Building Web-based Applications with MERNStack

Milestone: Project Report

Group 13

Harini Prasad Vasisht

Sushmitha Sudharsan

+1(984)374-4836 (Harini Prasad Vasisht)

+1(857)565-8800 (Sushmitha Sudharsan)

vasisht.h@northeastern.edu

sudharsan.s@northeastern.edu

Percentage of Effort Contributed by Student1: 50%

Percentage of Effort Contributed by Student2: 50%

Signature of Student 1 : harini

Signature of Student 2 : Sushmitha

Submission Date :12/08/2024

USE CASE STUDY REPORT

Group No.: Group 13

Student Names: Harini Prasad Vasisht and Sushmitha Sudharsan

Executive Summary:
The objective of this project was to design and implement a comprehensive task management web application utilizing the MERN stack to streamline and optimize task organization for end users. This system was envisioned to facilitate features such as user authentication, task creation, updates, retrieval, and deletion, as well as advanced functionalities like task tagging, commenting, file attachments, and notifications. Ensuring scalability, security, and a seamless user experience were paramount in the development process. For data modelling, the schema incorporated entities such as users, tasks, notes, comments, tags, and logs, all defined with robust relationships to manage and retrieve data efficiently.

EER and UML diagrams were modelled to represent the system's conceptual design, and the relational model was implemented using MongoDB as a NoSQL database. The application backend was developed using Express.js for efficient routing and HTTP request handling, while the frontend was built using React.js to deliver a responsive and dynamic user experience. Node.js served as the runtime environment for the backend, enabling event-driven, non-blocking operations. The system employed secure password encryption and JWT-based session authentication for user security.

The application demonstrated resilience in managing user interactions and processing complex task-related operations. MongoDB's flexibility allowed for efficient storage and retrieval of hierarchical data. Additionally, notifications for approaching task deadlines and logging of key actions ensured enhanced usability and system reliability. This project lays the groundwork for potential future enhancements, such as integrating machine learning for task prioritization and extending collaboration features for team-based task management.

I. Introduction

In today’s fast-paced world, managing tasks effectively has become a necessity rather than a luxury. Whether it is juggling personal commitments or coordinating team projects, having a tool that simplifies organization can significantly reduce stress and improve productivity. This project aims to address this need by developing an intelligent task management application built on the modern MERN stack.

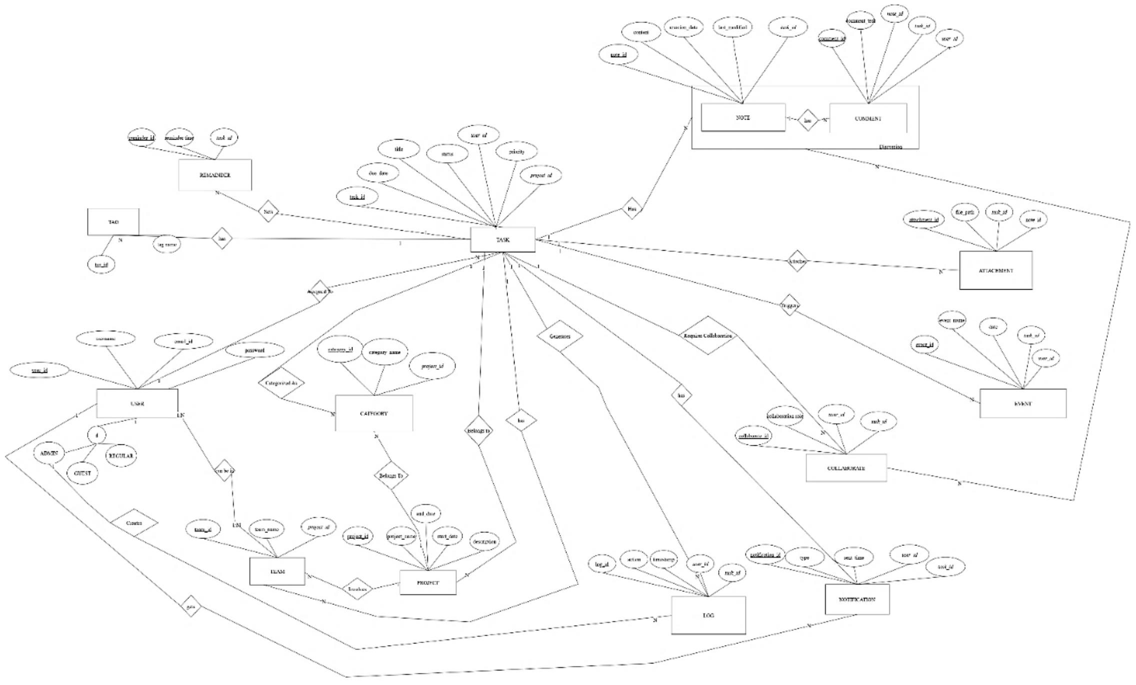
What sets this application apart is its focus on user experience, security, and adaptability. It is designed to go beyond basic task management, offering features like seamless authentication, tagging, file attachments, and real-time notifications for deadlines. With a dynamic and responsive interface powered by React.js and the scalability of MongoDB for data storage, this application is built to grow with its users.

By leveraging the capabilities of the MERN stack, this project combines innovative technology with practical functionality, providing users with a powerful tool to organize their lives and work more efficiently. From individuals managing daily to-do lists to teams collaborating on complex projects, this task management system aspires to make organization effortless, intuitive, and impactful.

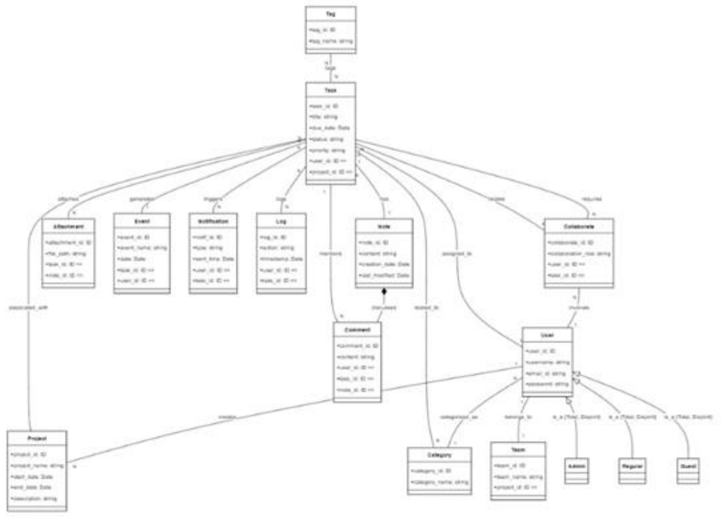
Requirements

1. Each user must have a unique email and username.
2. Passwords must be securely stored using encryption.
3. Users must be authenticated using JWT.
4. Tasks should be linked to a specific user.
5. Each task should have a title, status, and due date.
6. Users should be able to update or delete tasks.
7. Each session should have an associated user and JWT token.
8. Sessions should expire after a defined period of inactivity.
9. Each note must be linked to a task.
10. Notes can be created, updated, or deleted.
11. Users can add multiple notes to a single task.
12. Tags should be reusable across tasks and users.
13. Each task can have multiple tags, and each tag can be applied to multiple tasks.
14. Notifications must be sent for critical events like approaching due dates.
15. Each notification should be linked to a user.
16. All-important actions (e.g., creating or deleting tasks) should be logged.
17. Logs should be accessible for auditing and error detection.
18. Comments must be linked to specific tasks or notes.
19. Users can create, edit, or delete comments on tasks or notes.
20. Files must be attached to specific tasks or notes.
21. Users should be able to upload, view, and delete attachments.

II. Conceptual Data Modeling EER:



UML:



III. Mapping Conceptual Model to Relational Model

III. Mapping Conceptual Model to Relational Model

1. **User (user_id, username, password, email)**
   - user_id is the primary key.
   - username, password, and email cannot be NULL.
2. **Task (task_id, title, status, due_date, user_id, project_id)**
   - task_id is the primary key.
   - title, status, and due_date cannot be NULL.
   - user_id is the foreign key referring to user_id in the relation User and cannot be NULL.
   - project_id is the foreign key referring to project_id in the relation Project and can be NULL.
3. **Category (category_id, category_name, project_id)**
   - category_id is the primary key.
   - category_name cannot be NULL.
   - project_id is the foreign key referring to project_id in the relation Project and can be NULL.
4. **Log (log_id, action, task_id, user_id, timestamp)**
   - log_id is the primary key.
   - action and timestamp cannot be NULL.
   - task_id is the foreign key referring to task_id in the relation Task and can be NULL.
   - user_id is the foreign key referring to user_id in the relation User and can be NULL.
5. **Notification (notification_id, event_type, sent_time, user_id, task_id)**
   - notification_id is the primary key.
   - event_type and sent_time cannot be NULL.
   - user_id is the foreign key referring to user_id in the relation User and cannot be NULL.
   - task_id is the foreign key referring to task_id in the relation Task and can be NULL.
6. **Note (note_id, content, creation_date, last_modified, task_id)**
   - note_id is the primary key.
   - content, creation_date, and last_modified cannot be NULL.
   - task_id is the foreign key referring to task_id in the relation Task and cannot be NULL.
7. **Comment (comment_id, comment_text, note_id, task_id, user_id)**
   - comment_id is the primary key.
   - comment_text and user_id cannot be NULL.
   - note_id is the foreign key referring to note_id in the relation Note and cannot be NULL.
   - task_id is the foreign key referring to task_id in the relation Task and can be NULL.
8. **Reminder (reminder_id, reminder_time, task_id)**
   - reminder_id is the primary key.
   - reminder_time cannot be NULL.
   - task_id is the foreign key referring to task_id in the relation Task and cannot be NULL.
9. **Tag (tag_id, tag_name)**
   - tag_id is the primary key.
   - tag_name cannot be NULL.
10. **Attachment (attachment_id, file_path, task_id, note_id)**
    - attachment_id is the primary key.
    - file_path cannot be NULL.
    - task_id is the foreign key referring to task_id in the relation Task and can be NULL.
    - note_id is the foreign key referring to note_id in the relation Note and can be NULL.
11. **Project (project_id, project_name, start_date, end_date, description)**
    - project_id is the primary key.
    - project_name cannot be NULL.
    - start_date, end_date, and description can be NULL.
12. **Team (team_id, team_name, project_id)**
    - team_id is the primary key.

  - team_name cannot be NULL.
  - project_id is the foreign key referring to project_id in the relation Project and can be NULL.

13. **Collaborate (collaborate_id, user_id, note_id, collaboration_role)**
  - collaborate_id is the primary key.
  - collaboration_role cannot be NULL.
  - user_id is the foreign key referring to user_id in the relation User and cannot be NULL.
  - note_id is the foreign key referring to note_id in the relation Note and can be NULL.

14. **Event (event_id, event_name, date, task_id, user_id)**
  - event_id is the primary key.
  - event_name and date cannot be NULL.
  - task_id is the foreign key referring to task_id in the relation Task and can be NULL.
  - user_id is the foreign key referring to user_id in the relation User and can be NULL.

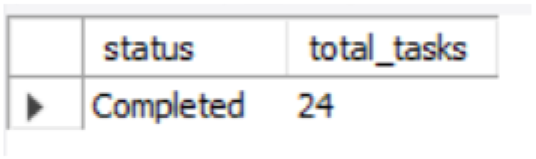
IV. Implementation of Relation Model via MySQL and NoSQL MySQL Implementation:

Random data was generated for all the tables using Mockaroo. Python was then used to ingest the data into MySQL Database post its creation.

Following are few queries that have been run along with their outputs:

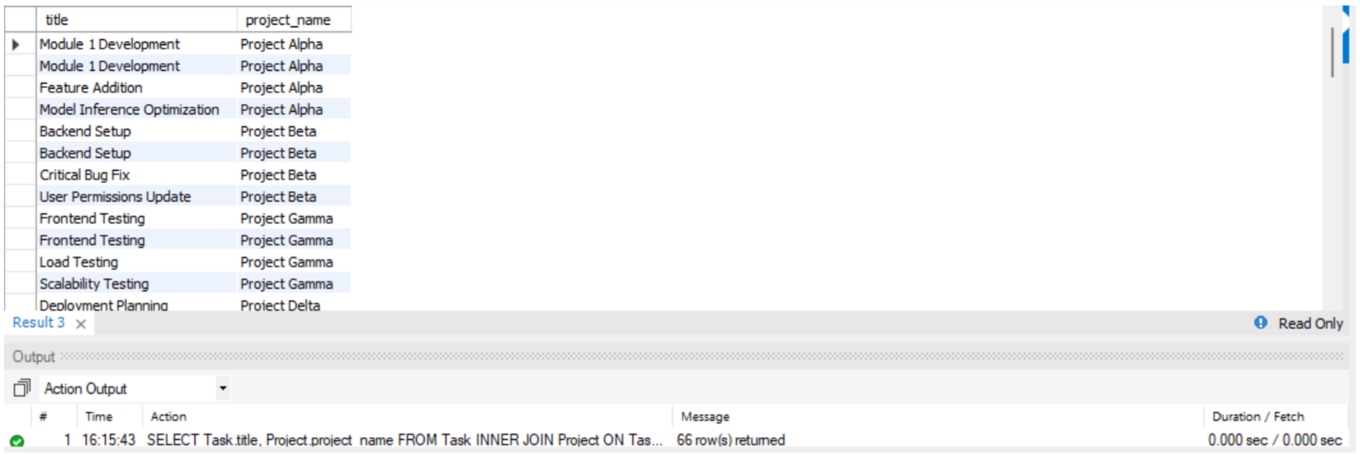
Query 1: To count the number of tasks completed.

SELECT status, COUNT(*) AS total_tasks FROM Task WHERE status = 'Completed' GROUP BY status;



	status	total_tasks
▶	Completed	24

Query 2: To list all task titles along with their corresponding project names.
SELECT Task.title, Project.project_name
FROM Task
INNER JOIN Project ON Task.project_id = Project.project_id;

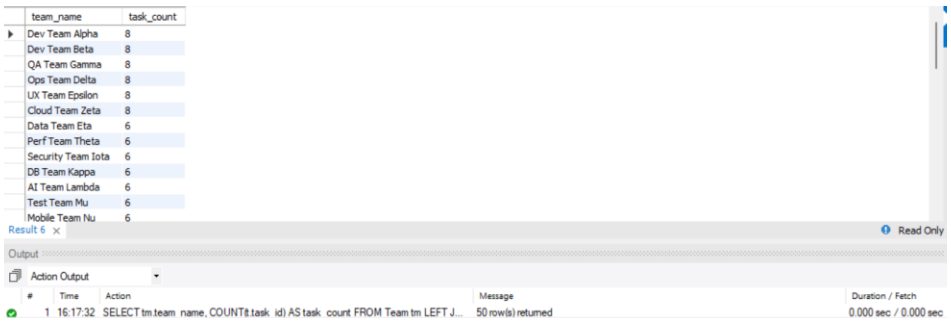


	title	project_name
▶	Module 1 Development	Project Alpha
	Module 1 Development	Project Alpha
	Feature Addition	Project Alpha
	Model Inference Optimization	Project Alpha
	Backend Setup	Project Beta
	Backend Setup	Project Beta
	Critical Bug Fix	Project Beta
	User Permissions Update	Project Beta
	Frontend Testing	Project Gamma
	Frontend Testing	Project Gamma
	Load Testing	Project Gamma
	Scalability Testing	Project Gamma
	Deployment Planning	Project Delta

#	Time	Action	Message	Duration / Fetch
1	16:15:43	SELECT Task.title, Project.project_name FROM Task INNER JOIN Project ON Tas...	66 row(s) returned	0.000 sec / 0.000 sec

Query 3: List all teams and the number of tasks associated with each, including teams that currently have no tasks.

SELECT tm.team_name, COUNT(t.task_id) AS task_count FROM Team tm LEFT JOIN Task t ON tm.project_id = t.project_id GROUP BY tm.team_name;



team_name	task_count
Dev Team Alpha	8
Dev Team Beta	8
QA Team Gamma	8
Ops Team Delta	8
UX Team Epsilon	8
Cloud Team Zeta	8
Data Team Eta	6
Perf Team Theta	6
Security Team Iota	6
DB Team Kappa	6
AI Team Lambda	6
Test Team Mu	6
Mobile Team Nu	6

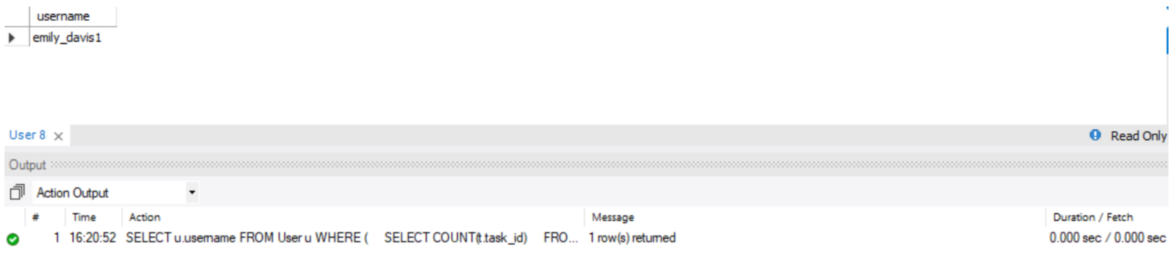
Result 6 Read Only

Output

Action Output

```
#  Time      Action                                                        Message              Duration / Fetch
1  16:17:32  SELECT tm.team_name, COUNT(t.task_id) AS task_count FROM Team tm LEFT J...  50 row(s) returned   0.000 sec / 0.000 sec
```

Query 4: Find users who have completed more than the average number of tasks across all users.
SELECT u.username FROM User u WHERE (SELECT COUNT(t.task_id) FROM Task t WHERE t.user_id = u.user_id AND t.status = 'Completed') > (SELECT AVG(task_count) FROM (SELECT COUNT(task_id) AS task_count FROM Task GROUP BY user_id) AS subquery);



username
emily_davis1

User 8 Read Only

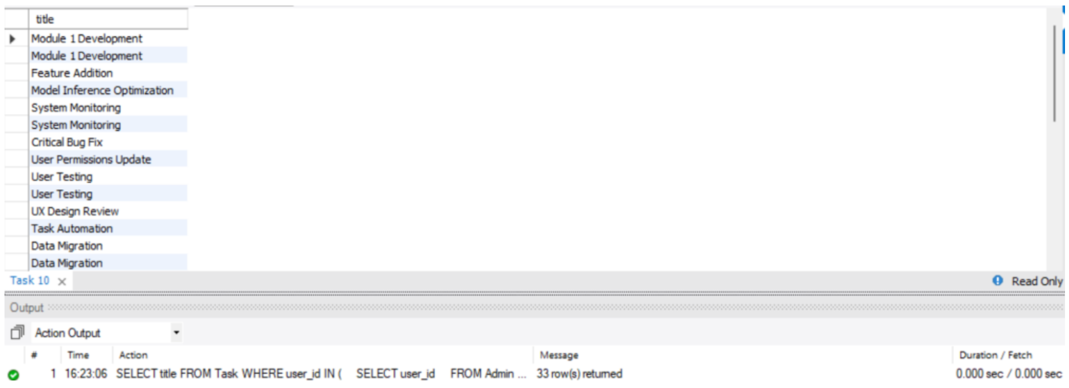
Output

Action Output

```
#  Time      Action                                                  Message             Duration / Fetch
1  16:20:52  SELECT u.username FROM User u WHERE (   SELECT COUNT(t.task_id)   FRO...  1 row(s) returned   0.000 sec / 0.000 sec
```

Query 5: Retrieve tasks assigned to users who are admins.

SELECT title FROM Task WHERE user_id IN (SELECT user_id FROM Admin);



title
Module 1 Development
Module 1 Development
Feature Addition
Model Inference Optimization
System Monitoring
System Monitoring
Critical Bug Fix
User Permissions Update
User Testing
User Testing
UX Design Review
Task Automation
Data Migration
Data Migration

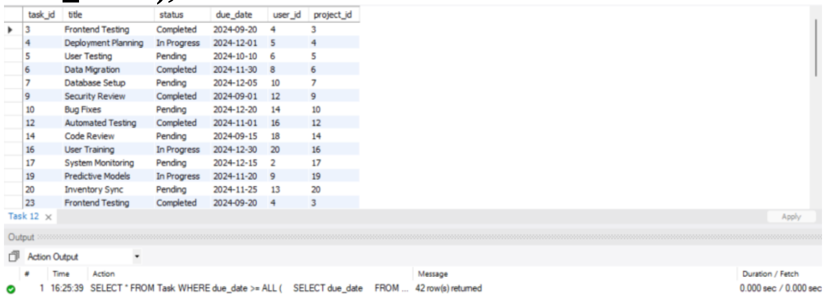
Task 10 Read Only

Output

Action Output

```
#  Time      Action                                                       Message              Duration / Fetch
1  16:23:06  SELECT title FROM Task WHERE user_id IN (   SELECT user_id   FROM Admin ...  33 row(s) returned   0.000 sec / 0.000 sec
```

Query 6: Retrieve tasks with due dates greater than all due dates of tasks assigned to user 1.
SELECT * FROM Task WHERE due_date >= ALL (SELECT due_date FROM Task WHERE user_id = 1);

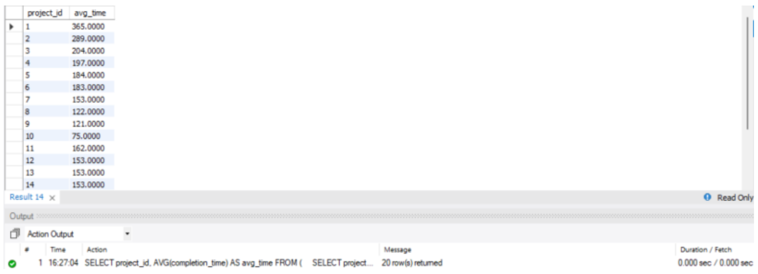


task_id	title	status	due_date	user_id	project_id
3	Frontend Testing	Completed	2024-09-20	4	3
4	Deployment Planning	In Progress	2024-12-01	5	4
5	User Testing	Pending	2024-10-10	6	5
6	Data Migration	Completed	2024-11-30	8	6
7	Database Setup	Pending	2024-12-05	10	7
9	Security Review	Completed	2024-09-01	12	9
10	Bug Fixes	Pending	2024-12-20	14	10
12	Automated Testing	Completed	2024-11-01	16	12
14	Code Review	Pending	2024-09-15	18	14
16	User Training	In Progress	2024-12-30	20	16
17	System Monitoring	Pending	2024-12-15	2	17
19	Predictive Models	In Progress	2024-11-20	9	19
20	Inventory Sync	Pending	2024-11-25	13	20
23	Frontend Testing	Completed	2024-09-20	4	3

```
Task 12
Output
Action Output
#  Time      Action                                                            Message              Duration / Fetch
1  16:25:39  SELECT * FROM Task WHERE due_date >= ALL (   SELECT due_date   FROM ...  42 row(s) returned   0.000 sec / 0.000 sec
```

Query 7: Retrieve projects where the number of associated tasks exceeds the average number of tasks per project.

```
SELECT project_name
FROM Project P
WHERE (SELECT COUNT(*) FROM Task T WHERE T.project_id = P.project_id) >
    (SELECT AVG(task_count) FROM (
        SELECT COUNT(*) AS task_count FROM Task GROUP BY project_id
    ) AS subquery);
```



project_id	avg_time
1	365.0000
2	289.0000
3	204.0000
4	197.0000
5	184.0000
6	183.0000
7	153.0000
8	122.0000
9	121.0000
10	75.0000
11	162.0000
12	153.0000
13	153.0000
14	153.0000

```
Result 14                                                                                    Read Only
Output
Action Output
#  Time      Action                                                                  Message              Duration / Fetch
1  16:27:04  SELECT project_id, AVG(completion_time) AS avg_time FROM (   SELECT project...  20 row(s) returned   0.000 sec / 0.000 sec
```

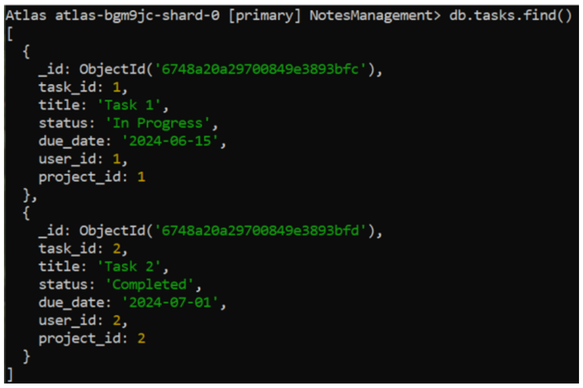
NoSQL Implementation:

Collections were created for all the MySQL tables on MongoDB.

Following are few queries that have been run on MongoShell along with their outputs:

Query 1: To find all the tasks

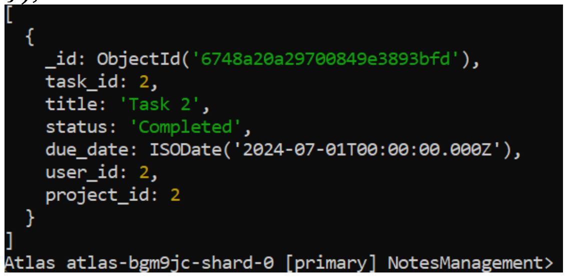
db.tasks.find();



```
Atlas atlas-bgm9jc-shard-0 [primary] NotesManagement> db.tasks.find()
[
  {
    _id: ObjectId('6748a20a29700849e3893bfc'),
    task_id: 1,
    title: 'Task 1',
    status: 'In Progress',
    due_date: '2024-06-15',
    user_id: 1,
    project_id: 1
  },
  {
    _id: ObjectId('6748a20a29700849e3893bfd'),
    task_id: 2,
    title: 'Task 2',
    status: 'Completed',
    due_date: '2024-07-01',
    user_id: 2,
    project_id: 2
  }
]
```

Query 2: Retrieve tasks with a status of "Completed" and a due_date before "2024-12-31".

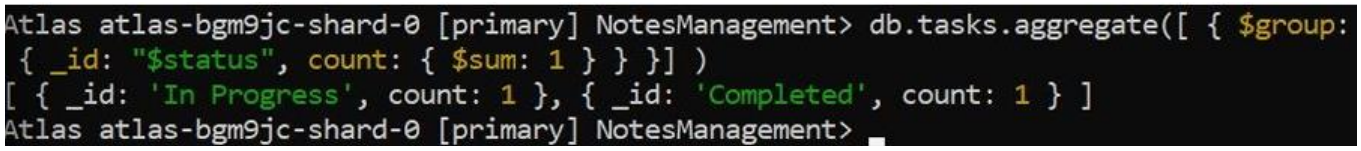
```
db.tasks.find({
  status: "Completed",
  due_date: { $lt: ISODate("2024-12-31T00:00:00Z") }
});
```



```
[
  {
    _id: ObjectId('6748a20a29700849e3893bfd'),
    task_id: 2,
    title: 'Task 2',
    status: 'Completed',
    due_date: ISODate('2024-07-01T00:00:00.000Z'),
    user_id: 2,
    project_id: 2
  }
]
Atlas atlas-bgm9jc-shard-0 [primary] NotesManagement>
```

Query 3: Count the total number of tasks grouped by their status (e.g., "In Progress," "Completed") in the database.

```
db.tasks.aggregate([{$group: { _id: "$status",count: { $sum: 1 }}}]);
```

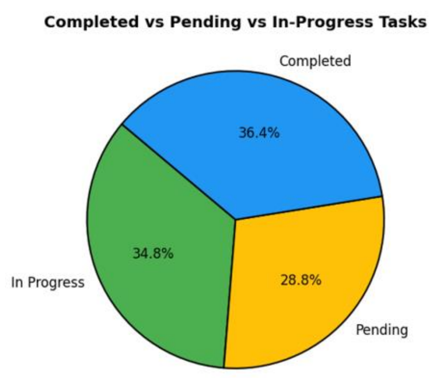


```
Atlas atlas-bgm9jc-shard-0 [primary] NotesManagement> db.tasks.aggregate([ { $group:
 { _id: "$status", count: { $sum: 1 } } }] )
[ { _id: 'In Progress', count: 1 }, { _id: 'Completed', count: 1 } ]
Atlas atlas-bgm9jc-shard-0 [primary] NotesManagement>
```

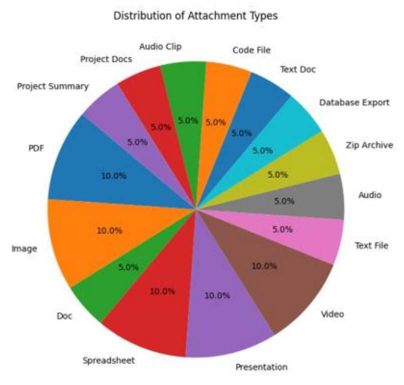
V. Database Access via R or Python

Python is used to access the MySQL database using the libraries mysql.connector and pymysql. The connection is made by creating an engine and then using engine.connect(). The read_sql function from Pandas library is then used to retrieve the data generated by the query into a dataframe. Matplotlib and Seaborn libraries are used to create visualizations for the retrieved data as shown below:

Graph 1

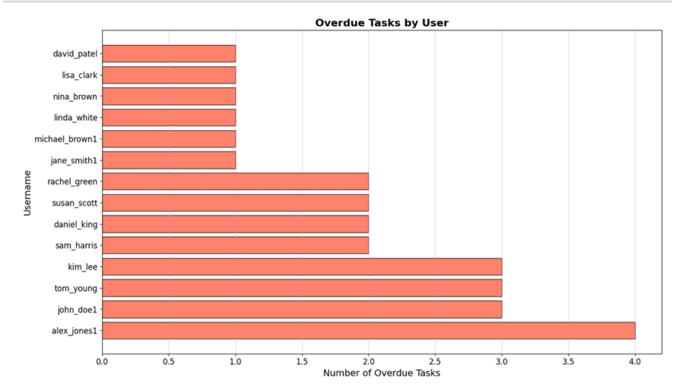


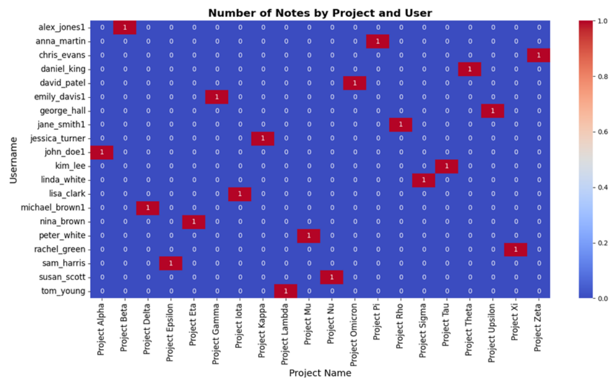
Graph 2



Graph 3

Graph 4





VII. Summary and recommendation

This project successfully demonstrates the development of a scalable, secure, and feature-rich task management web application using the MERN stack. The application integrates MongoDB, Express.js, React.js, and Node.js to provide a cohesive solution for managing daily tasks with advanced functionalities. Key features such as secure user authentication, task tagging, notifications, and logging were implemented to enhance usability and reliability. The system also supports the creation, modification, and deletion of tasks, as well as the ability to add attachments, notes, and comments, making it highly adaptable for individual and collaborative use. The conceptual design was meticulously mapped to both relational and NoSQL data models, ensuring robust handling of data relationships. Features like JWT-based session management and encrypted password storage emphasized security. MongoDB's flexibility in handling hierarchical data and the responsiveness of the React.js frontend contributed to an efficient and user-friendly system.

Advantages

1. Improved Productivity
   The application simplifies task management by providing features like tagging, notifications, and prioritization, enabling users to organize and complete tasks efficiently.
2. Scalable and Flexible
   Leveraging MongoDB’s flexible data structure and the MERN stack's architecture, the application can adapt to growing user bases and evolving requirements seamlessly.
3. Secure and Reliable
   With JWT-based authentication, encrypted password storage, and robust session handling, the application ensures user data security and reliable performance.
4. User-Friendly Design
   The React.js-powered interface offers a responsive and intuitive experience, making task management accessible and easy to use for all users.

Recommendations

1. Mobile Accessibility
   Develop a dedicated mobile application to provide users with seamless on-the-go task management, ensuring convenience and better accessibility across devices.
2. Real-Time Collaboration
   Introduce real-time collaboration features, such as shared task editing, live updates, and integrated communication tools, to support team-based workflows and dynamic projects.
3. AI and Analytics Integration
   Implement AI-driven task prioritization and analytics dashboards to help users optimize productivity, track progress, and make data-informed decisions about their workflows.