Final Project

Web Engineering

Sir Zaheer Sani

Ali Hamza 20i1881

Mahad Rahat 20i1808

Syed Muhammad Mian Abdullah 20i0457

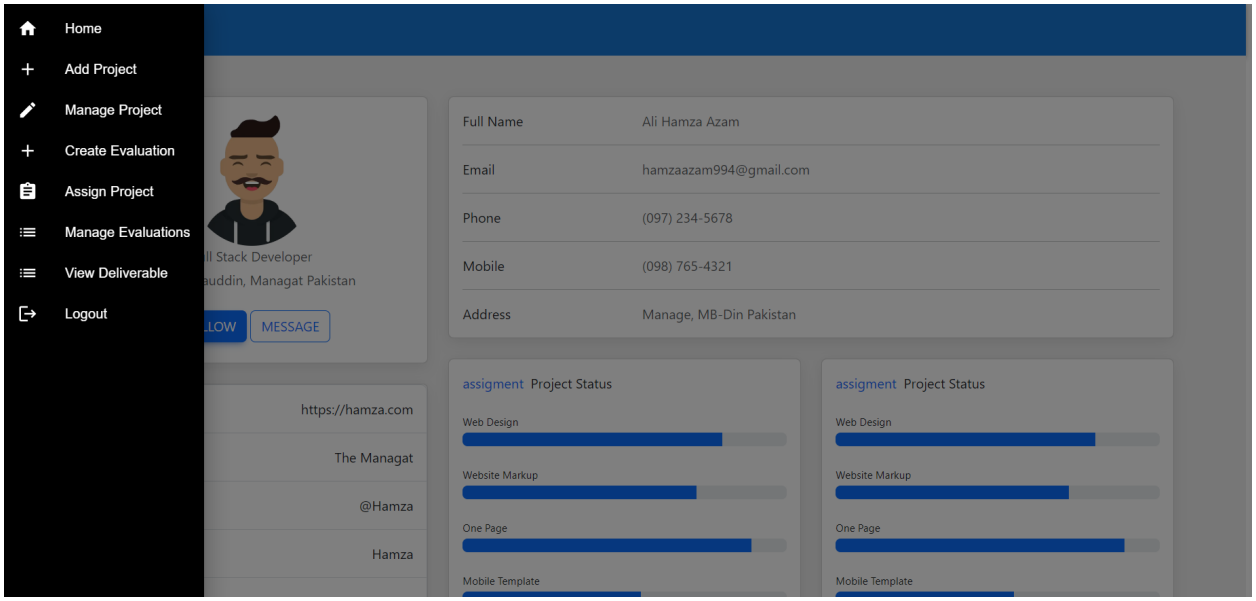
11 June, 2023

Work Division

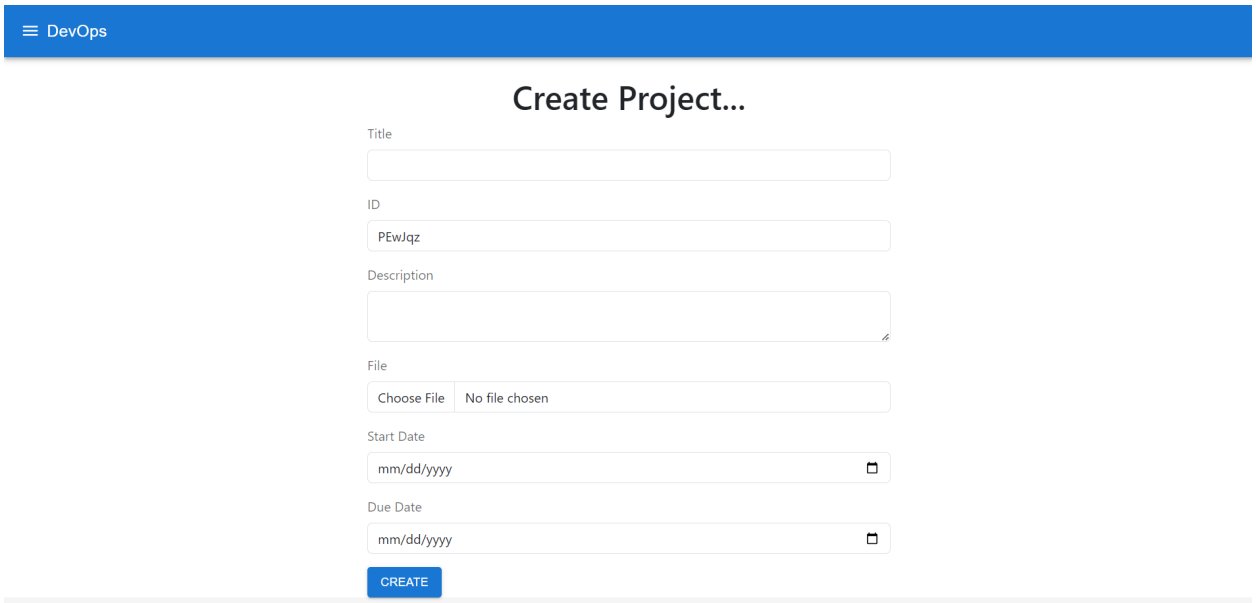
Ali Hamza 20i1881

UI Screen Shots

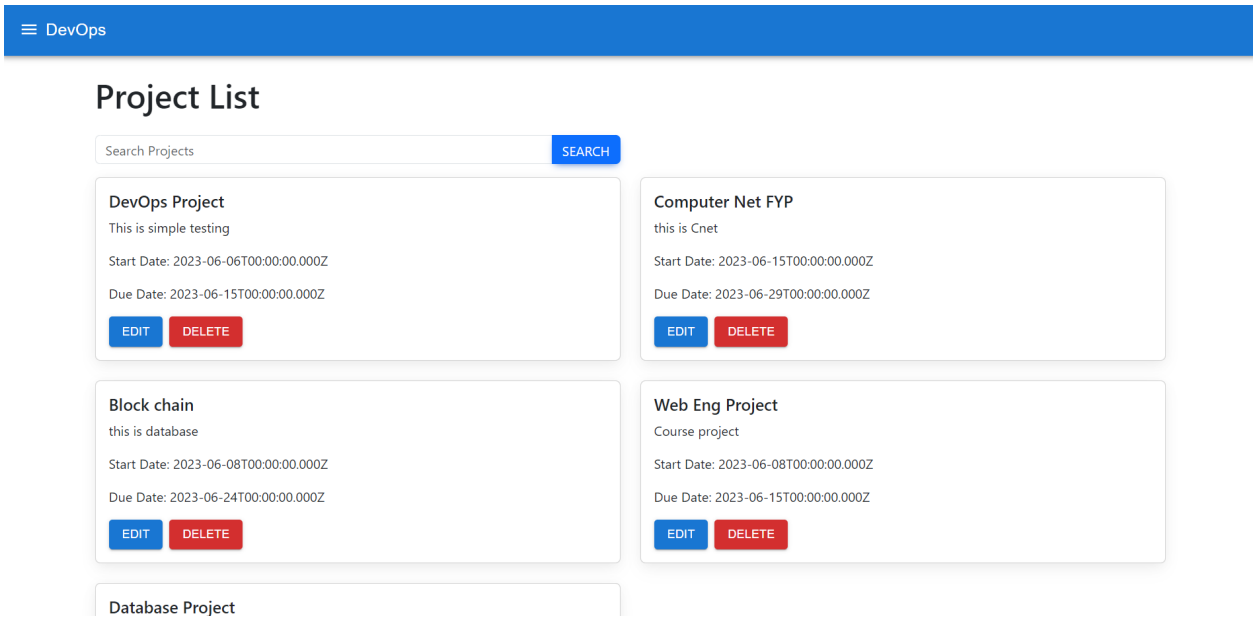
Home Page:



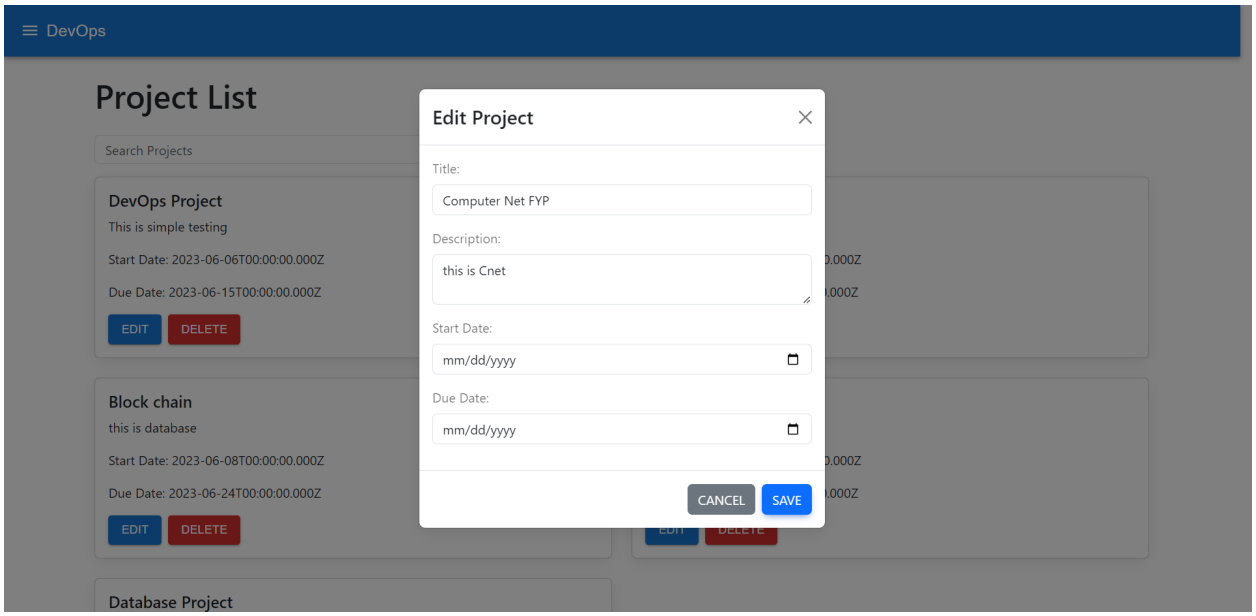
Create Project:



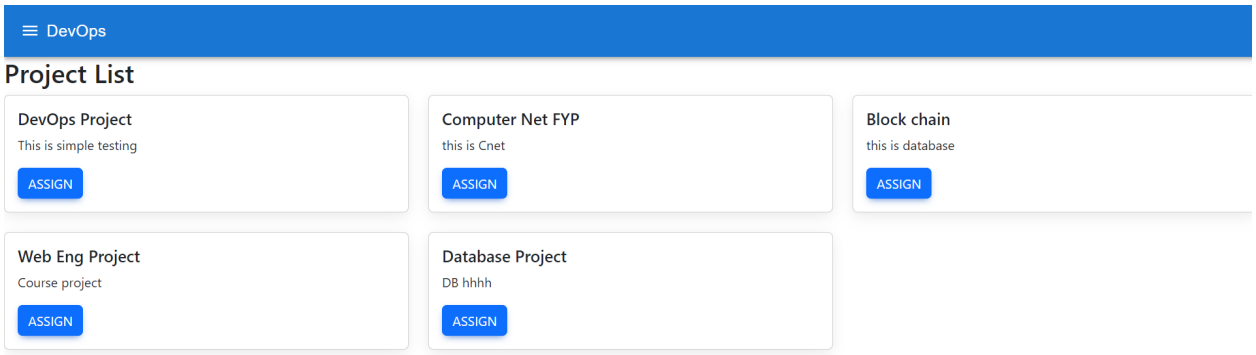
Manage Projects:



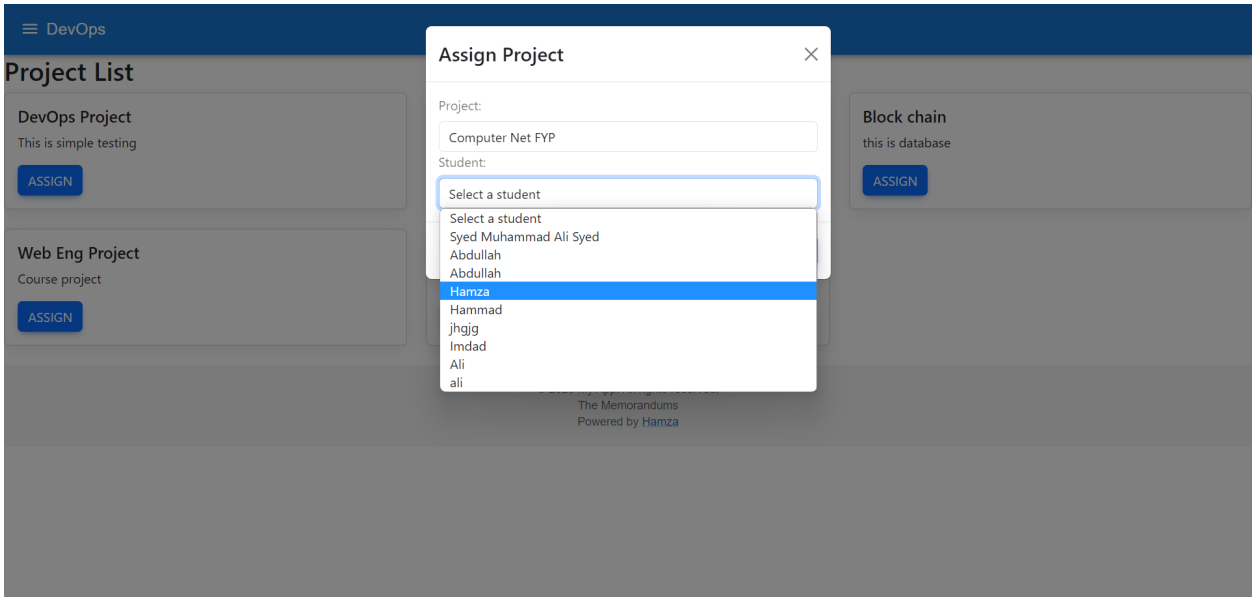
Edit Projects:



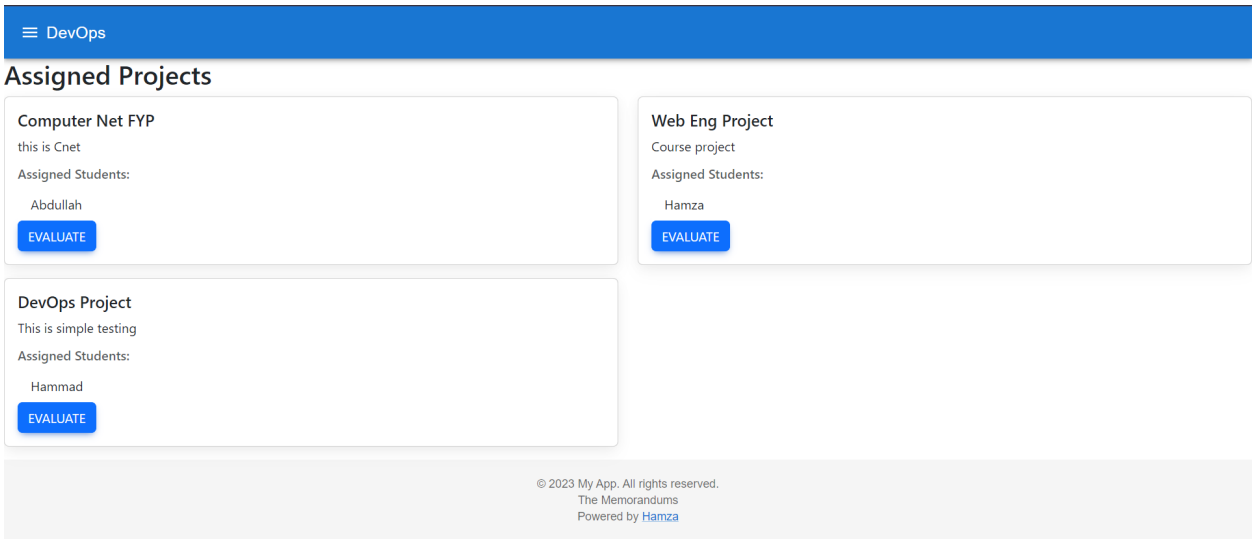
Assign Project:



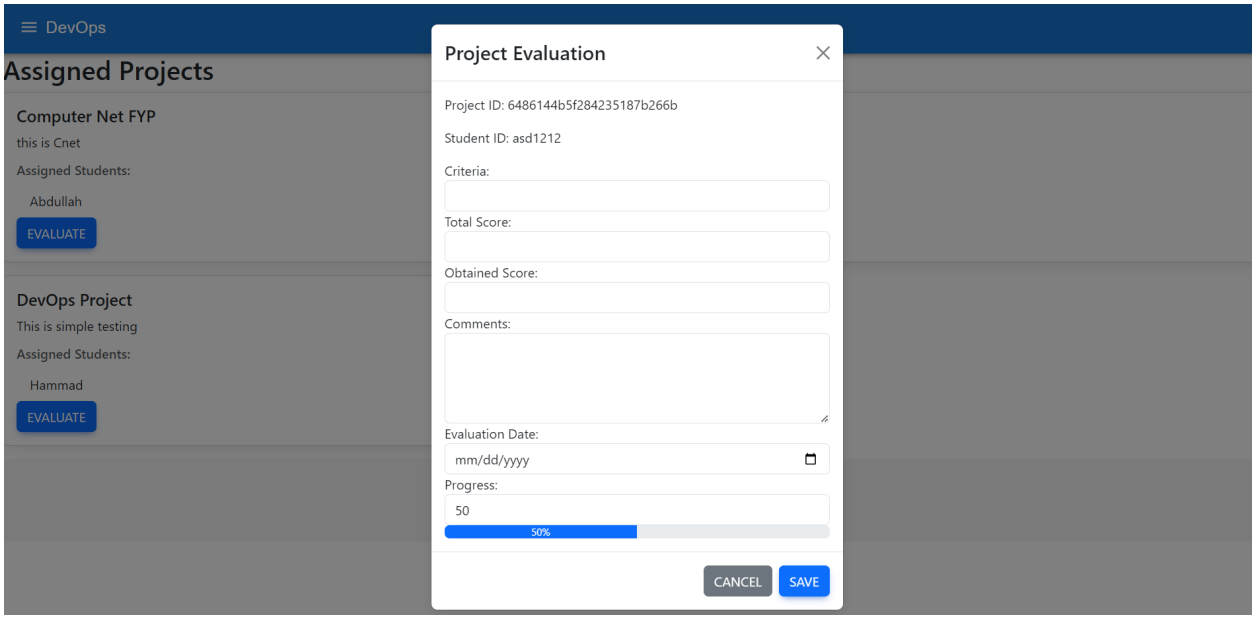
Assign project to student:



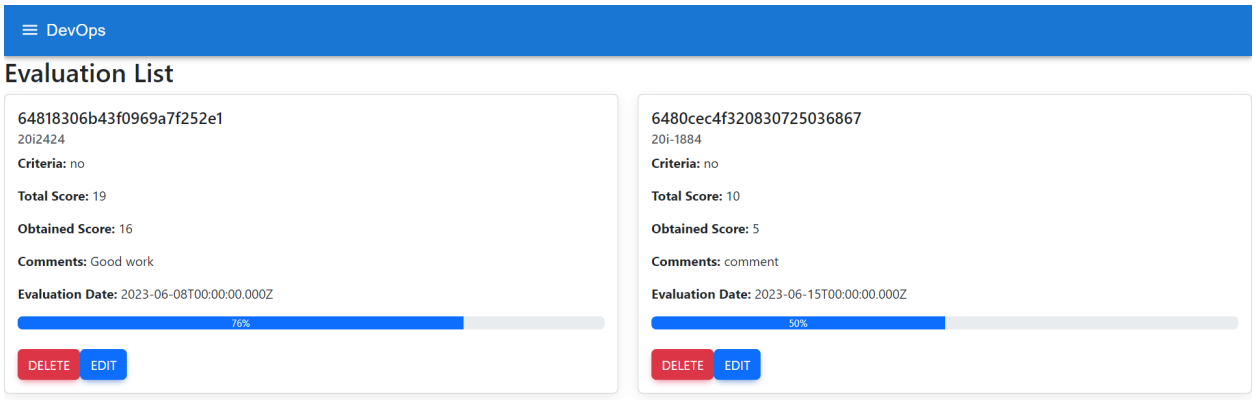
Display Assigned Project:



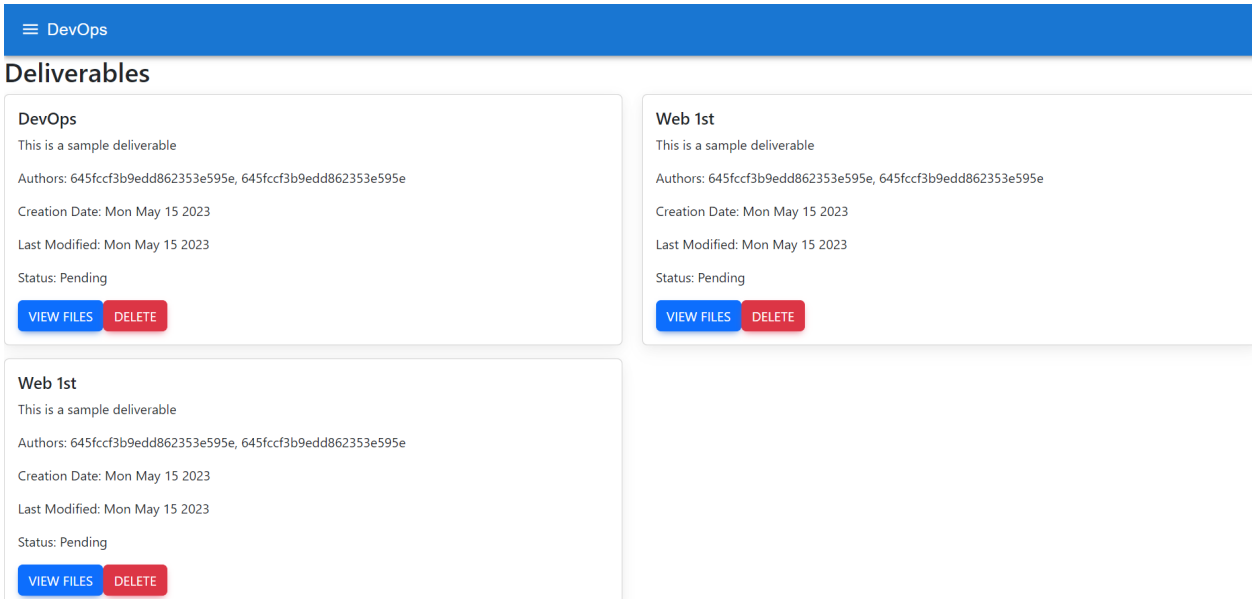
Evaluate Assigned Projects:

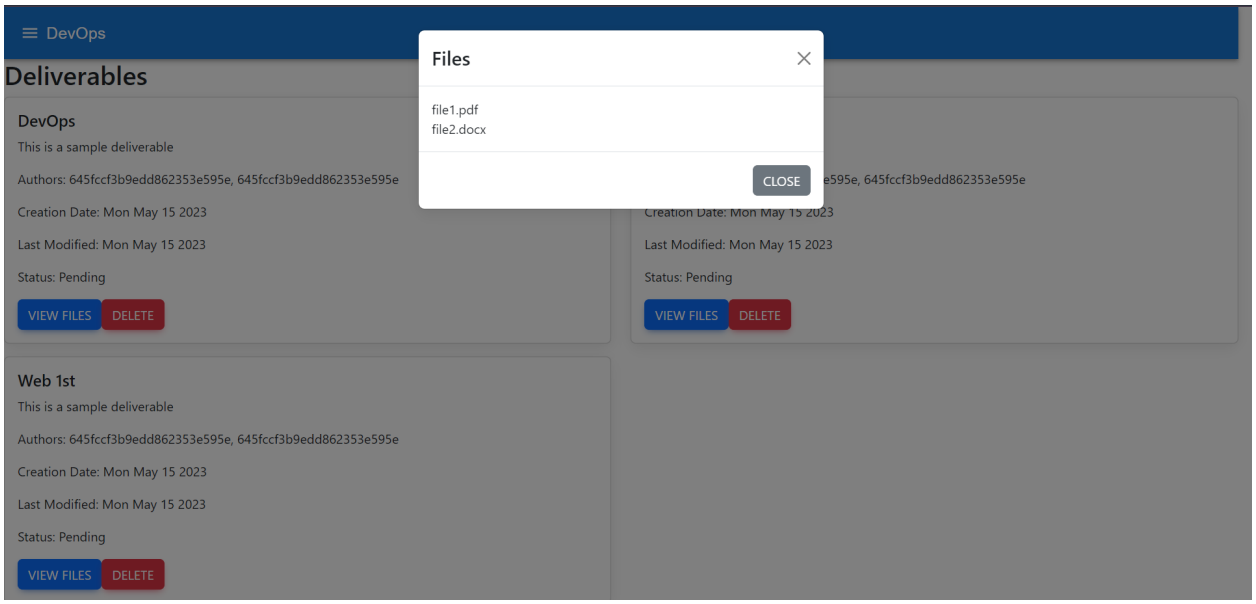


View Evaluations:



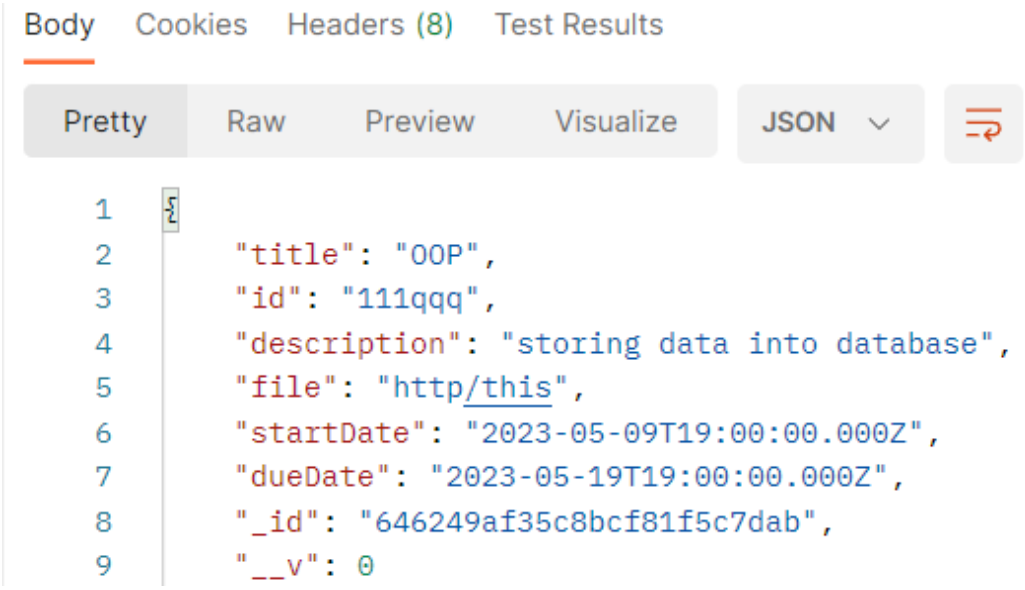
Deliverables view:





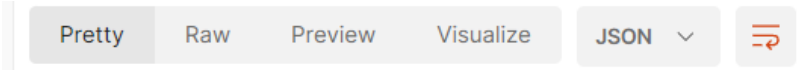
- **Sample request and response of the APIs**

1. http://localhost:3001/projects/createProject



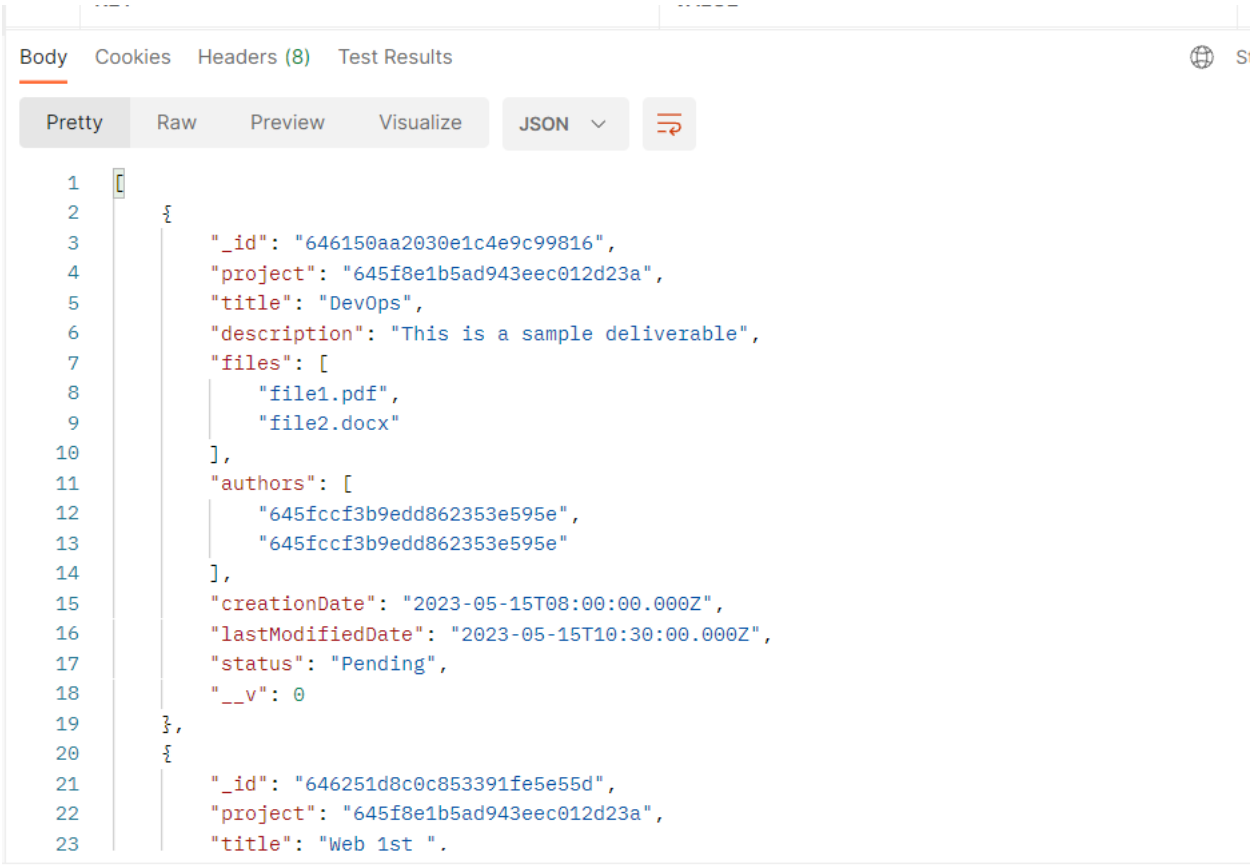

```
{
    "title": "OOP",
    "id": "111qqq",
    "description": "storing data into database",
    "file": "http/this",
    "startDate": "2023-05-09T19:00:00.000Z",
    "dueDate": "2023-05-19T19:00:00.000Z",
    "_id": "646249af35c8bcf81f5c7dab",
    "__v": 0
```


2. http://localhost:3001/deliverables/createDeliverable

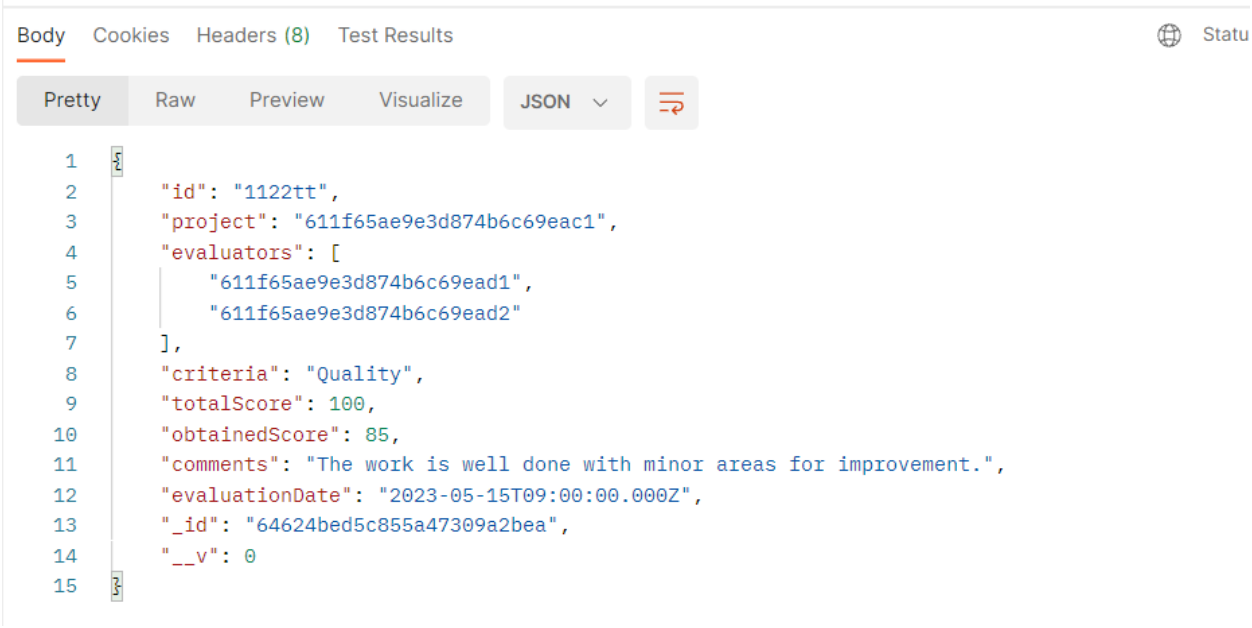



```
{
    "id": "55pp",
    "project": "645f8e1b5ad943eec012d23a",
    "title": "Web 1st",
    "description": "This is a sample deliverable",
    "files": [
        "file1.pdf",
        "file2.docx"
    ],
    "authors": [
        "645fccf3b9edd862353e595e",
        "645fccf3b9edd862353e595e"
    ],
    "creationDate": "2023-05-15T08:00:00.000Z",
    "lastModifiedDate": "2023-05-15T10:30:00.000Z",
    "status": "Pending",
    "_id": "646252f77a9339920f35cb3d",
    "__v": 0
}
```

3. http://localhost:3001/deliverables/getAllDeliverables

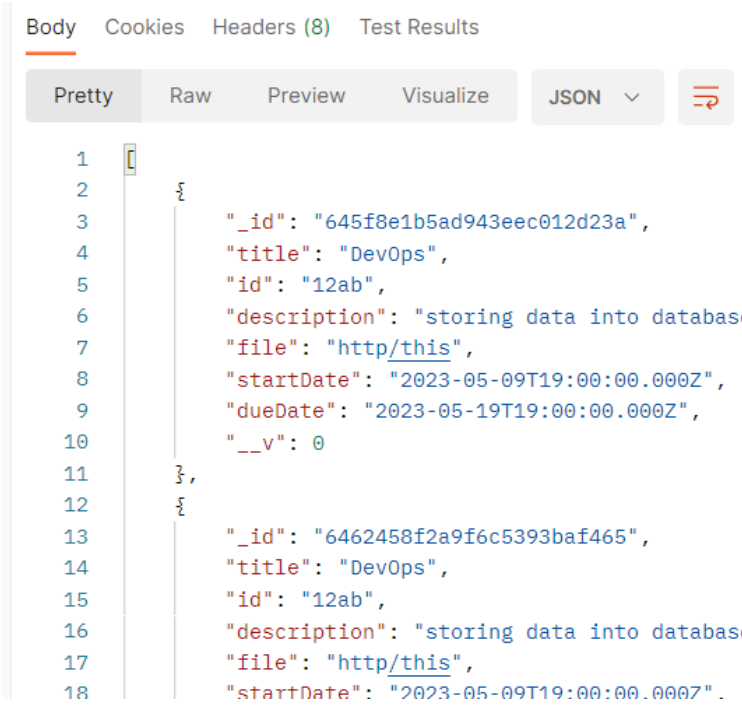


4. http://localhost:3001/evaluations/createEvaluation



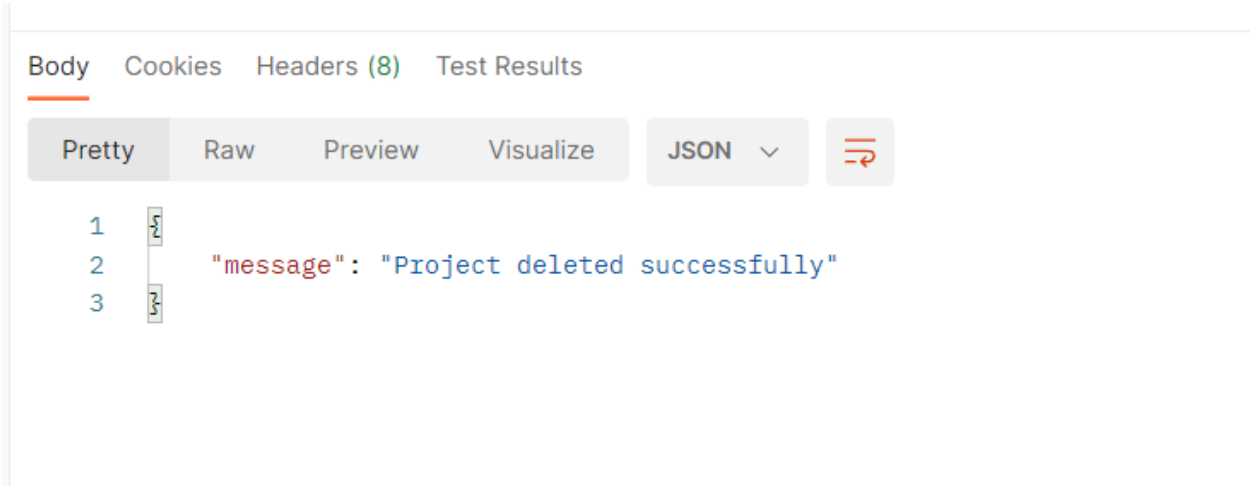

```
{
    "id": "1122tt",
    "project": "611f65ae9e3d874b6c69eac1",
    "evaluators": [
        "611f65ae9e3d874b6c69ead1",
        "611f65ae9e3d874b6c69ead2"
    ],
    "criteria": "Quality",
    "totalScore": 100,
    "obtainedScore": 85,
    "comments": "The work is well done with minor areas for improvement.",
    "evaluationDate": "2023-05-15T09:00:00.000Z",
    "_id": "64624bed5c855a47309a2bea",
    "__v": 0
}
```


5. http://localhost:3001/projects/getAllProjects

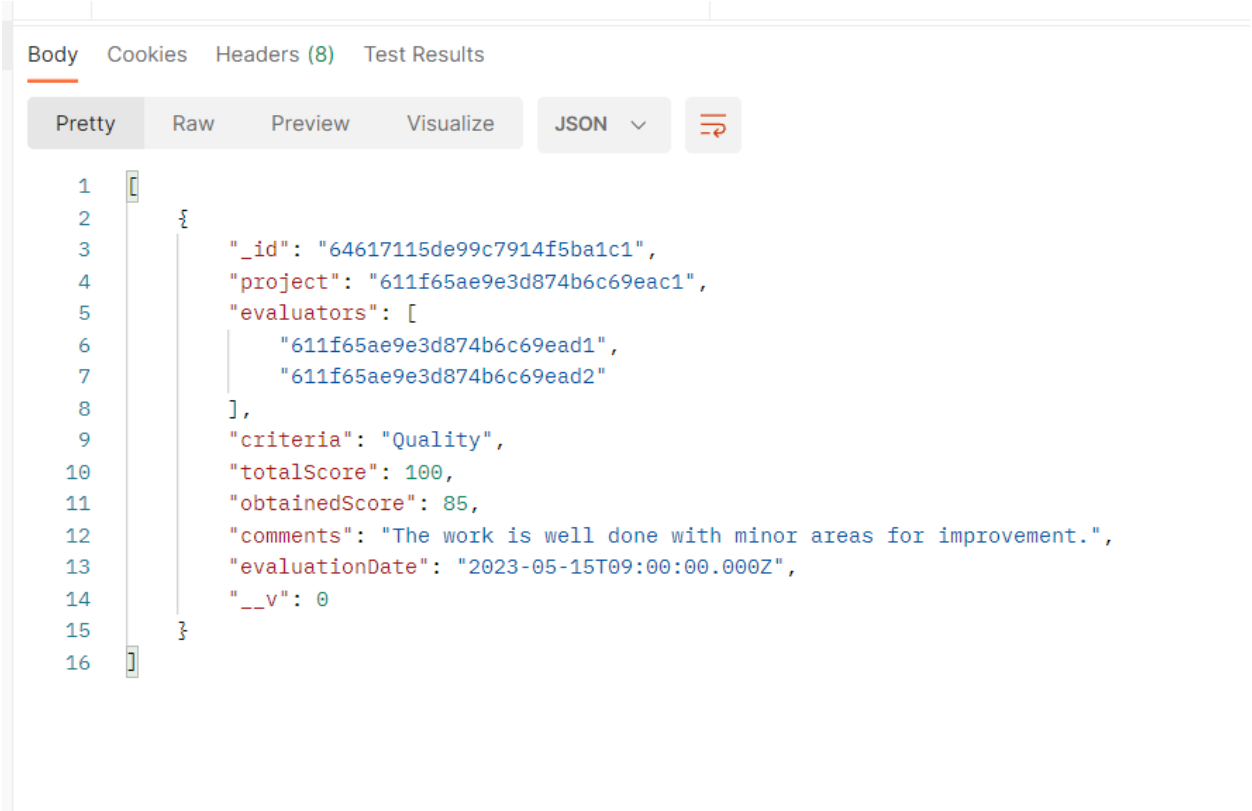



```
[
    {
        "_id": "645f8e1b5ad943eec012d23a",
        "title": "DevOps",
        "id": "12ab",
        "description": "storing data into database",
        "file": "http/this",
        "startDate": "2023-05-09T19:00:00.000Z",
        "dueDate": "2023-05-19T19:00:00.000Z",
        "__v": 0
    },
    {
        "_id": "6462458f2a9f6c5393baf465",
        "title": "DevOps",
        "id": "12ab",
        "description": "storing data into database",
        "file": "http/this",
        "startDate": "2023-05-09T19:00:00.000Z",
```

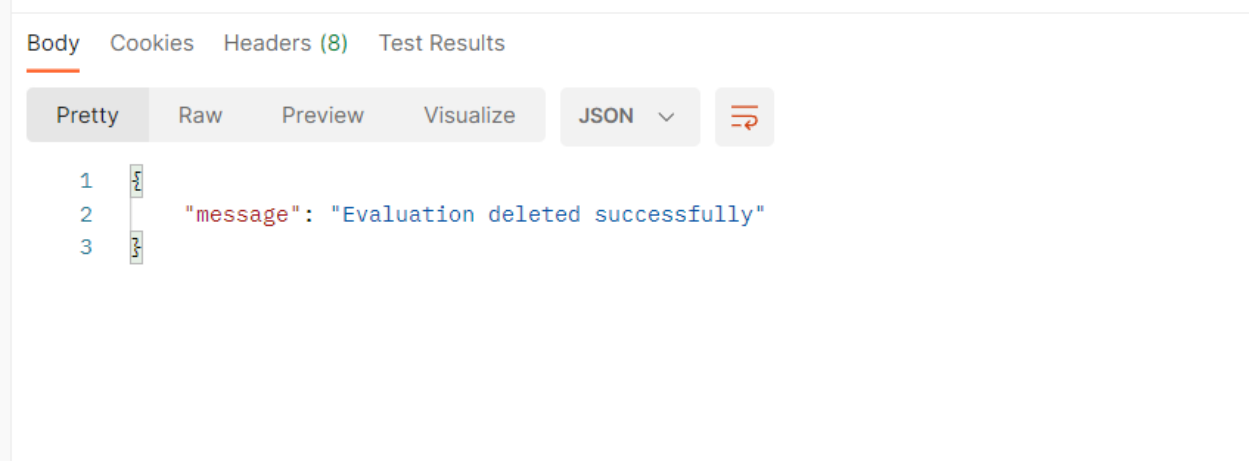
6. http://localhost:3001/projects/deleteProject/:id?id=12qwert



7. http://localhost:3001/evaluations/getAllEvaluations



8. http://localhost:3001/evaluations/deleteEvaluation/:id?id=1122tt



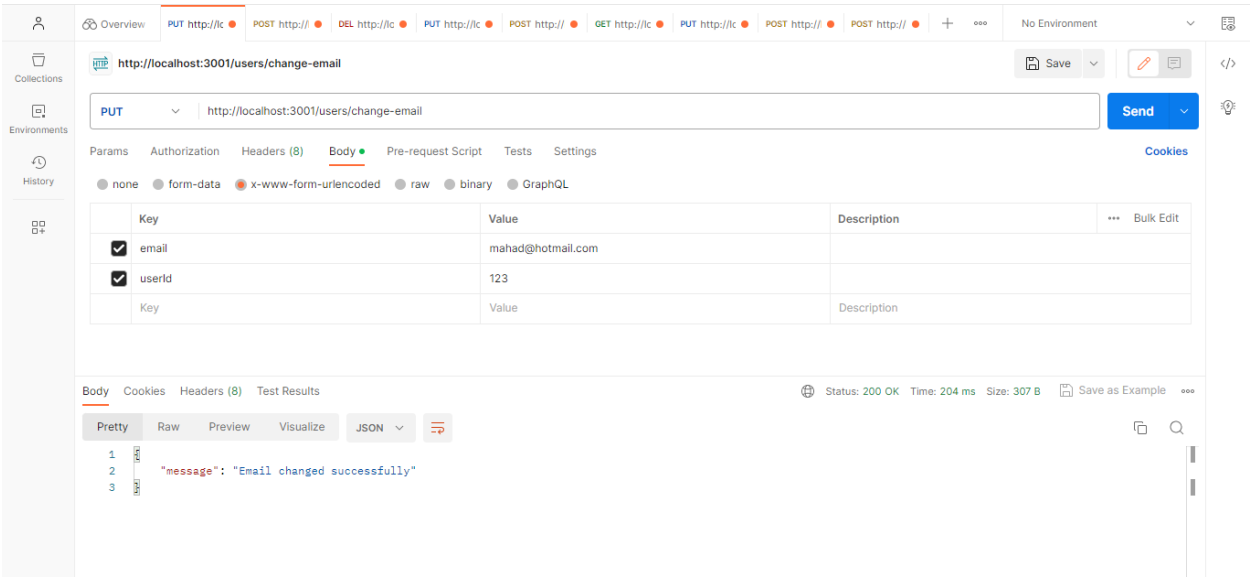
9. http://localhost:3001/deliverables/deleteDeliverable/:id

Mahad Rahat

- Sample request and response of the APIs (which you have developed)

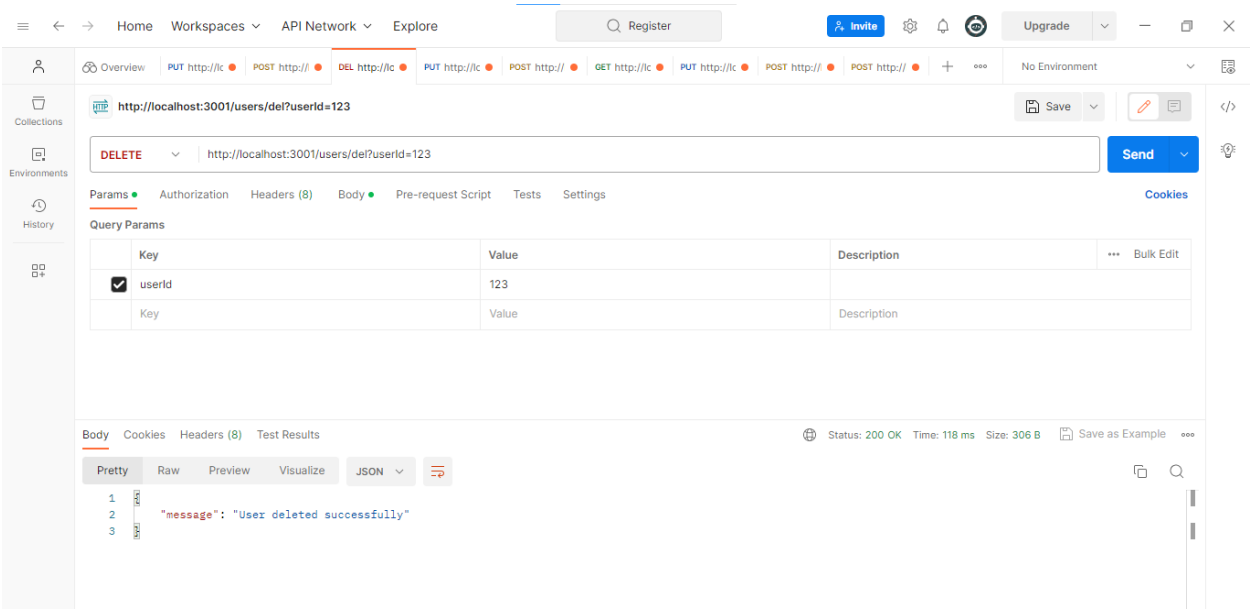
1. Changing email of a user

http://localhost:3001/users/change-email



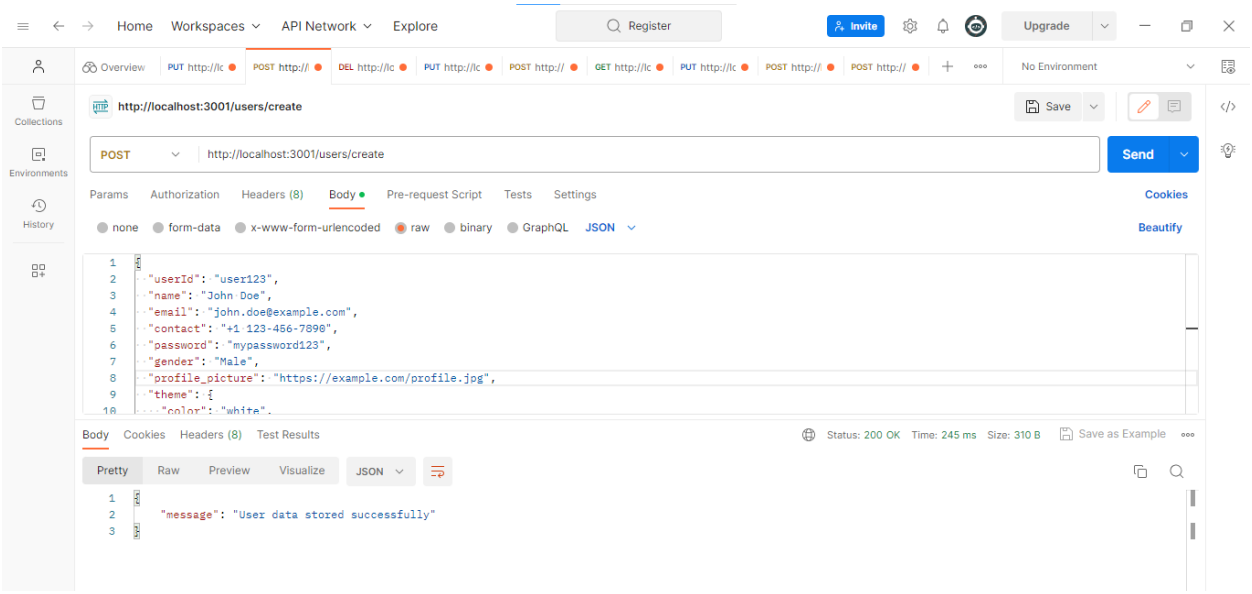
2. Deletion of a user

http://localhost:3001/users/del?userId=123



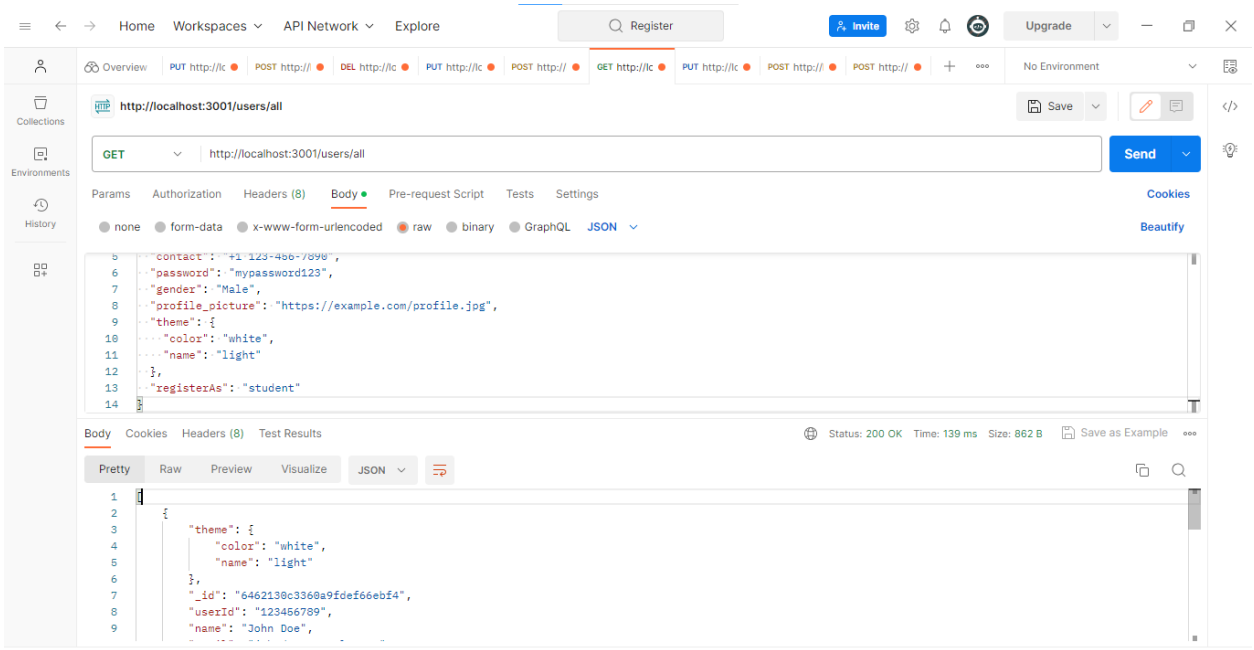
3. User Creation:

http://localhost:3001/users/create



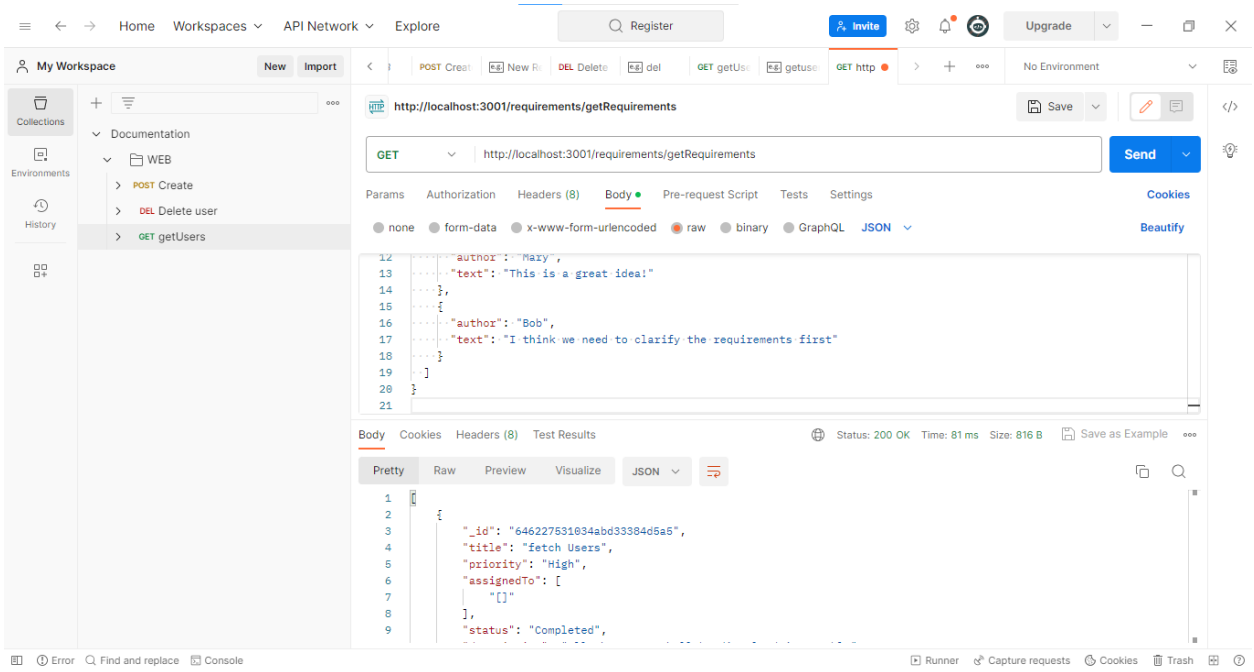
4. Getting all the users

http://localhost:3001/users/all

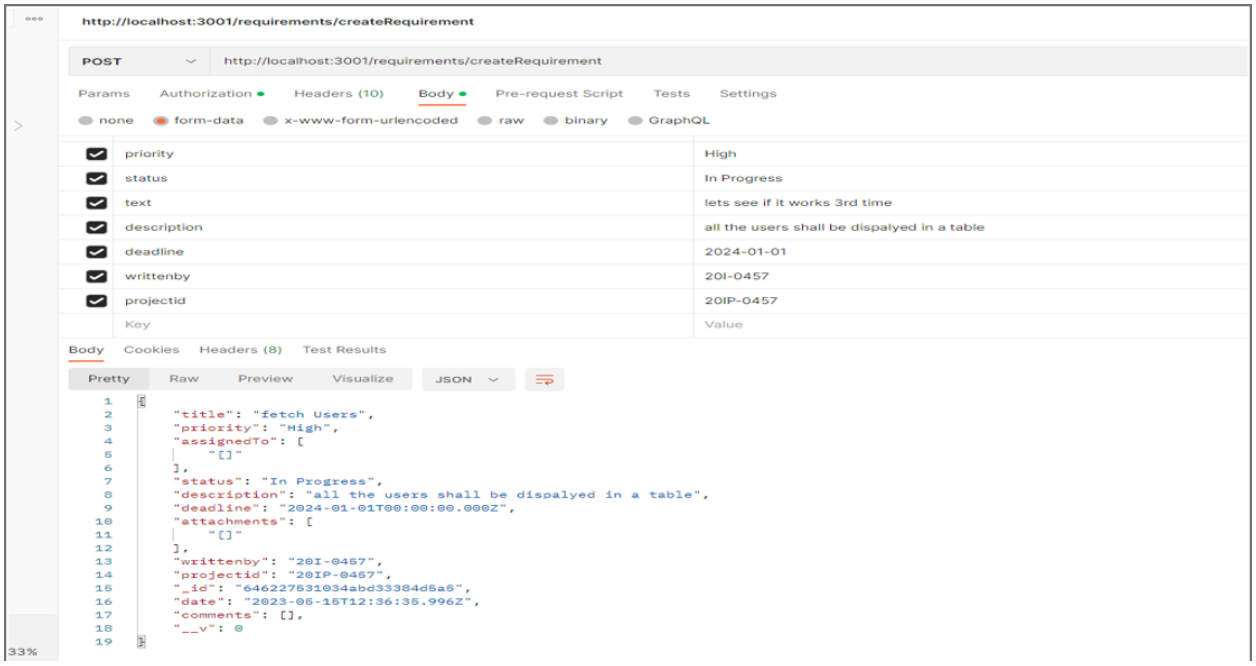


5. Getting all requirements

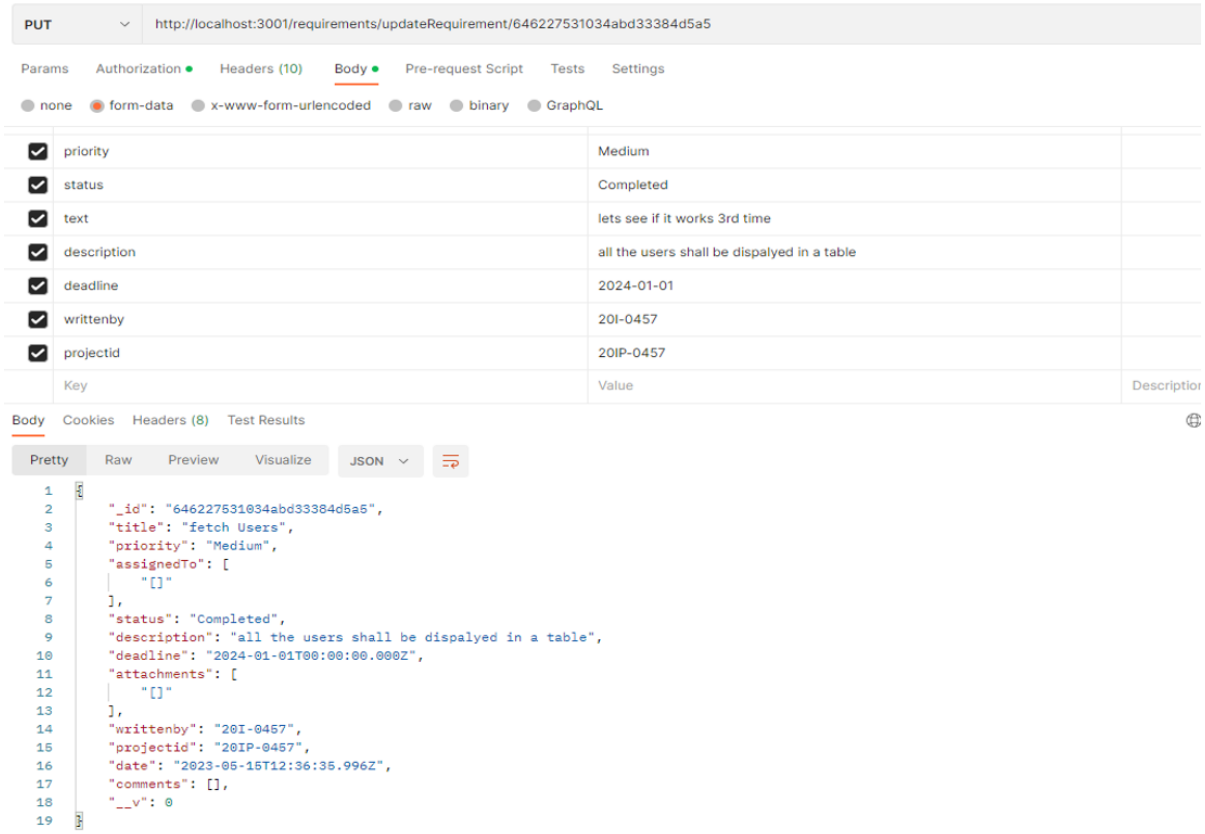
http://localhost:3001/requirements/getRequirements



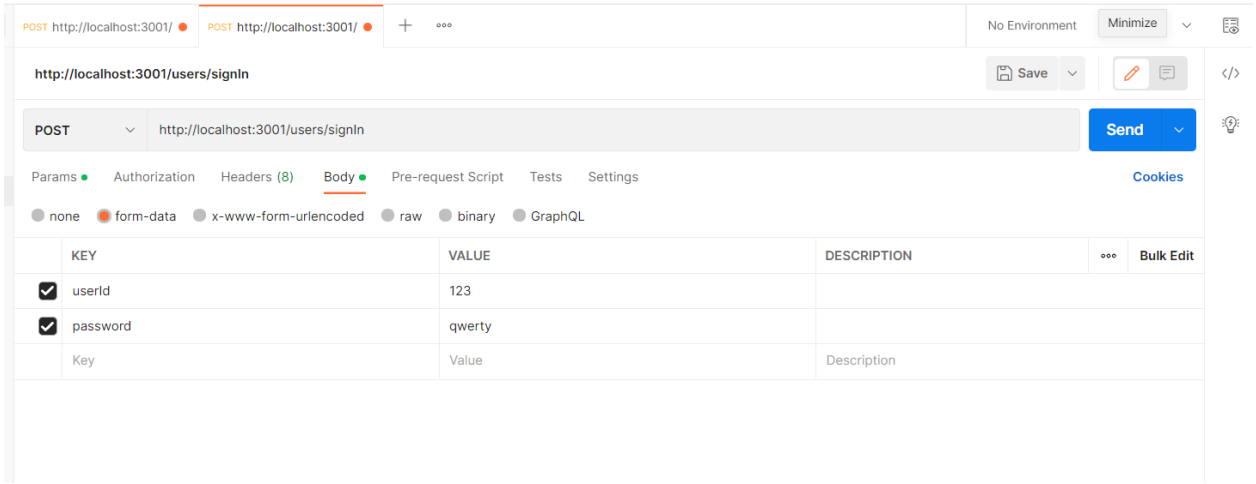
6: Creating Requirements (Teacher):



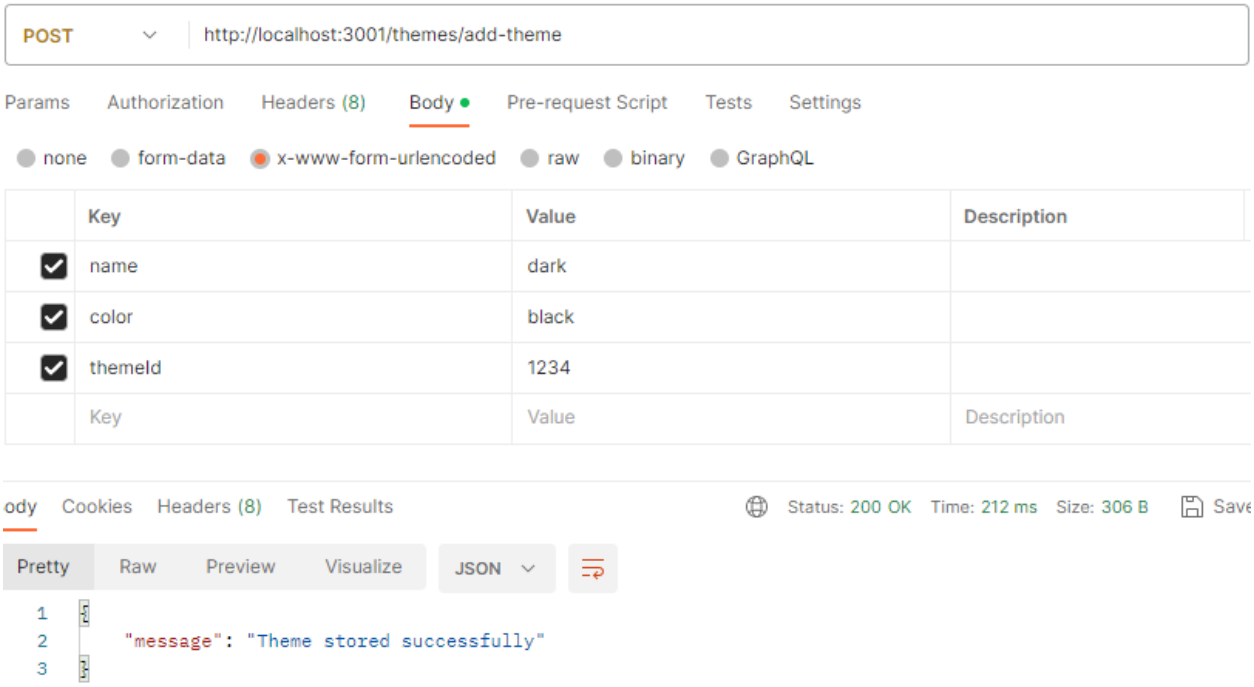
7. Update requirements



8: SignIn with jwt

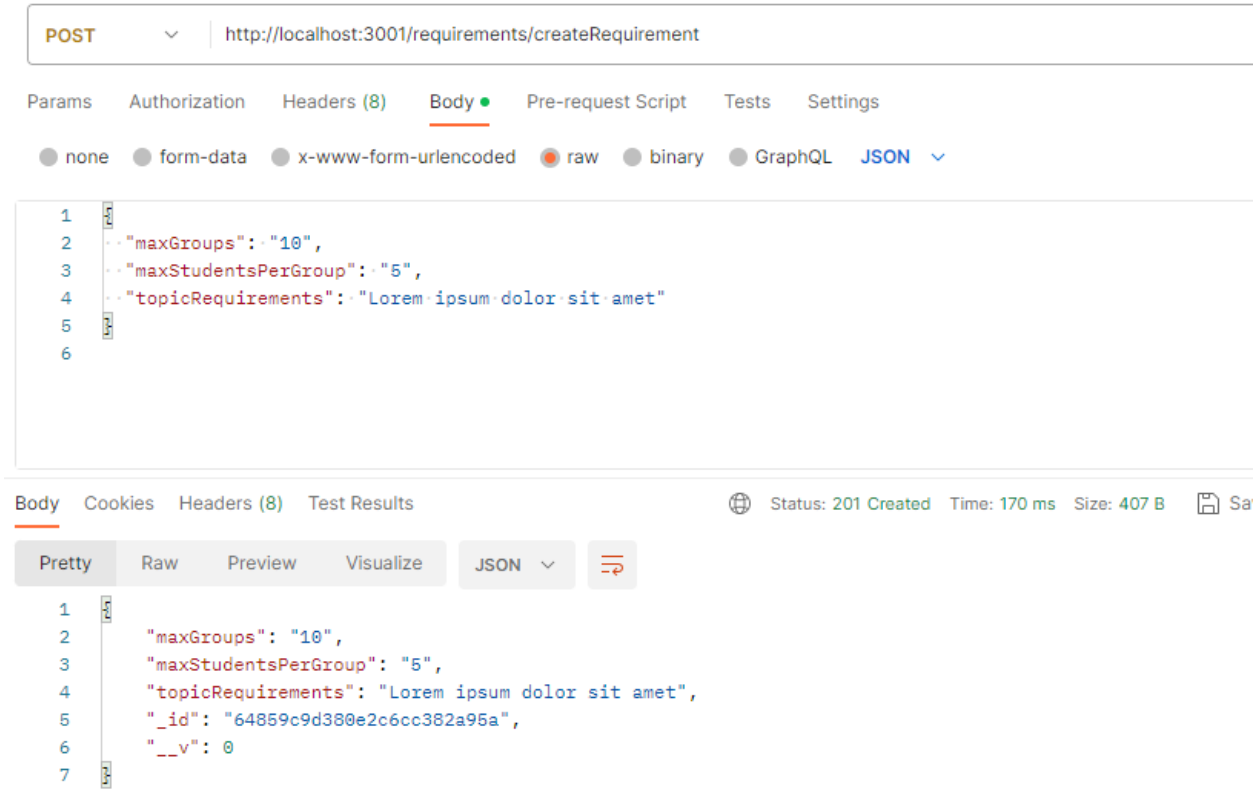


10. Creating theme:



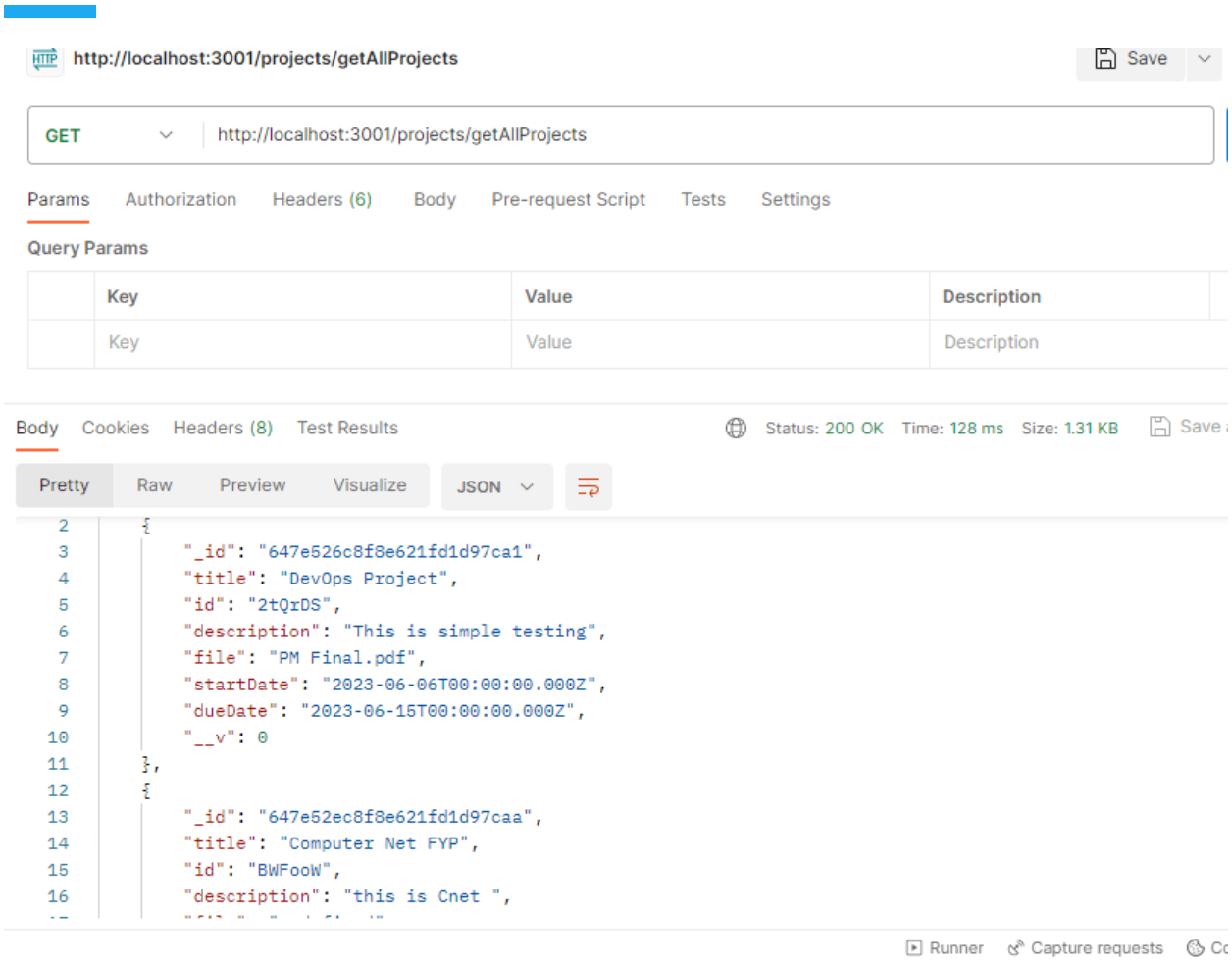
11. Create requirements (admin):

http://localhost:3001/requirements/createRequirement

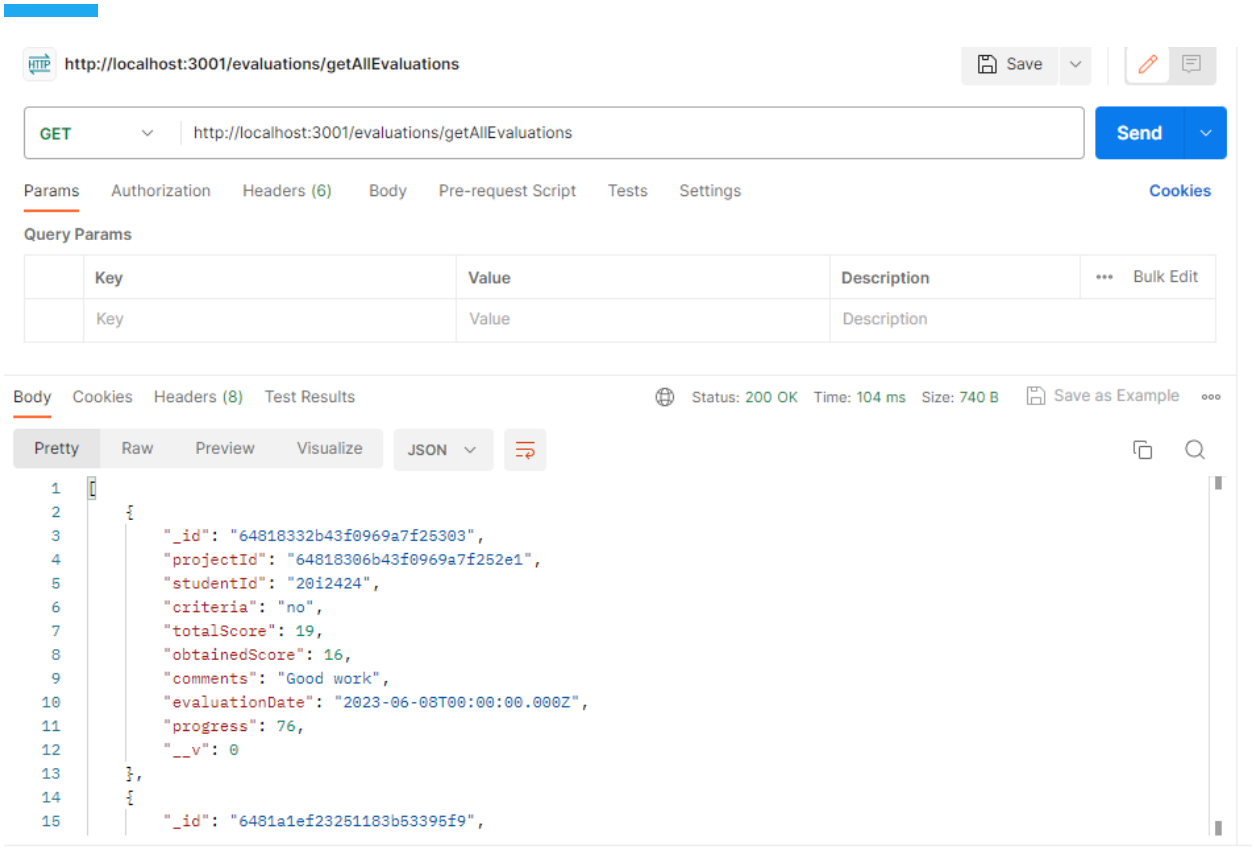


12. View projects:

http://localhost:3001/projects/getAllProjects

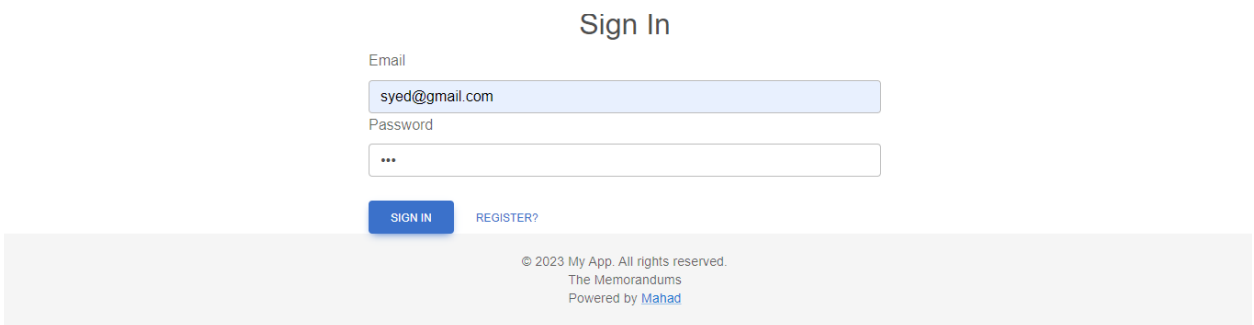


13. View evaluation and project status:

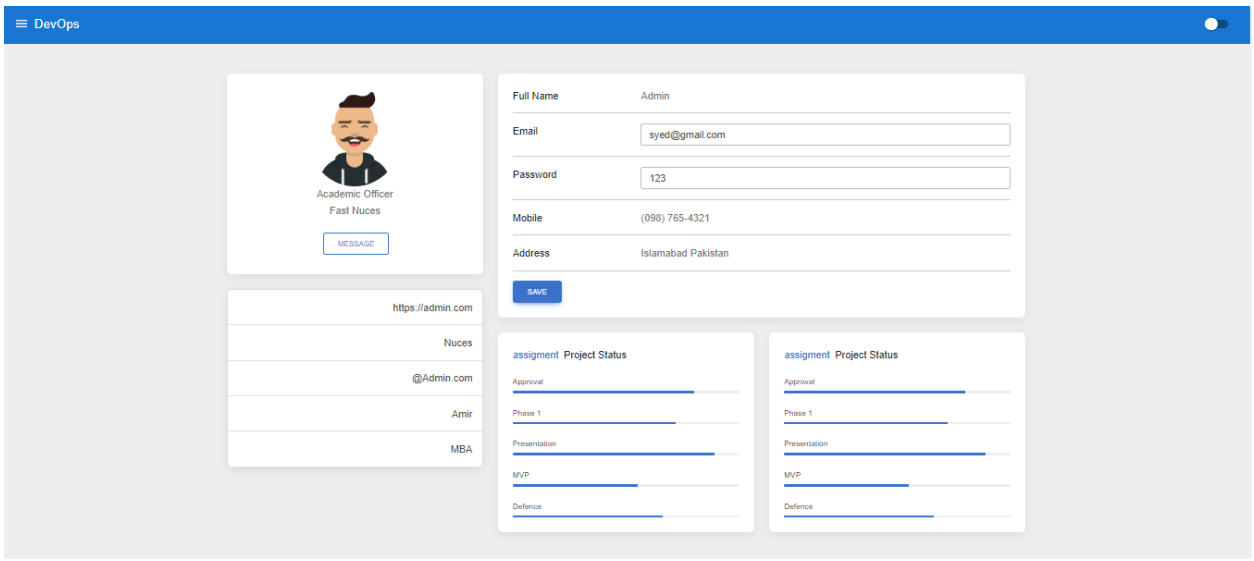


Screenshots of UI:

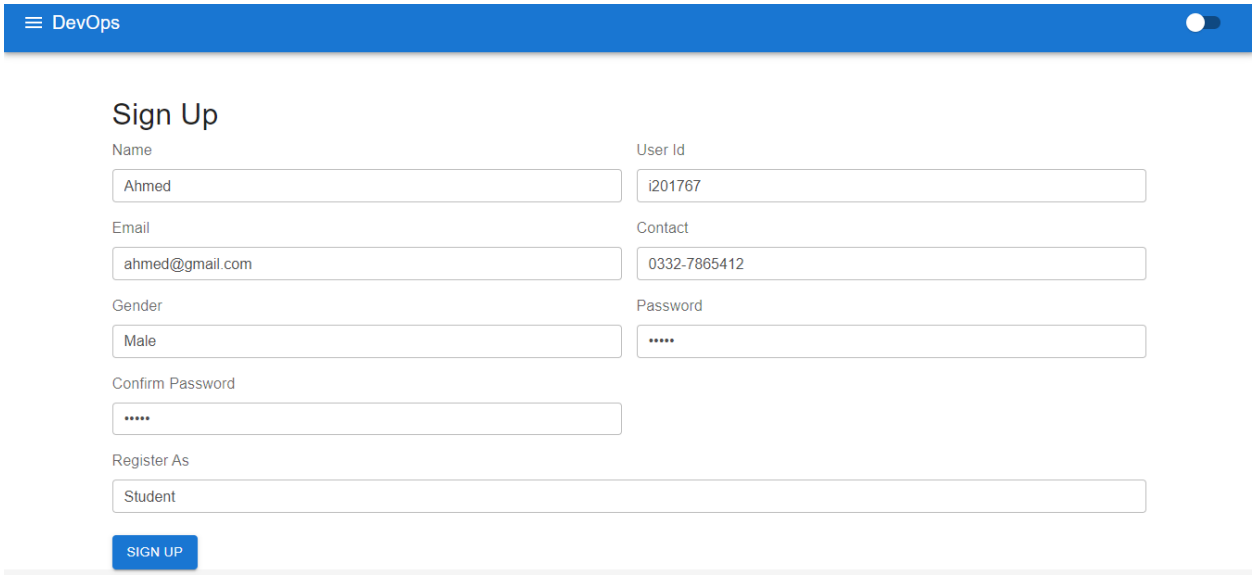
1- Sign In:



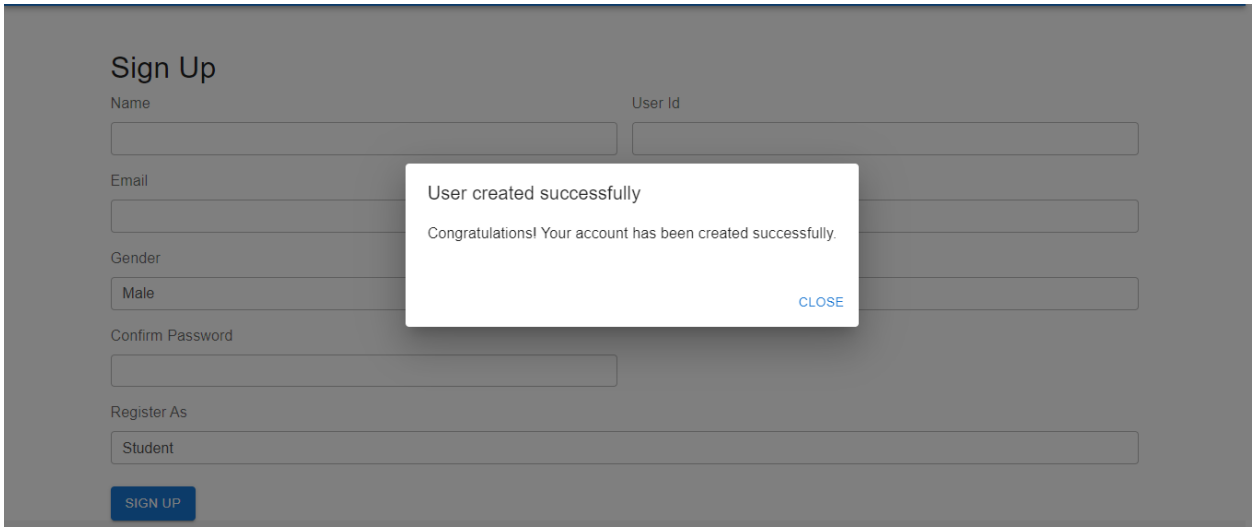
2-Home/Dashboard:



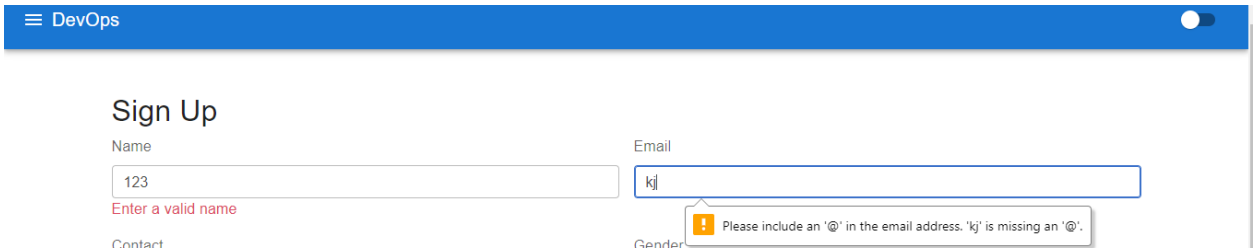
3- Create User:



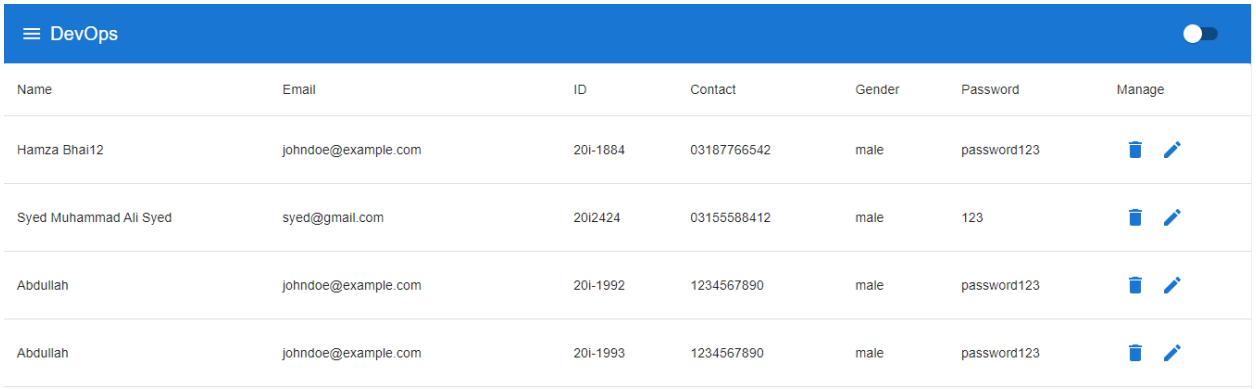
3.1: User successfully stored in database:



3.2 Input validation:



4- Manage User:



4.1: Update user (change email/password):

Imdad	imdad@gmail.com	Ali	21313	Male	1234	
Ali	ali@gmail.com	2	313213	Male	1122	

Edit User

Name: Imdad Email: imdad123@gmail.com Contact: 21313 Gender: Male Password: •••••••

Save

Email and password is changed

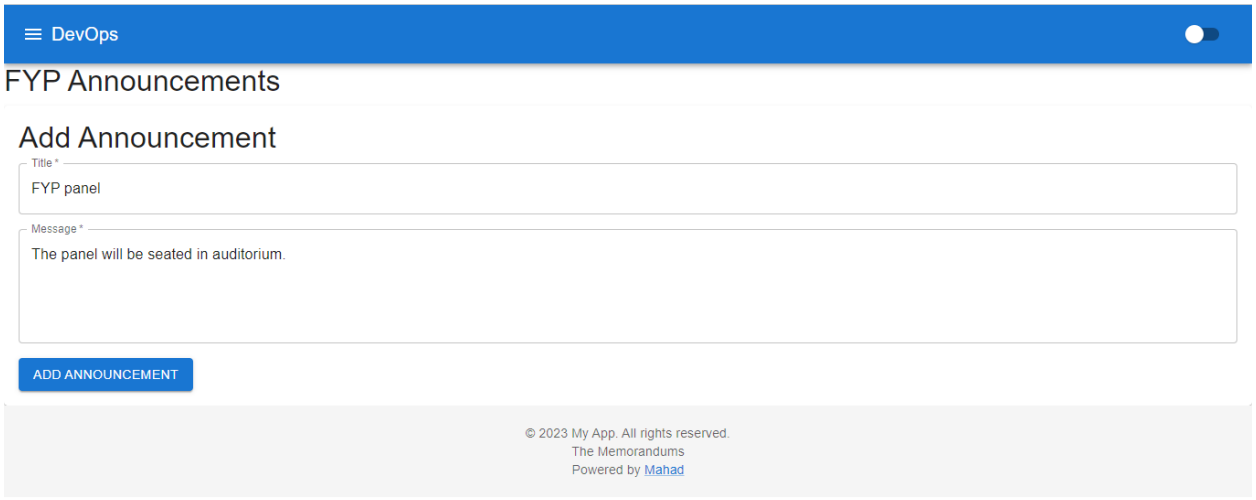
Imdad	imdad123@gmail.com	Ali	21313	Male	1234567	

4.2: Delete user:

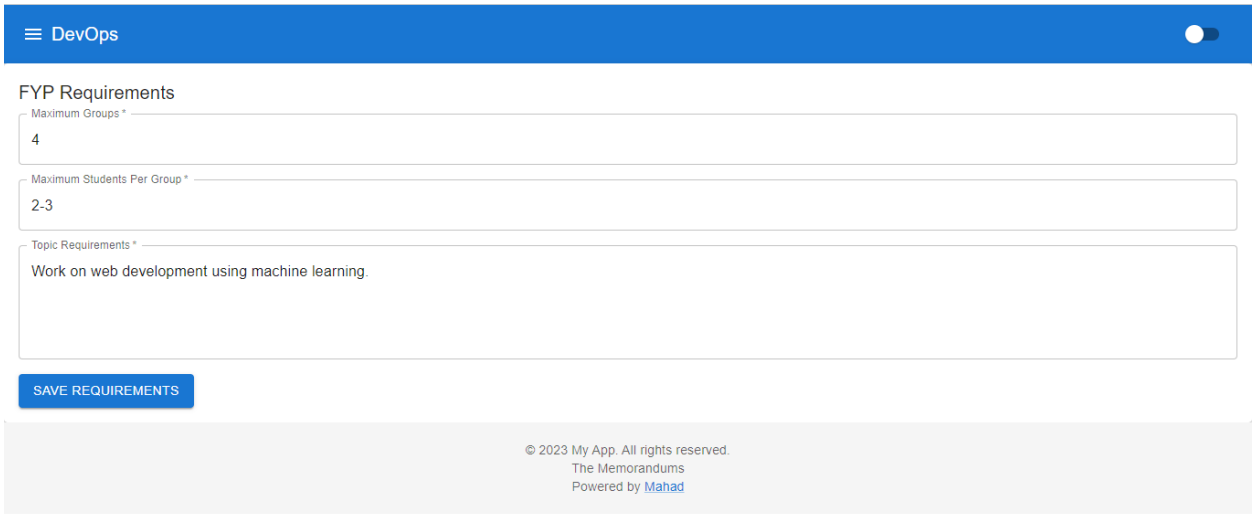
The user at the first row gets deleted.

Name	Email	ID	Contact	Gender	Password	Manage
Syed Muhammad Ali Syed	syed@gmail.com	20i2424	03155588412	male	123	
Abdullah	johndoe@example.com	20i-1992	1234567890	male	password123	
Abdullah	johndoe@example.com	20i-1993	1234567890	male	password123	
Hamza	hamzaazam994@gmail.com	asd1212	1	MALE	12345	
Hammad	hamad@gmail.com	11231	051-8978768	MALE	1234	

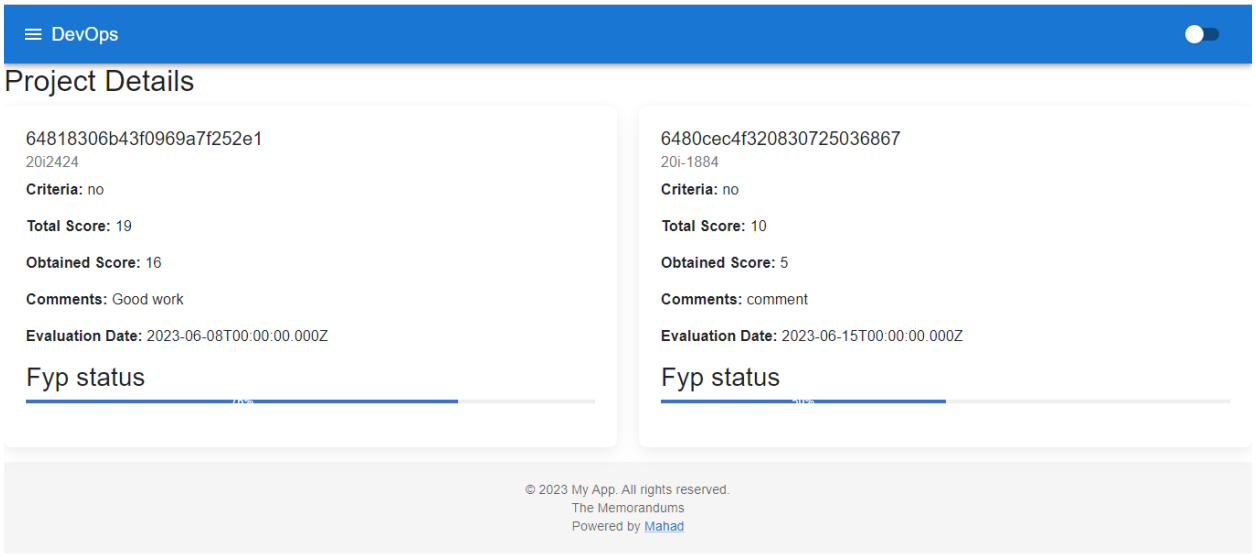
5- Make an announcement:



6- Create general requirements for FYP:



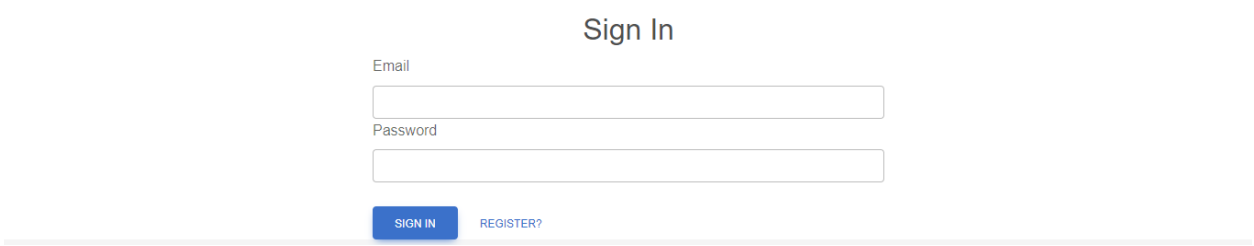
7- View Evaluations and project status:



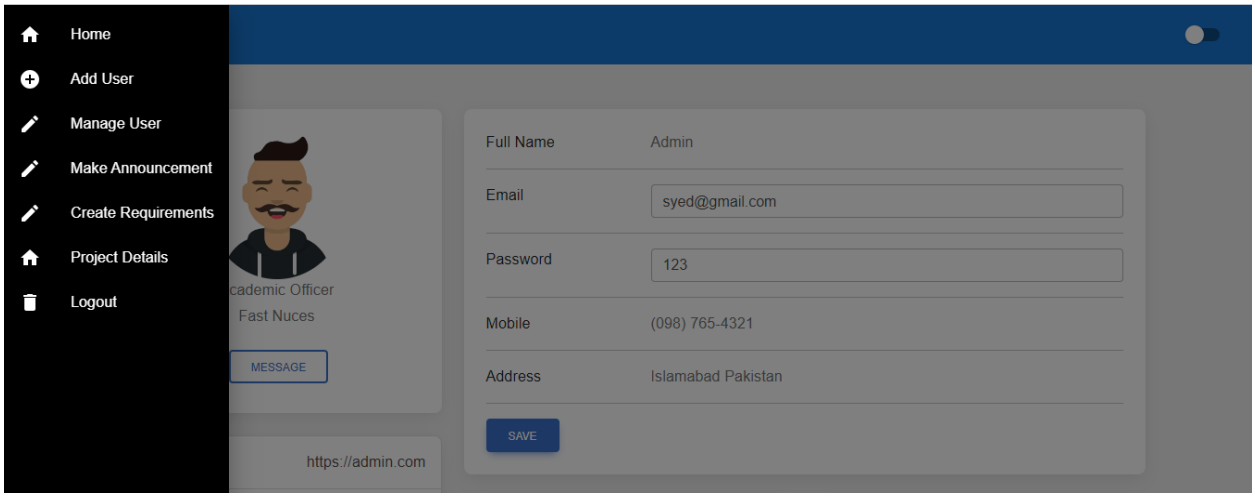
8- Change Theme:

9- Logout:

Logging out redirects you to the sign in page.



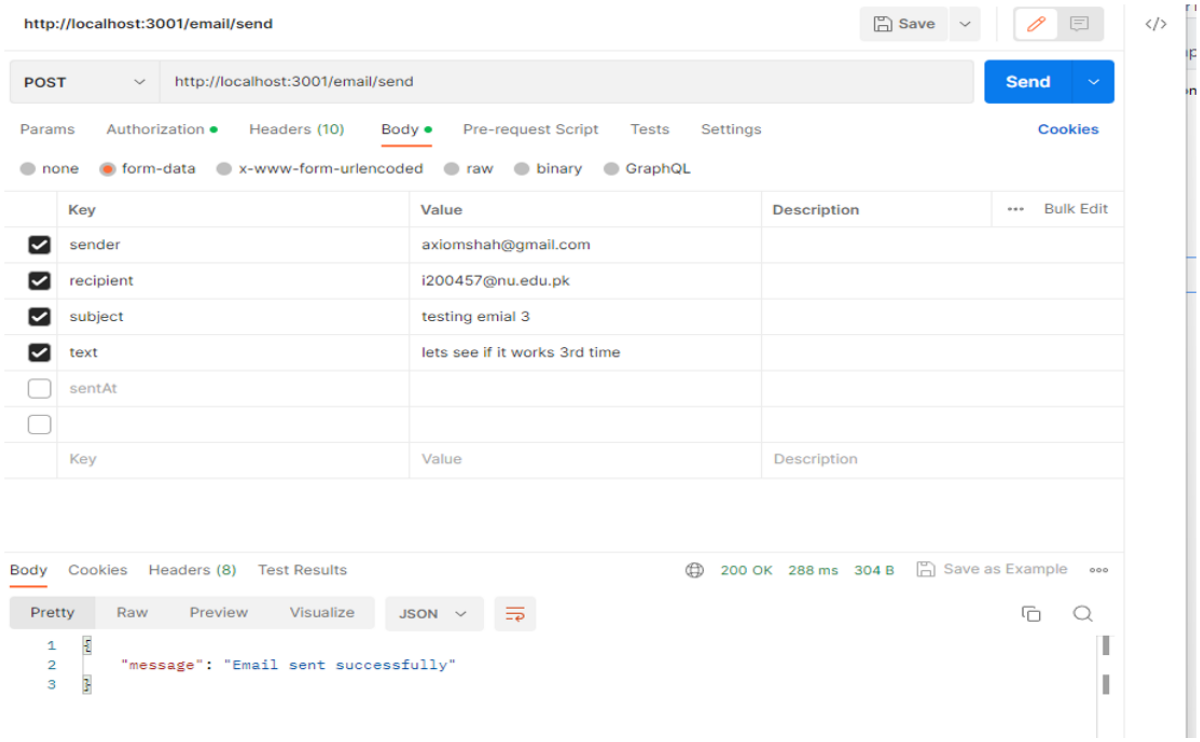
10- Header Component:



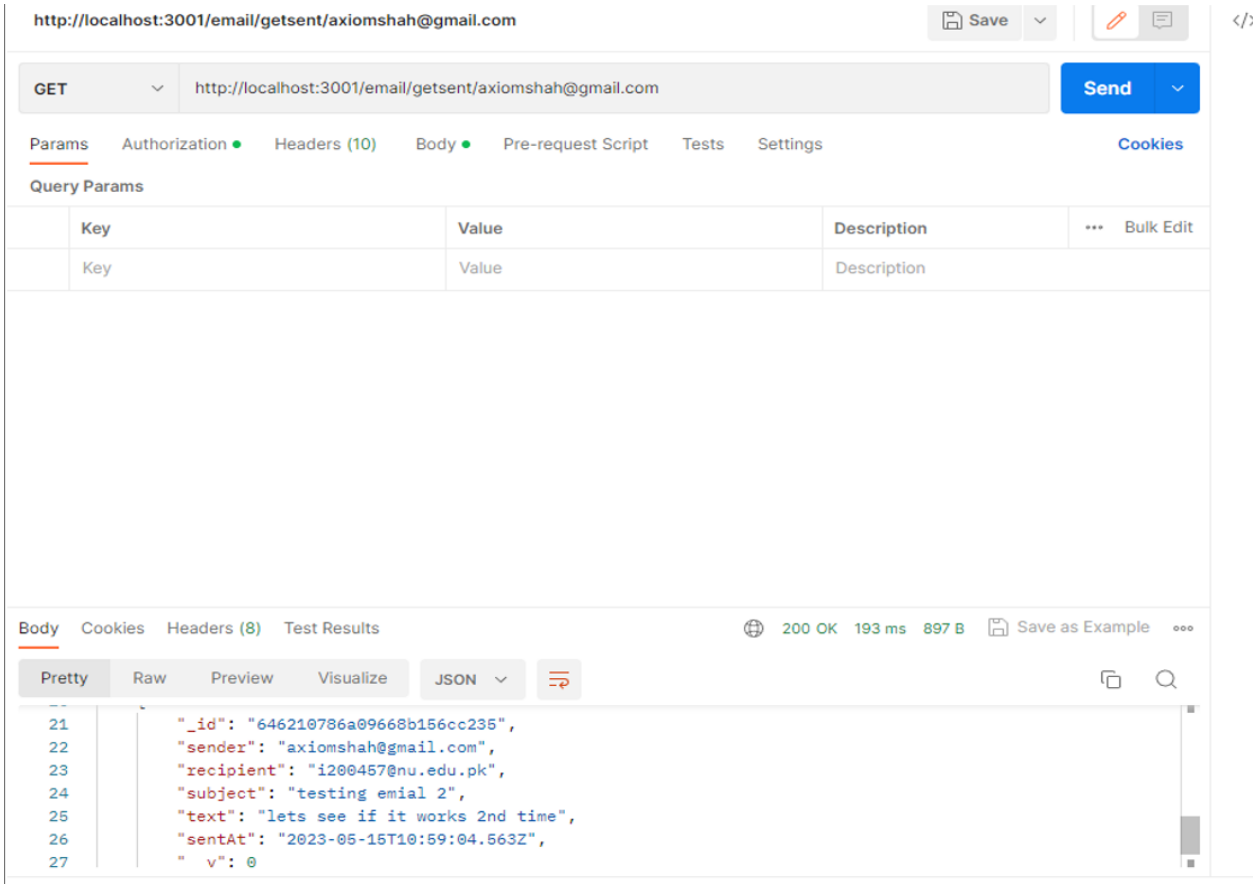
Mian Abdullah

Email Module

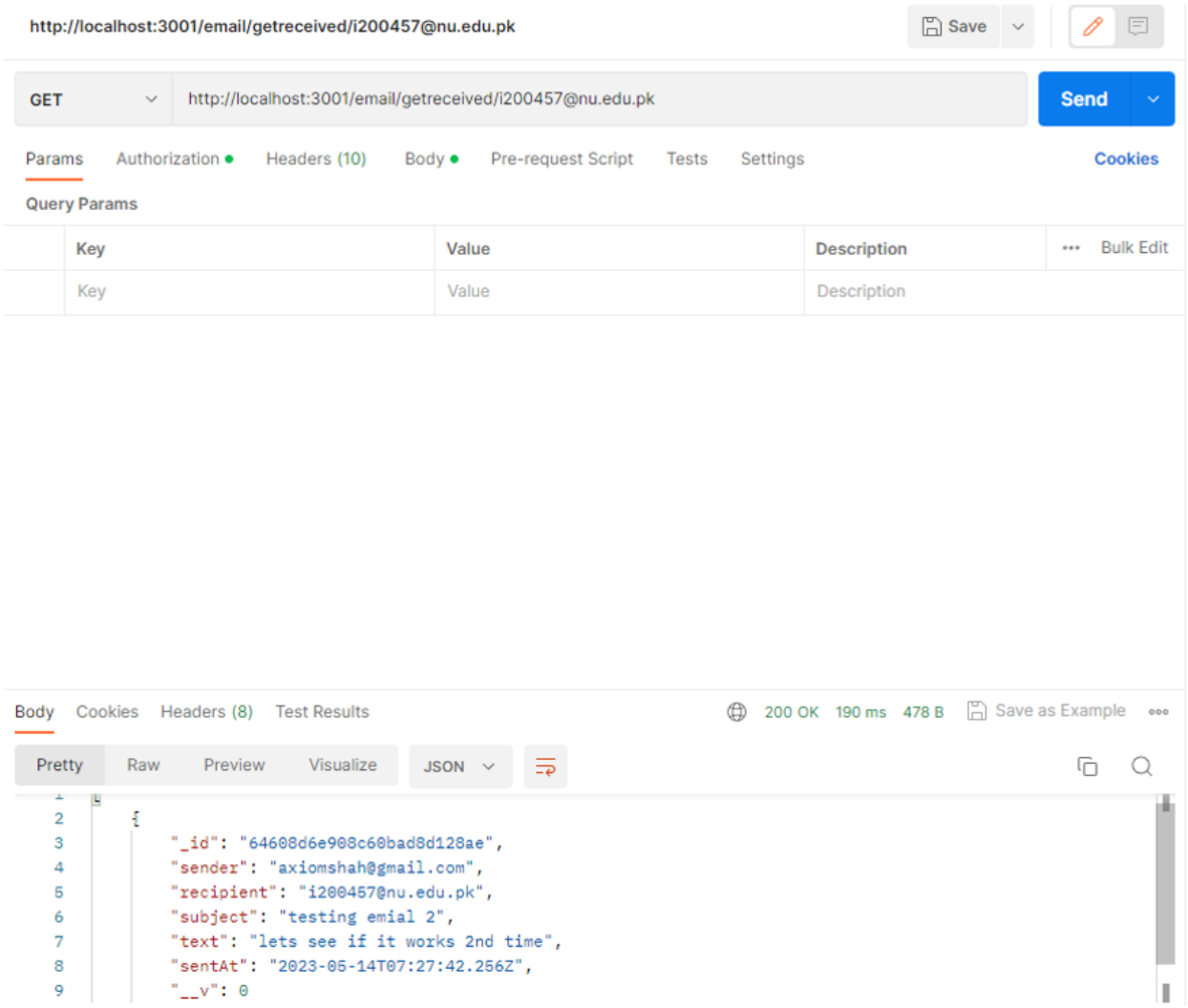
Send an email to a particular person



Get all emails that were send by a particular user

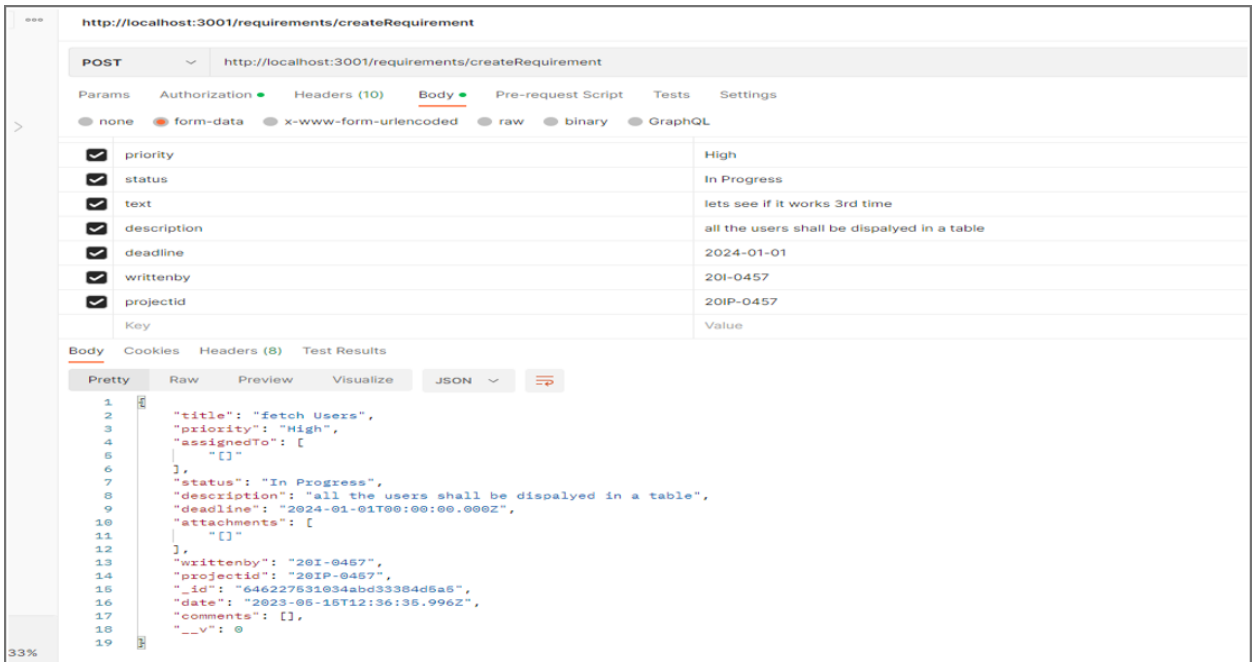


Get all emails that were receivied by a particular user

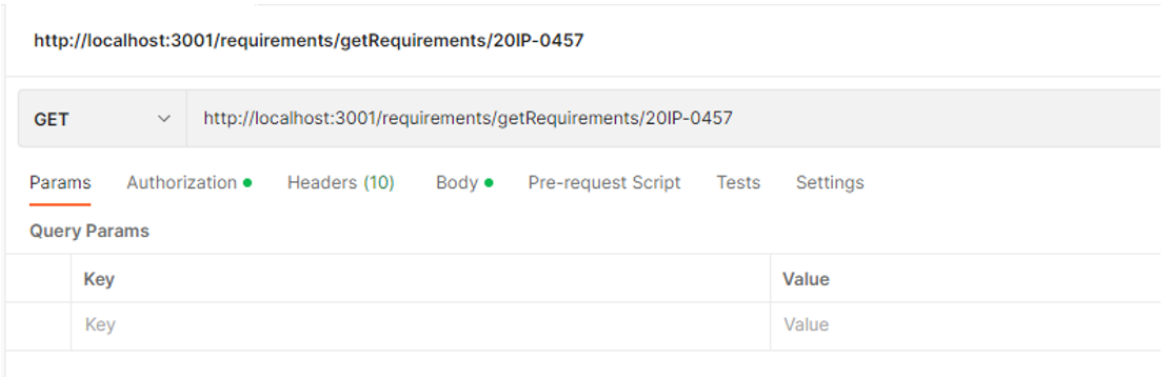


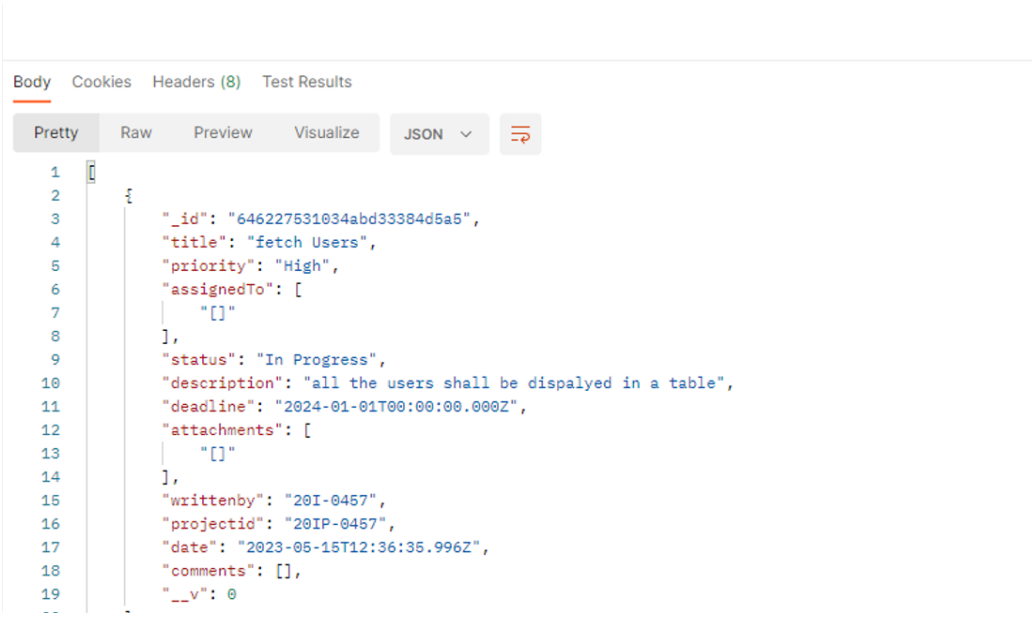
Requirement Module

Create a Requirement



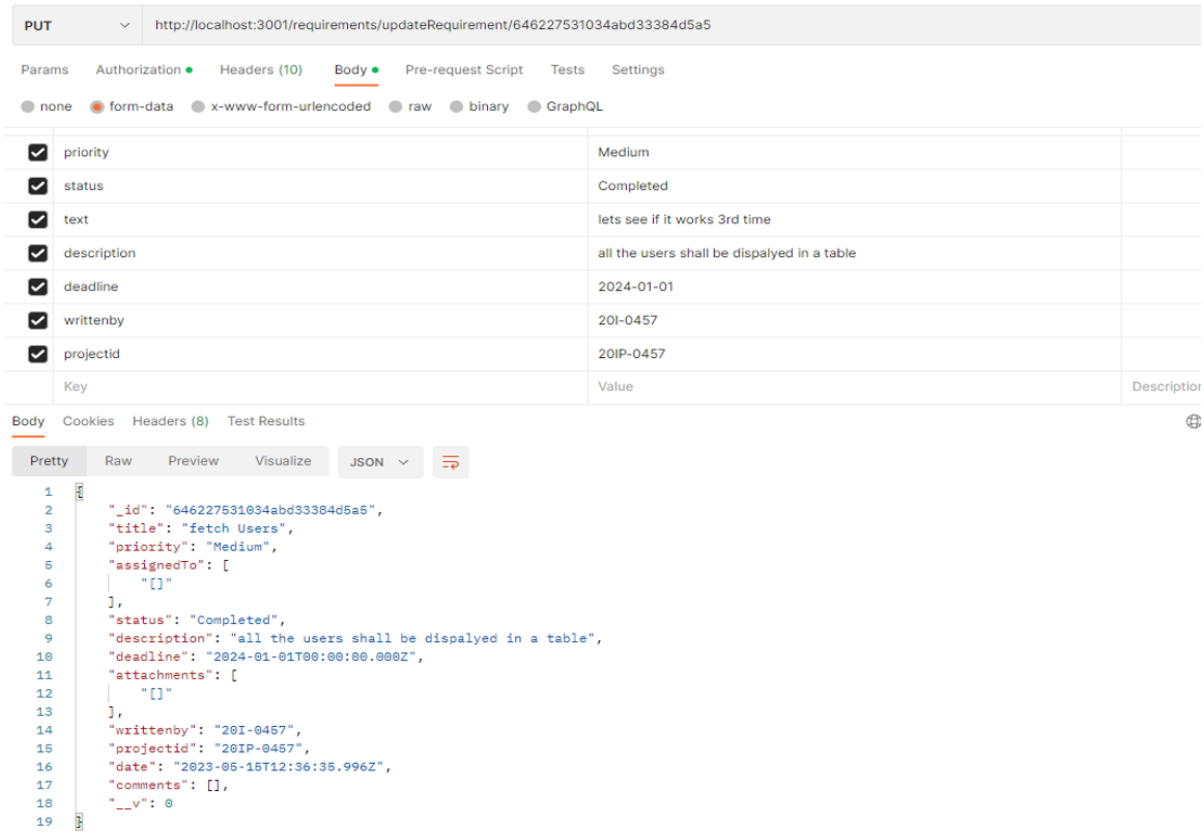
Find requirement by Projectid



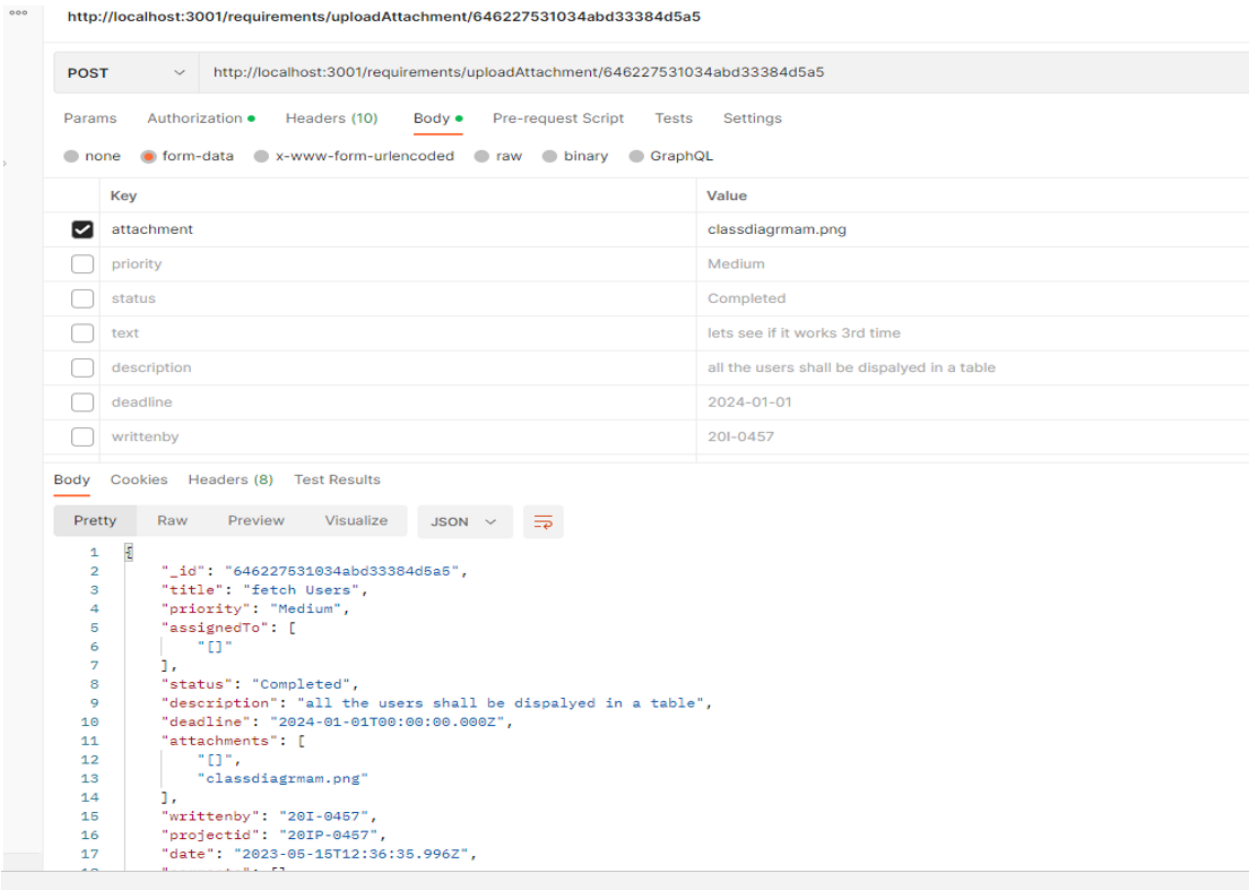


update requirement

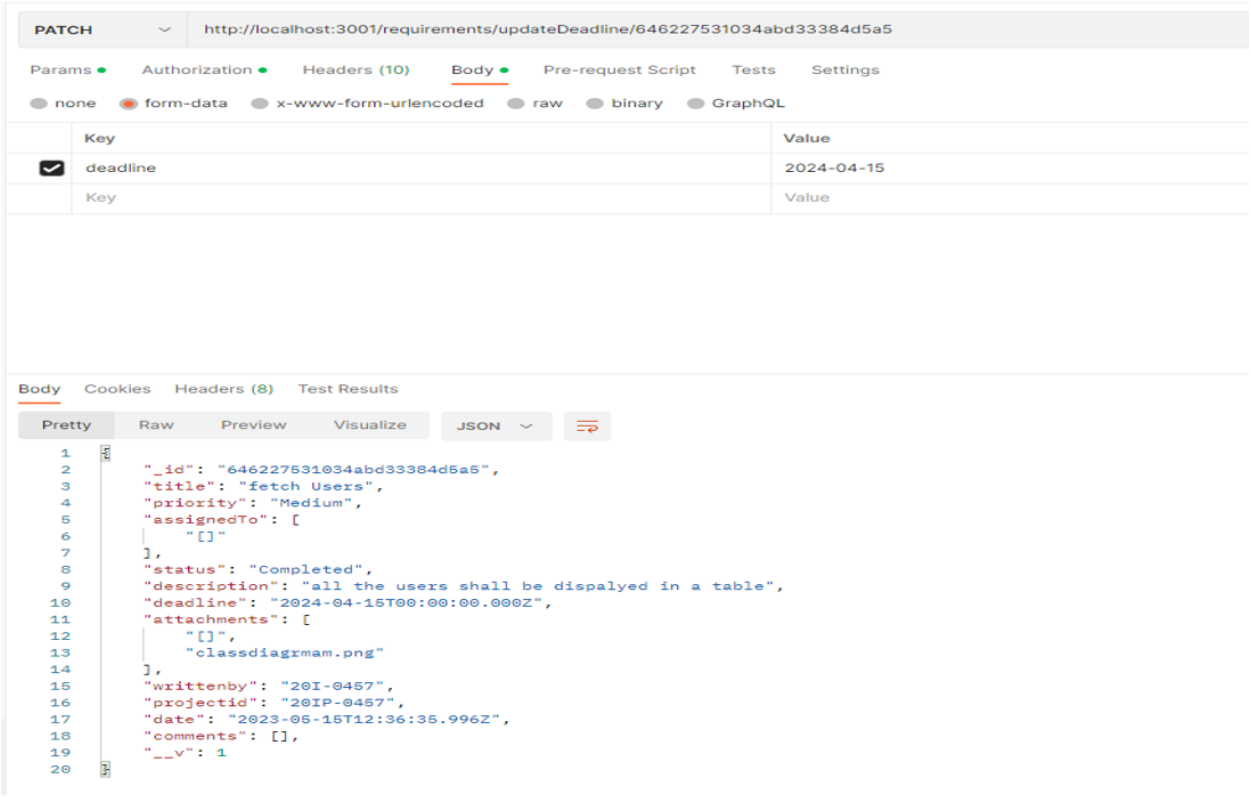
_Updating the priority to Medium and status to Complete


PUT
http://localhost:3001/requirements/updateRequirement/646227531034abd33384d5a5
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
priority
Medium
status
Completed
text
lets see if it works 3rd time
description
all the users shall be dispalyed in a table
deadline
2024-01-01
writtenby
20I-0457
projectid
20IP-0457
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"_id": "646227531034abd33384d5a5",
"title": "fetch Users",
"priority": "Medium",
"assignedTo": [
"[]"
],
"status": "Completed",
"description": "all the users shall be dispalyed in a table",
"deadline": "2024-01-01T00:00:00.000Z",
"attachments": [
"[]"
],
"writtenby": "20I-0457",
"projectid": "20IP-0457",
"date": "2023-05-15T12:36:35.996Z",
"comments": [],
"__v": 0
}

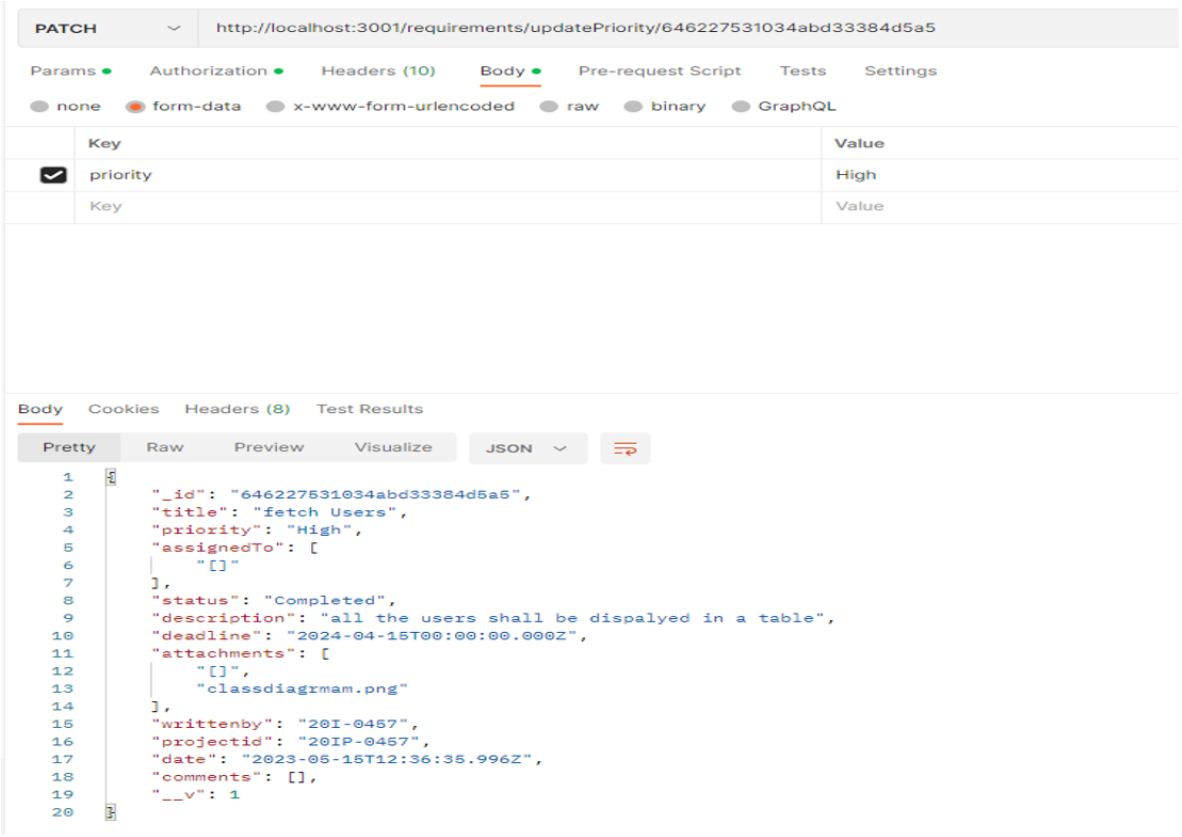
Upload attachment to a requirement



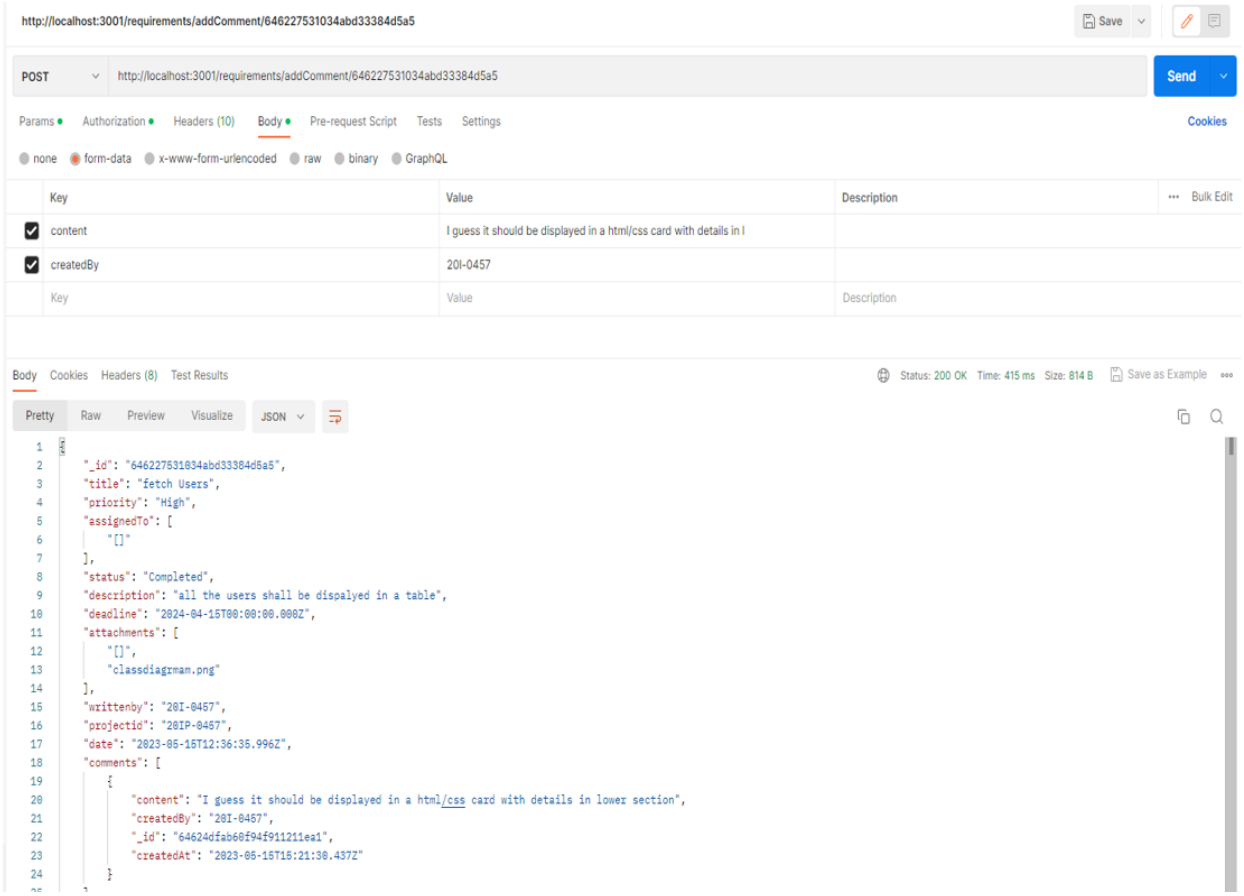
Update Deadline of requirement


PATCH
http://localhost:3001/requirements/updateDeadline/646227531034abd33384d5a5
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
deadline
2024-04-15
Key
Value
Body
Cookies
Headers (8)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"_id": "646227531034abd33384d5a5",
"title": "fetch Users",
"priority": "Medium",
"assignedTo": [
"[]"
],
"status": "Completed",
"description": "all the users shall be dispalyed in a table",
"deadline": "2024-04-15T00:00:00.000Z",
"attachments": [
"[]",
"classdiagrmam.png"
],
"writtenby": "20I-0457",
"projectid": "20IP-0457",
"date": "2023-05-15T12:36:35.996Z",
"comments": [],
"__v": 1
}

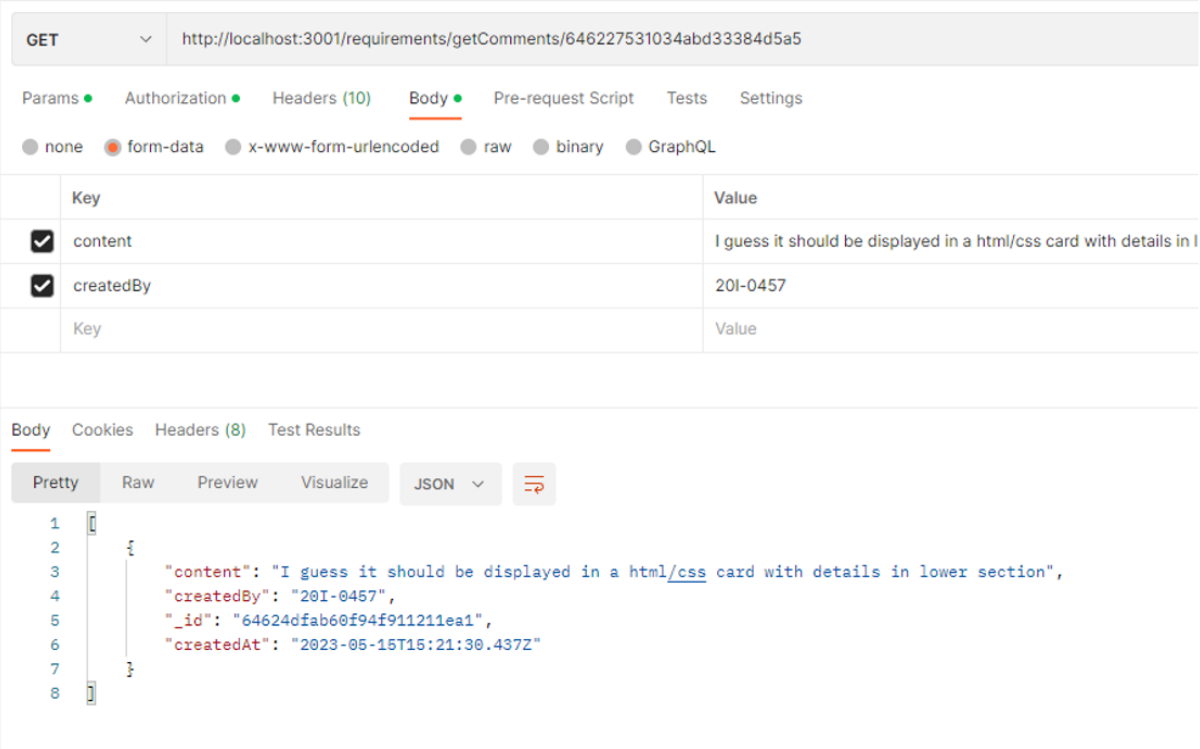
Update Priority of requirement



Add comments to requirement

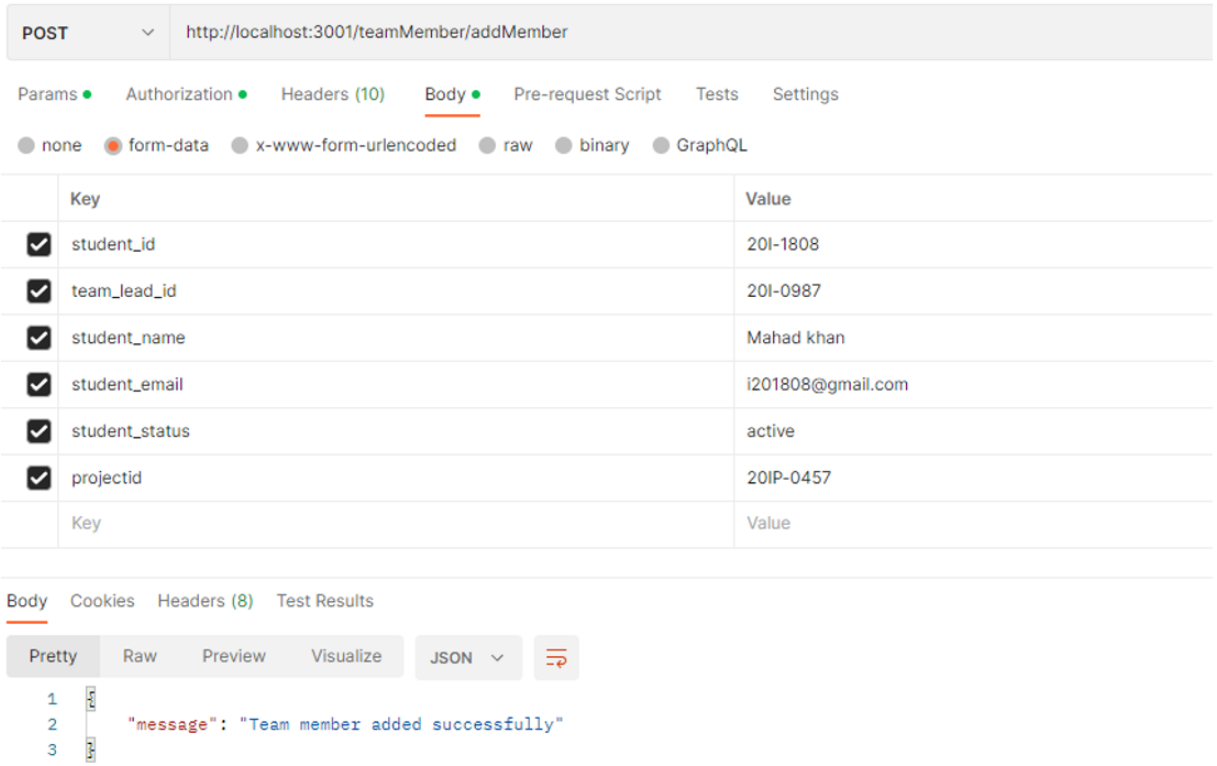


Display comments of a requirement

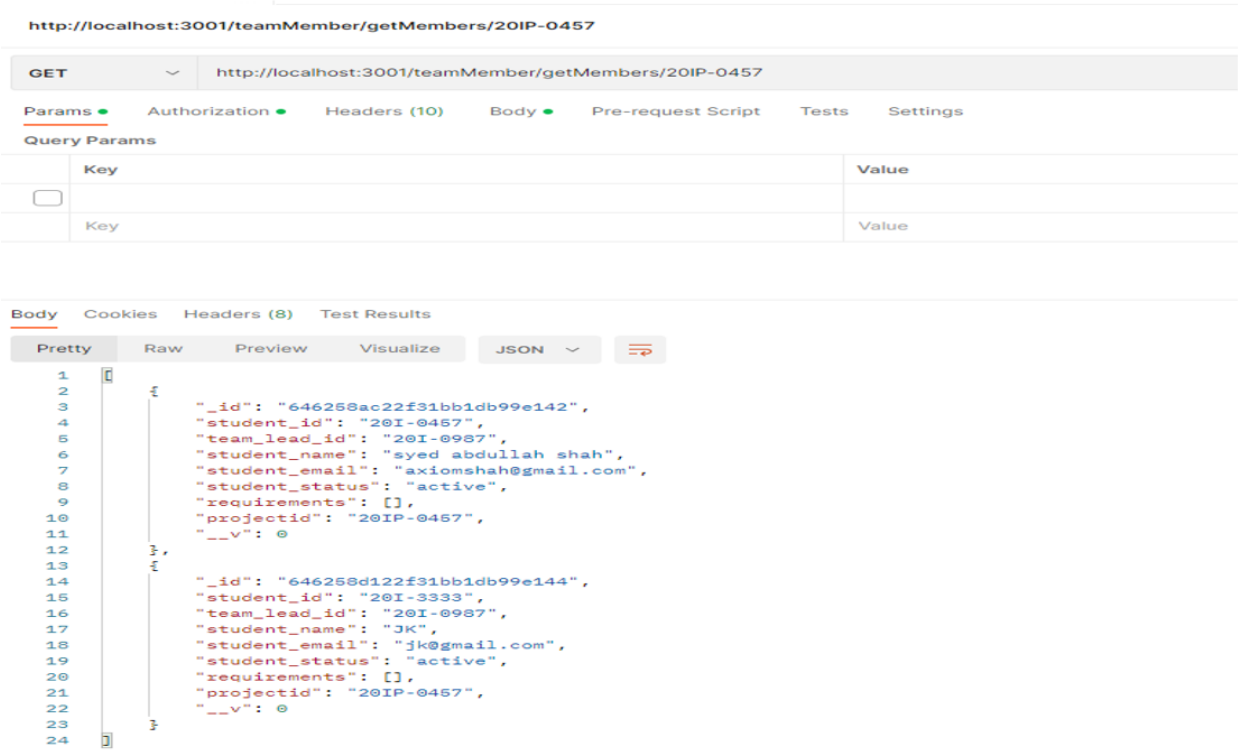


Team Member Module

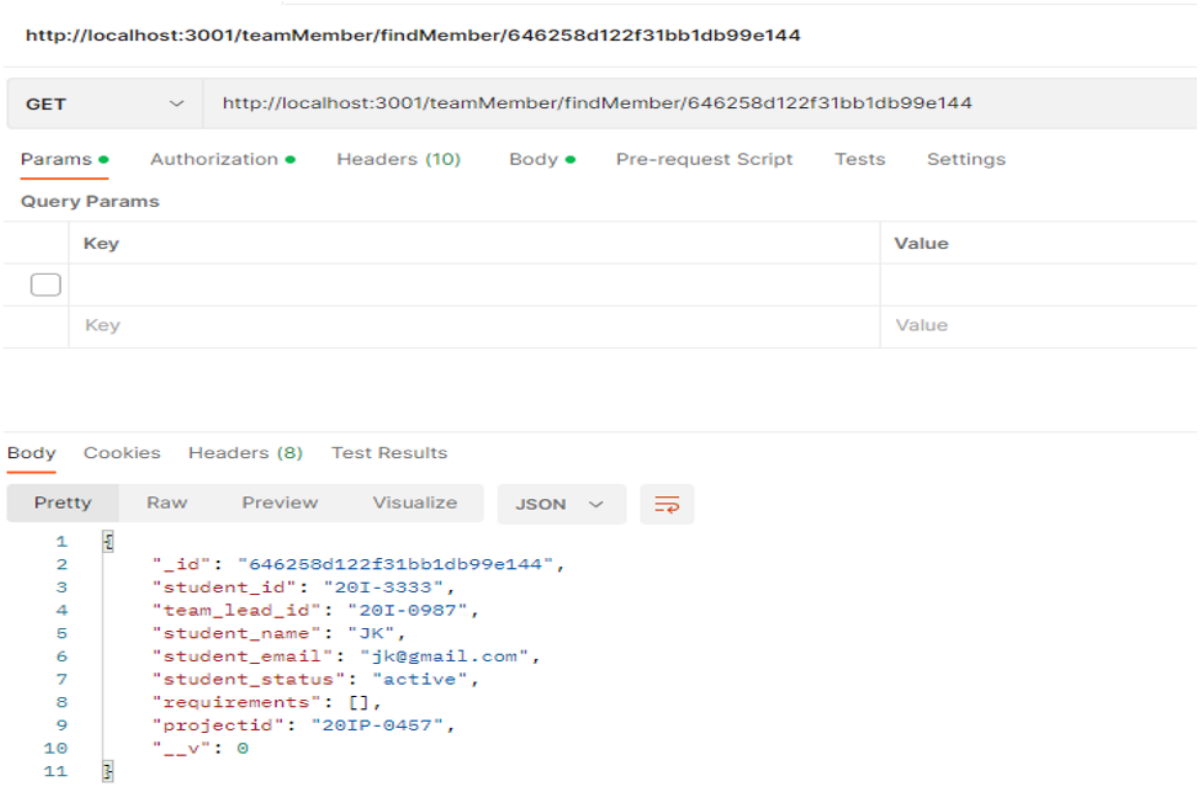
Adding team member



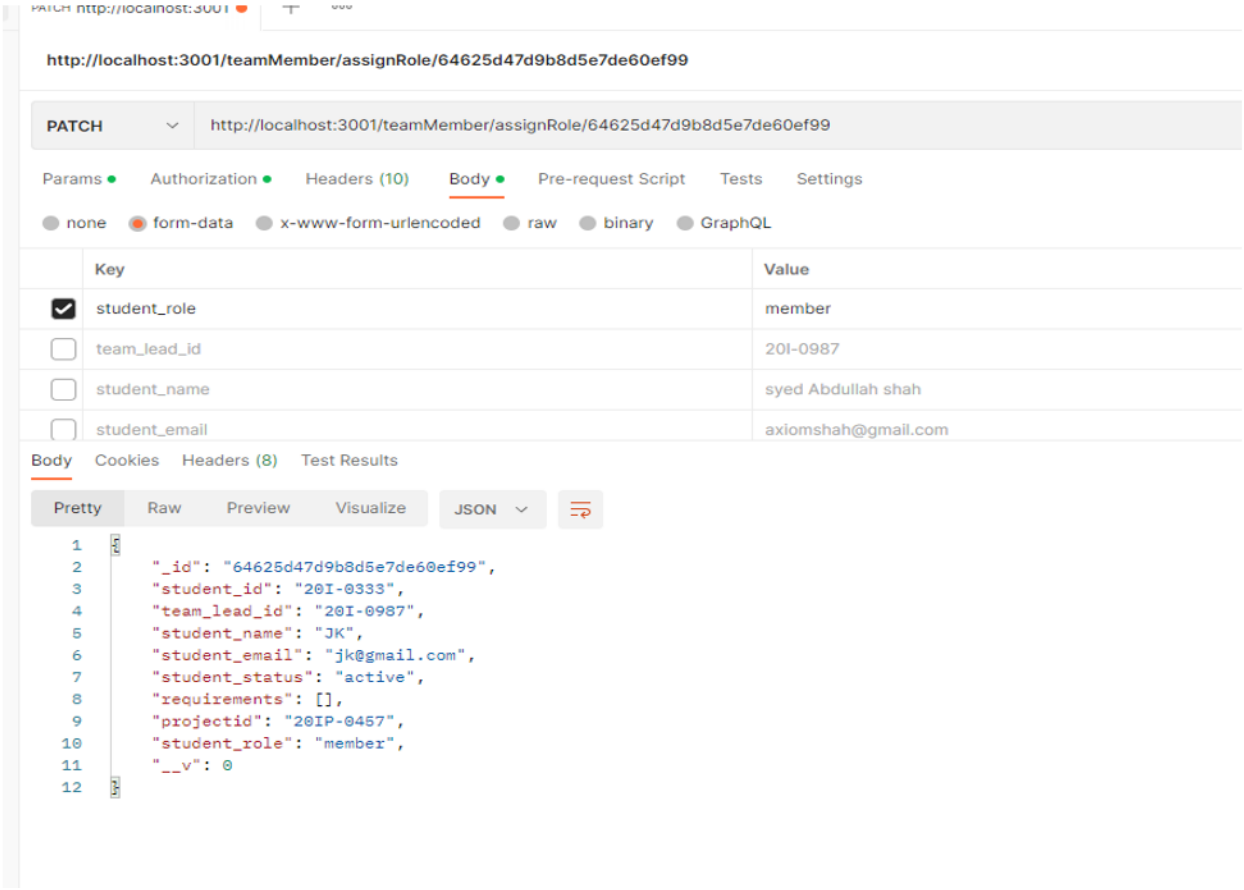
Getting all team member by Projectid



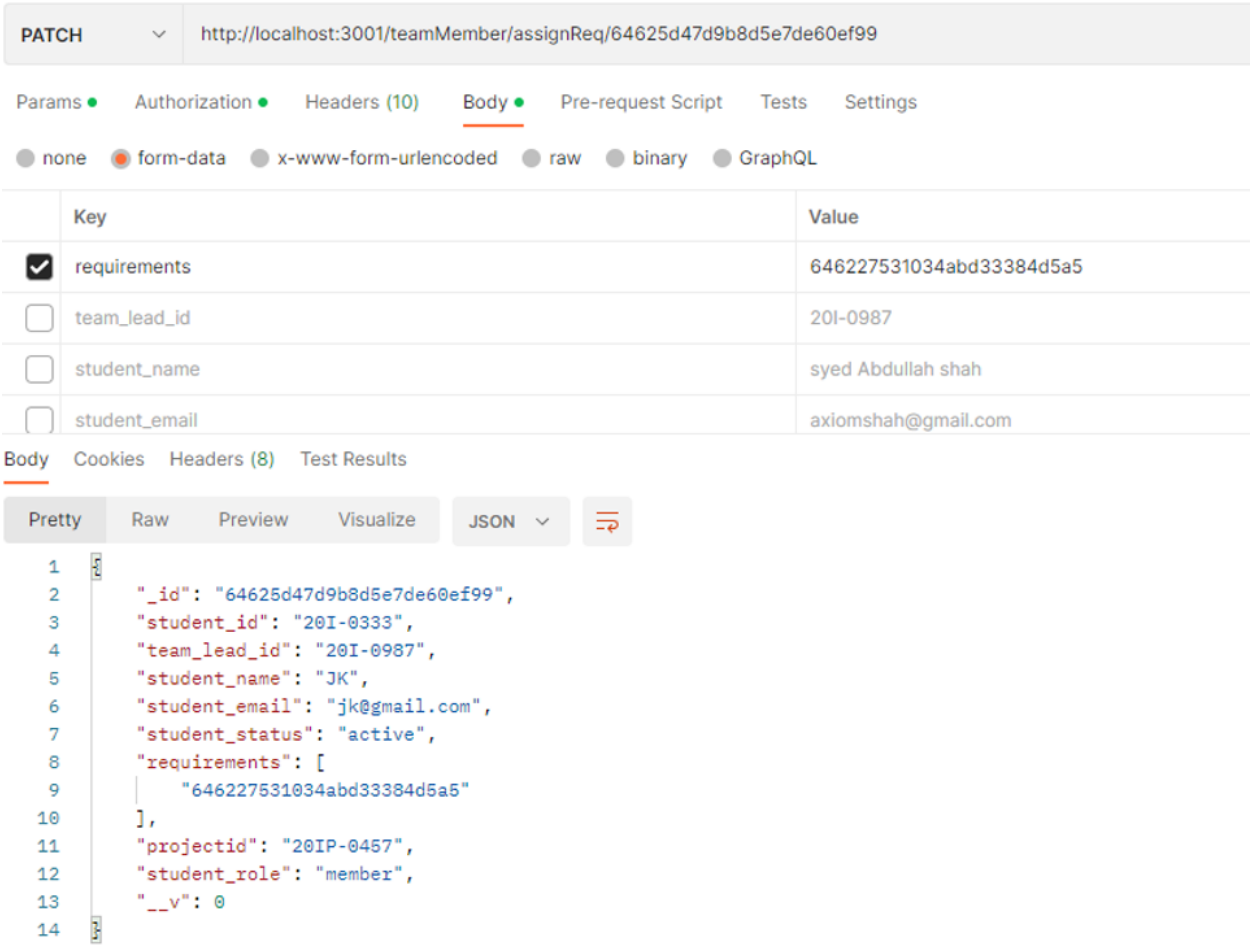
Getting a single team member by id



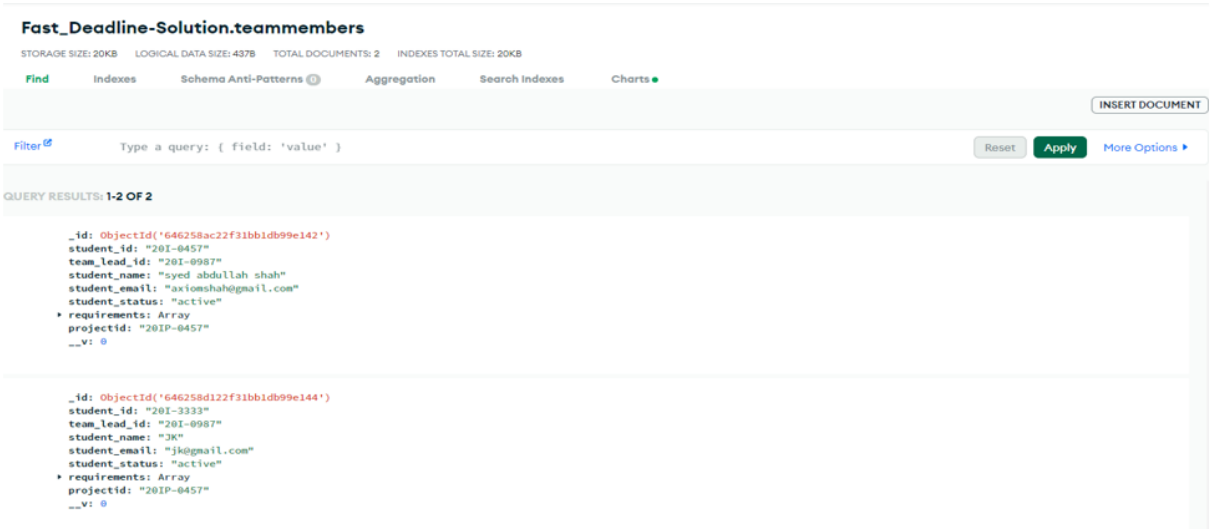
Updating the role of a team member

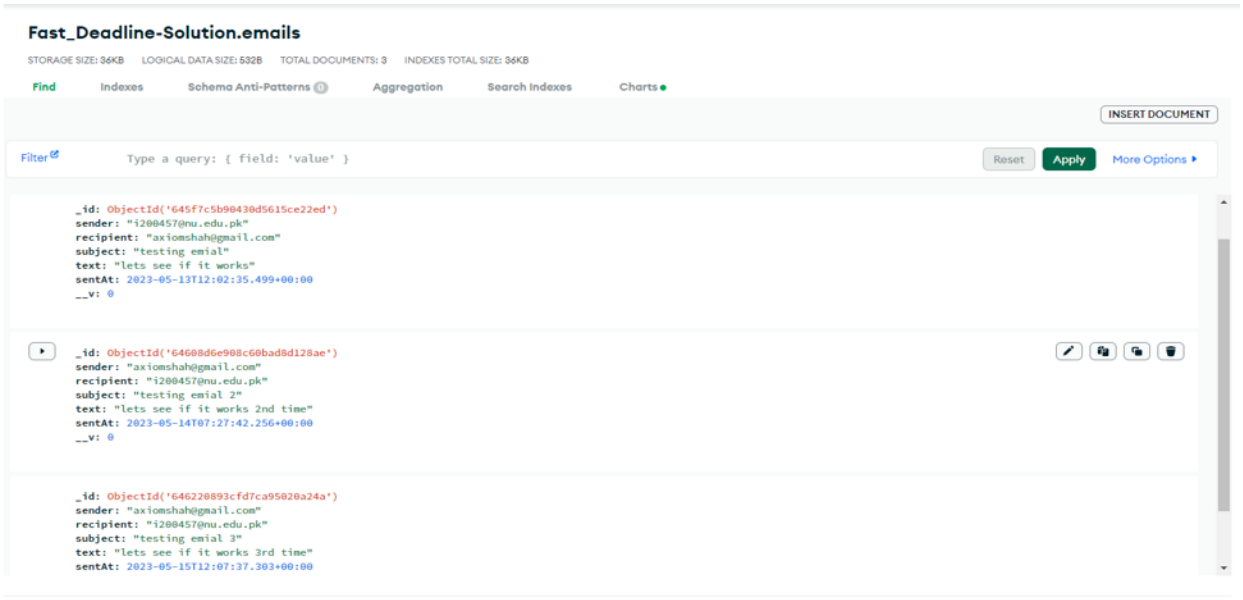

PATCH http://localhost:3001
http://localhost:3001/teamMember/assignRole/64625d47d9b8d5e7de60ef99
PATCH
http://localhost:3001/teamMember/assignRole/64625d47d9b8d5e7de60ef99
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
student_role
member
team_lead_id
20I-0987
student_name
syed Abdullah shah
student_email
axiomshah@gmail.com
Body
Cookies
Headers (8)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"_id": "64625d47d9b8d5e7de60ef99",
"student_id": "20I-0333",
"team_lead_id": "20I-0987",
"student_name": "JK",
"student_email": "jk@gmail.com",
"student_status": "active",
"requirements": [],
"projectid": "20IP-0457",
"student_role": "member",
"__v": 0
}

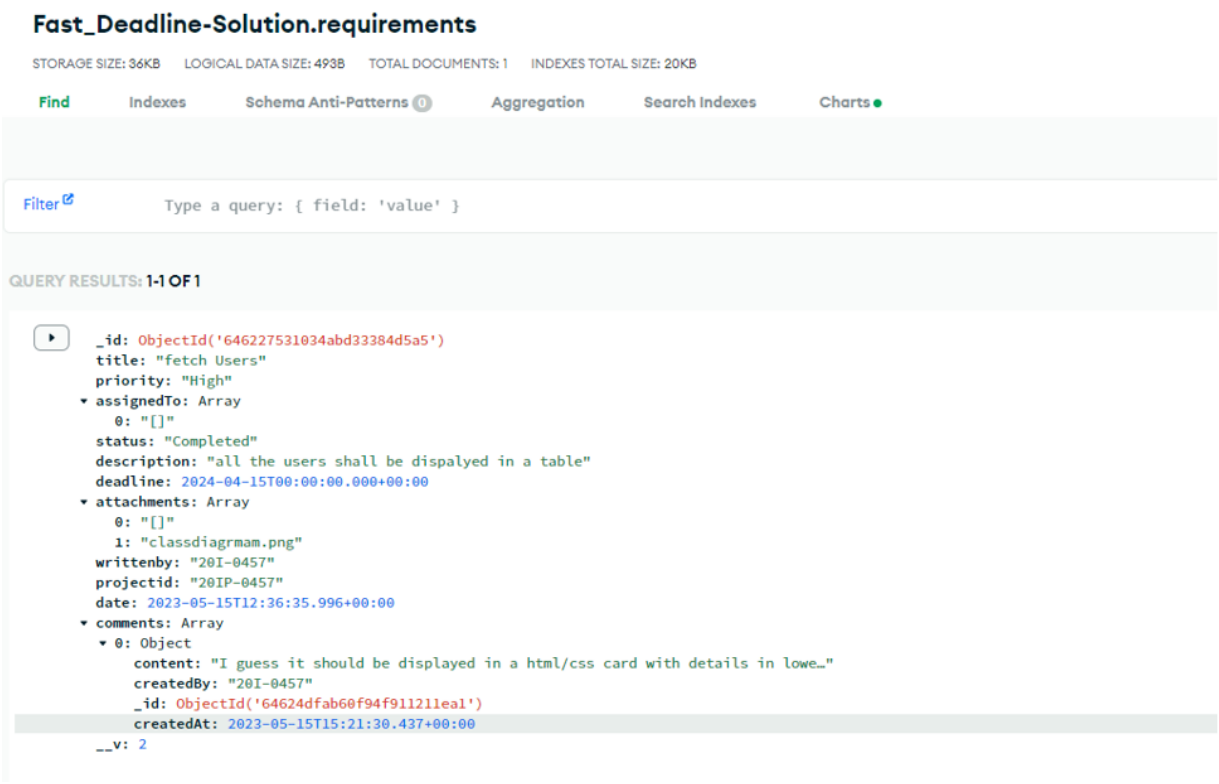
Assigning a requriement to a team member



DatabaseMongodb


Fast_Deadline-Solution.teammembers
STORAGE SIZE: 20KB LOGICAL DATA SIZE: 437B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 20KB
Find Indexes Schema Anti-Patterns Aggregation Search Indexes Charts
INSERT DOCUMENT
Filter Type a query: { field: 'value' } Reset Apply More Options
QUERY RESULTS: 1-2 OF 2
_id: ObjectId('646258ac22f31bb1db99e142')
student_id: "20I-0457"
team_lead_id: "20I-0987"
student_name: "syed abdullah shah"
student_email: "axiomshah@gmail.com"
student_status: "active"
requirements: Array
projectid: "20IP-0457"
__v: 0
_id: ObjectId('646258d122f31bb1db99e144')
student_id: "20I-3333"
team_lead_id: "20I-0987"
student_name: "JK"
student_email: "jk@gmail.com"
student_status: "active"
requirements: Array
projectid: "20IP-0457"
__v: 0


Fast_Deadline-Solution.emails
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 532B TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 36KB
Find Indexes Schema Anti-Patterns Aggregation Search Indexes Charts
INSERT DOCUMENT
Filter Type a query: { field: 'value' } Reset Apply More Options
_id: ObjectId('645f7c5b90430d5615ce22ed')
sender: "i200457@nu.edu.pk"
recipient: "axiomshah@gmail.com"
subject: "testing emial"
text: "lets see if it works"
sentAt: 2023-05-13T12:02:35.499+00:00
__v: 0
_id: ObjectId('64608d6e908c60bad8d128ae')
sender: "axiomshah@gmail.com"
recipient: "i200457@nu.edu.pk"
subject: "testing emial 2"
text: "lets see if it works 2nd time"
sentAt: 2023-05-14T07:27:42.256+00:00
__v: 0
_id: ObjectId('646220893cfd7ca95020a24a')
sender: "axiomshah@gmail.com"
recipient: "i200457@nu.edu.pk"
subject: "testing emial 3"
text: "lets see if it works 3rd time"
sentAt: 2023-05-15T12:07:37.303+00:00



Frontend Screens

Requirements Display

The Requirement Screen is a user-friendly interface designed to manage and update requirements. It features a table with columns for Deadline, Written By, Assigned To, Title, Priority, and Status. Users can easily view and track requirements using this format.

The table includes a Delete button, allowing users to remove requirements from the screen. This button ensures flexibility in requirement management.

An Update button is available, which opens a right-side bar or modal. This interface allows users to modify requirement details, such as the deadline, assignment, title, priority, and status. It provides a convenient way to update requirements without leaving the screen.

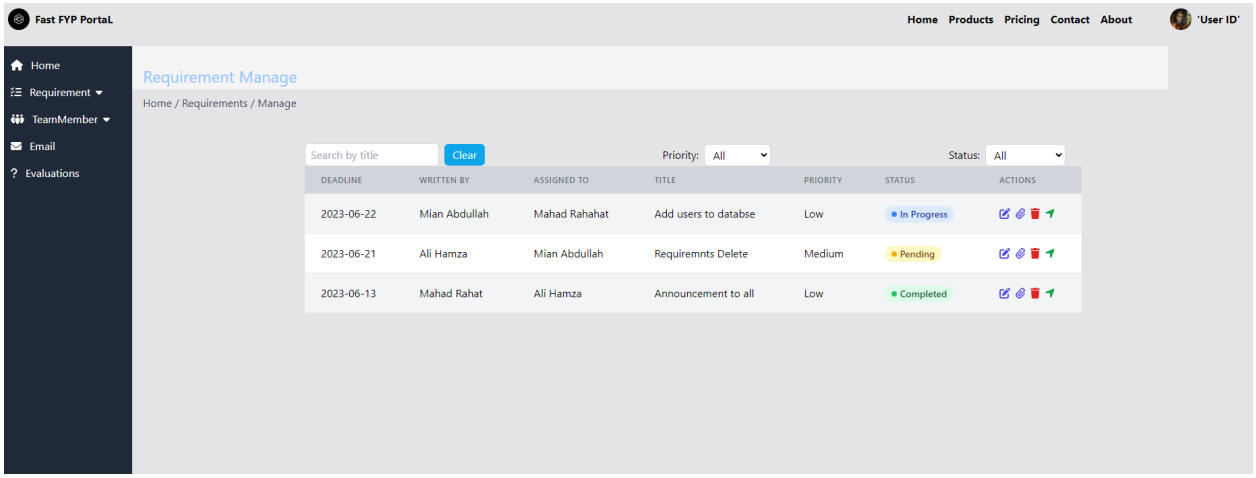
Additionally, there is an Add Attachments button, enabling users to attach files or documents related to a requirement. This feature enhances the documentation and organization of requirement-related information.

To facilitate efficient requirement management, a search bar is included. Users can enter keywords or phrases to search for specific requirements based on their titles or descriptions.

Filters are provided for further convenience. Users can filter requirements based on their status, such as In Progress, Pending, or Completed. Priority filters are also available, allowing users to focus on High, Medium, or Low priority requirements.

The Requirement Screen offers a comprehensive solution for managing requirements. It provides a visually organized and accessible table format, with buttons for deleting, updating, and attaching files. The search bar and filters enhance usability and enable users to locate and track requirements efficiently.

Overall, the Requirement Screen optimizes the requirement management process, streamlining tasks and providing a user-friendly experience for users.



Requirement Add

The requirement form allows users to add requirements with the following attributes:

Title: Users can provide a title for the requirement.

Priority: The requirement can be assigned a priority level, choosing from options such as Low, Medium, or High.

Assigned To: Users can assign the requirement to one or more individuals.

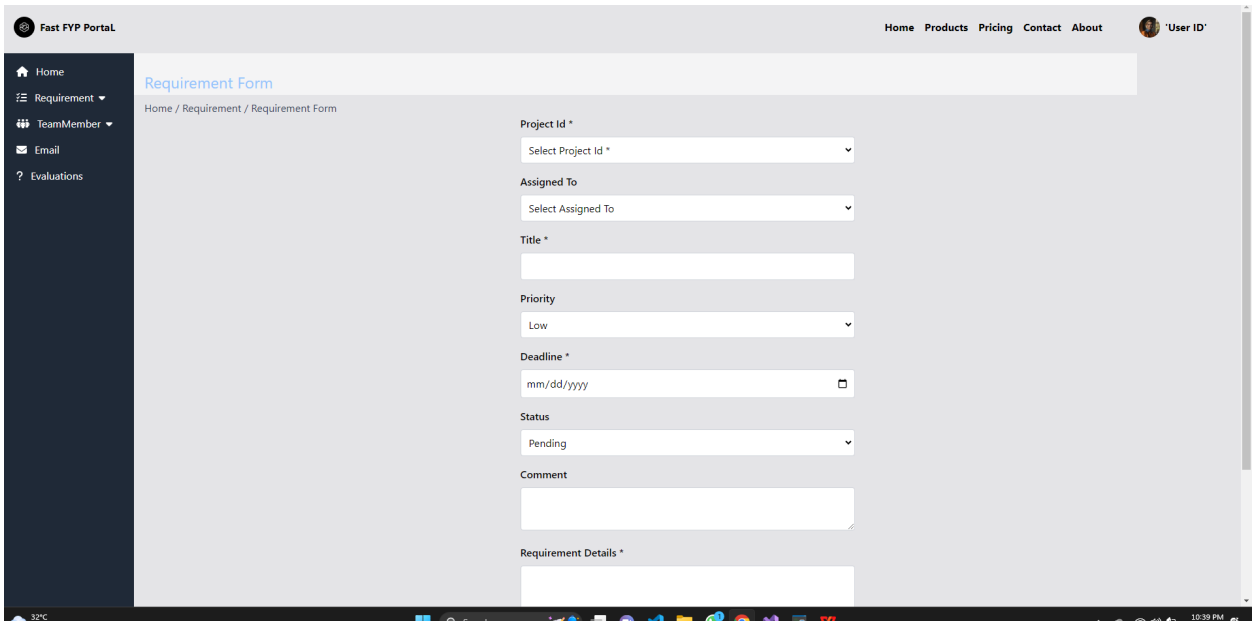
Status: The requirement can have a status, categorized as Pending, In Progress, or Completed.

Date: The date when the requirement was added is automatically recorded.

Description: Users can provide a detailed description of the requirement.

Deadline: The requirement must have a specified deadline.

Attachments: Users can attach files or documents related to the requirement.



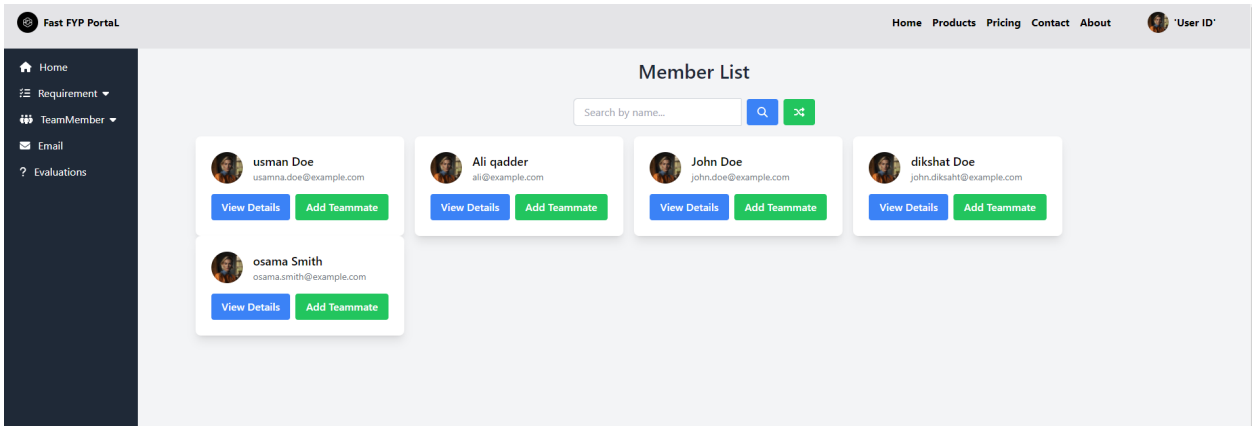
Team Member Display

The User Card Screen is designed to display user cards, each containing a picture, email, and username. The screen provides two buttons: "View Details" and "Add Teammate". The "View Details" button allows users to access comprehensive information about the user, while the "Add Teammate" button enables users to add the user to their team.

Additionally, the screen features a search bar that allows users to search for specific users by name or any other relevant criteria. This functionality enhances the user's ability to find and connect with specific individuals.

To enhance user interaction, a "Shuffle" button is provided. When clicked, it displays the next 20 users on the screen, providing variety and avoiding stagnation in user card display.

Overall, the User Card Screen provides a visually appealing and user-friendly interface for viewing user information. The combination of user cards, "View Details" and "Add Teammate" buttons, search functionality, and the ability to shuffle cards ensures a seamless user experience in managing and connecting with team members.



Manage Teammates

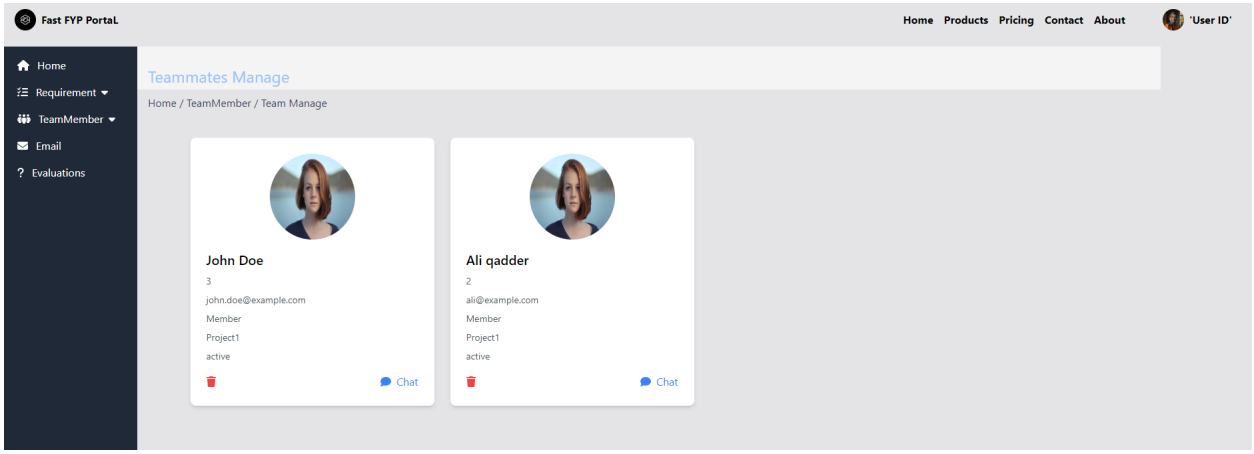
The Teammates Screen displays a list of your team members along with their details. Each team member is represented by a card, showcasing information such as their name, email, role, and other relevant details.

The screen provides an option to delete team members, allowing you to remove individuals from your team as needed. This feature ensures flexibility in managing and maintaining your team roster.

By visually presenting team member details in a structured manner, the screen allows you to quickly view important information about each teammate. This aids in better collaboration and communication within the team.

The deletion option offers a streamlined way to remove team members who may no longer be part of your team, ensuring accurate and up-to-date team records.

Overall, the Teammates Screen offers an organized and efficient interface for managing your team. It allows you to easily access teammate details and provides the necessary functionality to remove individuals from your team when necessary, promoting effective team management and collaboration.



Emails

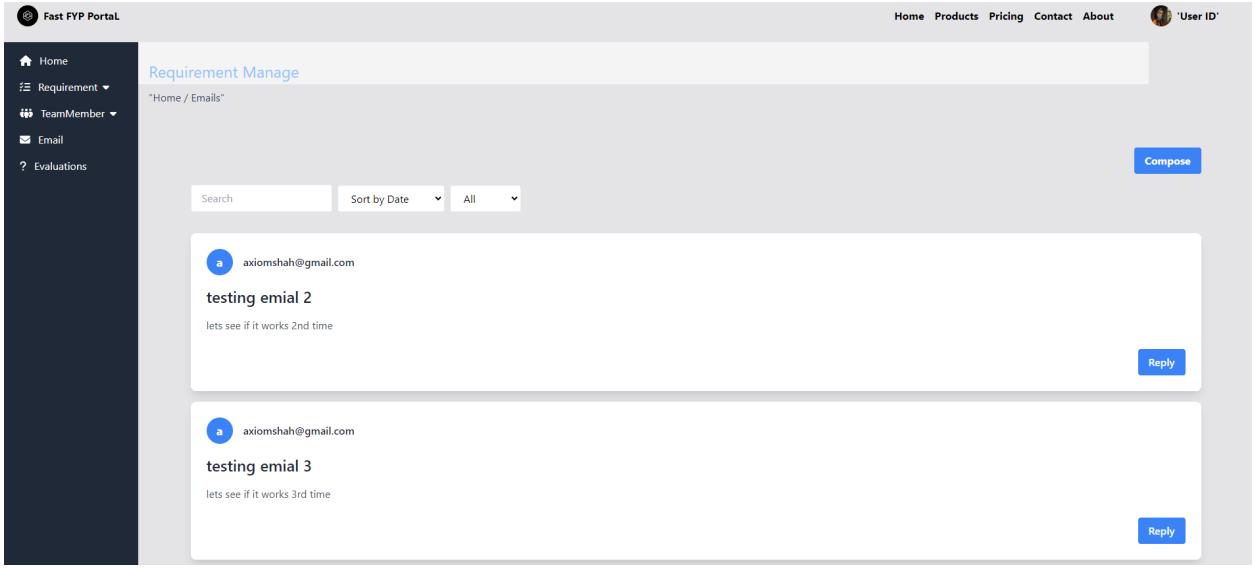
The Email Inbox Screen presents emails in row cards, providing details such as the sender's name, subject, and a brief preview of the email content. Each email card also includes a "Reply" button, allowing users to respond to the sender with the same subject line.

In addition to the email cards, the screen features a "Compose" button. When clicked, it opens a small screen or modal where users can send emails to anyone. The compose screen includes fields to specify the recipient(s), subject, and the body of the email.

This design provides an organized and user-friendly interface for managing emails. The email cards allow users to quickly scan through their inbox and view essential information about each email. The "Reply" button simplifies the process of replying to a specific email while keeping the subject intact.

The "Compose" button offers a convenient way to create new emails. By opening a separate screen or modal, users can focus on composing their message without distractions. The provided fields for recipient(s), subject, and email body ensure a comprehensive and efficient email composition experience.

Overall, the Email Inbox Screen optimizes email management by providing an intuitive interface for viewing and replying to emails, along with a dedicated compose feature for sending new messages.



Evaluation View

The Evaluation Screen showcases the evaluations of the FYP (Final Year Project) project in a user-friendly manner for a sample Project. The screen consists of cards that present the evaluation criteria and their corresponding details.

The evaluation criteria include:

Innovation: This criterion assesses the level of creativity and originality in the project (Weightage: 20%, Score: 8).

Technical Complexity: It evaluates the technical intricacy and complexity of the project (Weightage: 30%, Score: 7).

Implementation: This criterion measures the execution and implementation of the project (Weightage: 20%, Score: 9).

Documentation: It examines the quality and comprehensiveness of the project documentation (Weightage: 15%, Score: 8).

Collaboration: This criterion evaluates the level of teamwork and collaboration within the project team (Weightage: 10%, Score: 9).

Timeliness: It assesses the adherence to project timelines and deadlines (Weightage: 5%, Score: 7).

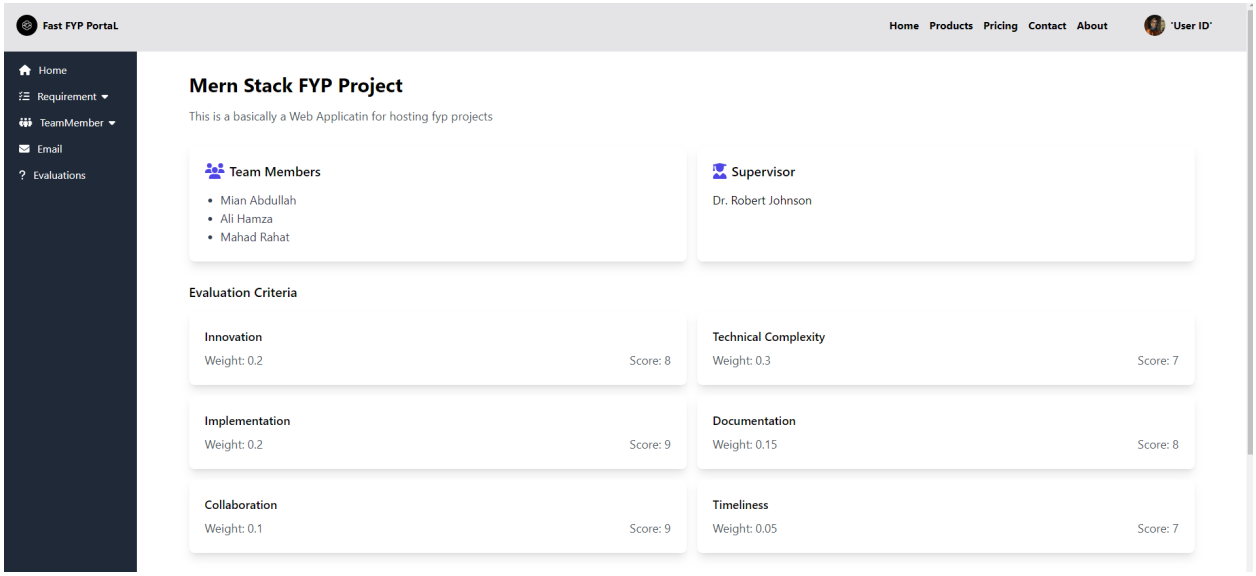
Additionally, the screen displays scores for the presentation, divided into three categories: content (Score: 8), delivery (Score: 9), and visuals (Score: 7).

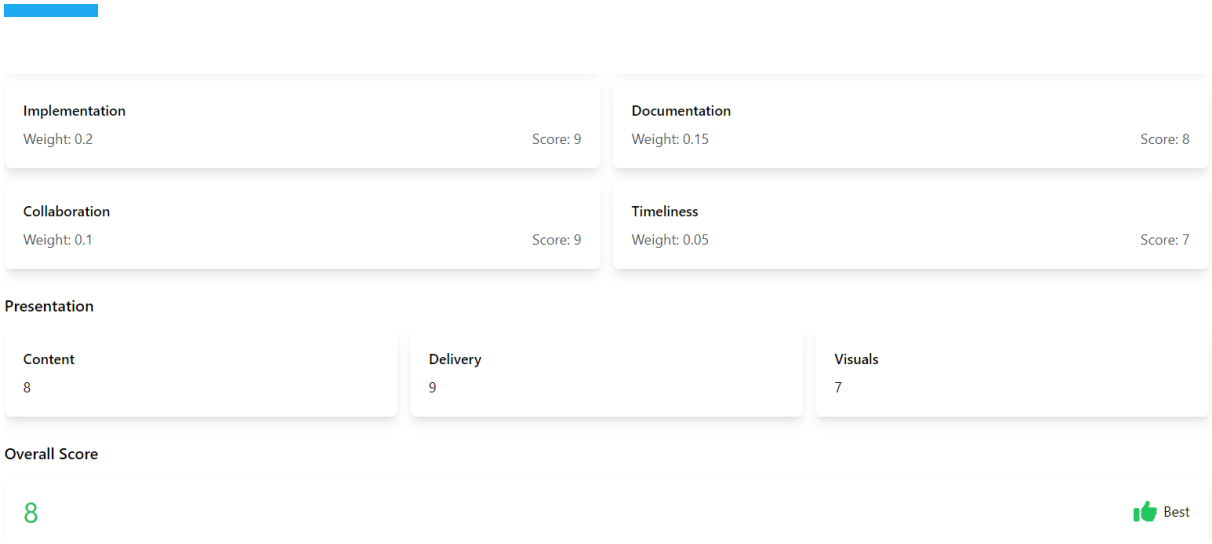
Based on the evaluation scores, an overall score is calculated. If the overall score is considered good, a thumb-up icon is displayed. However, if the overall score falls below a certain threshold, a thumb-down icon is shown to indicate a lower evaluation.

The Evaluation Screen provides a comprehensive overview of the project's performance across various evaluation criteria. The card-based layout facilitates easy comprehension and quick assessment of each criterion's weightage and score.

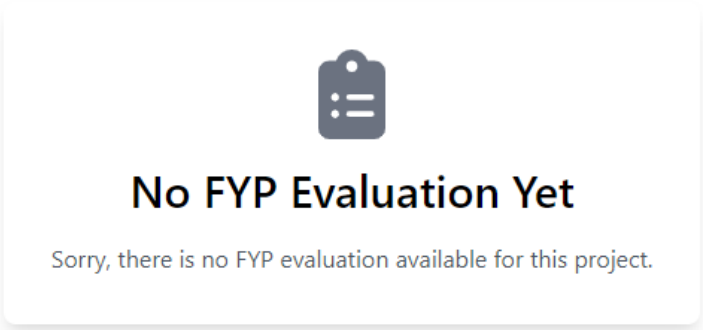
By visually indicating the overall evaluation result with a thumb-up or thumb-down icon, the screen allows users to quickly identify the project's performance level. This visual representation enhances decision-making and provides a clear assessment of the project's evaluation results.

Overall, the Evaluation Screen optimizes the evaluation process by presenting evaluation criteria and their scores in a clear and accessible manner, ensuring efficient assessment and evaluation of the FYP project.

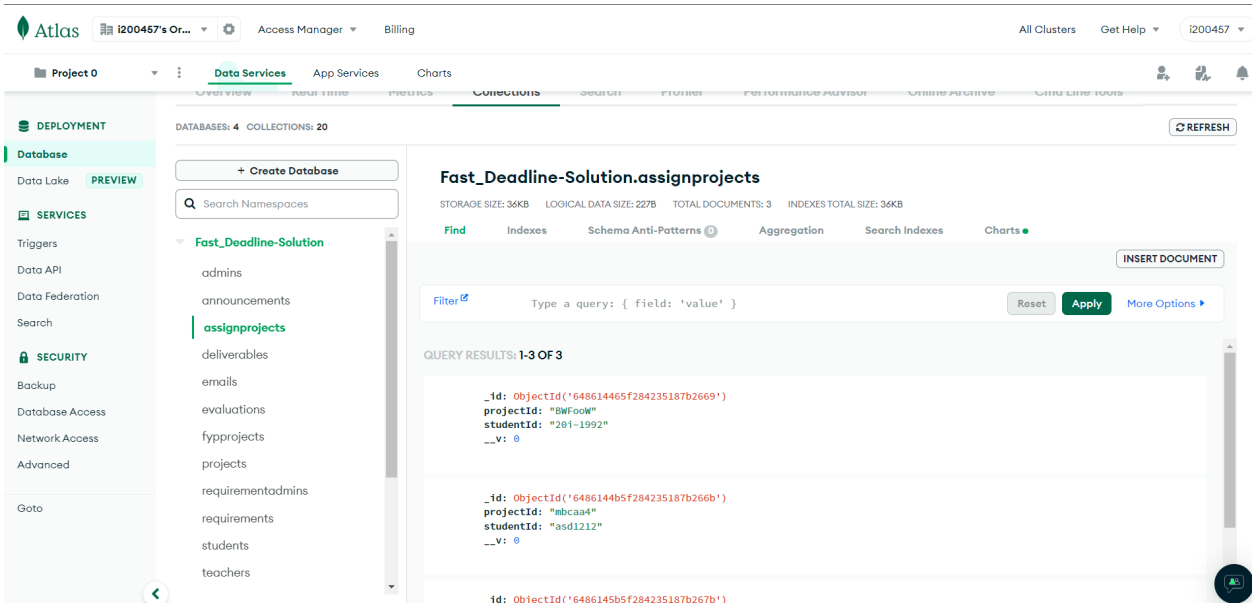




Also it shows a no fyp evaluation if it has not been given by the teacher or FYP Panel.



Database Schemas



The group utilizes a unified database for their operations.

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const assignProjectSchema = new Schema ({

    projectId: {
        type: String,
        require: true
    }
    ,
    studentId: {
        type: String,
        require: true
    }
})

const AssignProject = mongoose.model("AssignProject",
assignProjectSchema);
module.exports = AssignProject;
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const ProjectSchema = new Schema({


  id: {
    type: String,
    required: true,
  },
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  file: {
    type: String,
    required: true,
  },
  startDate: {
    type: Date,
    require: true,
  },
  dueDate: {
    type: Date,
    require: true,
  }
});

const Project = mongoose.model("Projects", ProjectSchema);
module.exports = Project;
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const EvaluationSchema = new Schema({
  projectId: {
    type: String,
    required: true,
  },
  studentId: {
    type: String,
    required: true,
  },
  criteria: {
    type: String,
    required: true,
  },
  totalScore: {
    type: Number,
    required: true,
  },
  obtainedScore: {
    type: Number,
    required: true,
  },
  comments: {
    type: String,
    required: true,
  },
  evaluationDate: {
    type: Date,
    default: Date.now,
  },
  progress: {
    type: Number,
    required: true,
  },
});

const Evaluation = mongoose.model("Evaluation", EvaluationSchema);
module.exports = Evaluation;
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const User = require("./userModel"); // Import the User schema file
const Project = require("./projectModel");

const DeliverableSchema = new Schema({

  id: {
    type : String,
    required: true,
  }
  ,
  project: {
    type: Schema.Types.ObjectId,
    ref: "Project",
    required: true,
  },
  title: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  files: [{
    type: String,
    required: true,
  }],
  authors: [{
    type: Schema.Types.ObjectId,
    ref: "User",
    required: true,
  }],
  creationDate: {
    type: Date,
    default: Date.now,
  },
  lastModifiedDate: {
    type: Date,
    default: Date.now,
  },
  status: {
    type: String,
    required: true,
  },

});

const Deliverable = mongoose.model("Deliverable", DeliverableSchema);
module.exports = Deliverable;
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const adminSchema = new Schema({
    userId: {
      type: Schema.Types.ObjectId,
      ref: 'users',
      required: true
    }

});
const admin = mongoose.model("admin", adminSchema);
```

```
//create schema for themes
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const themeSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    color: {
        type: String,
        required: true
    },
    themeId: {
        type: String,
        required: true
    },

});

module.exports = mongoose.model("theme", themeSchema);
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const ProjectSchema = new Schema({
  title: {
    type: String,
    required: true,
  },
  id: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  file: {
    type: String,
    required: true,
  },
  startDate: {
    type: Date,
    require: true,
  },
  dueDate: {
    type: Date,
    require: true,
  }
});

const Project = mongoose.model("Projects", ProjectSchema);
module.exports = Project;
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const assignProjectSchema = new Schema ({

    projectId: {
        type: String,
        require: true
    }
    ,
    studentId: {
        type: String,
        require: true
    }
})

const AssignProject = mongoose.model("AssignProject",
assignProjectSchema);
module.exports = AssignProject;
```

```
//make schema
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const announcementsSchema = new Schema({


    title: {
        type: String,
        required: true
    },

    message: {
        type: String,
        required: true
    },
});
module.exports = mongoose.model("announcement", announcementsSchema);
```

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

const requirementSchema = new mongoose.Schema({
  maxGroups: {
    type: String,
    required: true,
  },
  maxStudentsPerGroup: {
    type: String,
    required: true,
  },
  topicRequirements: {
    type: String,
    required: true,
  },

});

const RequirementAdd = mongoose.model('RequirementAdmin',
requirementSchema);

module.exports = RequirementAdd;
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const studentSchema = new Schema({
    userId: {
      type: Schema.Types.ObjectId,
      ref: 'users',
      required: true
    },

    teamlead:{
        type: Boolean,
        default: false
    },


});
module.exports = mongoose.model("student", studentSchema);
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;


const teacher = new Schema({
    userId: {
        type: Schema.Types.ObjectId,
        ref: 'users',
        required: true
    },

    designation: {

        type: String,
        required: true
    },

});
module.exports = mongoose.model("teacher", teacher);
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;
const UserSchema = new Schema ({
    userId: {
        type: String,
        required: true
    },
     name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    contact: {

        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
    gender: {
        type: String,
        default: "Male"
    },
    theme: {
        color:{
            type: String,
            default:"white"
        },
        name:{
            type: String,
            default:"light"
        }
    },
    registerAs:{

        type: String,
        enum: ['student', 'teacher', 'admin'],
        required: true
    }
});

module.exports = mongoose.model("users", UserSchema);
```

Email Schema

```
const mongoose = require('mongoose');  841.1k (gzipped: 227.5k)

const emailSchema = new mongoose.Schema({
  sender: {
    type: String,
    required: true,
  },
  recipient: {
    type: String,
    required: true,
  },
  subject: {
    type: String,
    required: true,
  },
  text: {
    type: String,
    required: true,
  },
  sentAt: {
    type: Date,
    default: Date.now,
  },
});

const Email = mongoose.model('Email', emailSchema);

module.exports = Email;
```

Requirement Schema

```
const mongoose = require('mongoose');  841.1k (gzipped: 227.5k)

const commentSchema = new mongoose.Schema({
  content: {
    type: String,
    required: true,
  },
  createdBy: {
    type: String,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
});

const requirementSchema = new mongoose.Schema({


  title: {
    type: String,
    required: true,
  },
  priority: {
    type: String,
    enum: ['Low', 'Medium', 'High'],
    default: 'Low',
  },
  assignedTo: {
    type: Array,
    default:"[]"
  },
  status: {
    type: String,
    enum: ['Pending', 'In Progress', 'Completed'],
    default: 'Pending',
  },
```

```
  date: {
    type: Date,
    default: Date.now,
  },
  description: {
    type: String,
    required: true,
  },
  deadline: {
    type: Date,
    required: true,
  },
  attachments: {
    type: Array,
    default:"[]"
  },
  comments:{
        type: [commentSchema],

  } ,
  writtenby: {//roll number
    type: String,
    required: true,
  },
  projectid: {// unique project id
    type: String,
    required: true,
  }
);

const Requirement = mongoose.model('Requirement', requirementSchema);

module.exports = Requirement;
```

Team Member Schema

```
const mongoose = require('mongoose');  841.1k (gzipped: 227.5k)

const teamMemberSchema = new mongoose.Schema({
  student_id: {
    type: String,
    required: true,
  },
  team_lead_id: {
    type: String,
    required: true,
  },
  student_name: {
    type: String,
    required: true,
  },
  student_email: {
    type: String,
    required: true,
    unique: true,
  },
  student_status: {
    type: String,
    required: true,
  },
  requirements: {
      type: Array,
    },
    projectid: {
      type: String,
      required: true,
    },
    student_role: {
      type: String,
      default:"Member"
    },
```