

# Final Project

# Web Engineering

Sir Zaheer Sani

**Ali Hamza 20i1881**

**Mahad Rahat 20i1808**

**Syed Muhammad Mian Abdullah 20i0457**

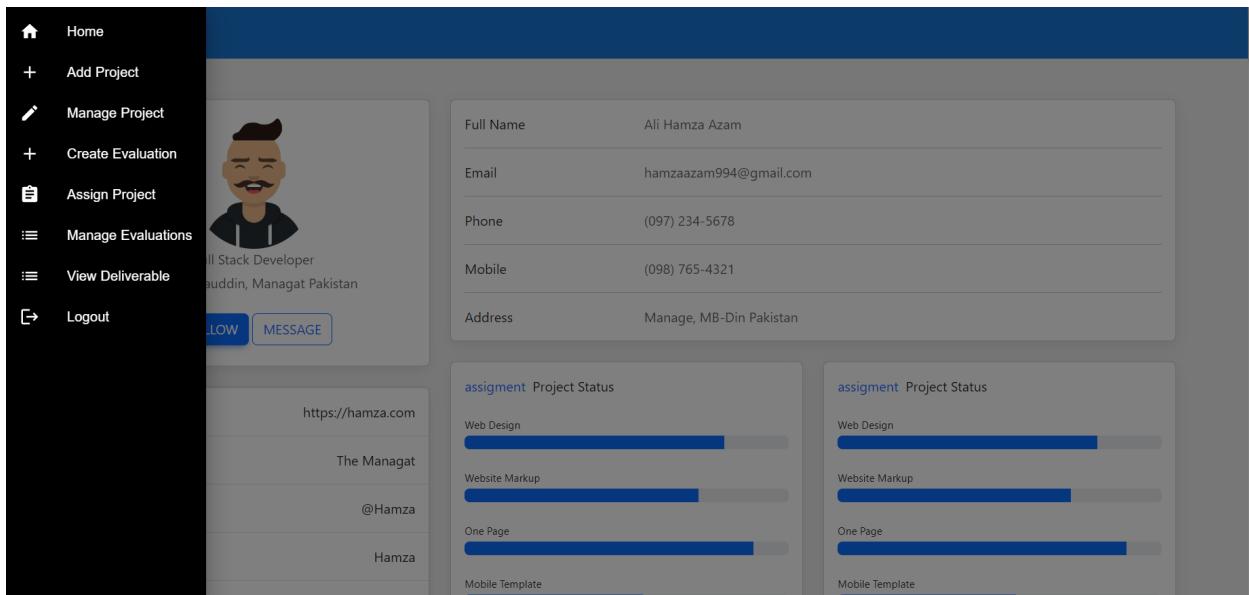
11 June, 2023

## Work Division

Ali Hamza 20i1881

## UI Screen Shots

### Home Page:



## Create Project:

☰ DevOps

### Create Project...

Title

ID

Description

File

Start Date

Due Date

## Manage Projects:

☰ DevOps

### Project List

Search Projects

<b>DevOps Project</b> This is simple testing Start Date: 2023-06-06T00:00:00.000Z Due Date: 2023-06-15T00:00:00.000Z <input type="button" value="EDIT"/> <input type="button" value="DELETE"/>	<b>Computer Net FYP</b> this is Cnet Start Date: 2023-06-15T00:00:00.000Z Due Date: 2023-06-29T00:00:00.000Z <input type="button" value="EDIT"/> <input type="button" value="DELETE"/>
<b>Block chain</b> this is database Start Date: 2023-06-08T00:00:00.000Z Due Date: 2023-06-24T00:00:00.000Z <input type="button" value="EDIT"/> <input type="button" value="DELETE"/>	<b>Web Eng Project</b> Course project Start Date: 2023-06-08T00:00:00.000Z Due Date: 2023-06-15T00:00:00.000Z <input type="button" value="EDIT"/> <input type="button" value="DELETE"/>
<b>Database Project</b>	

## Edit Projects:

The screenshot shows a 'Project List' interface with a modal dialog titled 'Edit Project'. The modal contains fields for 'Title' (Computer Net FYP), 'Description' (this is Cnet), 'Start Date' (mm/dd/yyyy), and 'Due Date' (mm/dd/yyyy). It includes 'CANCEL' and 'SAVE' buttons. In the background, the 'Project List' shows three projects: 'DevOps Project', 'Block chain', and 'Database Project', each with edit and delete buttons.

## Assign Project:

The screenshot shows a 'Project List' interface with an 'ASSIGN' button next to each project entry. The projects listed are 'DevOps Project', 'Computer Net FYP', 'Block chain', 'Web Eng Project', and 'Database Project'. Each project entry includes a brief description and an 'ASSIGN' button.

## Assign project to student:

The screenshot shows a web application interface for managing projects. On the left, there's a 'Project List' section with three items: 'DevOps Project', 'Web Eng Project', and 'Block chain'. The 'DevOps Project' item is selected, and its details are shown: 'This is simple testing' and a blue 'ASSIGN' button. A modal window titled 'Assign Project' is open over the list. It contains fields for 'Project:' (set to 'Computer Net FYP') and 'Student:' (a dropdown menu showing a list of students). The student 'Hamza' is highlighted in the dropdown list. At the bottom of the modal, there's a note: 'The Memorandums Powered by Hamza'.

## Display Assigned Project:

The screenshot shows a 'Assigned Projects' section. It displays three projects: 'Computer Net FYP', 'Web Eng Project', and 'DevOps Project'. Each project card includes a brief description, a list of assigned students, and a blue 'EVALUATE' button. The 'Computer Net FYP' card shows 'Assigned Students: Abdullah'. The 'Web Eng Project' card shows 'Assigned Students: Hamza'. The 'DevOps Project' card shows 'Assigned Students: Hammad'. At the bottom of the page, there's a footer note: '© 2023 My App. All rights reserved. The Memorandums Powered by Hamza'.

## Evaluate Assigned Projects:

The screenshot shows a modal window titled "Project Evaluation" overlaid on a main dashboard. The dashboard header says "Assigned Projects" and has sections for "Computer Net FYP" and "DevOps Project".

**Project Evaluation Modal Fields:**

- Project ID: 6486144b5f284235187b266b
- Student ID: asd1212
- Criteria: (input field)
- Total Score: (input field)
- Obtained Score: (input field)
- Comments: (input field)
- Evaluation Date: mm/dd/yyyy (input field)
- Progress: 50% (progress bar)

Buttons at the bottom: CANCEL (gray) and SAVE (blue).

## View Evaluations:

The screenshot shows a list of evaluations with two entries.

ID	Criteria	Total Score	Obtained Score	Comments	Evaluation Date
64818306b43f0969a7f252e1	no	19	16	Good work	2024-06-08T00:00:00.000Z
6480cec4f320830725036867	no	10	5	comment	2024-06-15T00:00:00.000Z

Each entry includes a progress bar and buttons for DELETE (red) and EDIT (blue).

## Deliverables view:

The screenshot shows a list of deliverables. The first two items are grouped under a header 'DevOps' and the last two under 'Web 1st'. Each item has a 'VIEW FILES' and 'DELETE' button.

Category	Name	Description	Authors	Creation Date	Last Modified	Status	Actions
DevOps	DevOps	This is a sample deliverable	Authors: 645fcf3b9edd862353e595e, 645fcf3b9edd862353e595e	Mon May 15 2023	Mon May 15 2023	Pending	<a href="#">VIEW FILES</a> <a href="#">DELETE</a>
	Web 1st	This is a sample deliverable	Authors: 645fcf3b9edd862353e595e, 645fcf3b9edd862353e595e	Mon May 15 2023	Mon May 15 2023	Pending	<a href="#">VIEW FILES</a> <a href="#">DELETE</a>
Web 1st	Web 1st	This is a sample deliverable	Authors: 645fcf3b9edd862353e595e, 645fcf3b9edd862353e595e	Mon May 15 2023	Mon May 15 2023	Pending	<a href="#">VIEW FILES</a> <a href="#">DELETE</a>
	DevOps	This is a sample deliverable	Authors: 645fcf3b9edd862353e595e, 645fcf3b9edd862353e595e	Mon May 15 2023	Mon May 15 2023	Pending	<a href="#">VIEW FILES</a> <a href="#">DELETE</a>

A modal window titled 'Files' is open, displaying two files: 'file1.pdf' and 'file2.docx'. A 'CLOSE' button is at the bottom right of the modal.

Category	Name	Description	Authors	Creation Date	Last Modified	Status	Actions
DevOps	DevOps	This is a sample deliverable	Authors: 645fcf3b9edd862353e595e, 645fcf3b9edd862353e595e	Mon May 15 2023	Mon May 15 2023	Pending	<a href="#">VIEW FILES</a> <a href="#">DELETE</a>
	Web 1st	This is a sample deliverable	Authors: 645fcf3b9edd862353e595e, 645fcf3b9edd862353e595e	Mon May 15 2023	Mon May 15 2023	Pending	<a href="#">VIEW FILES</a> <a href="#">DELETE</a>
Web 1st	Web 1st	This is a sample deliverable	Authors: 645fcf3b9edd862353e595e, 645fcf3b9edd862353e595e	Mon May 15 2023	Mon May 15 2023	Pending	<a href="#">VIEW FILES</a> <a href="#">DELETE</a>
	DevOps	This is a sample deliverable	Authors: 645fcf3b9edd862353e595e, 645fcf3b9edd862353e595e	Mon May 15 2023	Mon May 15 2023	Pending	<a href="#">VIEW FILES</a> <a href="#">DELETE</a>

- Sample request and response of the APIs

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

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON 

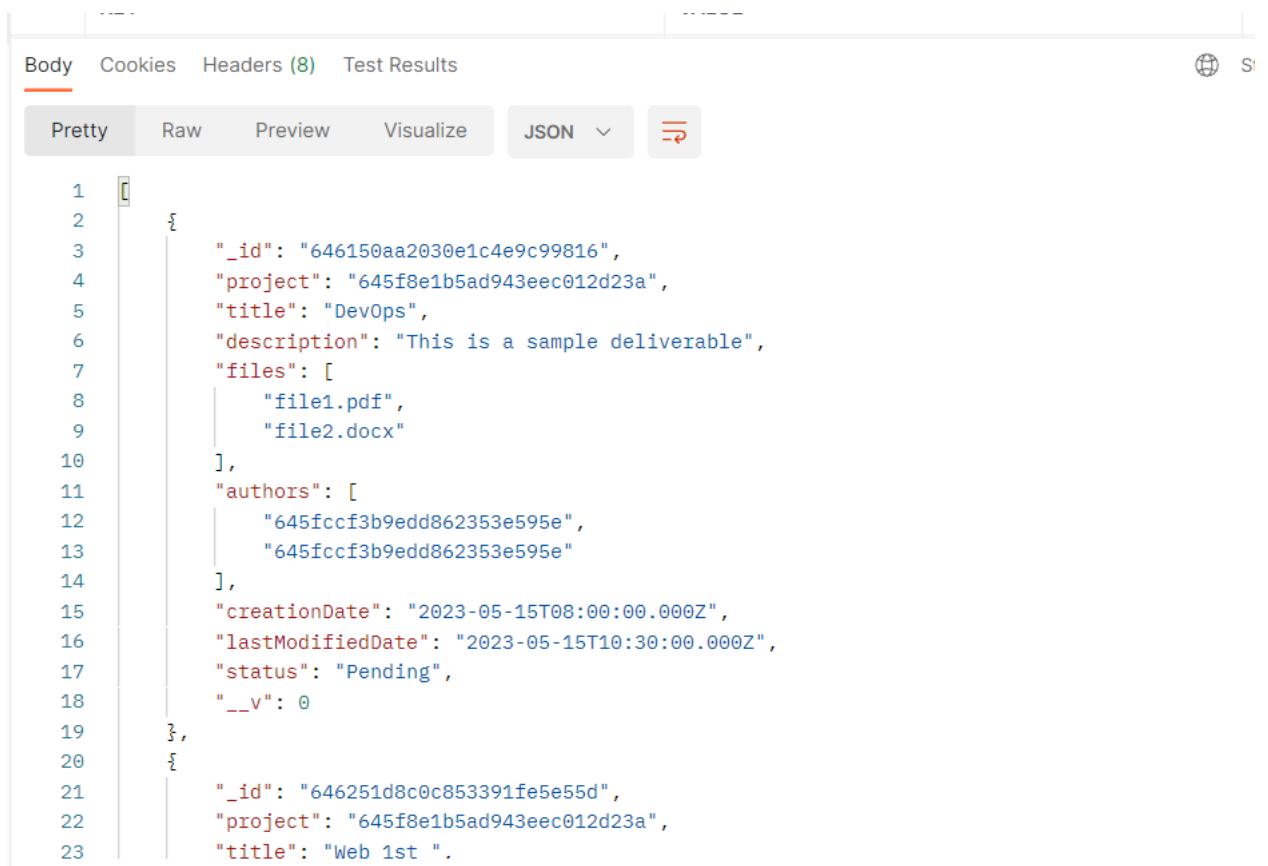
```
1 {  
2   "title": "OOP",  
3   "id": "111qqq",  
4   "description": "storing data into database",  
5   "file": "http/this",  
6   "startDate": "2023-05-09T19:00:00.000Z",  
7   "dueDate": "2023-05-19T19:00:00.000Z",  
8   "_id": "646249af35c8bcf81f5c7dab",  
9   "__v": 0
```

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

Pretty Raw Preview Visualize JSON 

```
1 {  
2   "id": "55pp",  
3   "project": "645f8e1b5ad943eec012d23a",  
4   "title": "Web 1st",  
5   "description": "This is a sample deliverable",  
6   "files": [  
7     "file1.pdf",  
8     "file2.docx"  
9   ],  
10  "authors": [  
11    "645fccf3b9edd862353e595e",  
12    "645fccf3b9edd862353e595e"  
13  ],  
14  "creationDate": "2023-05-15T08:00:00.000Z",  
15  "lastModifiedDate": "2023-05-15T10:30:00.000Z",  
16  "status": "Pending",  
17  "_id": "646252f77a9339920f35cb3d",  
18  "__v": 0  
19 }
```

### 3. <http://localhost:3001/deliverables/getAllDeliverables>



The screenshot shows a REST client interface with the following details:

- Headers (8)**: The number of headers present in the request.
- Pretty**: The current view mode for the JSON response.
- Raw**: An option to switch to raw text view.
- Preview**: An option to switch to a preview view.
- Visualize**: An option to switch to a visualization view.
- JSON**: The current view mode for the JSON response.
- Copy**: A button to copy the selected JSON fragment.

The JSON response body is displayed with line numbers from 1 to 23:

```
1  {
2   "id": "646150aa2030e1c4e9c99816",
3   "project": "645f8e1b5ad943eec012d23a",
4   "title": "DevOps",
5   "description": "This is a sample deliverable",
6   "files": [
7     "file1.pdf",
8     "file2.docx"
9   ],
10  "authors": [
11    "645fccf3b9edd862353e595e",
12    "645fccf3b9edd862353e595e"
13  ],
14  "creationDate": "2023-05-15T08:00:00.000Z",
15  "lastModifiedDate": "2023-05-15T10:30:00.000Z",
16  "status": "Pending",
17  "__v": 0
18},
19{
20  "_id": "646251d8c0c853391fe5e55d",
21  "project": "645f8e1b5ad943eec012d23a",
22  "title": "Web 1st ".
```

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

The screenshot shows a REST API testing interface with the following details:

- Headers (8):** Status
- Pretty:** Selected tab.
- Raw:** Raw JSON view.
- Preview:** Preview pane.
- Visualize:** Visualize pane.
- JSON:** JSON dropdown.
- Copy:** Copy button.

```
1
2   "id": "1122tt",
3   "project": "611f65ae9e3d874b6c69eac1",
4   "evaluators": [
5     "611f65ae9e3d874b6c69ead1",
6     "611f65ae9e3d874b6c69ead2"
7   ],
8   "criteria": "Quality",
9   "totalScore": 100,
10  "obtainedScore": 85,
11  "comments": "The work is well done with minor areas for improvement.",
12  "evaluationDate": "2023-05-15T09:00:00.000Z",
13  "_id": "64624bed5c855a47309a2bea",
14  "__v": 0
15 }
```

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

The screenshot shows a REST API testing interface with the following details:

- Headers (8):** Status
- Pretty:** Selected tab.
- Raw:** Raw JSON view.
- Preview:** Preview pane.
- Visualize:** Visualize pane.
- JSON:** JSON dropdown.
- Copy:** Copy button.

```
1
2 {
3   "_id": "645f8e1b5ad943eec012d23a",
4   "title": "DevOps",
5   "id": "12ab",
6   "description": "storing data into database",
7   "file": "http/this",
8   "startDate": "2023-05-09T19:00:00.000Z",
9   "dueDate": "2023-05-19T19:00:00.000Z",
10  "__v": 0
11 },
12 {
13   "_id": "6462458f2a9f6c5393baf465",
14   "title": "DevOps",
15   "id": "12ab",
16   "description": "storing data into database",
17   "file": "http/this",
18   "startDate": "2023-05-09T19:00:00.000Z".
```

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

The screenshot shows a REST client interface with the following details:

- Body:** The tab is selected.
- Headers (8):** There are 8 headers listed.
- Pretty:** The JSON response is displayed in a readable, indented format.
- Raw:** The raw JSON response is shown below the Pretty view.

```
1 {  
2   "message": "Project deleted successfully"  
3 }
```

- Preview:** A preview of the JSON structure is shown.
- Visualize:** A visualization tool icon.
- JSON:** A dropdown menu currently set to "JSON".
- Copy:** A copy icon.

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

The screenshot shows a REST client interface with the following details:

- Body:** The tab is selected.
- Headers (8):** There are 8 headers listed.
- Pretty:** The JSON response is displayed in a readable, indented format.
- Raw:** The raw JSON response is shown below the Pretty view.

```
1 {  
2   "_id": "64617115de99c7914f5ba1c1",  
3   "project": "611f65ae9e3d874b6c69eac1",  
4   "evaluators": [  
5     "611f65ae9e3d874b6c69ead1",  
6     "611f65ae9e3d874b6c69ead2"  
7   ],  
8   "criteria": "Quality",  
9   "totalScore": 100,  
10  "obtainedScore": 85,  
11  "comments": "The work is well done with minor areas for improvement.",  
12  "evaluationDate": "2023-05-15T09:00:00.000Z",  
13  "__v": 0  
14 }  
15 ]  
16 }
```

- Preview:** A preview of the JSON structure is shown.
- Visualize:** A visualization tool icon.
- JSON:** A dropdown menu currently set to "JSON".
- Copy:** A copy icon.

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

The screenshot shows a REST client interface with the following details:

- Header bar: Body, Cookies, Headers (8), Test Results.
- Toolbars: Pretty (selected), Raw, Preview, Visualize, JSON dropdown, and a copy icon.
- Response body:

```
1  {
2   "message": "Evaluation deleted successfully"
3 }
```

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

The screenshot shows a REST client interface with the following details:

- Header bar: Body, Cookies, Headers (8), Test Results, Status: 2C.
- Toolbars: Pretty (selected), Raw, Preview, Visualize, JSON dropdown, and a copy icon.
- Response body:

```
1  {
2   "message": "Deliverable deleted successfully"
3 }
```

# 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>

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (selected), 'Environments', and 'History'. The main area shows a 'PUT' request to 'http://localhost:3001/users/change-email'. The 'Body' tab is selected, showing form-data with 'email' set to 'mahad@hotmail.com' and 'userId' set to '123'. Below the body, the response status is 200 OK, with a JSON response body containing a single key 'message' with the value 'Email changed successfully'.

Key	Value	Description	... Bulk Edit
email	mahad@hotmail.com		
userId	123		
Key	Value	Description	

Body Cookies Headers (8) Test Results  
Pretty Raw Preview Visualize JSON

```
1   "message": "Email changed successfully"
2
3
```

Status: 200 OK Time: 204 ms Size: 307 B Save as Example

## 2. Deletion of a user

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

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

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> userId	123		
Key	Value	Description	

```

1   "message": "User deleted successfully"
2
3

```

Status: 200 OK Time: 118 ms Size: 306 B Save as Example

### 3. User Creation:

http://localhost:3001/users/create

POST http://localhost:3001/users/create

Body

```

1 {
2   "userId": "user123",
3   "name": "John Doe",
4   "email": "john.doe@example.com",
5   "contact": "+1 123-456-7890",
6   "password": "mypassword123",
7   "gender": "Male",
8   "profile_picture": "https://example.com/profile.jpg",
9   "theme": {
10     "color": "white"
11   }
12 }

```

Status: 200 OK Time: 245 ms Size: 310 B Save as Example

### 4. Getting all the users

http://localhost:3001/users/all

http://localhost:3001/users/all

```

5 "contact": "+1 123-456-7890",
6 "password": "mypassword123",
7 "gender": "Male",
8 "profile_picture": "https://example.com/profile.jpg",
9 "theme": {
10   "color": "white",
11   "name": "light"
12 },
13 "registerAs": "student"
14
    
```

Status: 200 OK Time: 139 ms Size: 862 B

## 5. Getting all requirements

http://localhost:3001/requirements/getRequirements

http://localhost:3001/requirements/getRequirements

```

12   "author": "Mazy",
13   "text": "This is a great idea!"
14 },
15 {
16   "author": "Bob",
17   "text": "I think we need to clarify the requirements first"
18 }
19 ]
20
    
```

Status: 200 OK Time: 81 ms Size: 816 B

## 6: Creating Requirements (Teacher):

http://localhost:3001/requirements/createRequirement

POST http://localhost:3001/requirements/createRequirement

Params: Authorization, Headers (10), Body, Pre-request Script, Tests, Settings

Body (form-data)

Key	Value
priority	High
status	In Progress
text	lets see if it works 3rd time
description	all the users shall be displayed in a table
deadline	2024-01-01
writtenby	20I-0457
projectid	20IP-0457

Body (Pretty)

```
1 "title": "fetch Users",
2 "priority": "High",
3 "assignedTo": [
4   []
5 ],
6 "status": "In Progress",
7 "description": "all the users shall be displayed in a table",
8 "deadline": "2024-01-01T00:00:00.000Z",
9 "attachments": [
10   []
11 ],
12 "writtenby": "20I-0457",
13 "projectid": "20IP-0457",
14 "_id": "646227631e34ab33384d6a6",
15 "date": "2023-06-16T12:36:36.996Z",
16 "comments": [],
17 "__v": 0
```

## 7. Update requirements

The screenshot shows a POST request to `http://localhost:3001/requirements/updateRequirement/646227531034abd33384d5a5`. The Body tab is selected, displaying a table of fields and their values:

Key	Value	Description
priority	Medium	
status	Completed	
text	lets see if it works 3rd time	
description	all the users shall be displayed in a table	
deadline	2024-01-01	
writtenby	20IP-0457	
projectid	20IP-0457	
Key	Value	Description

Below the table, there are tabs for Body, Cookies, Headers (8), and Test Results. The Body tab is active, showing a JSON preview:

```

1
2   "_id": "646227531034abd33384d5a5",
3   "title": "fetch Users",
4   "priority": "Medium",
5   "assignedTo": [
6     []
7   ],
8   "status": "Completed",
9   "description": "all the users shall be displayed in a table",
10  "deadline": "2024-01-01T00:00:00.000Z",
11  "attachments": [
12    []
13  ],
14  "writtenby": "20IP-0457",
15  "projectid": "20IP-0457",
16  "date": "2023-06-16T12:36:35.996Z",
17  "comments": [],
18  "__v": 0
19

```

## 8: SignIn with jwt

The screenshot shows a POST request to `http://localhost:3001/users/signin`. The Body tab is selected, displaying a table of form-data fields:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
userId	123			
password	qwerty			
Key	Value	Description		

## 10. Creating theme:

POST <http://localhost:3001/themes/add-theme>

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

	Key	Value	Description
<input checked="" type="checkbox"/>	name	dark	
<input checked="" type="checkbox"/>	color	black	
<input checked="" type="checkbox"/>	themId	1234	

	Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 212 ms Size: 306 B Save

Pretty Raw Preview Visualize JSON

```
1 {  
2   "message": "Theme stored successfully"  
3 }
```

## 11. Create requirements (admin):

<http://localhost:3001/requirements/createRequirement>

POST <http://localhost:3001/requirements/createRequirement>

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1
2   "maxGroups": "10",
3   "maxStudentsPerGroup": "5",
4   "topicRequirements": "Lorem ipsum dolor sit amet"
5
6
```

Body Cookies Headers (8) Test Results Status: 201 Created Time: 170 ms Size: 407 B [Save](#)

Pretty Raw Preview Visualize **JSON** [Copy](#)

```
1
2   "maxGroups": "10",
3   "maxStudentsPerGroup": "5",
4   "topicRequirements": "Lorem ipsum dolor sit amet",
5   "_id": "64859c9d380e2c6cc382a95a",
6   "__v": 0
7
```

## 12. View projects:

<http://localhost:3001/projects/getAllProjects>

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3001/projects/getAllProjects`. The response status is 200 OK, with a time of 128 ms and a size of 1.31 KB. The response body is displayed in Pretty JSON format, showing two project documents:

```
2 {
  "_id": "647e526c8f8e621fd1d97ca1",
  "title": "DevOps Project",
  "id": "2tQzrS",
  "description": "This is simple testing",
  "file": "PM Final.pdf",
  "startDate": "2023-06-06T00:00:00.000Z",
  "dueDate": "2023-06-15T00:00:00.000Z",
  "__v": 0
},
{
  "_id": "647e52ec8f8e621fd1d97caa",
  "title": "Computer Net FYP",
  "id": "BWFooW",
  "description": "this is Cnet",
  "__v": 0
}
```

### 13. View evaluation and project status:

http://localhost:3001/evaluations/getAllEvaluations

GET http://localhost:3001/evaluations/getAllEvaluations

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (8) Test Results Status: 200 OK Time: 104 ms Size: 740 B Save as Example

Pretty Raw Preview Visualize JSON

```

1 {
2   "_id": "64818332b43f0969a7f25303",
3   "projectId": "64818306b43f0969a7f252e1",
4   "studentId": "20i2424",
5   "criteria": "no",
6   "totalScore": 19,
7   "obtainedScore": 16,
8   "comments": "Good work",
9   "evaluationDate": "2023-06-08T00:00:00.000Z",
10  "progress": 76,
11  "__v": 0
12 },
13 {
14   "_id": "6481a1ef23251183b53395f9",
15 }
```

## Screenshots of UI:

### 1- Sign In:

Sign In

Email

Password

[SIGN IN](#)

[REGISTER?](#)

© 2023 My App. All rights reserved.  
The Memorandums  
Powered by [Mahad](#)

## 2-Home/Dashboard:

The dashboard interface includes a sidebar with a user icon and the text "Academic Officer Fast Nuces". Below the sidebar, there's a "MESSAGE" button. The main area shows a user profile with "Full Name: Admin", "Email: syed@gmail.com", "Password: 123", "Mobile: (098) 765-4321", and "Address: Islamabad Pakistan". A "SAVE" button is present. To the left, a sidebar displays a timeline with entries: "https://admin.com" (Nuces), "@Admin.com" (Amir), and "MBA". On the right, two project status cards show "assignment Project Status" for "Approval", "Phase 1", "Presentation", "MVP", and "Defence".

## 3- Create User:

The sign-up form titled "Sign Up" contains fields for "Name" (Ahmed), "User Id" (i201767), "Email" (ahmed@gmail.com), "Contact" (0332-7865412), "Gender" (Male), "Password" (\*\*\*\*\*), "Confirm Password" (\*\*\*\*\*), and "Register As" (Student). A "SIGN UP" button is at the bottom.

### 3.1: User successfully stored in database:

**Sign Up**

Name	User Id
<input type="text"/>	<input type="text"/>
Email	<input type="text"/>
Gender	<input type="text"/>
Male	<input type="text"/>
Confirm Password	<input type="text"/>
Register As	<input type="text"/>
Student	<input type="text"/>
<b>SIGN UP</b>	

User created successfully

Congratulations! Your account has been created successfully.

**CLOSE**

### 3.2 Input validation:

**DevOps**

**Sign Up**

Name	Email
<input type="text" value="123"/>	<input type="text" value="kj"/>
Enter a valid name	
Contact	Gender
<input type="text"/> <span style="border: 1px solid red; padding: 2px;">Please include an '@' in the email address. 'kj' is missing an '@'.</span>	

### 4- Manage User:

**DevOps**

Name	Email	ID	Contact	Gender	Password	Manage
Hamza Bhai12	johndoe@example.com	20i-1884	03187766542	male	password123	
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	

#### 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:  Email:  Contact:  Gender:  Password:

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	2012424	03155588412	male	123	
Abdullah	johndoe@example.com	201-1992	1234567890	male	password123	
Abdullah	johndoe@example.com	201-1993	1234567890	male	password123	
Hamza	hamzaazam994@gmail.com	as01212	1	MALE	12345	
Hammad	hamad@gmail.com	11231	051-8978768	MALE	1234	

### 5- Make an announcement:

≡ DevOps

## FYP Announcements

### Add Announcement

Title \*

Message \*

**ADD ANNOUNCEMENT**

© 2023 My App. All rights reserved.  
The Memorandums  
Powered by [Mahad](#)

## 6- Create general requirements for FYP:

≡ DevOps

## FYP Requirements

Maximum Groups \*

Maximum Students Per Group \*

Topic Requirements \*

**SAVE REQUIREMENTS**

© 2023 My App. All rights reserved.  
The Memorandums  
Powered by [Mahad](#)

## 7- View Evaluations and project status:

The screenshot shows a mobile application interface with a blue header bar containing the text "≡ DevOps" and a toggle switch. Below the header are two cards, each representing a project. The left card has the following details:

- Project ID: 64818306b43f0969a7f252e1
- Version: 20i2424
- Criteria: no
- Total Score: 19
- Obtained Score: 16
- Comments: Good work
- Evaluation Date: 2023-06-08T00:00:00.000Z

The right card has the following details:

- Project ID: 6480cec4f320830725036867
- Version: 20i-1884
- Criteria: no
- Total Score: 10
- Obtained Score: 5
- Comments: comment
- Evaluation Date: 2023-06-15T00:00:00.000Z

Both cards have a "Fyp status" section at the bottom.

At the bottom of the screen, there is a footer bar with the text "© 2023 My App. All rights reserved.", "The Memorandums", and "Powered by Mahad".

## 8- Change Theme:

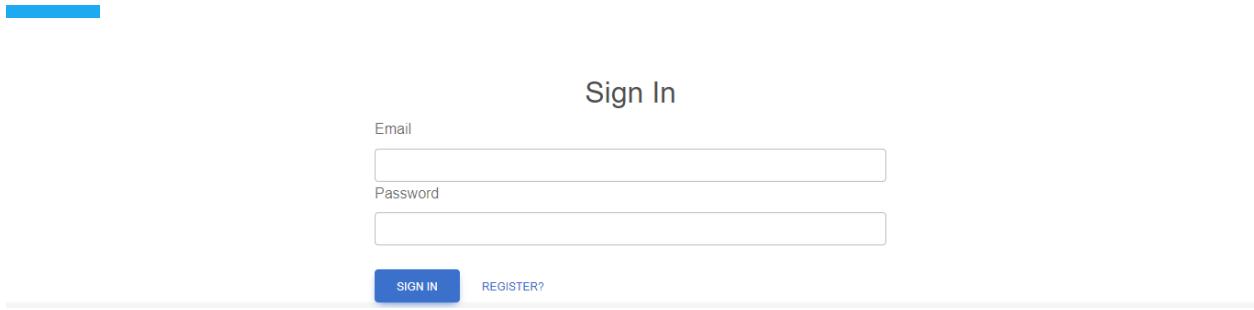
The screenshot shows a mobile application interface with a dark theme. At the top is a header bar with the text "≡ DevOps" and a toggle switch. Below the header is a "Sign Up" form. The form fields are arranged in pairs of two columns:

Name	Email
<input type="text"/>	<input type="text"/>
Contact	Gender
<input type="text"/>	<input type="text" value="Male"/>
Password	Confirm Password
<input type="text"/>	<input type="text"/>
Register As	
<input type="text" value="Student"/>	

At the bottom of the form is a blue "SIGN UP" button.

## 9- Logout:

Logging out redirects you to the sign in page.



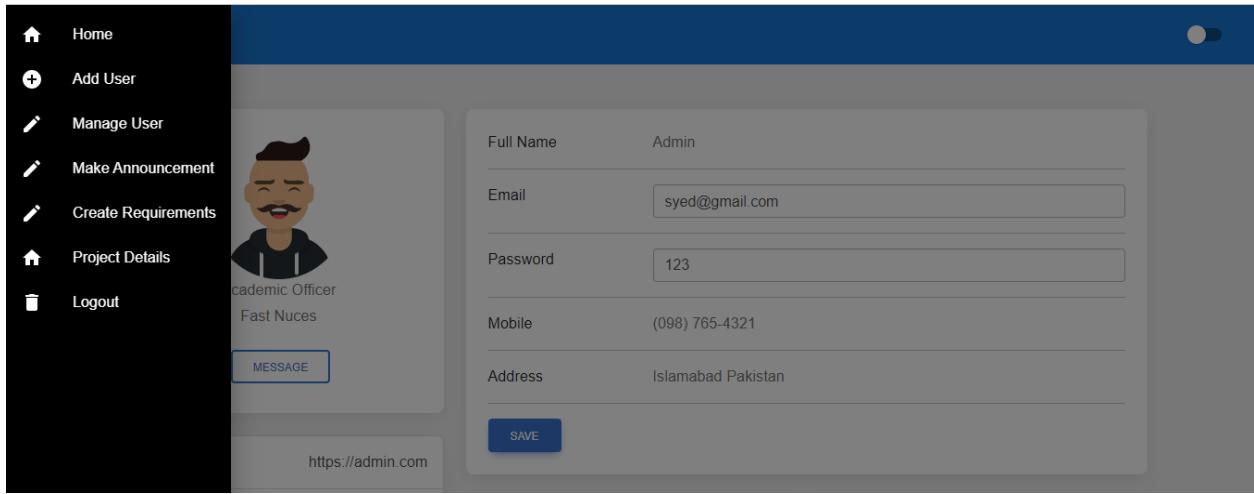
Sign In

Email

Password

SIGN IN REGISTER?

## 10- Header Component:



The screenshot shows a dashboard interface. On the left is a dark sidebar with white icons and text:

- Home
- Add User
- Manage User
- Make Announcement
- Create Requirements
- Project Details
- Logout

On the right, there's a user profile section with a placeholder image, the name "Academic Officer", and the text "Fast Nuces". Below this is a "MESSAGE" button and a URL "https://admin.com". The main area contains a form for user registration or update:

Full Name	Admin
Email	syed@gmail.com
Password	123
Mobile	(098) 765-4321
Address	Islamabad Pakistan

A "SAVE" button is at the bottom of the form.

# Mian Abdullah

## Email Module

Send an email to a particular person

The screenshot shows a POST request to `http://localhost:3001/email/send`. The request body is defined as `form-data` with the following fields:

Key	Value	Description
sender	axiomshah@gmail.com	
recipient	i200457@nu.edu.pk	
subject	testing emial 3	
text	lets see if it works 3rd time	
sentAt		

The response tab shows the following JSON output:

```
1   "message": "Email sent successfully"
2
3
```

## Get all emails that were send by a particular user

The screenshot shows the Postman application interface. At the top, the URL `http://localhost:3001/email/getsent/axiomshah@gmail.com` is entered. Below the URL, there are tabs for **GET**, **Authorization**, **Headers (10)**, **Body**, **Pre-request Script**, **Tests**, and **Settings**. The **Params** tab is currently selected. In the **Query Params** section, there is a table with columns **Key**, **Value**, **Description**, and **Bulk Edit**. There are two rows in the table, both labeled "Key". Below the table, there are tabs for **Body**, **Cookies**, **Headers (8)**, and **Test Results**. The **Body** tab is selected, showing a JSON response. The response body is:

```
21   "_id": "646210786a09668b156cc235",
22   "sender": "axiomshah@gmail.com",
23   "recipient": "i200457@nu.edu.pk",
24   "subject": "testing emial 2",
25   "text": "lets see if it works 2nd time",
26   "sentAt": "2023-05-15T10:59:04.563Z",
27   "v": 0
```

At the bottom right of the Body panel, there are status indicators: 200 OK, 193 ms, 897 B, and a "Save as Example" button.

## Get all emails that were received by a particular user

The screenshot shows the Postman application interface. At the top, the URL is set to `http://localhost:3001/email/getreceived/i200457@nu.edu.pk`. Below the URL, the method is selected as `GET`. The `Send` button is highlighted in blue. The `Params` tab is active, showing a single entry for the `Query Params` section with the key `Key` and value `Value`. The `Body` tab is also visible at the bottom.

Below the request details, the response is displayed. The status bar indicates `200 OK`, `190 ms`, and `478 B`. The `Pretty` tab is selected, showing the JSON response:

```
2
3   {
4     "_id": "64608d6e908c60bad8d128ae",
5     "sender": "axiomshah@gmail.com",
6     "recipient": "i200457@nu.edu.pk",
7     "subject": "testing emial 2",
8     "text": "lets see if it works 2nd time",
9     "sentAt": "2023-05-14T07:27:42.256Z",
10    "__v": 0
```

## Requirement Module

### Create a Requirement

The screenshot shows a Postman interface for creating a requirement. The URL is `http://localhost:3001/requirements/createRequirement`. The method is `POST`. The body is set to `form-data`. The following fields are filled:

Key	Value
priority	High
status	In Progress
text	lets see if it works 3rd time
description	all the users shall be displayed in a table
deadline	2024-01-01
writtenby	20I-0457
projectid	20IP-0457

The JSON response is:

```
1
2   "title": "fetch Users",
3   "priority": "High",
4   "assignedTo": [
5     []
6   ],
7   "status": "In Progress",
8   "description": "all the users shall be displayed in a table",
9   "deadline": "2024-01-01T00:00:00.000Z",
10  "attachments": [
11    []
12  ],
13  "writtenby": "20I-0457",
14  "projectid": "20IP-0457",
15  "_id": "646227631934ab53384d6a6",
16  "date": "2023-06-16T12:36:35.996Z",
17  "comments": [],
18  "__v": 0
19
```

### Find requirement by Projectid

The screenshot shows a Postman interface for finding requirements by project ID. The URL is `http://localhost:3001/requirements/getRequirements/20IP-0457`. The method is `GET`. The body is set to `form-data`. The following query parameters are filled:

Key	Value

The screenshot shows a REST API tool interface with the following details:

- Body:** Contains the JSON representation of a requirement document.
- Cookies:** Headers (8) Test Results
- Pretty:** Selected (highlighted in red).
- Raw:** Raw JSON view.
- Preview:** Preview tab.
- Visualize:** Visualize tab.
- JSON:** JSON dropdown.

```
1  [
2  {
3    "_id": "646227531034abd33384d5a5",
4    "title": "fetch Users",
5    "priority": "High",
6    "assignedTo": [
7      []
8    ],
9    "status": "In Progress",
10   "description": "all the users shall be displayed in a table",
11   "deadline": "2024-01-01T00:00:00.000Z",
12   "attachments": [
13     []
14   ],
15   "writtenby": "20I-0457",
16   "projectid": "20IP-0457",
17   "date": "2023-05-15T12:36:35.996Z",
18   "comments": [],
19   "__v": 0
--
```

## 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

<input checked="" type="checkbox"/>	priority	Medium
<input checked="" type="checkbox"/>	status	Completed
<input checked="" type="checkbox"/>	text	lets see if it works 3rd time
<input checked="" type="checkbox"/>	description	all the users shall be dispalyed in a table
<input checked="" type="checkbox"/>	deadline	2024-01-01
<input checked="" type="checkbox"/>	writtenby	20I-0457
<input checked="" type="checkbox"/>	projectid	20IP-0457
	Key	Description

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1
2   "_id": "646227531034abd33384d5a5",
3   "title": "fetch Users",
4   "pxriority": "Medium",
5   "assignedTo": [
6     []
7   ],
8   "status": "Completed",
9   "description": "all the users shall be dispalyed in a table",
10  "deadline": "2024-01-01T00:00:00.000Z",
11  "attachments": [
12    []
13  ],
14  "writtenby": "20I-0457",
15  "projectid": "20IP-0457",
16  "date": "2023-06-16T12:36:35.996Z",
17  "comments": [],
18  "__v": 0
19

```

## Upload attachment to a requirement

The screenshot shows a Postman interface with the following details:

URL: <http://localhost:3001/requirements/uploadAttachment/646227531034abd33384d5a5>

Method: POST

Headers (10):

- Content-Type: multipart/form-data
- Accept: application/json
- Content-Length: 1000
- Host: localhost:3001
- Connection: keep-alive
- User-Agent: PostmanRuntime/7.32.0
- TE: trailers
- Cache-Control: no-cache
- Postman-Token: 3a2a2a2a-2a2a-2a2a-2a2a-2a2a2a2a2a2a

Body (form-data):

Key	Value
attachment	classdiagramm.png
priority	Medium
status	Completed
text	lets see if it works 3rd time
description	all the users shall be displayed in a table
deadline	2024-01-01
writtenby	20I-0457

JSON Response (Pretty):

```
1  {
2    "_id": "646227531034abd33384d5a5",
3    "title": "fetch Users",
4    "priority": "Medium",
5    "assignedTo": [
6      []
7    ],
8    "status": "Completed",
9    "description": "all the users shall be displayed in a table",
10   "deadline": "2024-01-01T00:00:00.000Z",
11   "attachments": [
12     [],
13     "classdiagramm.png"
14   ],
15   "writtenby": "20I-0457",
16   "projectid": "20IP-0457",
17   "date": "2023-05-15T12:36:35.996Z",
18   "updated": "2023-05-15T12:36:35.996Z"
}
```

## 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
<input checked="" type="checkbox"/> deadline	2024-04-15
Key	Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1  "id": "646227531034abd33384d5a5",
2  "title": "fetch Users",
3  "priority": "Medium",
4  "assignedTo": [
5  |   []
6  ],
7  "status": "Completed",
8  "description": "all the users shall be dispalyed in a table",
9  "deadline": "2024-04-15T00:00:00.000Z",
10 "attachments": [
11 |   [],
12 |   "classdiagramm.png"
13 ],
14 "writtenby": "20I-0457",
15 "projectid": "20IP-0457",
16 "date": "2023-05-15T12:36:35.996Z",
17 "comments": [],
18 "___v": 1
19
20
```

## Update Priority of requirement

The screenshot shows the Postman application interface. At the top, it displays a 'PATCH' method and the URL `http://localhost:3001/requirements/updatePriority/646227531034abd33384d5a5`. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (10)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. Under 'Body', the 'form-data' tab is selected. A table is present with one row, where 'priority' is set to 'High'. The 'Body' tab is also active, showing a JSON response with line numbers from 1 to 20. The JSON content is as follows:

```
1  {
2   "id": "646227531034abd33384d5a5",
3   "title": "fetch Users",
4   "priority": "High",
5   "assignedTo": [
6     []
7   ],
8   "status": "Completed",
9   "description": "all the users shall be dispalyed in a table",
10  "deadline": "2024-04-15T00:00:00.000Z",
11  "attachments": [
12    [],
13    "classdiagram.png"
14  ],
15  "writtenby": "20I-0457",
16  "projectid": "20IP-0457",
17  "date": "2023-05-15T12:36:35.996Z",
18  "comments": [],
19  "__v": 1
20 }
```

## Add comments to requirement

The screenshot shows the Postman application interface. At the top, the URL is `http://localhost:3001/requirements/addComment/646227531034abd33384d5a5`. Below the URL, there are buttons for Save, Edit, and Delete. The main area is a POST request to the same URL. The Body tab is selected, showing two fields: "content" with the value "I guess it should be displayed in a html/css card with details in I" and "createdBy" with the value "20I-0457". There are also tabs for Params, Authorization, Headers (10), Pre-request Script, Tests, and Settings. Below the body, there are sections for "Key" and "Value" under "Description". At the bottom, there are tabs for Body, Cookies, Headers (8), and Test Results. The Test Results tab shows a successful response with Status: 200 OK, Time: 415 ms, Size: 814 B, and a "Save as Example" button.

```
1: {
2:   "_id": "646227531034abd33384d5a5",
3:   "title": "fetch Users",
4:   "priority": "High",
5:   "assignedTo": [
6:     []
7:   ],
8:   "status": "Completed",
9:   "description": "all the users shall be dispalyed in a table",
10:  "deadline": "2024-04-15T08:00:00Z",
11:  "attachments": [
12:    [],
13:    "classdiagram.png"
14:  ],
15:  "writtenby": "20I-0457",
16:  "projectid": "28IP-0457",
17:  "date": "2023-05-15T12:36:35.996Z",
18:  "comments": [
19:    {
20:      "content": "I guess it should be displayed in a html/css card with details in lower section",
21:      "createdBy": "20I-0457",
22:      "_id": "64624dfab6f94f911211ea1",
23:      "createdAt": "2023-05-15T15:21:38.437Z"
24:    }
25:  ]
26: }
```

## Display comments of a requirement

The screenshot shows the Postman interface with a GET request to `http://localhost:3001/requirements/getComments/646227531034abd33384d5a5`. The 'Body' tab is selected, showing two parameters: 'content' and 'createdBy'. The 'content' parameter has a value of "I guess it should be displayed in a html/css card with details in lower section". The 'createdBy' parameter has a value of "20I-0457". Below the body, the JSON response is displayed in a pretty-printed format:

```
1 [ {  
2   "content": "I guess it should be displayed in a html/css card with details in lower section",  
3   "createdBy": "20I-0457",  
4   "_id": "64624dfab60f94f911211ea1",  
5   "createdAt": "2023-05-15T15:21:30.437Z"  
6 } ]
```

## Team Member Module

### Adding team member

The screenshot shows the Postman application interface for making a POST request to the endpoint `http://localhost:3001/teamMember/addMember`. The request body is defined as form-data with the following fields:

Key	Value
student_id	20I-1808
team_lead_id	20I-0987
student_name	Mahad khan
student_email	i201808@gmail.com
student_status	active
projectid	20IP-0457

The response body is displayed in JSON format:

```
1 "message": "Team member added successfully"
```

## Getting all team member by Projectid

http://localhost:3001/teamMember/getMembers/20IP-0457

The screenshot shows the Postman application interface. At the top, there is a header bar with tabs for 'GET' (selected), 'Params', 'Authorization', 'Headers (10)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. Below this is a 'Query Params' section with a table:

Key	Value
Key	Value

Below the header bar, there are tabs for 'Body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'Body' tab is selected and contains a 'Pretty' view of the JSON response. The JSON output is as follows:

```
1
2   [
3     {
4       "_id": "646268ac22f31bb1db99e142",
5       "student_id": "20I-0457",
6       "team_lead_id": "20I-0987",
7       "student_name": "syed abdullah shah",
8       "student_email": "axiomshah@gmail.com",
9       "student_status": "active",
10      "requirements": [],
11      "projectid": "20IP-0457",
12      "__v": 0
13    },
14    {
15      "_id": "646258d122f31bb1db99e144",
16      "student_id": "20I-3333",
17      "team_lead_id": "20I-0987",
18      "student_name": "jk",
19      "student_email": "jk@gmail.com",
20      "student_status": "active",
21      "requirements": [],
22      "projectid": "20IP-0457",
23      "__v": 0
24    }
]
```

## Getting a single team member by id

http://localhost:3001/teamMember/findMember/646258d122f31bb1db99e144

The screenshot shows the Postman application interface. At the top, there is a header bar with tabs for 'GET' (selected), 'Authorization', 'Headers (10)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. Below the header, the URL 'http://localhost:3001/teamMember/findMember/646258d122f31bb1db99e144' is entered. Under the 'Params' tab, there is a table with two rows. The first row has an empty 'Key' column and an empty 'Value' column. The second row has a 'Key' column containing 'Key' and a 'Value' column containing 'Value'. In the main body area, there are tabs for 'Body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'Body' tab is selected and contains a JSON editor. The JSON response is displayed in a pretty-printed format with line numbers from 1 to 11 on the left. The JSON object includes fields like '\_id', 'student\_id', 'team\_lead\_id', 'student\_name', 'student\_email', 'student\_status', 'requirements', 'projectid', and '\_\_v'. The 'JSON' tab is also visible in the editor.

```
1  "_id": "646258d122f31bb1db99e144",
2  "student_id": "20I-3333",
3  "team_lead_id": "20I-0987",
4  "student_name": "JK",
5  "student_email": "jk@gmail.com",
6  "student_status": "active",
7  "requirements": [],
8  "projectid": "20IP-0457",
9  "__v": 0
```

## Updating the role of a team member

The screenshot shows a Postman interface with the following details:

- Request URL:** PATCH http://localhost:3001/teamMember/assignRole/64625d47d9b8d5e7de60ef99
- Method:** PATCH
- Headers:** (10)
- Body:** (selected)
- Params:** (green dot)
- Authorization:** (green dot)
- Tests:**
- Settings:**

The body contains the following data:

Key	Value
<input checked="" type="checkbox"/> student_role	member
<input type="checkbox"/> team_lead_id	20I-0987
<input type="checkbox"/> student_name	syed Abdullah shah
<input type="checkbox"/> student_email	axiomshah@gmail.com

Below the table, there are tabs for Body, Cookies, Headers (8), and Test Results. The Body tab is selected, showing the JSON response:

```
1  "_id": "64625d47d9b8d5e7de60ef99",
2  "student_id": "20I-0333",
3  "team_lead_id": "20I-0987",
4  "student_name": "JK",
5  "student_email": "jk@gmail.com",
6  "student_status": "active",
7  "requirements": [],
8  "projectid": "20IP-0457",
9  "student_role": "member",
10 " __v": 0
```

## Assigning a requirement to a team member

The screenshot shows the Postman application interface. At the top, it displays a 'PATCH' method and the URL `http://localhost:3001/teamMember/assignReq/64625d47d9b8d5e7de60ef99`. Below the URL, there are tabs for 'Params', 'Authorization', 'Headers (10)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is currently selected and has a red underline. Under the 'Body' tab, there are several input fields:

Key	Value
<input checked="" type="checkbox"/> requirements	646227531034abd33384d5a5
<input type="checkbox"/> team_lead_id	20I-0987
<input type="checkbox"/> student_name	syed Abdullah shah
<input type="checkbox"/> student_email	axiomshah@gmail.com

Below these fields, there are tabs for 'Body', 'Cookies', 'Headers (8)', and 'Test Results'. The 'Body' tab is selected and has a red underline. Under the 'Body' tab, there are three buttons: 'Pretty', 'Raw', and 'Visualize'. The 'Pretty' button is selected and has a grey background. To the right of these buttons is a dropdown menu set to 'JSON'. Below the buttons, the JSON payload is displayed:

```
1 {  
2   "_id": "64625d47d9b8d5e7de60ef99",  
3   "student_id": "20I-0333",  
4   "team_lead_id": "20I-0987",  
5   "student_name": "JK",  
6   "student_email": "jk@gmail.com",  
7   "student_status": "active",  
8   "requirements": [  
9     "646227531034abd33384d5a5"  
10  ],  
11  "projectid": "20IP-0457",  
12  "student_role": "member",  
13  "__v": 0  
14 }
```

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: "jih@gmail.com"
student_email: "jih@gmail.com"
student_status: "active"
requirements: Array
projectid: "20IP-0457"
__v: 0
```

### Fast\_Deadline-Solution.emails

STORAGE SIZE: 6KB 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('645fc5b90430d5615ce22ed')
sender: "i200457@nu.edu.pk"
recipient: "axiomshah@gmail.com"
subject: "testing email"
text: "lets see if it works"
sentAt: 2023-05-13T12:02:35.499+00:00
__v: 0

_id: ObjectId('64600d6e908c60bad8d128ae')
sender: "axiomshah@gmail.com"
recipient: "i200457@nu.edu.pk"
subject: "testing email 2"
text: "lets see if it works 2nd time"
sentAt: 2023-05-14T07:27:42.256+00:00
__v: 0

_id: ObjectId('646220893cf7ca95020a24a')
sender: "axiomshah@gmail.com"
recipient: "i200457@nu.edu.pk"
subject: "testing email 3"
text: "lets see if it works 3rd time"
sentAt: 2023-05-15T12:07:37.303+00:00
__v: 0
```

The screenshot shows a MongoDB interface with the following details:

- STORAGE SIZE:** 36KB    **LOGICAL DATA SIZE:** 493B    **TOTAL DOCUMENTS:** 1    **INDEXES TOTAL SIZE:** 20KB
- Find**, **Indexes**, **Schema Anti-Patterns** (0), **Aggregation**, **Search Indexes**, **Charts**
- Filter**: Type a query: { field: 'value' }
- QUERY RESULTS: 1-1 OF 1**
- Document Data:**

```

_id: ObjectId('646227531034abd33384d5a5')
title: "fetch Users"
priority: "High"
assignedTo: []
  0: []
status: "Completed"
description: "all the users shall be dispalyed in a table"
deadline: 2024-04-15T00:00:00.000+00:00
attachments: []
  0: []
  1: "classdiagram.png"
writtenby: "20I-0457"
projectid: "20IP-0457"
date: 2023-05-15T12:36:35.996+00:00
comments: []
  0: Object
    content: "I guess it should be displayed in a html/css card with details in lowe_"
    createdBy: "20I-0457"
    _id: ObjectId('64624dfab60f94f911211ea1')
    createdAt: 2023-05-15T15:21:30.437+00:00
--v: 2

```

## 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.

Deadline	Written By	Assigned To	Title	Priority	Status	Actions
2023-06-22	Mian Abdullah	Mahad Rahat	Add users to database	Low	<span>In Progress</span>	<span>View</span> <span>Edit</span> <span>Delete</span> <span>Mark as Complete</span>
2023-06-21	Ali Hamza	Mian Abdullah	Requirements Delete	Medium	<span>Pending</span>	<span>View</span> <span>Edit</span> <span>Delete</span> <span>Mark as Complete</span>
2023-06-13	Mahad Rahat	Ali Hamza	Announcement to all	Low	<span>Completed</span>	<span>View</span> <span>Edit</span> <span>Delete</span> <span>Mark as Complete</span>

## 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.

The screenshot shows a web application interface titled 'Requirement Form' under the 'Requirement' section of the 'Fast FYP Portal'. The left sidebar includes links for Home, Requirement (with a dropdown), TeamMember (with a dropdown), Email, and Evaluations. The main form area has fields for Project Id (dropdown), Assigned To (dropdown), Title (text input), Priority (dropdown), Deadline (date input), Status (dropdown), Comment (text input), and Requirement Details (text input). The top right corner shows a user profile icon and the text 'User ID'. The bottom status bar indicates the temperature is 32°C and the time is 10:39 PM.

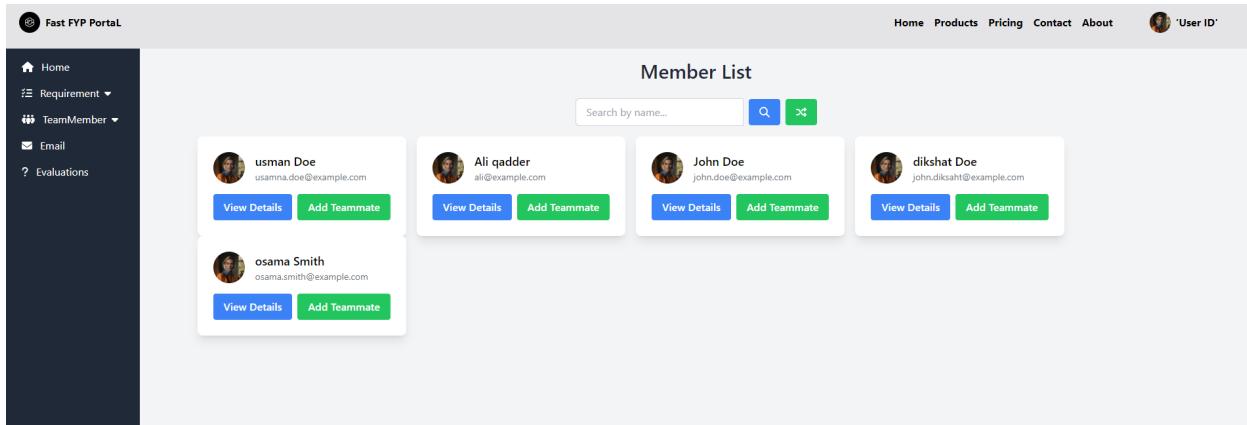
## Team Member Display

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

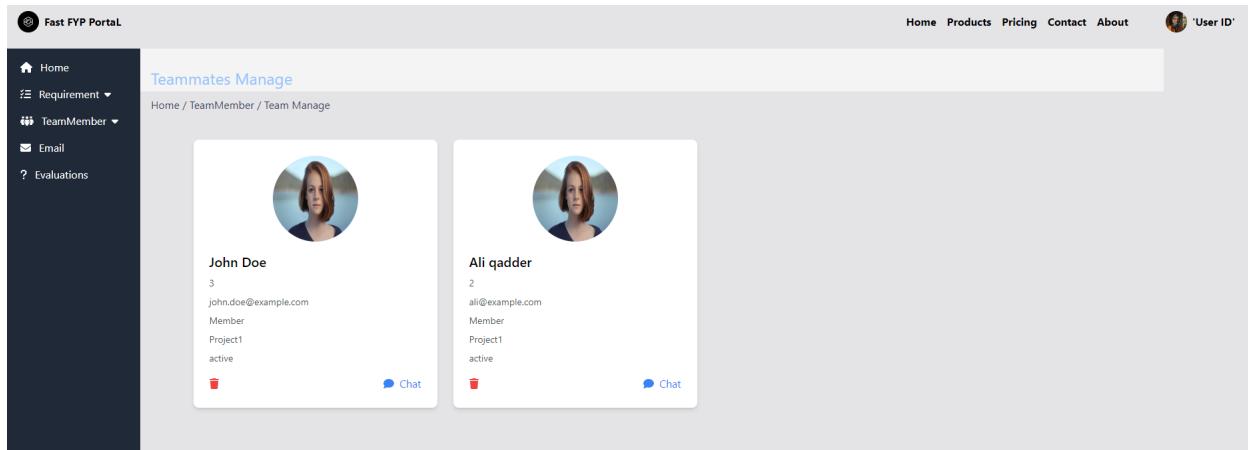
The Teammates Screen displays a list of your team members along with their details. Each team member is represented by a card, showcasing information such as their name, email, role, and other relevant details.

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.



The screenshot shows the 'Teammates Manage' screen of the Fast FYP Portal. The left sidebar has a dark theme with white icons and text, listing 'Home', 'Requirement', 'TeamMember' (which is expanded), 'Email', and 'Evaluations'. The main area has a light gray background and displays two team member profiles in cards:

- John Doe**:  
3  
john.doe@example.com  
Member  
Project1  
active  
Delete Chat
- Ali qadder**:  
2  
ali@example.com  
Member  
Project1  
active  
Delete Chat

The top right corner shows a user profile icon and the text 'User ID'.

## Emails

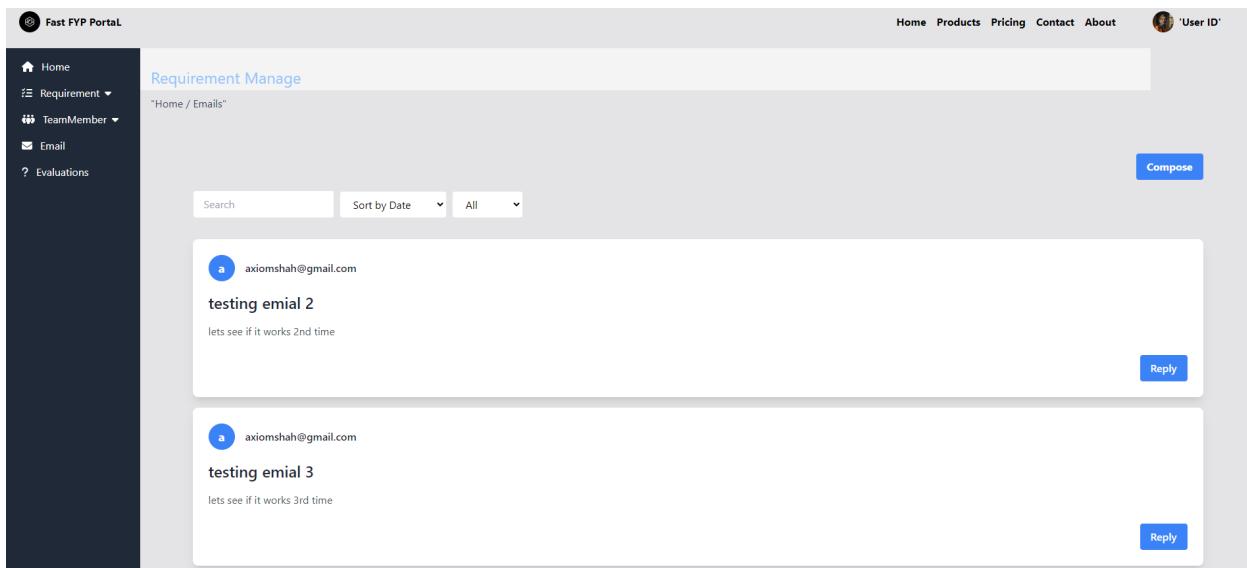
The Email Inbox Screen presents emails in row cards, providing details such as the sender's name, subject, and a brief preview of the email content. Each email card also includes a "Reply" button, allowing users to respond to the sender with the same subject line.

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.

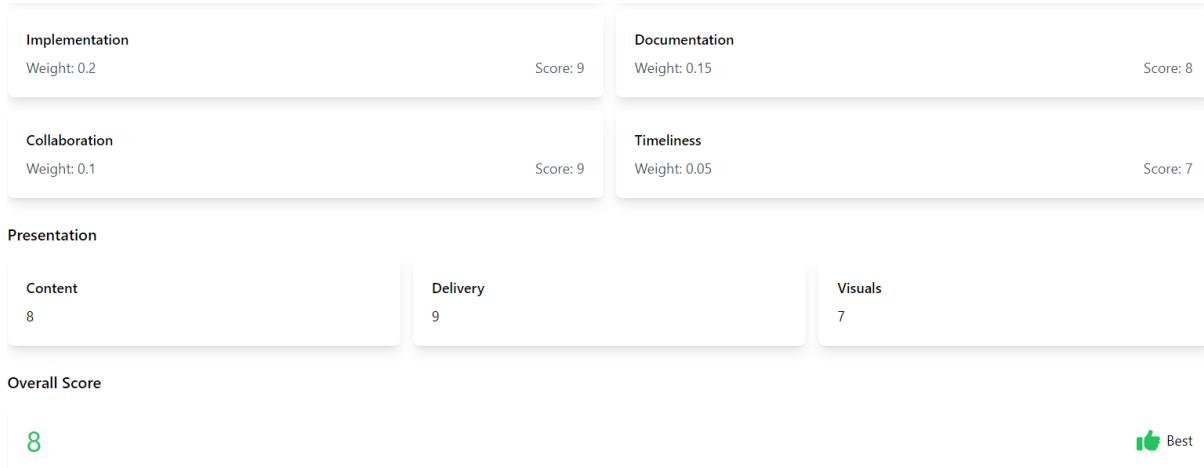
The Evaluation Screen provides a comprehensive overview of the project's performance across various evaluation criteria. The card-based layout facilitates easy comprehension and quick assessment of each criterion's weightage and score.

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.

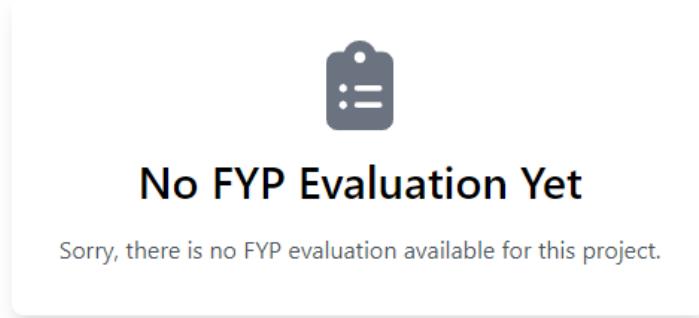
Overall, the Evaluation Screen optimizes the evaluation process by presenting evaluation criteria and their scores in a clear and accessible manner, ensuring efficient assessment and evaluation of the FYP project.

The screenshot shows the 'Mern Stack FYP Project' page. At the top left is the 'Fast FYP Portal' logo. The top right features a navigation bar with 'Home', 'Products', 'Pricing', 'Contact', 'About', and a user profile icon labeled 'User ID'. On the far left is a dark sidebar with a navigation menu: 'Home', 'Requirement', 'TeamMember', 'Email', and 'Evaluations'. The main content area has a title 'Mern Stack FYP Project' and a subtitle 'This is basically a Web Application for hosting fyp projects'. Below this, there are two sections: 'Team Members' (listing Mian Abdullah, Ali Hanza, and Mahad Rahat) and 'Supervisor' (listing Dr. Robert Johnson). The central part of the page is titled 'Evaluation Criteria' and contains six cards arranged in a grid:

Evaluation Criteria	Weight	Score	Weight	Score
Innovation	0.2	8	0.3	7
Implementation	0.2	9	0.15	8
Collaboration	0.1	9	0.05	7
Technical Complexity				
Documentation				
Timeliness				



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



# Database Schemas

The screenshot shows the MongoDB Atlas interface for a project named 'Project 0'. The left sidebar includes sections for Deployment, Services, and Security. Under 'Database', 'assignprojects' is selected, showing collections like 'admins', 'announcements', and 'students'. The main panel displays the 'Fast\_Deadline-Solution.assignprojects' collection with 3 documents. The first document is:

```
_id: ObjectId('648614465f284235187b2669')
projectId: "BWFooW"
studentId: "201-1992"
__v: 0
```

The second document is:

```
_id: ObjectId('6486144b5f284235187b266b')
projectId: "mbrcaa4"
studentId: "asd1212"
__v: 0
```

The third document is:

```
_id: ObjectId('6486145b5f284235187b267b')
```

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"
  },
})
```