

Chapter 1

Introduction

1.1 Background

Context:

In the fast-paced environment of education, students often find it challenging to manage their academic workload efficiently. They are expected to juggle multiple tasks, such as assignments, projects, exams, and deadlines, across various subjects. Traditional methods, such as paper planners or manually written notes, are inefficient, often leading to missed deadlines, disorganized schedules, and increased stress. With the proliferation of smartphones and web-based applications, there is an opportunity to shift towards a digital study planner and organizer that helps students stay on top of their tasks and deadlines.

Problem:

Many students struggle to keep track of their assignments, exams, and projects, resulting in missed deadlines, late submissions, and a lack of structured planning. The existing methods of managing academic schedules, such as using physical planners or relying on memory, are often ineffective and prone to errors. Students often forget deadlines, underestimate the time required for tasks, and face difficulties in prioritizing their work.

Opportunity:

With the rise of technology, especially mobile apps and web applications, there is a clear opportunity to develop a digital solution that automates task tracking, sends reminders, and helps students plan their academic workload effectively. A Study Planner and Organizer system can not only track tasks but also provide reminders via email or push notifications to ensure students stay on top of their academic responsibilities.

1.2 Problem Statement

Overview of the Problem:

Students in educational institutions often struggle with managing their academic tasks, particularly when they have numerous assignments and deadlines. The manual tracking of tasks and the reliance on memory often result in disorganization and inefficiency. As a result, students face difficulties in prioritizing tasks, remembering deadlines, and managing their time effectively. This leads to stress, missed deadlines, and poor academic performance.

Specific Issues:

Difficulty in Tracking Tasks: Students have trouble keeping track of their assignments, projects, and deadlines, leading to confusion and missed deadlines.

Lack of Reminders:

Without automated reminders, students tend to forget about upcoming deadlines, which may affect the quality of their work or lead to late submissions.

Poor Time Management:

Without proper planning, students often struggle with balancing their study time, assignments, and extracurricular activities, leading to procrastination and ineffective time utilization.

Inconsistent Task Organization:

Students might rely on multiple platforms or manual systems that are not integrated, which can cause disorganization and confusion. **Lack of Real-time Updates:** Students cannot receive real-time updates or notifications about task deadlines or changes, affecting their ability to stay organized.

1.3 Objective of the System

The primary objective of the Study Planner and Organizer System is to create an automated, accurate, and efficient digital tool for students to track their academic tasks, set deadlines, and receive reminders, helping them stay organized and manage their time effectively.

Key Goals:

- **Ease of Use:** The system should allow students to easily add tasks, set deadlines, and view their schedules without complex navigation.
- **Automated Reminders:** Send reminders through email or push notifications when tasks are approaching their deadlines or when a task is incomplete.
- **Task Management:** Enable students to create, view, and manage tasks such as assignments, exams, and study sessions.
- **Progress Tracking:** Allow students to mark tasks as completed and track their progress over time.
- **Prioritization:** Enable students to prioritize tasks based on deadlines and importance.
- **Security and Privacy:** Ensure the secure storage and access of student data, including task details and personal information.
- **Scalability:** The system should be scalable to accommodate a large number of users and handle increasing task data

1.4 Significance of the System

Efficiency:

The Study Planner and Organizer system will automate and streamline the task management process for students. By digitizing the task-tracking process, students will save time and effort in managing their academic responsibilities. They will be able to focus more on studying and completing their tasks rather than worrying about missed deadlines or disorganization.

Accuracy:

By eliminating manual tracking and reliance on memory, the system ensures that tasks are correctly recorded, deadlines are met, and reminders are sent in real-time. This reduces the chances of human error, helping students stay on top of their responsibilities.

Real-Time Monitoring:

Students will have access to real-time updates on their task status. The system will notify students about upcoming deadlines or incomplete tasks, making it easier for them to prioritize and manage their academic workload effectively.

Data Analytics:

The system will provide analytics features that help students track their progress and identify patterns in their work habits. For example, students can see which types of tasks they tend to procrastinate on and adjust their study routines accordingly.

Cost-Effectiveness:

The system reduces the need for physical planners or other manual task-tracking methods. By centralizing task management in a digital format, it cuts down on paper usage, reduces the risk of misplacing information, and provides an environmentally friendly solution.

1.4 Scope of the Project

In Scope:

- **Development of a Web-Based or Mobile App :** A digital platform where students can add tasks, set deadlines, and track their progress.
- **Role-Based Access :** Students will be able to add and manage their tasks, while administrators can manage system settings.
- **Email Notifications :** The system will send reminders via email for upcoming deadlines and incomplete tasks.
- **Task Management :** Features for adding, editing, and deleting tasks, as well as marking tasks as completed.

- **Analytics :** The system will provide data visualization tools to track task completion rates and student progress over time.
- **Security :** Ensure proper data encryption and secure storage of student data.

Out of Scope:

- **Grading System :** The system will not include features related to grading or student performance tracking.
- **Social Features :** There will be no integration with social media platforms or collaborative study features.
- **Advanced Scheduling :** While the system will allow task management, advanced scheduling (like group projects or shared calendars) is not part of the project unless specify.

1.5 Methodology

Approach:

The Study Planner and Organizer will be developed using modern web and mobile technologies. The platform will be designed to be responsive, easy to use, and accessible across devices. The back-end will be built using a server-side language like Node.js or Python (Flask/Django), with a relational database like MySQL or PostgreSQL for storing task data.

Agile Development:

The development process will follow an Agile methodology, involving iterative cycles of design, development, and testing. Regular feedback from potential users (students) will help refine the features and usability of the system at each stage.

Testing:

The system will undergo several rounds of testing, including:

- **Unit Testing :** Testing individual components (e.g., task creation, reminders) to ensure correct functionality.
- **Integration Testing :** Testing the interactions between various parts of the system (e.g., task scheduling and email notification).
- **User Acceptance Testing :** Gathering feedback from students to ensure the system meets their needs and expectations.

1.6 Target Audience

Students:

The primary users of the system, who will use it to manage their academic tasks, track deadlines, and receive notifications.

Administrators:

Administrators will manage system settings, including user accounts, task categories, and security settings.

1.7 Overview of the Report

This report is structured to detail the Study Planner and Organizer system's development. The following chapters will cover:

- **Chapter 2:** System Design – This chapter will describe the architecture and design of the system, including the database schema, user interface design, and key functionalities
- **Chapter 3:** Implementation – This chapter will discuss the technologies used in building the system, including the front-end and back-end development processes.
- **Chapter 4:** Testing and Validation – This chapter will detail the testing approach, including unit tests, integration tests, and user feedback.
- **Chapter 5:** Results and Discussions – This chapter will present the results from the testing phase, as well as any challenges faced during development and implementation.
- **Chapter 6:** Conclusion and Future Enhancements – The final chapter will summarize the project and suggest areas for future improvement, such as adding advanced features like collaborative task management or integrating with calendar apps.

Conclusion

The Study Planner and Organizer system aims to solve the common problem of task management for students by automating the process of tracking assignments, exams, and other academic tasks. With features like task management, automated reminders, progress tracking, and real-time notifications, students can stay organized, reduce stress, and improve their academic performance. This digital solution will provide a user-friendly, efficient, and cost-effective tool that meets the growing demand for smart study management in today's digital age

Chapter 2

System Design

This chapter outlines the technical design of the Study Planner and Organizer System, detailing its architecture, components, and functionality. The primary objective of the system is to help users organize their study tasks, set due dates, and receive reminders for incomplete tasks. The design approach aims to create an efficient, user-friendly platform that ensures tasks are managed effectively and deadlines are met.

2.1 System Architecture

1. User Interface (UI) Module:

- Components: Login form, task creation form, task list view, reminder settings interface.
- Responsibilities: Interact with the user, capture input, display task lists and reminders.

2. Authentication Module:

- Components: User credential validator, session manager.
- Responsibilities: Manage user login and session states.

3. Task Management Module:

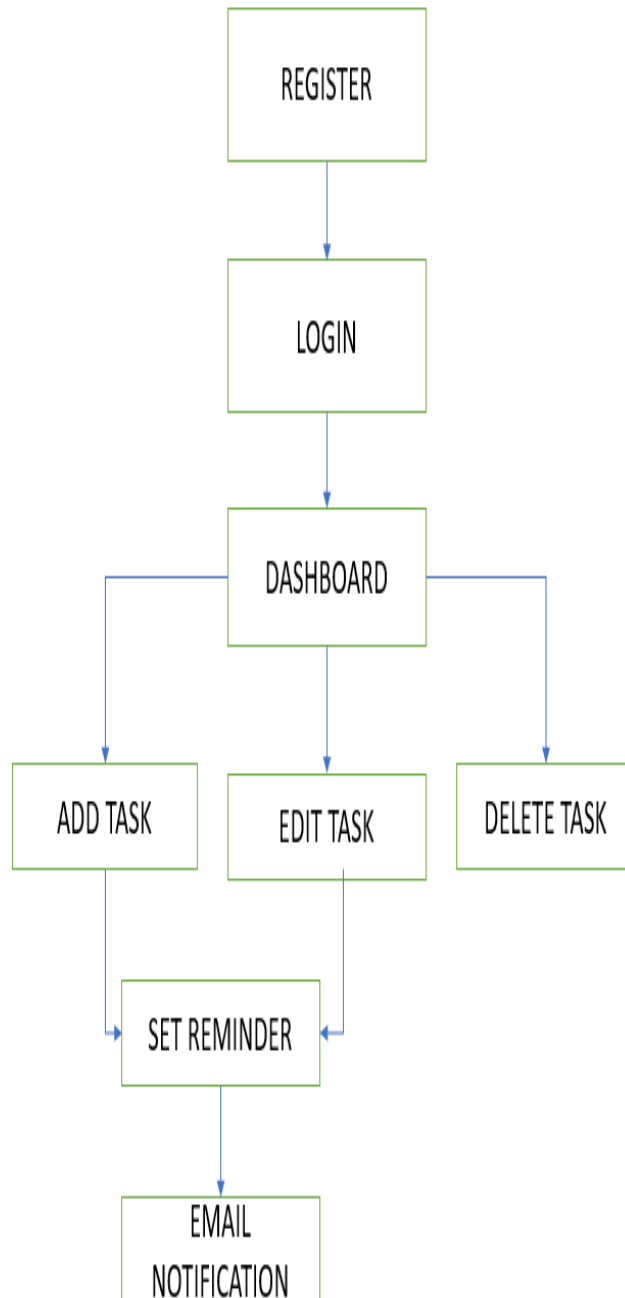
- Components: Task repository, task processing engine.
- Responsibilities: Handle task creation, modification, deletion, and status updates (completed/incomplete).

4. Reminder Management Module:

- Components: Reminder scheduler, notification sender.
- Responsibilities: Manage reminder settings, trigger notifications based on task due dates.

5. Database Module:

- Components: User data storage, task data storage, reminders data storage.
- Responsibilities: Persist user accounts, tasks, and reminders.

ARCHITECTURE DIAGRAM:

Chapter 3

Implementation

Implementation of the Study Planner and Organizer System

This chapter outlines the steps taken to implement the Study Planner and Organizer System, covering the backend, frontend, database, and integration processes. It describes the technologies used, the structure of the codebase, and any special development techniques.

3.1 Backend Implementation:

//Email sending code:

1. Import nodemailer module:
 - Require 'nodemailer' to use email-sending functionalities.
2. Configure email transporter:
 - Create a transporter using `nodemailer.createTransport`.
 - Set the service to 'gmail' for Gmail's SMTP server.
 - Provide authentication credentials (email and password or app-specific password).
 - Set TLS options (e.g., `rejectUnauthorized: false` for development purposes).
3. Define the `sendEmail` function:
 - Accept parameters: 'to' (recipient email), 'subject' (email subject), 'text' (email body content).
 - Define the 'mailOptions' object:
 - Set 'from' to the sender's email.
 - Set 'to' to the recipient's email.
 - Set 'subject' to the email's subject.
 - Set 'text' to the email's body content.
4. Send the email:
 - Use `transporter.sendMail` with the 'mailOptions' object.
 - Use 'await' to wait for the email sending process to finish.
 - If the email is sent successfully, log the response (`info.response`).
 - If an error occurs, catch the error and log it.
5. Export the `sendEmail` function:
 - Export the `sendEmail` function to be used elsewhere in the app.

Detailed Flow of Operations:

1. Import Nodemailer :
The nodemailer library is imported to enable sending emails using SMTP.
2. Configure Transporter:
The transporter object is created using `nodemailer.createTransport`, which configures the email service (Gmail in this case).
The authentication details (user and pass) are provided to allow Nodemailer to send emails through the Gmail account.
TLS options are set to allow insecure connections (useful for development, but not recommended for production).

3. sendEmail Function:

The sendEmail function is defined to accept three parameters: to, subject, and text.

Inside the function, mailOptions is created, specifying the email's sender, recipient, subject, and body content.

transporter.sendMail(mailOptions) sends the email asynchronously.

If the email is sent successfully, a success message is logged to the console with the response.

If there is an error during sending, the error is caught and logged to the console.

4. Exporting sendEmail:

The sendEmail function is exported using module.exports so that it can be imported and used in other parts of the application (e.g., in controllers or routes that trigger email sending).

3.2 Frontend Implementation:

//Register page:

BEGIN Register Component

DECLARE state variables:

username (String)

email (String)

password (String)

error (String)

isSubmitting (Boolean)

DECLARE navigate (function)

FUNCTION handleSubmit(event)

PREVENT default form submission

CLEAR error message

SET isSubmitting to true

// Validate password with regex

IF password does not match regex pattern

SET error message for password validation

SET isSubmitting to false

RETURN

TRY to send a POST request to the registration API

SET request body with username, email, and password

SET headers for content type application/json

WAIT for API response

IF response status is 201 (success)

SHOW success message: "User registered successfully"

NAVIGATE to /login page

ELSE

SET error message from response or default message

```
CATCH any errors during the request
  LOG error and SET error message to "Failed to register"
```

```
FINALLY, SET isSubmitting to false
```

```
FUNCTION render()
  RENDER a Container with:
    RENDER a Card
      RENDER Title ("Register")
      IF there is an error message:
        RENDER error message in red color
      RENDER Form:
        RENDER Input fields for username, email, and password
        RENDER a submit button with the label "Register" or "Registering..." depending on
        isSubmitting status
      RENDER a TextCenter with a message linking to the login page
```

```
END Register Component
```

```
// Styled components
DECLARE Container with flexbox layout, centering content
DECLARE Card with background, padding, and styling
DECLARE Title with bold text and blue color
DECLARE ErrorMessage with red color and centered text
DECLARE Form with flex column layout
DECLARE InputContainer with margin for spacing
DECLARE Label with block display
DECLARE Input with styling for width, padding, focus, and hover effects
DECLARE Button with background color, padding, and hover effects
DECLARE TextCenter with centered text
DECLARE StyledLink with blue text and hover effect
```

Key steps in the pseudocode:

1. State management: The component uses state variables to store the user's input, errors, and submission status.
2. Form submission handling: The `handleSubmit` function performs input validation (for password strength), sends the registration data to an API, and handles success or error responses.
3. Conditional rendering: Depending on the `isSubmitting` state, the button text changes, and if there's an error, it gets displayed.
4. UI components: Styled components are used to structure the layout and style the form.

//Login Page:

BEGIN Login Component

DECLARE state variables:

email (String)

password (String)

error (String)

DECLARE navigate (function)

FUNCTION handleSubmit(event)

PREVENT default form submission

CLEAR error message

// Ensure email and password are provided

IF email is empty OR password is empty

SET error message: "Email and password are required!"

RETURN

TRY to send a POST request to the login API

SET request body with email and password

SET headers for content type application/json

WAIT for API response

IF response status is 200 (success)

STORE JWT token in localStorage

STORE token in localStorage

CALL setIsAuthenticated(true) to update authentication status

SHOW success message: "Login successful!"

NAVIGATE to /dashboard (redirect to dashboard)

ELSE

SET error message from response or default message: "Invalid email or password"

CATCH any errors during the request

LOG error and SET error message to "Failed to login. Please try again."

FUNCTION render()

RENDER a Container with:

RENDER a Card

RENDER Title ("Login")

IF there is an error message:

RENDER error message in red color

RENDER Form:

RENDER Input fields for email and password

RENDER a submit button with the label "Login"

RENDER a TextCenter with a message linking to the register page

END Login Component

```
// Styled components
DECLARE Container with flexbox layout, centering content
DECLARE Card with background, padding, and styling
DECLARE Title with bold text and blue color
DECLARE ErrorMessage with red color and centered text
DECLARE Form with flex column layout
DECLARE InputContainer with margin for spacing
DECLARE Label with block display
DECLARE Input with styling for width, padding, focus, and hover effects
DECLARE Button with background color, padding, and hover effects
DECLARE TextCenter with centered text
DECLARE StyledLink with blue text and hover effect
```

Key steps in the pseudocode:

1. State management: The component uses state variables to handle user input for email and password, as well as any error messages.
 2. Form submission handling: The handleSubmit function ensures that both email and password fields are not empty, sends a POST request to the login API, and handles success or failure responses. On success, the JWT token is saved, and the user is redirected to the dashboard.
 3. Conditional rendering: If there is an error, it's displayed above the form. If login is successful, the user is redirected.
1. UI components: Styled components are used to organize and style the layout, input fields, and buttons.

//Dashboard:

```
// Initialize states: tasks, loading, loadingTasks, userName
define navigate (useNavigate)

function loadTasks() {
  // Set loadingTasks to true
  Retrieve authToken from localStorage

  if no authToken:
    Redirect to login

  Decode authToken and check expiration
  If expired:
```

```
    Remove authToken and Redirect to login
    Extract email from decoded token
    Set userName (email part before @)

    Fetch tasks from API (/api/tasks)
    If success:
        Set tasks state
        Set loadingTasks to false
    If error:
        Handle error (e.g., redirect to login if unauthorized)
}
```

```
function markAsCompleted(taskId) {
    Retrieve authToken
    If no authToken:
        Redirect to login

    Send PUT request to mark task as completed
    Update tasks state if successful
    Handle error if any
}
```

```
function deleteTask(taskId) {
    Retrieve authToken
    If no authToken:
        Redirect to login

    Confirm delete action
    If confirmed:
        Send DELETE request to remove task
        Update tasks state if successful
    Handle error if any
}
```

```
function editTask(taskId) {
    Navigate to /edit-task/{taskId}
}
```

```
function handleLogout() {
    Remove authToken from localStorage
    Redirect to login
}
```

```
// useEffect: Call loadTasks on initial render
useEffect(() => {
    loadTasks()
}, [])
```

```
// Render UI:
Render header with username
```

```
    Render task list or loading message
    Provide task actions (mark as completed, delete, edit)
    Provide buttons for adding tasks and viewing email logs
    Provide logout button
  }
```

Key Points:

State management: Use of `useState` to manage task data, loading states, and username.

Token handling: Secure authentication using `authToken` from `localStorage` with expiration check.

Task management: Includes functions for marking tasks as completed, deleting, editing, and viewing tasks.

Routing: Uses React Router for navigation between different parts of the app (task editing, login, etc).

//Add Task :

1. Initialize State Variables:
 - `title`: store the task title (default is empty)
 - `due_date`: store the task due date (default is empty)
 - `reminder`: store the task reminder time (default is empty)
 - `error`: store any error messages (default is empty)
 - `loading`: boolean to track loading state (default is false)
2. Initialize `navigate` function for page redirection.
3. Define `handleSubmit` function to handle form submission:
 - Prevent default form submission behavior.
 - Retrieve the authentication token from `localStorage`.
 - If no token is found:
 - Alert user to login and return from the function.
 - Clear any previous error messages.
 - Prepare the task data (`title`, `due_date`, `reminder`).
4. Handle Task Submission:
 - Set loading state to true.
 - Make a POST request to the backend API with task data:
 - Include the authentication token in the Authorization header.
 - Set the Content-Type header as `application/json`.
 - If the request fails:
 - Parse the error details and throw an error with the message.
 - If the request is successful:
 - Parse the response JSON.

- Alert the user that the task was added successfully.
 - Navigate to the dashboard page.
5. Handle any errors:
- Log the error in the console.
 - Display an error message to the user.
6. Reset the loading state to false after the API request completes.
7. Return JSX:
- Render a container with a heading "Add New Task".
 - If there is an error, display the error message.
 - Render the form with the following fields:
 - Task Title input field.
 - Due Date input field (datetime-local).
 - Reminder input field (datetime-local).
 - Submit Button (disabled when loading).
8. Define styled components:
- Container: Styles for the overall form layout.
 - Heading: Centered heading for the page.
 - Form: Flexbox layout for the form.
 - InputContainer: Styles for each input field container.
 - Label: Label styles for the input fields.
 - Input: Styling for input fields (text, datetime-local).
 - Button: Styling for the submit button (disabled state for loading).
 - Error: Styles for displaying error messages.

//Email logs:

- On component mount (useEffect):
 - Retrieve the authToken from localStorage:
 - If no token is found, redirect the user to the login page.
 - Send a GET request to the server (using axios):
 - Pass the token in the Authorization header for authentication.
 - On successful response:
 - Store the email logs in the emailLogs state.
 - Set loading to false.
 - On error:
 - If error status is 401 (Unauthorized):
 - Remove the token from localStorage and redirect to login page.
 - If other errors, set an error message.
- In the component return:
 - If loading is true, show a loading message.
 - If an error occurred, show an error message.
 - If no email logs are found, display a message: "No email logs found."
 - If email logs are available, render a list of email logs:
 - Each log displays subject, recipient (to), and sent date/time.

Detailed Flow of Operations:

1. Component Mount:

useEffect runs when the component is first mounted.

The authToken is retrieved from localStorage to check if the user is authenticated.

If no valid token is found, the user is redirected to the login page using the navigate hook.

A GET request is made using axios to fetch the email logs from the backend API, passing the token in the Authorization header for authentication.

2. Error Handling:

If the GET request is successful, the email logs are saved in the emailLogs state and the loading state is set to false.

If the request fails (e.g., token is invalid or expired), an error message is shown. In case of a 401 Unauthorized error, the token is removed from localStorage and the user is redirected to the login page.

3. Rendering the UI:

The component first checks if the data is still being fetched (loading state).

If it's still loading, a loading message is displayed.

If there's an error, an error message is shown to the user.

If email logs are available, the list of logs is displayed. Each log entry contains:

The subject of the email.

The recipient (to).

The date and time when the email was sent, formatted as a readable string.

Conclusion

The Study Planner and Organizer System provides an intuitive platform for users to manage their tasks, track deadlines, and receive timely reminders. By utilizing Node.js and Express.js

for the backend and React.js for the frontend, the system ensures fast, scalable, and user-friendly experiences. The database is structured to support efficient task management, with relationships between users, tasks, and reminders for seamless integration. The system enables users to stay on top of their study plans by offering clear task management and timely reminder.

Chapter-4

Testing

This chapter covers the testing processes and methodologies applied to the Study Planner and Organizer System. Testing is essential to identify and correct any issues, validate that the system meets functional and non-functional requirements, and ensure that it performs reliably under various conditions.

4.1 Testing Objectives

- **Verify System Functionality:** Execute a series of test cases to ensure that the system performs as expected.
- **User Role Access:** Ensure that all user roles (Admin, Student) can access the intended features without errors or unauthorized access.
- **Task Management Accuracy:** Test that tasks can be added, updated, and marked as complete with accurate due dates and reminders.
- **Reminder Functionality:** Verify that reminders for incomplete tasks are triggered on time.
- **Security:** Confirm that the system properly handles invalid inputs and unauthorized access, ensuring secure data access.
- **Performance and Reliability:** Evaluate the system's performance and reliability under load, particularly when handling a large number of tasks or users.

4.2 Testing Environment

- **Hardware:** Laptop/PC with a minimum of 8GB RAM and multi-core processor.
- **Software:** Backend and frontend hosted on local servers (Node.js for backend, React for frontend).
Database: MongoDB.
Testing Tools: Postman for API testing, Jest/Mocha for unit tests, and Cypress for end-to-end (E2E) testing.
- **Operating System:** Windows 10/macOS/Linux.
- **Browsers:** Google Chrome, Mozilla Firefox, and Microsoft Edge for cross-browser testing.

4.3 Types of Testing

4.3.1 Unit Testing

- **Objective:** To test individual components or functions in isolation to verify their correctness.
- **Tools:** Jest or Mocha for backend logic testing, and React Testing Library for frontend component testing.
- **Example Test Cases:**
 - **User Authentication:** Verifies that the login function correctly authenticates users based on their credentials.
 - **Task Creation:** Tests that tasks can be successfully created with the correct details (name, description, due date).

Reminder Function: Verifies that reminders are triggered at the correct time for incomplete task.

Integration Testing

- **Objective:** To test the interaction between different modules of the system (e.g., frontend and backend, backend and database).
- **Example Test Cases:**
 - **Task Submission:** Ensures that the frontend correctly sends task data to the backend and that it is stored correctly in the database.
 - **Reminder Triggering:** Verifies that the backend correctly sends reminders for incomplete tasks based on their due dates.
 - **User Role Access:** Confirms that only authorized users can access specific features (e.g., only admins can manage users, only students can view their tasks).

4.3.2 Functional Testing

- **Objective:** To test the system against functional requirements to ensure it meets specified user needs.
- **Test Scenarios:**
 - Login Page:**

Users can log in with their registered email and password.
If the credentials are incorrect, an error message will be displayed.
 - Registration Page:**

New users can register by providing their email, name, and password.
Users will receive a confirmation email upon successful registration.
 - Add Task:**

Once logged in, users can add a new task by providing the task title, description, due ,and priority.
The system stores the task in the database.
 - Modify Task:**

Users can modify existing tasks (e.g., update the task title, description, due date, priority).
 - Delete Task:**

Users can delete tasks from their task list.
Users can delete a task by selecting the task and choosing the delete option.
 - Get Reminders:**

Users get reminders regarding the addition and completion of particular task.
 - Task Addition Reminder:**

Users will receive an email reminder when a new task is added to their list.
 - Task Completion Reminder:**

Users will receive an email reminder a day before the due date of a task and on the day it is due
The reminders will be sent via Gmail API or SMTP.

4.4 Test Cases

Below are sample test cases for various components:

Table 4.1: Test Cases

Test Case ID	Description	Test Steps	Expected Result	Status
TC-001	Login Functionality	Login with valid credentials	User is successfully logged in.	Pass
TC-002	User Registration	Open the registration page and enters the required details.	Account is created and user is redirected to login page.	Pass
TC-003	Add Task	Enter the task details.	Task is added and appears in the task list with correct details.	Pass
TC-004	Delete Task	Select an existing task and click delete to confirm deletion.	The selected task is deleted from the task list.	Pass
TC-005	Modify Task	Select an existing task and modify the task details.	Task details are uploaded in the task list.	Pass
TC-006	Email Reminder for Task.	Add a task with deadline, wait for the deadline time and check Gmail inbox for reminder email.	Reminder email is received on time.	Pass

Chapter 5

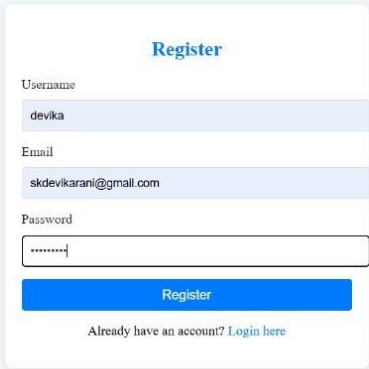
Results and Discussion

This chapter summarizes the results of the **Study Planner and Organizer** project, discussing its effectiveness, reliability, and alignment with the intended objectives. It also covers any challenges encountered, key insights, and recommendations for future improvements.

5.1 Results

Snapshots of the Project with description:

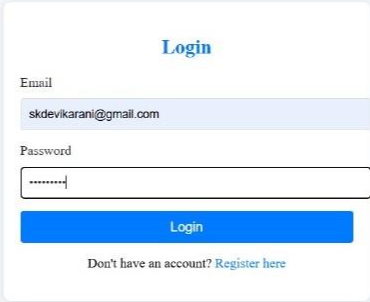
Snapshot 5.1.1:



The image shows a web registration form titled "Register". It contains the following fields and elements:

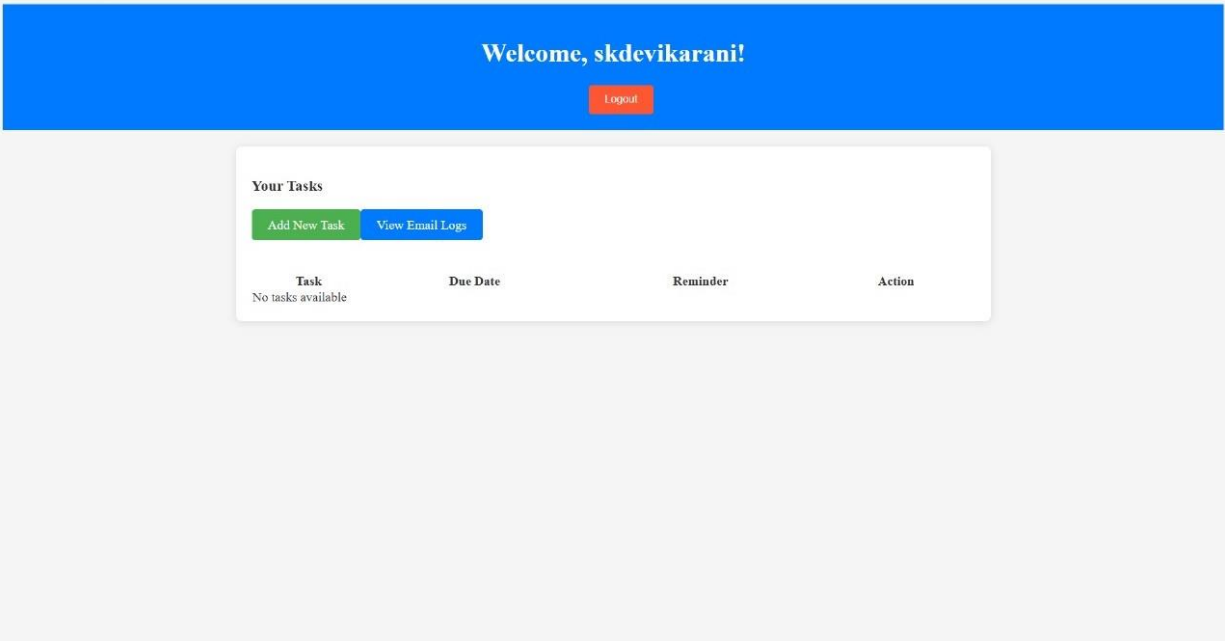
- Username:** A text input field containing the value "devika".
- Email:** A text input field containing the value "skdevikarani@gmail.com".
- Password:** A password input field with masked characters (asterisks).
- Register Button:** A blue button labeled "Register".
- Footer Link:** A link below the button that reads "Already have an account? [Login here](#)".

Snapshot 5.1.1: This is the snapshot of the registration page, where new users can create an account to access the Study Planner and Organizer system.

Snapshot 5.1.2:

A login form titled "Login" is centered on a light blue background. The form has a white background and a thin grey border. It contains two input fields: "Email" with the value "skdevikarani@gmail.com" and "Password" with masked characters "*****". Below the password field is a blue "Login" button. At the bottom of the form, there is a link that says "Don't have an account? [Register here](#)".

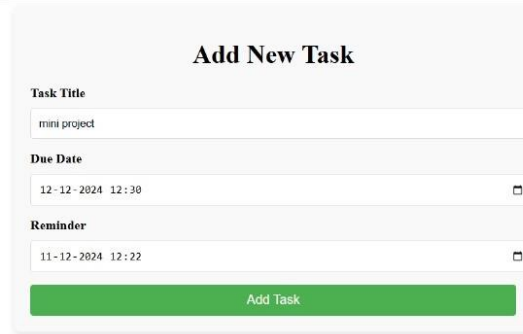
Snapshot 5.1.2: This is the snapshot of the login page, allowing users to securely sign in to their personalized task management dashboard.

Snapshot 5.1.3:

The dashboard interface features a blue header bar with the text "Welcome, skdevikarani!" and a red "Logout" button. Below the header, there is a white box titled "Your Tasks" containing two buttons: "Add New Task" (green) and "View Email Logs" (blue). Underneath these buttons is a table with the following structure:

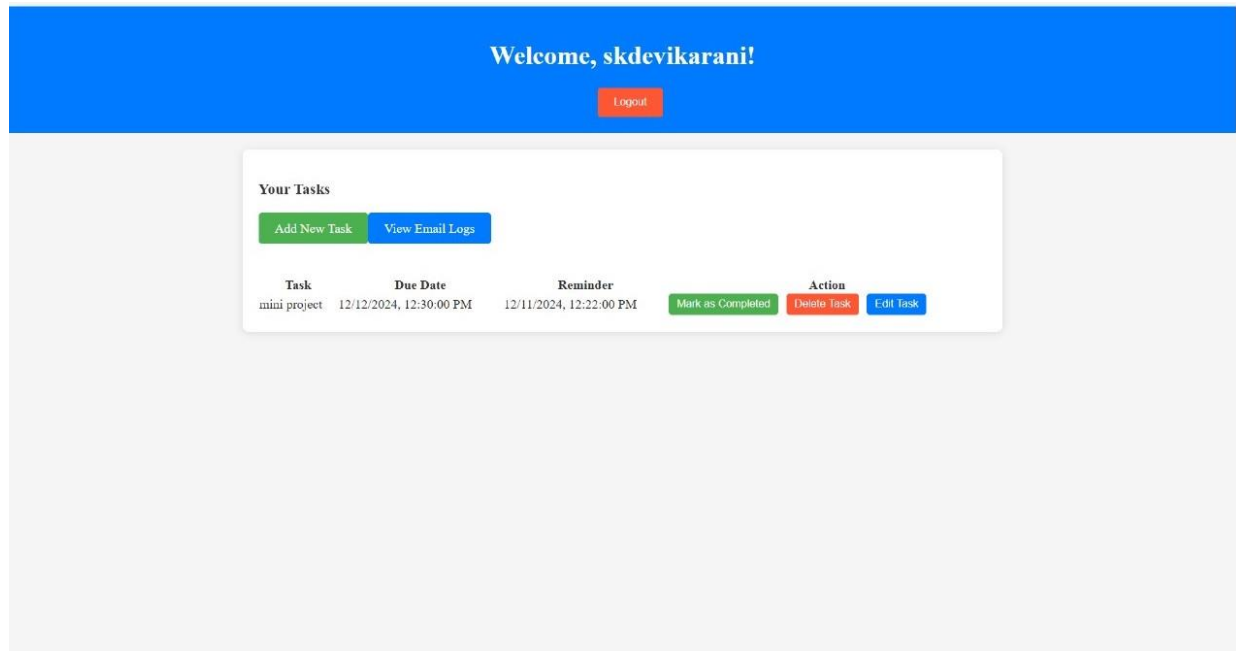
Task	Due Date	Reminder	Action
No tasks available			

Snapshot 5.1.3: This is the snapshot of the dashboard, displaying an overview of the user's tasks, deadlines, and completion status.

Snapshot 5.1.4:

The 'Add New Task' form is a light gray box with a title 'Add New Task' at the top. It contains three input fields: 'Task Title' with the text 'mini project', 'Due Date' with the date and time '12-12-2024 12:30', and 'Reminder' with the date and time '11-12-2024 12:22'. Each input field has a small calendar icon on the right. At the bottom of the form is a green button labeled 'Add Task'.

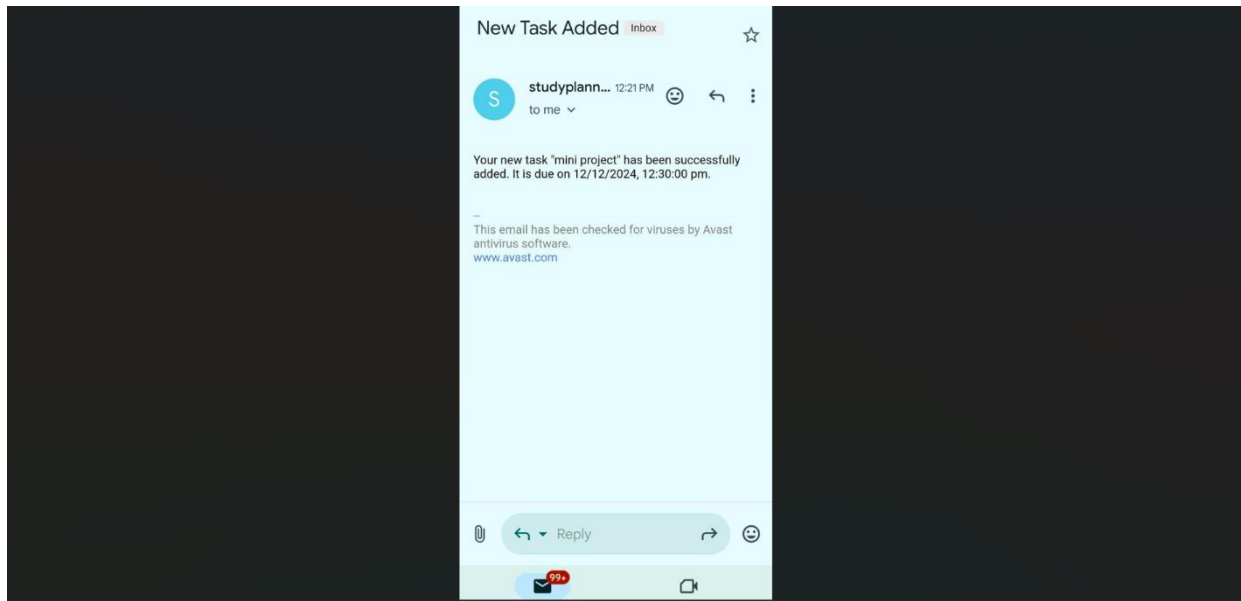
Snapshot 5.1.4: This is the snapshot of the 'Add New Task' page, where users can enter task details such as title, description, and due date.

Snapshot 5.1.5:

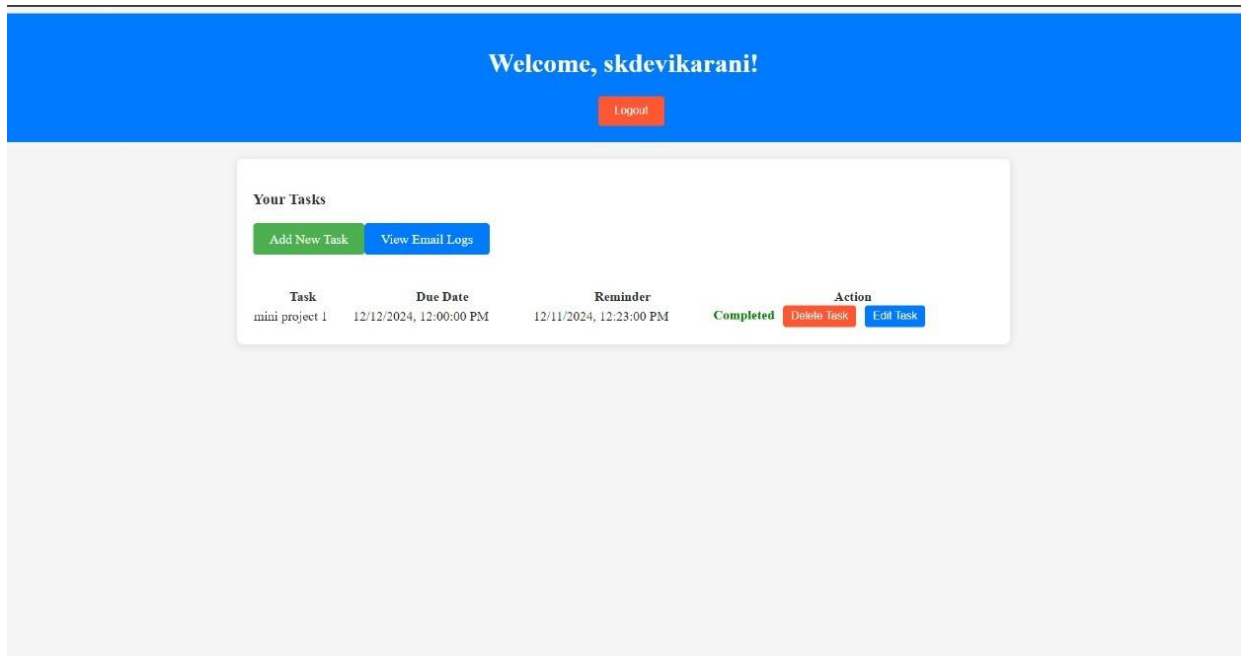
The user dashboard features a blue header with the text 'Welcome, skdevikarani!' and a 'Logout' button. Below the header is a 'Your Tasks' section. This section contains two buttons: 'Add New Task' (green) and 'View Email Logs' (blue). Below these buttons is a table with the following data:

Task	Due Date	Reminder	Action
mini project	12/12/2024, 12:30:00 PM	12/11/2024, 12:22:00 PM	Mark as Completed Delete Task Edit Task

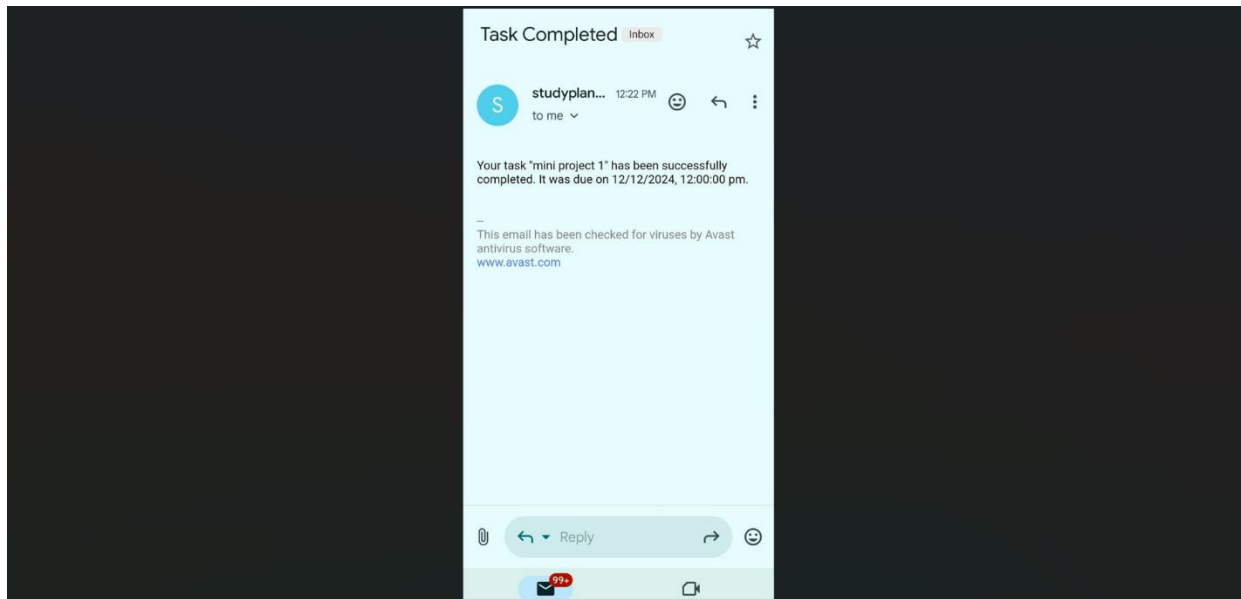
Snapshot 5.1.5: This is the snapshot showing a confirmation , after a new task has been successfully added to the system.

Snapshot 5.1.6:

Snapshot 5.1.6: This is the snapshot of the email notification sent to the user, confirming the successful addition of a task along with its due date.

Snapshot 5.1.7:

Snapshot 5.1.7: This is the snapshot of the task list showing a task that has been marked as completed.

Snapshot 5.1.8:

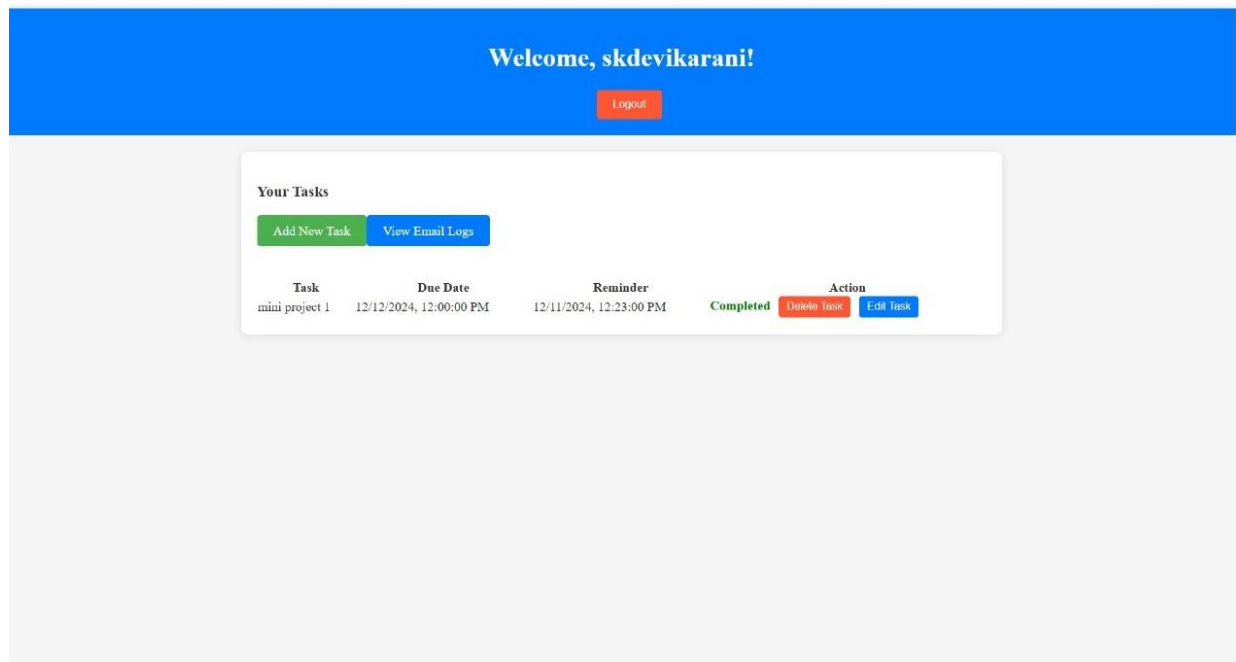
Snapshot 5.1.8: This is the snapshot of the email notification confirming that a task has been completed successfully.

Snapshot 5.1.9:A screenshot of the "Edit Task" form. It contains the following fields:

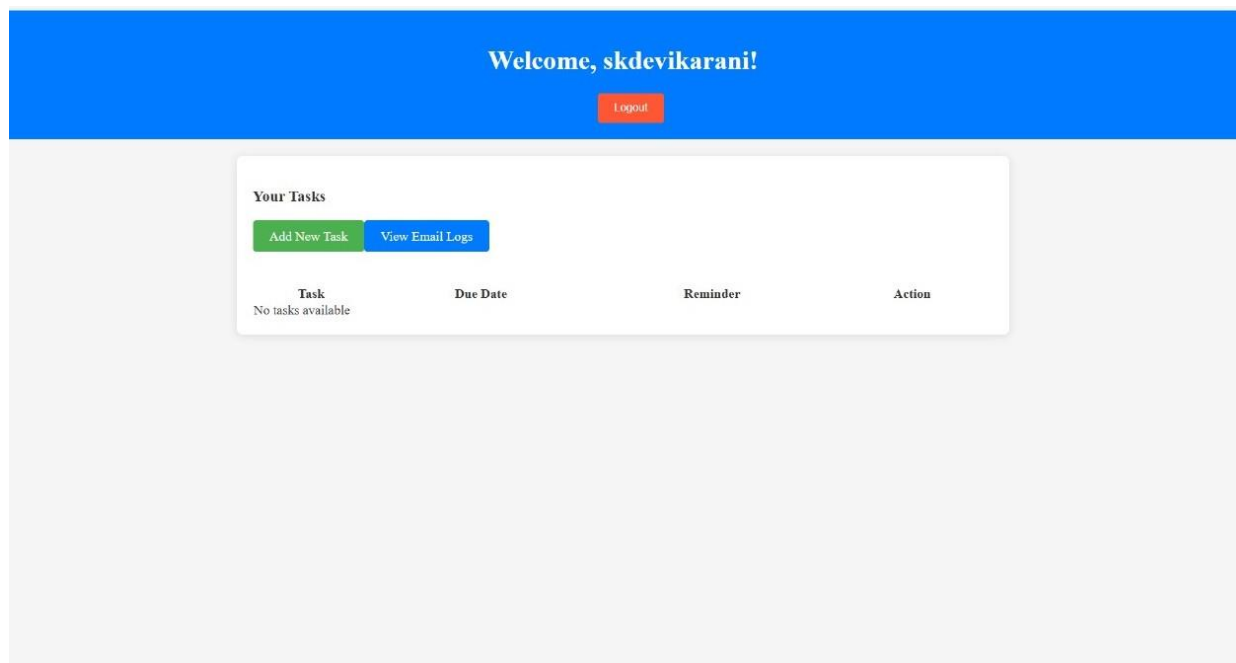
- Task Title:** mini project 1
- Due Date:** 12-12-2024 12:00
- Reminder:** 11-12-2024 12:23
- Completed:** ☐

A green "Update Task" button is at the bottom.

Snapshot 5.1.9: This is the snapshot of the task editing page, where users can modify details of an existing task.

Snapshot 5.1.10:

Snapshot 5.1.10: This is the snapshot showing the updated task details after modifications have been made.

Snapshot 5.1.11:

Snapshot 5.1.11: This is the snapshot of the task deletion page, where users can remove tasks from their list.

Snapshot 5.1.12:

Email Logs

- **Task Completed** - skdevikarani@gmail.com - 12/11/2024, 12:22:35 PM
- **New Task Added** - skdevikarani@gmail.com - 12/11/2024, 12:21:21 PM
- **New Task Added** - skdevikarani@gmail.com - 12/10/2024, 4:17:21 PM
- **Task Completed** - skdevikarani@gmail.com - 12/10/2024, 1:27:43 PM
- **New Task Added** - skdevikarani@gmail.com - 12/10/2024, 1:27:31 PM
- **Task Completed** - skdevikarani@gmail.com - 12/9/2024, 11:57:21 AM
- **Task Completed** - skdevikarani@gmail.com - 12/9/2024, 11:57:20 AM
- **New Task Added** - skdevikarani@gmail.com - 12/9/2024, 11:56:59 AM
- **Task Completed** - skdevikarani@gmail.com - 12/9/2024, 11:20:36 AM
- **Task Completed** - skdevikarani@gmail.com - 12/9/2024, 11:20:35 AM
- **New Task Added** - skdevikarani@gmail.com - 12/9/2024, 11:20:13 AM
- **Task Completed** - skdevikarani@gmail.com - 12/8/2024, 10:23:25 PM
- **New Task Added** - skdevikarani@gmail.com - 12/8/2024, 10:22:08 PM

Snapshot 5.1.12: This is the snapshot of the email logs page, displaying a record of all email notifications sent to the user regarding tasks.

Snapshot 5.1.13:

Snapshot 5.1.13: This is the snapshot of the logout page, allowing users to securely log out of their account and end their session.

5.2 Discussion

5.2.1 Challenges Encountered:

- **User Authentication Issues:** Ensuring secure login and registration was a challenge, especially when handling different types of password hashing and verification methods for enhanced security.
- **Task Duplication:** Preventing users from adding duplicate tasks was difficult, and required extra logic to ensure unique task titles or deadlines.
- **Email Delivery Reliability:** Sometimes email notifications were delayed or not received, primarily due to issues with email service providers or API limits. Ensuring timely and consistent email reminders was a challenge.
- **User Experience:** Optimizing the interface for different screen sizes and ensuring an intuitive task management experience, especially in the dashboard, required continuous refinement. Additionally, providing clear feedback messages (e.g., task added successfully, task marked as completed) was vital for a positive user experience.

5.2.2 Limitations of the current system:

- **Manual Task Management:** The system currently requires users to manually add, edit, or delete tasks. Automation features, such as auto-adding tasks based on calendar events or class schedules, are not yet supported.
- **Email-Based Reminders:** The reminder system is limited to email notifications and lacks real-time reminders such as push notifications, which could enhance user engagement.
- **No Task Prioritization:** The system does not currently support task prioritization (e.g., marking tasks as high, medium, or low priority), limiting the ability for users to prioritize their workload effectively.
- **Limited Platform Support:** The system is primarily designed for desktop use and is not optimized for mobile or tablet devices. A dedicated mobile app or mobile-responsive web design would increase accessibility.
- **No Offline Functionality:** Users are required to be connected to the internet to interact with the system, which might be a limitation in areas with poor or no internet access.

Chapter 6

Conclusion and Future Enhancements

1. Conclusion

- The **Study Planner and Organizer** system successfully achieved its primary goal of helping users manage and track academic tasks efficiently. By allowing users to log in, add tasks with due dates, and set reminders for incomplete tasks, the system offers a user-friendly solution that enhances time management and task completion.
- Overall, the Study Planner and Organizer system demonstrates the benefits of digital task management, improving productivity by allowing students and teachers to stay on top of their tasks and deadlines. The reminder feature has proven to be particularly valuable in helping users track and complete tasks on time, reducing the likelihood of missed deadlines.
- The system provides a reliable way for students to manage their study schedules, organize tasks, and receive timely reminders, thereby improving overall academic performance. Teachers can also use the platform to track tasks assigned to students, ensuring better planning and organization of academic work.

2. Future Enhancements

To further increase the effectiveness and usability of the Study Planner and Organizer, the following enhancements are recommended:

Push Notifications and Reminders

- Integrating push notifications into the system would allow reminders to be sent directly to users about their pending tasks and deadlines. For instance, students would be alerted when their tasks are nearing their due date or if they have incomplete tasks.
- Teachers and students could benefit from push notifications that remind them about upcoming assignments, deadlines, or task updates, ensuring that no task is missed. Additionally, reminders for recurring tasks or long-term goals could help users stay on track with their schedules.

Mobile Application Development

- Developing dedicated mobile applications for **Android** and **iOS** platforms would significantly improve accessibility and user experience. A mobile app would make it easier for users to interact with the system on the go, whether they are adding new tasks, updating deadlines, or checking reminders.
- The app would provide a more seamless experience for users with mobile-optimized features, such as real-time task syncing, offline task management, and notifications, allowing users to manage their tasks from anywhere.

Task Categorization and Prioritization

- Adding features for **task categorization** (e.g., academic, personal, work-related) and **prioritization** (e.g., high, medium, low) would further enhance task organization and management. Users would be able to better focus on important tasks, ensuring that they meet deadlines for priority items.
- Task categorization could also allow for specialized task views (e.g., by subject or course) to help students and teachers filter tasks based on specific areas of focus.

Integration with Calendar Apps

- Integrating the system with popular calendar apps (e.g., Google Calendar, Outlook) could allow users to visualize their tasks alongside their other commitments. Synchronization with a calendar would give users an at-a-glance overview of upcoming tasks and deadlines, and allow them to manage their time more effectively.

REFERENCES

Citation Format:

Technologies :

- Node.js : <https://nodejs.org/api/index.html>
- MongoDB : <https://www.mongodb.com/docs/>
- React : <https://react.dev/>
- Express.js : https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

Online Resources :

- <https://openai.com/chatgpt/overview/>
- <http://in.youtube.com/>
- <https://www.google.co.in/>