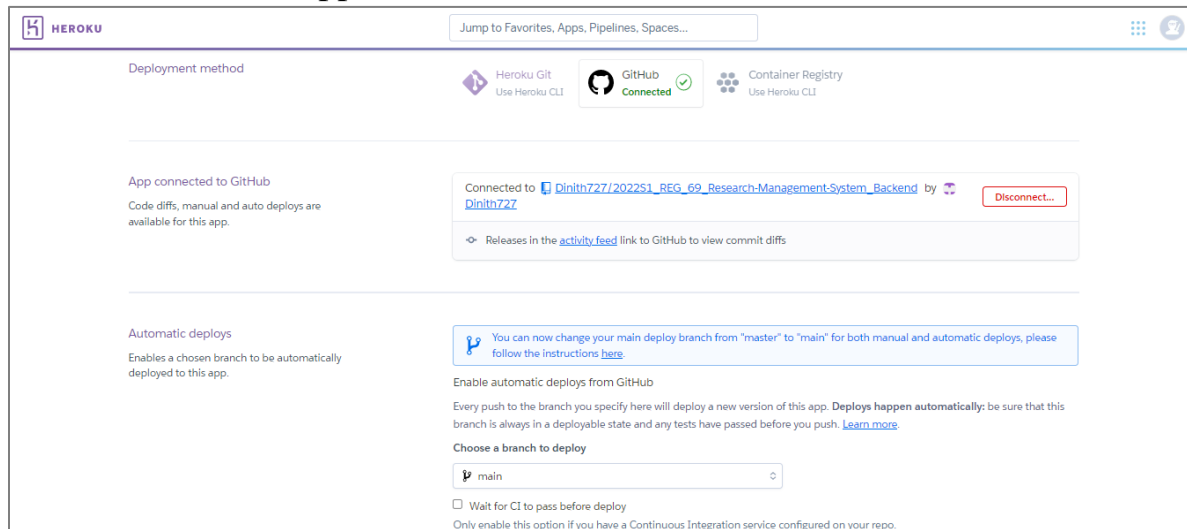
System Status: All Good

48

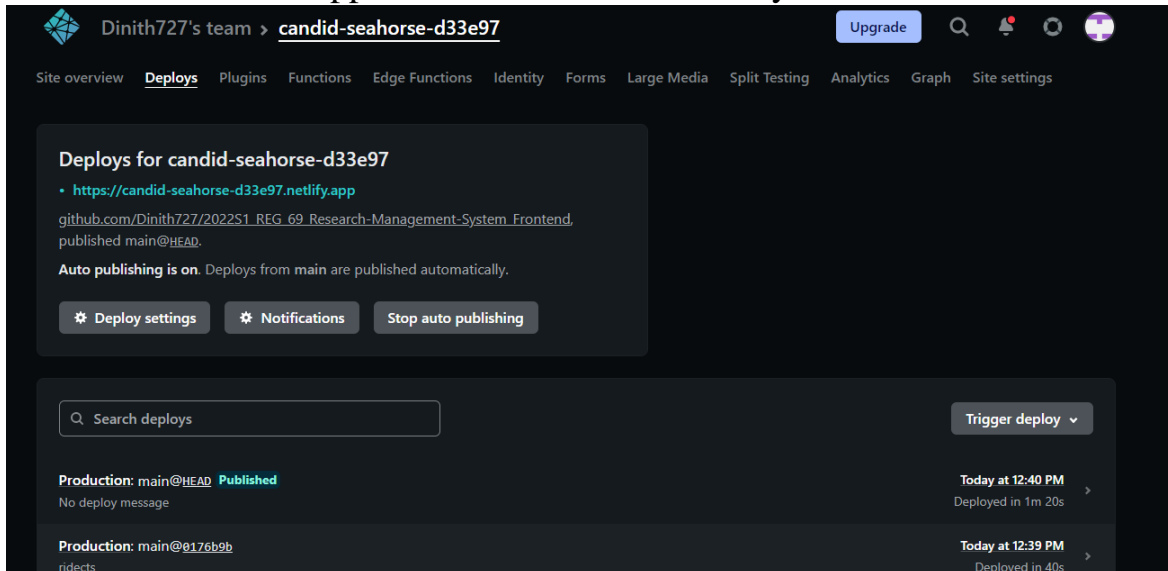


## 5. Proof of Hosting

- The backend of the application is hosted via Heroku.



- The front end of the application is hosted via Netlify.



- By clicking the following link, you can see the project is hosted