National College of Ireland

Technical Report

Task Management Web Application

12th- May 2024

Bachelor of Science (Honours) in Computing

Software Development

Academic Year i.e., 2023/2024

Satyam Sehgal

X19104464

X19104464@student.ncirl.ie

Contents

Contents 1
Executive Summary 3
1.0 Introduction 3
1.1. Background 3
1.2. Aims 3
1.3. Technology 4
2.0 System 5
2.1. Requirements 5
2.1.1. Functional Requirements 5
2.1.1.1. Use Case Diagram 7
2.1.1.2. Requirement 1: Create Tasks 7
Description & Priority 7
Use Case 7
2.1.1.3. Requirement 2: Read Tasks 8
Description & Priority 8
Use Case 8
2.1.1.4. Requirement 3: Edit Tasks 9
Description & Priority 9
Use Case 9
2.1.1.5. Requirement 4: Delete Tasks 10
Description & Priority 10
Use Case 10
2.1.1.6. Requirement 5: Filter Tasks through Categories 11
Description & Priority 11
Use Case 11
2.1.1.7. Requirement 6: Manage Tasks with Drag and Drop 12
Description & Priority 12
Use Case 12
2.1.1.8. Requirement 7: View Future and Past Tasks 13
Description & Priority 13
Use Case 13
2.1.1.9. Requirement 8: View Specific Week Tasks 14
Description & Priority 14

Use Case 14
2.1.1.10. Requirement 9: View Specific Week Tasks 15
Description & Priority 15
Use Case 15
2.1.1.11. Requirement 10: Create Template/Archive Tasks 16
Description & Priority 16
Use Case 16
2.1.1.12. Requirement 2: Read Template/Archive Tasks 17
Description & Priority 17
Use Case 17
2.1.1.13. Requirement 3: Edit Template/Archive Tasks 18
Description & Priority 18
Use Case 18
2.1.1.14. Requirement 4: Delete Template/Archive Tasks 19
Description & Priority 19
Use Case 19
2.1.1.15. Requirement 14: Filter Template Tasks through Categories 20
Description & Priority 20
Use Case 20
2.1.1.16. Requirement 15: Manage Template Tasks with Drag and Drop 21
Description & Priority 21
Use Case 21
2.1.2. Data Requirements 22
2.1.3. User Requirements 23
2.1.4. Environmental Requirements 23
2.1.5. Usability Requirements 24
2.2. Design & Architecture 24
2.3. Implementation 25
2.4. Graphical User Interface (GUI) 55
2.5. Testing 61
2.6. Evaluation 65
3.0 Conclusions 66
4.0 Further Development or Research 67
5.0 References 68
6.0 Appendices 68
6.1. Project Proposal 68

1.1. Reflective Journals 74

Executive Summary

The project aims to develop a user-friendly Task Management Web Application modified for individual use. It focuses on simplicity, ease of use, and real-time updates, distinguishing it from existing tools designed for team collaboration with no automation. The technical approach involves an Agile methodology, emphasizing regular progress and adaptation. The development stack includes Next.js for the frontend, Nest.js for the backend, and Redis for data management. Real-time updates are conducted through the WebSocket technology and Bull Queues/Web workers.

1.0 Introduction

1.1. Background

I got motivated by the absence of straightforward tools for individual task management. Most tools out there are made for teams and work life, which can be confusing for individuals. So, I decided to make a simple and easy-to-use tool just for personal use with a vast number of features. I will pay a lot of attention to making it user-friendly and adding features like real-time updates and a lot of automation tasks i.e., notification sending will relay if a task is still pending or not.

1.2. Aims

My Project aims to achieve the following:

1. User-Friendly Task Management
2. Simplicity and Ease of Use
3. Real-Time Updates
4. Personal Task Organization priority vise
5. Simple task automations
6. Task Prioritization
7. Archive Tasks
8. Task Categories Management
9. Automation of Recurring Tasks and more
10. Agile Development and Adaptability
11. Testing
12. Deployment and User Training

1.3. Technology

The Task Management Web Application is built using a variety of technologies to ensure it works well, is easy to use, and can keep information updated in real time. The technologies used for different parts of the project are as follows:

Frontend Development

Technology: JavaScript, Next, HTML, CSS

Backend Development

Technology: Nest.js, Redis

Real-Time Updates

Technology: WebSocket

Version, Special Integrations Control, AI Automation

Technology: Bull Queues, Web Sockets, Git, REST API, JWT, Swagger

This combination of these technologies will help me create a strong, flexible, and easy-to-use app that meets the project's goals. By these technologies it’s possible to provide a smooth and productive user experience while also handling specific requirements like real-time updates and task prioritization. By version control, I can ensure that the code is stable and manageable throughout the development process.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements

Feature	Requirement Description
Create Tasks	Users can create new tasks by entering the task name, category, and priority, starting time, repeat days in a week (if any) and estimated task time. The application will store the task information.
Read Tasks	Users can view their existing tasks in a 2-column list format. The tasks will be displayed based on their category and priority.
Edit Tasks	Users have the ability to modify task details, such as the task name, category, or priority level. This feature allows for easy updates to task information.
Delete Tasks	Users can remove unwanted tasks from their task list. This feature helps keep the task list clean and organized.
Filter Tasks through Categories	Users can select multiple category filters to view relevant tasks. When no filters are set then all tasks are viewed.
Manage Tasks with Drag and Drop	The application provides a drag and drop functionality that allows users to easily create a new task by dropping into the create new task form
View Future and Past tasks	Users can scroll through the dates and select required month to view the tasks of specific days in the future or the past
View specific week tasks	Users can skip whole weeks to view tasks present in the future or past week accordingly
Create task template for Archive Tasks	Users can create template/archive Tasks so they can create tasks from already created templates without entering all the task information again.
View Template/Archive Tasks	Users can view their existing template/archive tasks in a 2-column list format. The template/archive

	tasks will be displayed based on their category and priority.
Edit Template/Archive Tasks	Users have the ability to modify Template/Archive task details, such as the task name, category, or priority level. This feature allows for easy updates to task information.
Delete Template/Archive Tasks	Users can remove unwanted Template/Archive tasks from their task list. This feature helps keep the Template/Archive task list clean and organized.
Filter Template/Archive Tasks through Categories	Users can select multiple category filters to view relevant Template/Archive tasks. When no filters are set then all Template/Archive tasks are viewed.
Manage Template/Archive Tasks with Drag and Drop	The application provides a drag and drop functionality that allows users to easily create a new Template/Archive task by dropping into the create new task form
Create Categories for Tasks	Users can create new categories by selecting the icon from the list provided and adding a name for the category. Categories help in filtering tasks so users can have a clean view.
Delete Categories for Tasks	Users can delete unwanted categories. This will also delete all the tasks created under that category with the help of bull queue and websocket so if there are thousands of tasks then the frontend does not get stuck and a thread/process is started in backend to delete all relevant tasks and just show alert message on front end when the deletion is done.
Edit user profile password	Users can update password if they want to change the password in case of breach
Logout	Users can log out from the system if they want to.

2.1.1.1. Use Case Diagram

2.1.1.2. Requirement 1: Create Tasks

Description & Priority

High. Allows users to create new tasks with essential details such as name, category, priority, start time, repetition, and estimated duration.

Use Case

Scope

Task management within the application.

Description

Users can input task details into a form and submit it to create a new task.

Flow Description

Precondition

User must be logged in.

Activation

User navigates to the task creation form.

Main flow

1. User fills out the task creation form.
2. User submits the form.
3. System validates the input.
4. System stores the task information in the database.
5. System confirms task creation to the user.

Alternate flow

A1. If input validation fails, the system prompts the user to correct the data.

A2. If a user selects a past date, the system prompts the user to enter present or future date

Exceptional flow

E1. System failure during save operation results in error notification to the user.

2.1.1.3. Requirement 2: Read Tasks

Description & Priority

High. Enables users to view tasks in a categorized and prioritized list.

Use Case

Scope

Viewing tasks within the application.

Description

Users can view a list of their tasks organized by categories and priorities.

Flow Description

Precondition

Tasks exist in the system.

Activation

User opens the task viewing/planning page.

Main flow

1. User accesses the task list page.
2. System retrieves tasks from the database.
3. System displays tasks in a 2-column list format.

Alternate flow

A1. No tasks exist, and the system displays a "No Tasks" message.

Exceptional flow

E1. Failure to retrieve tasks results in an error message.

2.1.1.4. Requirement 3: Edit Tasks

Description & Priority

Medium. Allows users to modify details of existing tasks.

Use Case

Scope

Task modification within the application.

Description

Users can change task details like name, category, starting time, estimated time, repeat days in the week and priority,

Flow Description

Precondition

Tasks exist in the system.

Activation

User selects a task to edit.

Main flow

1. User clicks on the edit option of a task.
2. System displays the task details in an editable form.
3. User modifies the required fields and submits the form.
4. System validates and updates the task details.
5. System confirms the successful update to the user.

Alternate flow

A1. User aborts the edit operation.

A2. If input validation fails, the system prompts the user to correct the data.

A3. If a user selects a past date, the system prompts the user to enter present or future date

Exceptional flow

E1. System error during update results in an error message.

2.1.1.5. Requirement 4: Delete Tasks

Description & Priority

Medium. Provides functionality for users to delete tasks from their list.

Use Case

Scope

Task deletion within the application.

Description

Users can remove tasks they no longer need.

Flow Description

Precondition

Tasks exist in the system.

Activation

User selects a task to delete.

Main flow

1. User clicks on the delete option for a task.
2. System asks for confirmation to delete the task.
3. User confirms deletion.
4. System removes the task from the database.
5. System confirms deletion to the user.

Alternate flow

A1. User cancels the deletion process.

Exceptional flow

E1. Failure during deletion leads to an error message.

2.1.1.6. Requirement 5: Filter Tasks through Categories

Description & Priority

Medium. Allows users to filter tasks based on selected categories.

Use Case

Scope

Filtering tasks in the application.

Description

Users can apply category filters to the task list to narrow down the visible tasks.

Flow Description

Precondition

Tasks exist in the system.

Activation

User selects filter options from the category chips interface.

Main flow

1. User selects one or more categories from the filter options.
2. System filters tasks based on the selected categories.
3. System displays only the tasks that match the selected categories.

Alternate flow

A1. No tasks match the selected filters, and the system displays a "No Tasks Found" message.

Exceptional flow

E1. If there is a system error during filtering, an error message is shown.

2.1.1.7. Requirement 6: Manage Tasks with Drag and Drop

Description & Priority

Medium. Enables users to create a copy of a task using a drag-and-drop interface to create a new task form on the right pane.

Use Case

Scope

Creating new tasks within the application with drag and drop.

Description

Users can create a copy of a task using a drag-and-drop interface to create a new task form on the right pane. When the task gets dragged to the right pane then all the created tasks fields are populated with the dropped task fields.

Flow Description

Precondition

Tasks exist in the system.

Activation

User starts dragging a task.

Main flow

1. User clicks and holds a task.
2. User drags the task to the create new task form on the right pane.
3. User releases the task.
4. System updates the task's position and fills out the task info to create a new task form on the right pane.
5. System confirms the update.

Alternate flow

A1. User aborts the drag operation by releasing the task in the left tasks list pane.

Exceptional flow

E1. If the system fails to update the task position, an error message is displayed.

2.1.1.8. Requirement 7: View Future and Past Tasks

Description & Priority

Medium. Users can view tasks scheduled for future or past dates.

Use Case

Scope

Viewing tasks by date within the application.

Description

Users can navigate through the week swiper and month selector to view tasks for selected dates.

Flow Description

Precondition

Tasks exist for the selected dates.

Activation

User navigates to the week swiper and month selector.

Main flow

1. User selects a date from the week selector and a month from month selector.
2. System retrieves tasks for the chosen date.
3. System displays tasks for that date.

Alternate flow

A1. No tasks exist for the selected date, and the system displays a "No Tasks" message.

Exceptional flow

E1. Failure to retrieve tasks results in an error message.

2.1.1.9. Requirement 8: View Specific Week Tasks

Description & Priority

Medium. Allows users to view tasks for a specific week.

Use Case

Scope

Viewing weekly tasks in the application.

Description

Users can select a week to view all tasks assigned to that week.

Flow Description

Precondition

Tasks are scheduled in the system by week.

Activation

User selects a week from the week navigator.

Main flow

1. User selects a specific week.
2. System retrieves all tasks for that week.
3. System displays the tasks.

Alternate flow

A1. No tasks are scheduled for the selected week, prompting a "No Tasks" message.

Exceptional flow

E1. Failure to retrieve tasks results in an error message.

2.1.1.10. Requirement 9: View Specific Week Tasks

Description & Priority

Medium. Allows users to view tasks for a specific week.

Use Case

Scope

Viewing weekly tasks in the application.

Description

Users can select a week to view all tasks assigned to that week.

Flow Description

Precondition

Tasks are scheduled in the system by week.

Activation

User selects a week from the week navigator.

Main flow

4. User selects a specific week.
5. System retrieves all tasks for that week.
6. System displays the tasks.

Alternate flow

A1. No tasks are scheduled for the selected week, prompting a "No Tasks" message.

Exceptional flow

E1. Failure to retrieve tasks results in an error message.

2.1.1.11. Requirement 10: Create Template/Archive Tasks

Description & Priority

High. Allows users to create new template/archive for frequently used tasks with essential details such as name, category, priority, start time, repetition, and estimated duration. This saves time again in creating a task from scratch as a template is already available.

Use Case

Scope

Template/archive Task management within the application.

Description

Users can input template/archive task details into a form and submit it to create a new template/archive task.

Flow Description

Precondition

User must be logged in.

Activation

User navigates to the template/archive task creation form.

Main flow

6. User fills out the template/archive task creation form.
7. User submits the form.
8. System validates the input.
9. System stores the task information in the database.
10. System confirms task creation to the user.

Alternate flow

A1. If input validation fails, the system prompts the user to correct the data.

A2. If a user selects a past date, the system prompts the user to enter present or future date

Exceptional flow

E1. System failure during save operation results in error notification to the user.

2.1.1.12. Requirement 2: Read Template/Archive Tasks

Description & Priority

High. Enables users to view template/archive tasks in a categorized and prioritized list.

Use Case

Scope

Viewing template/archive tasks within the application.

Description

Users can view a list of their template/archive tasks organized by categories

Flow Description

Precondition

Template/archive Tasks exist in the system.

Activation

User opens the template/archive task page.

Main flow

4. User accesses the template/archive task list page.
5. System retrieves tasks from the database.
6. System displays tasks in a 2-column list format.

Alternate flow

A1. No tasks exist, and the system displays a "No Template/archive Tasks" message.

Exceptional flow

E1. Failure to retrieve tasks results in an error message.

2.1.1.13. Requirement 3: Edit Template/Archive Tasks

Description & Priority

Medium. Allows users to modify details of existing template/archive tasks.

Use Case

Scope

Template/archive Task modification within the application.

Description

Users can change template/archive task details like name, category, starting time, estimated time, repeat days in the week and priority,

Flow Description

Precondition

Template/archive Tasks exist in the system.

Activation

User selects a template/archive task to edit.

Main flow

6. User clicks on the edit option of a template/archive task.
7. System displays the template/archive task details in an editable form.
8. User modifies the required fields and submits the form.
9. System validates and updates the task details.
10. System confirms the successful update to the user.

Alternate flow

A1. User aborts the edit operation.

A2. If input validation fails, the system prompts the user to correct the data.

A3. If a user selects a past date, the system prompts the user to enter present or future date

Exceptional flow

E1. System error during update results in an error message.

2.1.1.14. Requirement 4: Delete Template/Archive Tasks

Description & Priority

Medium. Provides functionality for users to delete template/archive tasks from their list.

Use Case

Scope

Template/archive Task deletion within the application.

Description

Users can remove template/archive tasks they no longer need.

Flow Description

Precondition

Template/archive Tasks exist in the system.

Activation

User selects a template/archive task to delete.

Main flow

6. User clicks on the delete option for a template/archive task.
7. System asks for confirmation to delete the template/archive task.
8. User confirms deletion.
9. System removes the template/archive task from the database.
10. System confirms deletion to the user.

Alternate flow

A1. User cancels the deletion process.

Exceptional flow

E1. Failure during deletion leads to an error message.

2.1.1.15. Requirement 14: Filter Template Tasks through Categories

Description & Priority

Medium. Allows users to filter template/archive tasks based on selected categories.

Use Case

Scope

Filtering template/archive tasks in the application.

Description

Users can apply category filters to the template/archive task list to narrow down the visible tasks.

Flow Description

Precondition

Template/archive Tasks exist in the system.

Activation

User selects filter options from the category chips interface.

Main flow

4. User selects one or more categories from the filter options.
5. System filters tasks based on the selected categories.
6. System displays only the template/archive tasks that match the selected categories.

Alternate flow

A1. No tasks match the selected filters, and the system displays a "No template/archive Tasks Found" message.

Exceptional flow

E1. If there is a system error during filtering, an error message is shown.

2.1.1.16. Requirement 15: Manage Template Tasks with Drag and Drop

Description & Priority

Medium. Enables users to create a copy of a template/archive task using a drag-and-drop interface to create a new task form on the right pane.

Use Case

Scope

Creating new template/archive tasks within the application with drag and drop.

Description

Users can create a copy of a template/archive task using a drag-and-drop interface to create a new template/archive task form on the right pane. When the template/archive task gets dragged to the right pane then all the created template/archive tasks fields are populated with the dropped task fields.

Flow Description

Precondition

Template/archive Tasks exist in the system.

Activation

User starts dragging a template/archive task.

Main flow

6. User clicks and holds a template/archive task.
7. User drags the template/archive task to the create new template/archive task form on the right pane.
8. User releases the template/archive task.
9. System updates the template task's position and fills out the template/archive task info to create a new template/archive task form on the right pane.
10. System confirms the update.

Alternate flow

A1. User aborts the drag operation by releasing the template task in the left tasks list pane.

Exceptional flow

E1. If the system fails to update the task position, an error message is displayed.

2.1.2. Data Requirements

Task Data:

Task name, category, priority, starting time, repeat days, and estimated task time. Data integrity and validity must be ensured, particularly in preventing tasks from being scheduled in the past unintentionally.

- **User Data:**

  User credentials (e.g., username, password), preferences, and task customization settings. Data protection measures such as encryption for passwords and sensitive data are essential.

- **Template/Archive Data:**

  Information on templates or archived tasks that can be reused or referred to, which include all task properties.

- **Database Transactions:**

  ACID properties (Atomicity, Consistency, Isolation, Durability) must be maintained to ensure that task data is correctly processed, especially in operations that modify multiple entries like deleting categories.

- **Scalability and Performance:**

  The system should be able to handle a large volume of data efficiently, using technologies like Redis for caching and Bull Queue for managing intensive tasks without overloading the frontend.

2.1.3. User Requirements

- **Ease of Use:**

  An intuitive user interface that allows effortless navigation and interaction, especially for creating, editing, and deleting tasks.

- **Accessibility**

  The application should be accessible to users with disabilities, featuring keyboard navigability and screen reader compatibility.

- **Personalization**

  Users should be able to customize the interface and functionality, such as setting preferred categories or task filters.

- **Security**

  Secure authentication and authorization mechanisms must be in place to protect user data and prevent unauthorized access.

- **Real-time Interaction**

  Users expect real-time updates for task changes, which necessitates robust backend WebSocket implementations for immediate feedback.

2.1.4. Environmental Requirements

- **Cross-Platform Compatibility**

  The application should run seamlessly on various devices and browsers, adapting to different screen sizes and operating systems.

- **Server Requirements**

  The backend should be capable of handling multiple user requests simultaneously without degradation in performance, which might require scalable cloud hosting solutions.

- **Development Environment**

  Consistency across development, testing, and production environments to prevent issues during deployment. Use of containers or virtual environments is recommended to maintain consistency.

- **Compliance and Standards**

Adherence to data protection regulations (like GDPR) and web standards for accessibility and security.

2.1.5. Usability Requirements

- **User Feedback:**

  Incorporating user feedback mechanisms to continuously improve the interface based on actual usage patterns.

- **Error Handling:**

  Clear and helpful error messages should be provided. Users should be guided to resolve issues themselves, if possible.

- **Performance Metrics:**

  Load times and responsiveness must meet industry standards, with optimizations for slower networks.

- **Guidance and Help:**

  Embedded help sections or tooltips that explain how to perform various tasks within the application, improving the learning curve.

2.2. Design & Architecture

The design and architecture of the task management system are planned to meet both functional and non-functional requirements, ensuring robustness, scalability, and a seamless user experience. Here is an elaboration on the system's architecture and design:

1. RESTful API

The communication between the frontend and backend services is managed through RESTful APIs, which support a clear and well-defined set of operations. These APIs are stateless, meaning each call from the frontend to the backend must contain all the information the backend needs to understand the request, and the responses are self-descriptive. This statelessness simplifies the interaction between client and server, making the system more scalable and easier to manage. RESTful APIs also facilitate integration with other services and systems, providing flexibility to extend functionality or swap out components with minimal disruption.

2. Use of Frameworks

For the frontend development, the system utilizes Next.js, a React framework that supports features like server-side rendering and static site generation, making it ideal for building fast, SEO-friendly single-page applications (SPAs). This choice enhances the user experience

by speeding up load times and providing an interactive, dynamic user interface. The backend employs Nest.js, a progressive Node.js framework that uses TypeScript by default and supports a wide range of out-of-the-box application architecture patterns. Nest.js's use of modern JavaScript features and its alignment with Angular's structure make it a robust choice for building scalable server-side applications.

3. Security Measures

Security is a top priority in the system's design. JSON Web Tokens (JWT) are used for secure user authentication. A JWT is a compact, URL-safe means of representing claims to be transferred between two parties, allowing the server to verify tokens and establish the user's identity without needing to repeatedly query the database. All data transmissions are secured using HTTPS, ensuring that all data sent between the client and server is encrypted. Regular security audits are planned to identify and mitigate vulnerabilities, ensuring the application adheres to the latest security standards and practices.

4. Real-time Processing

To enhance the user experience with real-time interaction, the system incorporates WebSocket technology. This allows for two-way interactive communication sessions between the user's browser and the server. As a result, users receive immediate updates on their tasks, such as new messages or status changes, without needing to refresh the browser. Additionally, the system uses Bull Queues for managing background tasks such as sending notifications or processing recurring tasks. Bull Queues handle these operations asynchronously, preventing them from blocking or slowing down the main application processes.

2.3. Implementation

2.3.1. Sign In

2.3.1.1. Next.js - Front end code

page.tsx × LoginForm.tsx

src > app > [locale] > signin > page.tsx > SignInPage

```python
import { useTranslations } from 'next-intl';
import LoginForm from '@/components/organisms/LoginForm';

export default function SignInPage() {
  const t = useTranslations('Login');
  const translations = {
    loginUsernameTitle: t(Username),
    loginPasswordTitle: t(Password),
    forgotPassword: t(Forgot Password),
    login: t(Login),
    invalidCredentials: t(Invalid Credentials),
  };

  return (
    <div className='h-screen flex flex-col m-auto w-full items-center justify-center min-w-[1248px] '>
      <div className='w-[32rem] mt-8 p-1 rounded-md'>
        <span className='flex justify-center text-white text-2xl pt-4'>
          TASK MANAGEMENT APP
        </span>
        <div className='w-full flex justify-center pt-2'></div>
        <LoginForm translations={translations} />
      </div>
    </div>
  );
}
```

page.tsx × LoginForm.tsx ×

~/Documents/task-management/task-management-next-fe/src/app/[locale]/signin/page.tsx

```python
import { useState } from 'react';
import { signIn } from 'next-auth/react';
import CustomButton from '@/components/atoms/Button';
import { AuthProvider, Translations } from '@/constants/constants';
import { useSearchParams } from 'next/navigation';
import { SignIn } from '@/dto/User.dto';
import VectorSvg from '../atoms/svg/Vector';
import { toast } from 'react-toastify';
import Loading from '@/app/[locale]/signin/loading';
import PasswordInputSignIn from '../atoms/PasswordInputSignIn';

interface Props {
  translations: Translations;
}

export default function LoginForm(props: Props) {
  const [credentials, setCredentials] = useState<SignIn>({
    username: '',
    password: '',
  });
  const [loading, setLoading] = useState(false);

  const searchParams = useSearchParams();
  const search = searchParams.get('callbackUrl');

  async function verifyLogin(e: { preventDefault: () => void }) {
    setLoading(true);
    e.preventDefault();
    const result = await signIn(AuthProvider.CREDENTIALS, {
      redirect: false,
      username: credentials.username,
      password: credentials.password,
      callbackUrl: search ?? '/',
    });

    if (result) {
      if (result.error) {
        toast(props.translations.invalidCredentials, { type: 'error' });
        setLoading(false);
      } else if (result.url) {
        window.location.href = result.url;
      }
    }
  }

  const isLoginFormValid = credentials.username && credentials.password;
  return (
    <div>
      <form className='my-3 mx-10 text-white flex flex-col'>
        <div className='mt-2'>
          <label className='text-lg'>
```

Ln 14, Col 18 Spaces: 2 UTF-8 LF {} TypeScript JSX Prettier EN EN

```
page.tsx    LoginForm.tsx ×
src > components > organisms > LoginForm.tsx > Props
18  export default function LoginForm(props: Props) {
56          </div>
57          <input
58            name='username'
59            value={credentials.username}
60            placeholder={props.translations.loginUsernameTitle}
61            onChange={(e) =>
62              setCredentials({
63                ...credentials,
64                username: e.currentTarget.value,
65              })
66            }
67            className='mt-2 bg-background w-full rounded-md h-12 autofill:caret-amber-50'
68          ></input>
69
70          <div className='mt-2'>
71            <label className='text-lg'>
72              {props.translations.loginPasswordTitle}
73            </label>
74          </div>
75
76          <PasswordInputSignIn
77            type={'password'}
78            placeholder={props.translations.loginPasswordTitle}
79            value={credentials.password}
80            name='password'
81            className='mt-2 w-full'
82            onChange={(e) =>
83              setCredentials({
84                ...credentials,
85                password: e.currentTarget.value,
86              })
87            }
88          />
89
90          <div className='mt-8 mb-2 w-full flex justify-center'>
91            <CustomButton
92              color={!isLoginFormValid ? 'default' : 'primary'}
93              size='lg'
94              onClick={verifyLogin}
95              className='text-lg'
96              type='submit'
97              disabled={!isLoginFormValid}
98            >
99              {!loading && props.translations.login}
100
101             {!loading ? <VectorSvg /> : <Loading />}
102           </CustomButton>
103         </div>
104       </form>
105     </div>
106   );
107 }
Ln 14, Col 18  Spaces: 2  UTF-8  LF  {} TypeScript JSX  Prettier  EN  EN
```

Imports and Setup:

- useTranslations from next-intl is imported for internationalization, allowing the component to fetch translation strings based on a given key.
- LoginForm is imported which is a custom component for the login form.

Translation Setup:

- const t = useTranslations('Login'); initializes translation hook to use translation keys under the 'Login' namespace.
- const translations is an object that stores various translations needed for the LoginForm, such as titles and messages.

Render Method:

- The SignInPage function component returns JSX that defines the layout of the login page:
- A full-screen div with centered content that ensures the form is responsive and centered.
- A nested div that contains a title and the LoginForm component. The LoginForm receives the translation object as props.

LoginForm Component

Props and State Initialization:

- The component accepts props with a type of Props which includes translations of type Translations.
- useState is used to manage credentials (username and password) and loading state (indicating if the login is processing).

Handling URL Parameters:

useSearchParams from next/navigation is used to fetch query parameters, specifically to handle redirect scenarios after login.

Login Functionality:

- verifyLogin is an asynchronous function triggered on form submission:
- Prevents default form submission behavior.
- Sets loading to true indicating the start of the login process.
- Uses signIn from next-auth/react to perform authentication with credentials. If a callbackUrl is provided via URL parameters, it will redirect there after a successful login.
- Error handling using toast from react-toastify to show error messages.
- Redirects the user if the signIn process returns a URL.

Form Validation:

isLoginFormValid checks if both username and password fields are not empty, enabling the submit button only if both fields are filled.

Form Rendering:

Renders a form that includes:

- Input fields for username and password. These inputs update the credentials state on change.
- A custom button CustomButton that triggers verifyLogin on click. It is disabled based on isLoginFormValid.
- Conditional rendering inside the button to show a loading indicator or a vector image (VectorSvg) based on the loading state.

Styling and Accessibility:

The form uses Tailwind CSS classes for styling. The inputs and button are styled to be visually consistent and accessible, including responsive design considerations.

Security Features:

The password input uses a custom PasswordInputSignIn component, likely enhancing security features like masking the input and potentially adding additional security measures like strength validation.

2.3.1.2. Nest.js - Backend end code

TS auth.controller.ts × TS auth.service.ts TS users.repository.ts

src > modules > auth > TS auth.controller.ts > AuthController > loginManagerApp

```ts
import { ApiTags } from '@nestjs/swagger';
import { Body, Controller, Post } from '@nestjs/common';
import { AuthService } from './auth.service';
import { Anonymous } from '../../shared/decorators/anonymous.decorator';
import { LoginDto, LoginResponse } from './dto/login.dto';
import { ApiManagerAppLogin } from './auth.swagger';
import { Role } from '../users/entities/role.enum';

@ApiTags('auth')
@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @ApiManagerAppLogin()
  @Anonymous()
  @Post('login-manager-app')
  async loginManagerApp(@Body() loginDto: LoginDto): Promise<LoginResponse> {
    return await this.authService.login(loginDto, Role.Manager);
  }
}
```

TS auth.controller.ts TS auth.service.ts × TS users.repository.ts

src > modules > auth > TS auth.service.ts > AuthService > validateUser

```ts
@Injectable()
export class AuthService {
  constructor(
    private readonly jwtService: JwtService,
    private readonly usersValidationService: UsersValidationService,
    private readonly storesValidationService: StoresValidationService,
    private readonly usersRepository: UsersRepository,
    private readonly storesRepository: StoresRepository,
  ) {}

  async login(loginDto: LoginDto, role: Role): Promise<LoginResponse> {
    const user = await this.validateUser(loginDto, role);
    const token = await this.jwtService.signAsync(user);
    if (role === Role.Manager) {
      const lastLoggedInTime = dayjs().format(DATETIME_FORMAT_WITH_SECONDS);
      return { token, user, lastLoggedInTime } as LoginResponse;
    }

    return { token, user } as LoginResponse;
  }

  async validateUser(loginDto: LoginDto, role: Role): Promise<JwtPayload> {
    let user: User, storeId: string;
    if (role === Role.SuperAdmin) {
      user = await this.usersRepository.findByUsername(loginDto.username);
    } else {
      const stores = await this.storesRepository.findAll();
      for (const store of stores) {
        storeId = store.id   (property) AuthService.usersRepository: UsersRepository
        user = await this.usersRepository.findByUsername(loginDto.username, storeId);
        if (user) {
          this.storesValidationService.validateActiveStore(store);
          break;
        }
      }
    }

    this.usersValidationService.validateUserExists(user);
    this.usersValidationService.validateUserRole(user, role);
    await this.usersValidationService.validatePasswordMatching(loginDto.password, user.password);

    return { userId: user.id, username: user.username, storeId, role: user.role } as JwtPayload;
  }
}
```

```
TS auth.controller.ts   TS auth.service.ts   TS users.repository.ts ×
src > modules > users > TS users.repository.ts > UsersRepository
9   @Injectable()
10  export class UsersRepository {
11    constructor(private readonly repository: Repository, private readonly cls: CellClsService) {}
12
13    get dbKey(): string {
14      return DB_KEYS.USERS.ALL_BY_STORE(this.cls.storeId);
15    }
16
17    async findAll(storeId?: string): Promise<User[]> {
18      const dbKey = DB_KEYS.USERS.ALL_BY_STORE(storeId ?? this.cls.storeId);
19      return await this.repository.findAll<User>(dbKey);
20    }
21
22    async findAllUsernamesFromStores(stores: Store[]): Promise<string[]> {
23      const dbKeys = stores.map((store) => DB_KEYS.USERS.ALL_BY_STORE(store.id));
24      const allUsers = await this.repository.findMany<User>(...dbKeys, DB_KEYS.USERS.SUPER_ADMINS);
25      return allUsers.flatMap((users) => users.map((a) => a.username));
26    }
27
28    async findById(id: string, storeId?: stri  (property) SUPER_ADMINS: string
29      const dbKey = !storeId ? DB_KEYS.USERS.SUPER_ADMINS : DB_KEYS.USERS.ALL_BY_STORE(storeId);
30      return await this.repository.findOne<User>(dbKey, id);
31    }
32
33    async findByUsername(username: string, storeId?: string): Promise<User> {
34      const dbKey = !storeId ? DB_KEYS.USERS.SUPER_ADMINS : DB_KEYS.USERS.ALL_BY_STORE(storeId);
35      const users = await this.repository.findAll<User>(dbKey);
36      return users.find((u) => u.username.toLowerCase() === username.toLowerCase());
37    }
38
39    async findAllByRole(role: string, storeId: string): Promise<User[]> {
40      const dbKey = DB_KEYS.USERS.ALL_BY_STORE(storeId);
41      const users = await this.repository.findAll<User>(dbKey);
42      return users.filter((u) => u.role.toLowerCase() === role.toLowerCase());
43    }
44
45    async findAllDeviceTokensFromStore(storeId: string, role?: string): Promise<DeviceToken[]> {
46      const users = role ? await this.findAllByRole(role, storeId) : await this.findAll(storeId);
47      return users.filter((user) => user.deviceTokens).flatMap((user) => user.deviceTokens);
48    }
49
50    async create(user: Omit<User, 'id'>, storeId: string): Promise<User> {
51      const dbKey = DB_KEYS.USERS.ALL_BY_STORE(storeId);
52      const newUser: User = { id: uuidv4(), ...user };
53      await this.repository.create<User>(dbKey, newUser);
54      return newUser;
55    }
56
57    async update(user: User): Promise<void> {
58      await this.repository.update<User>(this.dbKey, user);
59    }
60  }
61
```

AuthController (NestJS Controller)

This is a typical NestJS controller handling authentication, specifically for managers logging into an application.

Imports and Decorators:

- @ApiTags, @Body, @Controller, and @Post are decorators from NestJS that are used to define the controller's API routes and documentation.
- @ApiManagerAppLogin and @Anonymous might be custom decorators to handle API logging and permit anonymous access respectively.

Controller Setup:

- The controller is tagged with 'auth' for API documentation and routing purposes.
- A route /auth/login-manager-app is established for handling login requests specifically for managers.

Dependency Injection:

AuthService is injected into the controller through the constructor, allowing the controller to delegate authentication logic to this service.

Login Method:

- loginManagerApp is an asynchronous method that handles POST requests to login managers.
- It takes LoginDto (Data Transfer Object) as input, which likely includes username and password.
- The method calls authService.login, passing the LoginDto and a role (Role.Manager).
- It returns a Promise of LoginResponse, which likely includes JWT tokens and other login-related information.

AuthService (NestJS Service)

This service handles the business logic for user authentication.

Imports and Dependency Injection:

Various classes and services are injected, including JwtService for handling JWT tokens, UsersValidationService, and repositories for user and store data management.

Login Function:

- The login function authenticates a user based on their role.
- It calls validateUser to ensure the user exists and their role and password match the credentials provided.
- For managers, it also logs the last login time.
- Returns LoginResponse with the JWT token and user information.

ValidateUser Function:

- Validates the existence of a user and their role based on the provided LoginDto.
- SuperAdmins are fetched directly; other users are validated based on associated stores.
- Passwords are validated, and user role checks are performed to ensure the user can perform the role-specific actions.

UsersRepository (Repository Pattern)

This section outlines the repository used for interacting with user data stored in a database.

Repository and CLS Service:

- Handles database operations abstractly, using a generic Repository service.
- Uses CellClsService to manage context and store identifiers during requests.
- User Data Retrieval and Management:

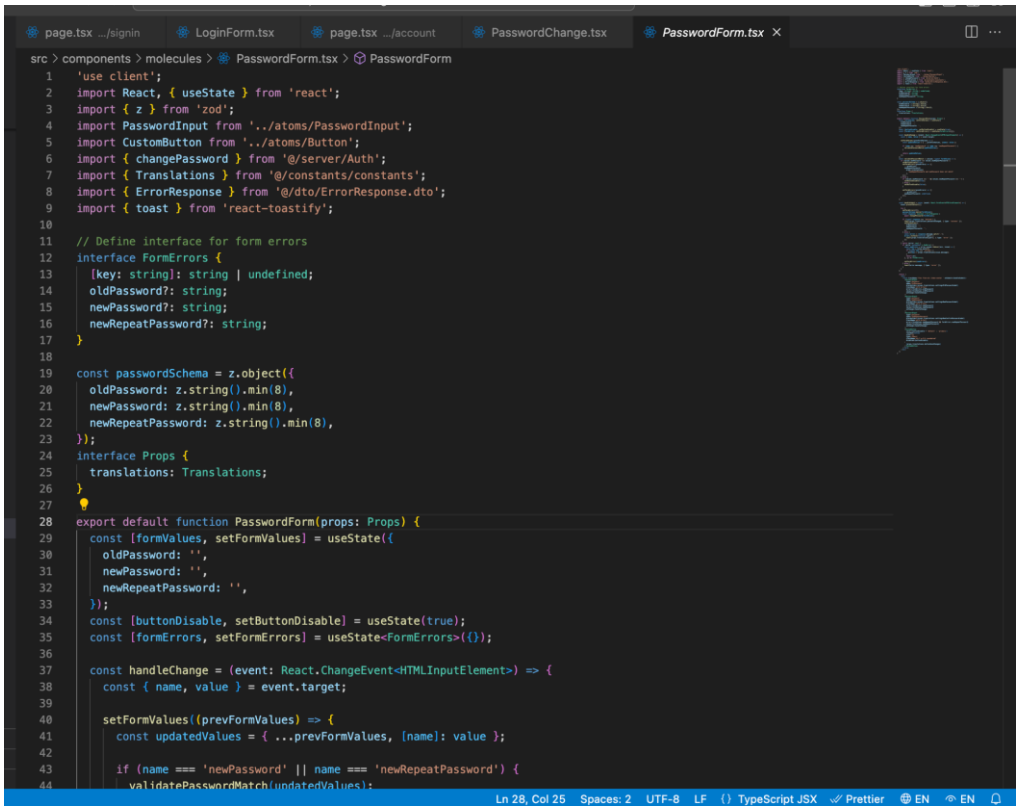
- Functions are provided to retrieve users based on various criteria (e.g., username, role, store).
- Includes complex operations like fetching all usernames from multiple stores and filtering users by role.
- Also includes CRUD operations to create and update user entries in the database.

Database Keys:

- Uses predefined keys for database operations, ensuring consistent and error-free data access.
- DB_KEYS is used to define structured keys based on store IDs and other parameters for organized data retrieval.

2.3.2. Update User Password

2.3.2.1. Next.js - Front end code



```
page.tsx .../signin   LoginForm.tsx   page.tsx .../account   PasswordChange.tsx   PasswordForm.tsx ×
src > components > molecules > PasswordForm.tsx > PasswordForm
28  export default function PasswordForm(props: Props) {
50    const validatePasswordMatch = (values: typeof formValues) => {
62        setButtonDisable(true);
63      } else {
64        setButtonDisable(false);
65      }
66
67      setFormErrors((prevErrors) => ({
68        ...prevErrors,
69        newRepeatPassword: undefined,
70      }));
71    }
72  };
73
74  const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
75    event.preventDefault();
76
77    try {
78      setFormErrors({});
79      passwordSchema.parse(formValues);
80      const response: boolean | ErrorResponse =
81        await changePassword(formValues);
82
83      if (typeof response === 'boolean') {
84        toast(props.translations.passwordChanged, { type: 'success' });
85        setFormValues({
86          oldPassword: '',
87          newPassword: '',
88          newRepeatPassword: '',
89        });
90      } else {
91        const errors = response.message.split('.');
92        errors.forEach(function (part) {
93          toast(props.translations[part], { type: 'error' });
94        });
95      }
96    } catch (error: any) {
97      if (error instanceof z.ZodError) {
98        const newErrors = error.issues.reduce((acc, issue) => {
99          const key = issue.path[0];
100         if (typeof key === 'string') {
101           acc[key] = props.translations[issue.message];
102         }
103         return acc;
104       }, {} as FormErrors);
105
106       setFormErrors(newErrors);
107     } else {
108       toast(error.message, { type: 'error' });
109     }
110   }
111 };
112
```

```
page.tsx .../signin   LoginForm.tsx   page.tsx .../account   PasswordChange.tsx   PasswordForm.tsx ×
src > components > molecules > PasswordForm.tsx > PasswordForm
28  export default function PasswordForm(props: Props) {
74    const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
109       }
110     }
111   };
112
113   return (
114     <div>
115       <form className='flex flex-col items-center ' onSubmit={handleSubmit}>
116         <PasswordInput
117           type='password'
118           name='oldPassword'
119           placeholder={props.translations.settingsOldPasswordLabel}
120           className='mt-8 w-full'
121           error={formErrors.oldPassword}
122           value={formValues.oldPassword}
123           onChange={handleChange}
124         />
125         <PasswordInput
126           type='password'
127           name='newPassword'
128           placeholder={props.translations.settingsNewPasswordLabel}
129           className='mt-8 w-full'
130           error={formErrors.newPassword}
131           value={formValues.newPassword}
132           onChange={handleChange}
133         />
134         <PasswordInput
135           type='password'
136           name='newRepeatPassword'
137           placeholder={props.translations.settingsNewConfirmPasswordLabel}
138           className='mt-8 w-full'
139           error={formValues.newRepeatPassword && formErrors.newRepeatPassword}
140           value={formValues.newRepeatPassword}
141           onChange={handleChange}
142         />
143         <CustomButton
144           color={buttonDisable ? 'default' : 'primary'}
145           rounded={false}
146           size='lg'
147           type='submit'
148           className='mt-7 w-2/4 rounded-md'
149           disabled={buttonDisable}
150         >
151           {props.translations.buttonSaveChanges}
152         </CustomButton>
153       </form>
154     </div>
155   );
156 }
157
```

Imports and Setup

- React and useState Hook: Used for managing component state.
- zod: A validation library to ensure data integrity by defining a schema for the password fields.
- PasswordInput and CustomButton: Reusable React components for input fields and buttons, respectively.
- changePassword: A function likely making an API call to update the user's password on the server.
- Translations and ErrorResponse Types: Used for internationalization and handling API response errors.

Component Structure and Logic

State Definitions:

- formValues holds the current values of the form inputs: old password, new password, and password confirmation.
- buttonDisable controls the disabled state of the submit button based on form validation.
- formErrors stores potential error messages for each input, which helps in displaying validation feedback to the user.

Password Validation Schema:

passwordSchema is defined using zod and sets a minimum length of 8 characters for each password field.

Event Handlers:

- handleChange updates the formValues state whenever an input field changes. It also checks if the new passwords match each time either the new password or password confirmation changes.
- validatePasswordMatch checks if the new password and its confirmation match. If not, it disables the submit button and sets an appropriate error message. If they do match and are not empty, it enables the submit button and clears the error.

Form Submission:

- handleSubmit is triggered when the form is submitted.
- Prevents the default form submission action.
- Clears previous errors and validates the form data against the defined schema.
- If validation passes, it calls the changePassword function with the form values.
- Handles the response: if successful (returns a boolean), it shows a success toast and resets the form values. If an error occurs (returns ErrorResponse), it parses the error message and displays it via toast notifications.
- Catches and handles any errors from the zod validation or other exceptions, setting form errors appropriately or showing a toast for unexpected errors.

2.3.2.2. Nest.js - Back end code

src > modules > users > TS users.controller.ts > ...

```
import { Body, Controller, Get, Patch, Query } from '@nestjs/common';
import { ApiBearerAuth, ApiTags } from '@nestjs/swagger';
import { ApiChangePassword, ApiIsPasswordChanged } from './users.swagger';
import { HasRoles } from '../../shared/decorators/roles/has-roles.decorator';
import { Role } from './entities/role.enum';
import { UsersService } from './services/users.service';
import { ChangePasswordDto } from './dto/change-password.dto';
import { DateQuery } from './dto/query-users.dto';

@ApiBearerAuth()
@ApiTags('users')
@Controller('users')
export class UsersController {
  constructor(private readonly usersService: UsersService) {}

  @ApiChangePassword()
  @Patch('change-password')
  @HasRoles(Role.Manager)
  async changePassword(@Body() changePasswordDto: ChangePasswordDto): Promise<boolean> {
    return await this.usersService.changePassword(changePasswordDto);
  }

```

rc > modules > users > services > TS users.service.ts > UsersService > changePassword

```
import { hash } from 'bcrypt';
import { EventEmitter2 } from '@nestjs/event-emitter';
import { EVENTS } from '../../../shared/constants/evens.constants';
import { UserPasswordChangedEvent } from '../dto/users.event';
import dayjs from 'dayjs';
import { DATETIME_FORMAT } from '../../../shared/constants/date.constants';
import { dateNowFormatted } from '../../../shared/utils/date.utils';
import { CellClsService } from '../../libs/cls/cell-cls.service';
import { UsersRepository } from '../users.repository';

@Injectable()
export class UsersService {
  constructor(
    private readonly usersValidationService: UsersValidationService,
    private readonly usersRepository: UsersRepository,
    private readonly eventEmitter: EventEmitter2,
    private readonly cls: CellClsService,
  ) {}

  async changePassword(changePasswordDto: ChangePasswordDto) {
    const user = await this.usersRepository.findById(this.cls.userId, this.cls.storeId);

    this.usersValidationService.validateUserExists(user);
    await this.usersValidationService.validatePasswordMatching(changePasswordDto.oldPassword, user.password);

    user.password = await hash(changePasswordDto.newPassword, 10);
    const passwordChangedAt = dateNowFormatted(DATETIME_FORMAT);
    user.lastPasswordChangedDate = passwordChangedAt;
    await this.usersRepository.update(user);

    const data: UserPasswordChangedEvent = {
      userId: this.cls.userId,
      storeId: this.cls.storeId,
      passwordChangedAt,
    };
    this.eventEmitter.emit(EVENTS.USERS.PASSWORD_CHANGED, data);

    return true;
  }

```

Imports and Setup:

- @Body, @Controller, @Patch, and @Query are NestJS decorators to define the controller and its methods.
- @ApiBearerAuth, @ApiTags from @nestjs/swagger for API documentation purposes.
- HasRoles is likely a custom decorator used to enforce role-based access control.

Controller Decorators:

- @ApiBearerAuth() indicates that the methods within the controller require HTTP Bearer Authentication.
- @ApiTags('users') categorizes the controller's endpoints under the 'users' tag in the Swagger documentation.
- @Controller('users') defines the base route for all endpoints defined within this controller.

Dependency Injection:

- UsersService is injected into the controller via the constructor. This service is responsible for the business logic associated with user operations.
- Endpoint - changePassword:
- @Patch('change-password') specifies a PATCH method for the endpoint, which is typically used for updating resources.
- @HasRoles(Role.Manager) restricts access to this endpoint to users who have a 'Manager' role.
- changePassword(@Body() changePasswordDto: ChangePasswordDto) takes the new password data from the request body.
- Returns a boolean promise that resolves to true if the password change was successful.

UsersService (NestJS Service)

Service Setup:

The service class is annotated with @Injectable(), making it a candidate for dependency injection.

Dependency Injection in Service:

Dependencies such as UsersValidationService, UsersRepository, EventEmitter2, and CellClsService are injected. These handle various aspects of user validation, database interaction, event management, and context-specific data respectively.

changePassword Method:

- User Retrieval: Retrieves the user based on userId and storeId from the CellClsService context, which likely manages data specific to the current request or session.
- Validation: Validates that the user exists and that the provided old password matches the stored password.
- Password Update: Hashes the new password using bcrypt and updates the user's password.

- Date Handling: Sets the lastPasswordChangedDate to the current date formatted according to a predefined format, ensuring the date is recorded for auditing or security purposes.
- Persistence: Saves the updated user information back to the database.
- Event Emission: Emits a password changed event using EventEmitter2. This could be used for logging, notifications, or other side effects triggered by a password change.

Event-Driven Behaviour:

After changing the password, the service emits an event (EVENTS.USERS.PASSWORD_CHANGED). This is a key feature of an event-driven architecture, where components react to events rather than directly invoking further actions. This allows for better separation of concerns and easier integration of side effects like notifications or logging.

2.3.3. Categories create, delete and view

2.3.3.1. Next.js - Front end code

```
interface SectionsInterface {
  sections: Array<Section>;
  translations: Translations;
}

export default function Sections(props: SectionsInterface) {
  const [isRename, setIsRename] = useState(false);
  const [selectedRenameSection, setSelectedRenameSection] = useState<Section>({
    iconName: '',
    id: '',
    name: '',
    plannedDays: [],
  });
  const [inputValue, setInputValue] = useState('');
  const [renameError, setRenameError] = useState({
    error: false,
    message: '',
  });

  const [allSections, setAllSections] = useState(props.sections);
  const [selectedSections, setSelectedSections] = useState<string[]>([]);
  const [showModal, setShowModal] = useState(false);
  const [showConfirmationModal, setShowConfirmationModal] = useState(false);
  const [reload, setReload] = useState(false);

  const context = useContext(sectionsData);
  const { sections, setSections } = context;

  const filterSections = useCallback(() => {
    const filteredSections = allSections.filter(
      (section) => !sections.includes(section.id)
    );
    setAllSections(filteredSections);
  }, [reload]);

  useEffect(() => {
    filterSections();
  }, [filterSections]);

  useEffect(() => {
    setAllSections(props.sections);
  }, [props.sections]);

  const toggleSectionSelection = (sectionId: string) => {
    if (selectedSections.includes(sectionId)) {
      setSelectedSections(selectedSections.filter((id) => id !== sectionId));
    } else {
      setSelectedSections([...selectedSections, sectionId]);
    }
  };
```

```
src > components > molecules > Sections.tsx > Sections > filterSections > useCallback() callback > filteredSections > allSections.filter() callback
23  export default function Sections(props: SectionsInterface) {
67    };
68
69    const handleConfirmationModal = () => {
70      if (selectedSections.length === 0) {
71        toast(props.translations.noSectionSelected, { type: 'warning' });
72      } else {
73        setShowConfirmationModal(true);
74      }
75    };
76
77    const deleteSections = async () => {
78      try {
79        setSections(selectedSections);
80        const response: boolean | ErrorResponse =
81          await deleteSection(selectedSections);
82        if (typeof response === 'boolean') {
83          toast(props.translations.deleteInProcess, { type: 'warning' });
84          setReload(!reload);
85        } else {
86          toast(props.translations.errorDeletingSection, { type: 'error' });
87        }
88      } catch (error: any) {
89        toast(error.message, { type: 'error' });
90      }
91      setSelectedSections([]);
92      setShowConfirmationModal(false);
93    };
94
95    function handleModalOpen() {
96      setShowModal(true);
97    }
98    const handleRename = (event: React.ChangeEvent<HTMLInputElement>) => {
99      setInputValue(event.target.value);
100   };
101   const renameSection = async (name: string, section: Section) => {
102     const data = {
103       id: section.id,
104       name: name,
105       plannedDays: section.plannedDays,
106       iconName: section.iconName,
107     };
108     const response: boolean | ErrorResponse = await editSection(data);
109     if (typeof response === 'boolean') {
110       toast(props.translations.sectionRenamedSuccessfully, { type: 'success' });
111     } else {
112       const erros = response.message.split('.');
113       erros.forEach(function (part) {
114         toast(props.translations[part], { type: 'error' });
115       });
116     }
117   };
```

```
src > components > molecules > Sections.tsx > Sections
23  export default function Sections(props: SectionsInterface) {
117   };
118   return (
119     <>
120       <ConfirmationModal
121         type='danger'
122         showModal={showConfirmationModal}
123         title={props.translations.areYouSure}
124         description={props.translations.modalTitle}
125         handleCancellation={() => {
126           setShowConfirmationModal(false);
127         }}
128         handleConfirmation={() => {
129           deleteSections();
130         }}
131         cancelText={props.translations.modalCancelTitle}
132         confirmText={props.translations.modalDeleteTitle}
133       />
134       <div className='h-3/4 w-2/5 text-white overflow-y-auto scrollbar-none'>
135         <div className='flex flex-col'>
136           <div>
137             {allSections.length > 0 ? (
138               allSections.map((section, index) => (
139                 <>
140                   {isRename && selectedRenameSection.id === section.id && (
141                     <>
142                       <div className='flex w-full relative'>
143                         <input
144                           autoComplete='off'
145                           type='text'
146                           value={inputValue}
147                           onChange={handleRename}
148                           className=' p-3 mt-4 border border-gray-500 rounded-md bg-transparent w-full'
149                           maxLength={25}
150                         />
151                         <span
152                           className='mt-1 absolute right-9 top-6 cursor-pointer'
153                           onClick={() => {
154                             const sectionNames = allSections.map((s) =>
155                               s.name.toUpperCase()
156                             );
157                             if (
158                               sectionNames.includes(inputValue.toUpperCase())
159                             ) {
160                               setRenameError({
161                                 error: true,
162                                 message:
163                                   props.translations.sectionAlreadyExists,
164                               });
165                             } else if (inputValue === '') {
166                               setRenameError({
167                                 error: true,
```

```
page.tsx .../signin   LoginForm.tsx   page.tsx .../account   PasswordChange.tsx   page.tsx .../department   Sections.tsx 1 ×
src > components > molecules > Sections.tsx > Sections
23   export default function Sections(props: SectionsInterface) {
138                 allSections.map((section, index) => (
153                           onClick={() => {
162                             message:
163                               props.translations.sectionAlreadyExists,
164                           });
165                           } else if (inputValue === '') {
166                             setRenameError({
167                               error: true,
168                               message:
169                                 props.translations.sectionCannotBeEmpty,
170                             });
171                           } else {
172                             setIsRename(false);
173                             renameSection(inputValue, selectedRenameSection);
174                             setSelectedRenameSection({} as Section);
175                             setRenameError({
176                               error: false,
177                               message: '',
178                             });
179                           }
180                         }}
181                         >
182                           <TickSVG />
183                         </span>
184                         <span
185                           className='mt-1 absolute right-4 top-7 cursor-pointer'
186                           onClick={() => {
187                             setIsRename(false);
188                             setSelectedRenameSection({} as Section);
189                             setRenameError({
190                               error: false,
191                               message: '',
192                             });
193                           }}
194                           >
195                             <CrossSvg size={15} />
196                           </span>
197                         </div>
198                         {renameError.error && (
199                           <p className='text-red-500 text-sm mt-1'>
200                             {renameError.message}
201                           </p>
202                         )}
203                       </>
204                     )}
205                     <div
206                       key={index}
207                       className={`p-3 mt-4 border border-gray-500 rounded-md cursor-pointer bg-${
208                         selectedSections.includes(section.id)
209                           ? ' bg-primary'
210                           : 'dark-80'
```

```
page.tsx .../signin   LoginForm.tsx   page.tsx .../account   PasswordChange.tsx   page.tsx .../department   Sections.tsx 1 ×
src > components > molecules > Sections.tsx > Sections
23   export default function Sections(props: SectionsInterface) {
138                 allSections.map((section, index) => (
207                       className={`p-3 mt-4 border border-gray-500 rounded-md cursor-pointer bg-${
208                         selectedSections.includes(section.id)
209                           ? ' bg-primary'
210                           : 'dark-80'
211                       } ${
212                         isRename && selectedRenameSection.id === section.id
213                           ? 'hidden'
214                           : 'block'
215                       } `}
216                       >
217                         <div className='relative'>
218                           <div
219                             className='flex justify-between'
220                             onClick={() => toggleSectionSelection(section.id)}
221                             >
222                             <div className='flex gap-2'>
223                               {selectedRenameSection.id !== section.id && (
224                                 <>
225                                   <SVGComponent iconName={section.iconName} />
226                                   <span className='font-medium'>
227                                     {section.name.length > 17
228                                       ? `${section.name.substring(0, 17)}...`
229                                       : section.name}
230                                   </span>
231                                 </>
232                               )}
233                             </div>
234                           </div>
235                           <div className='flex absolute top-0 right-0 cursor-pointer'>
236                             {selectedSections.includes(section.id) ? (
237                               <TickSVG />
238                             ) : (
239                               <span
240                                 onClick={() => {
241                                   setIsRename(true);
242                                   setSelectedRenameSection(section);
243                                   setInputValue(section.name);
244                                 }}
245                                 >
246                                   <RenameIcon />
247                                 </span>
248                               )}
249                             </div>
250                           </div>
251                         </div>
252                       </>
253                     ))
254                   ) : (
255                     <span className='flex flex-col justify-center items-center text-lg font-semibold'>
256                       {props.translations.emptySectionTitle}
```

```
                    </div>
                  </div>
                </>
              ))
            ) : (
              <span className='flex flex-col justify-center items-center text-lg font-semibold'>
                {props.translations.emptySectionTitle}
              </span>
            )}
          </div>
        </div>
      </div>

      <div>
        <Modal className='h-3/4 w-3/12 min-w-[26.75rem]' showModal={showModal}>
          <AddSection
            setShowModal={setShowModal}
            sections={props.sections}
            translations={props.translations}
          />
        </Modal>
      </div>

      <div className='flex w-2/5'>
        <CustomButton
          color='primary'
          size='md'
          className=' mt-4 w-9/12'
          onClick={handleModalOpen}
        >
          {props.translations.buttonAdd}
        </CustomButton>
        <CustomButton
          color='danger'
          size='md'
          className='ml-4 mt-4 w-9/12'
          onClick={handleConfirmationModal}
        >
          {props.translations.buttonDelete}
        </CustomButton>
      </div>
    </>
  );
}
```

Component Structure

- SectionsInterface: A TypeScript interface that specifies the props expected by the Sections component, including a list of sections and translations for localization.
- State Management:
- isRename, selectedRenameSection, inputValue, renameError: States used to manage the renaming of sections.
- allSections, selectedSections: States to hold sections data and track selected sections.
- showModal, showConfirmationModal, reload: Boolean states to control modal visibility and trigger re-render or refreshes.

Context Usage:

- The component uses the useContext hook to access sectionsData context, which likely shares sections data and methods across components.
- Functional Logic
- Filtering Sections: A filterSections function (memoized with useCallback) filters out sections already included in the context, ensuring the UI reflects the current state.

Event Handlers:

- toggleSectionSelection: Toggles selection of sections, used for actions like delete or edit.
- handleConfirmationModal: Opens a confirmation modal if sections are selected; otherwise, displays a warning toast.
- deleteSections: Deletes selected sections after confirmation, using an API call (deleteSection), and updates the UI accordingly.

- handleModalOpen: Opens a modal to possibly add a new section.
- handleRename: Handles changes to the input field for renaming a section.
- renameSection: Submits the new name for the section, updates the backend via editSection, and handles the UI response based on success or error.

useEffect for Filtering and Setting Sections:

- One useEffect is used to filter sections whenever there's a reload.
- Another useEffect ensures that allSections is synchronized with props.sections.
- UI Components and Interaction

Modals for Adding and Confirming Actions:

Modal and ConfirmationModal components are used for adding new sections and confirming deletions, respectively.

Dynamic List Rendering:

Sections are listed dynamically with options to select, rename, or delete. Conditionally rendered elements include input fields for renaming and SVG icons for actions.

Custom Buttons:

Buttons are used to trigger actions like opening modals and confirming deletions or additions.

Detailed Rendering Logic

Sections are rendered in a list with conditional elements for renaming:

- If isRename is true and the section matches selectedRenameSection, an input field and control icons (TickSVG and CrossSvg) are shown.
- Error messages are displayed if there’s an issue with the renaming process.
- Each section can be toggled for selection, and upon selection, it can be either renamed or marked with a TickSVG to denote selection.
- Actions like adding new sections or deleting selected ones are managed through buttons which trigger modals for further interactions.

2.3.3.2. Nest.js - Back-end code

```
auth.controller.ts   auth.service.ts   users.repository.ts   users.controller.ts 4   users.service.ts 1   sections.controller.ts ×   sections.queue.consumer.ts
rc > modules > sections > sections.controller.ts > SectionsController > updatePlanningStatus
import { Section } from './entity/section.entity';
import { UpdateSectionPlanningDto } from './dto/update-section-planning.dto';
import { HasRoles } from '../../shared/decorators/roles/has-roles.decorator';
import { Role } from '../users/entities/role.enum';
import { CreateSectionDto } from './dto/create-section.dto';
import { EditSectionDto } from './dto/edit-section.dto';

@ApiBearerAuth()
@ApiTags('sections')
@Controller('sections')
export class SectionsController {
  constructor(private readonly sectionsService: SectionsService) {}

  @ApiGetSections()
  @Get()
  @HasRoles(Role.Manager, Role.HQ)
  async findAll(): Promise<Section[]> {
    return await this.sectionsService.findAllByStoreId();
  }

  @ApiCreateSection()
  @Post()
  @HasRoles(Role.Manager)
  async create(@Body() createSectionDto: CreateSectionDto): Promise<boolean> {
    return await this.sectionsService.create(createSectionDto);
  }

  @ApiUpdatePlanningStatus()
  @Put('planning-status')
  @HasRoles(Role.Manager)
  async updatePlanningStatus(@Body() updateSectionPlanningDto: UpdateSectionPlanningDto): Promise<boolean> {
    return await this.sectionsService.updatePlanningStatus(updateSectionPlanningDto);
  }

  @ApiEditSection()
  @Put()
  @HasRoles(Role.Manager)
  async editSection(@Body() editSectionDto: EditSectionDto): Promise<boolean> {
    return await this.sectionsService.editSection(editSectionDto);
  }

  @ApiDeleteSection()
  @Delete()
  @HasRoles(Role.Manager)
  async delete(@Body() sectionIds: string[]) {
    return this.sectionsService.delete(sectionIds);
  }
}
```

```
auth.controller.ts   auth.service.ts   users.repository.ts   users.controller.ts 4   users.service.ts 1   sections.controller.ts   sections.service.ts ×
src > modules > sections > services > sections.service.ts > SectionsService > updatePlanningStatus
import { Injectable, NotFoundException } from '@nestjs/common';
import { Section } from '../entity/section.entity';
import { UpdateSectionPlanningDto } from '../dto/update-section-planning.dto';
import { SectionsValidationService } from './sections-validation.service';
import { EventEmitter2 } from '@nestjs/event-emitter';
import { EVENTS } from '../../../shared/constants/evens.constants';
import { SectionsEvent, SectionsPlanningEvent } from '../dto/sections.event';
import { EventSyncStatus } from '../../tasks/dto/tasks.event';
import { CreateSectionDto } from '../dto/create-section.dto';
import { CellClsService } from '../../libs/cls/cell-cls.service';
import { SectionsRepository } from '../sections.repository';
import { TasksService } from '../../tasks/services/tasks.service';
import { InjectQueue } from '@nestjs/bull';
import { Queue } from 'bull';
import { EditSectionDto } from '../dto/edit-section.dto';

@Injectable()
export class SectionsService {
  constructor(
    private readonly repository: SectionsRepository,
    private readonly tasksService: TasksService,
    private readonly sectionsValidationService: SectionsValidationService,
    private readonly eventEmitter: EventEmitter2,
    private readonly cls: CellClsService,
    @InjectQueue('SECTIONS_QUEUE') private queue: Queue,
  ) {}

  async findAllByStoreId(): Promise<Section[]> {
    return await this.repository.findAll();
  }

  async editSection(editSectionDto: EditSectionDto): Promise<boolean> {
    const updated = await this.repository.updateSectionName(editSectionDto.sectionId, editSectionDto.name);
    if (!updated) {
      throw new NotFoundException(`Section ID not found.`);
    }
    return updated;
  }

  async create(createSectionDto: CreateSectionDto): Promise<boolean> {
    const sections = await this.repository.findAll();
    this.sectionsValidationService.validateUniqueSectionName(createSectionDto.name, sections);
    this.sectionsValidationService.validateUniqueSectionIconName(createSectionDto.iconName, sections);

    const newSection = await this.repository.create(createSectionDto);
    const sectionEvent = {
      storeId: this.cls.storeId,
      syncStatus: EventSyncStatus.CREATED,
      sections: [newSection],
    } as SectionsEvent;
    this.eventEmitter.emit(EVENTS.SECTIONS.CHANGES, sectionEvent);

    return true;
Ln 62, Col 41   Spaces: 2   UTF-8   LF   {} TypeScript   Prett
```

TS auth.controller.ts | TS auth.service.ts | TS users.repository.ts | TS users.controller.ts 4 | TS users.service.ts 1 | TS sections.controller.ts | TS

src > modules > sections > services > TS sections.service.ts > SectionsService > updatePlanningStatus

```
18  export class SectionsService {
40    async create(createSectionDto: CreateSectionDto): Promise<boolean> {
53      return true;
54    }
55
56    async updatePlanningStatus({ date, isDone, sectionId }: UpdateSectionPlanningDto): Promise<boolean> {
57      const section = await this.repository.findById(sectionId);
58      this.sectionsValidationService.validateSectionExists(section);
59
60      const plannedDayExist = section.plannedDays.some((plannedDay) => plannedDay === date);
61      this.sectionsValidationService.validateIsSectionAlreadyPlanned(isDone, plannedDayExist);
62      this.sectionsValidationService.validateIsSectionAlreadyNotPlanned(isDone, plannedDayExist);
63
64      const sectionPlanningEvent = { storeId: this.cls.storeId, sectionId, date } as SectionsPlanningEvent;
65      if (isDone && !plannedDayExist) {
66        section.plannedDays.push(date);
67        await this.repository.update(section);
68        this.eventEmitter.emit(EVENTS.SECTIONS.PLANNING, {
69          ...sectionPlanningEvent,
70          syncStatus: EventSyncStatus.CREATED,
71        } as SectionsPlanningEvent);
72        return true;
73      }
74      if (!isDone && plannedDayExist) {
75        const plannedDayIndex = section.plannedDays.findIndex((plannedDay) => plannedDay === date);
76        section.plannedDays.splice(plannedDayIndex);
77        await this.repository.update(section);
78
79        this.eventEmitter.emit(EVENTS.SECTIONS.PLANNING, {
80          ...sectionPlanningEvent,
81          syncStatus: EventSyncStatus.DELETED,
82        } as SectionsPlanningEvent);
83
84        return true;
85      }
86    }
87
88    async delete(sectionIds: string[]) {
89      const storeId = this.cls.storeId;
90      return this.queue.add('DELETE_SECTION_TASKS', {
91        sectionIds,
92        storeId,
93      });
94    }
95
96    async closeQueue() {
97      this.queue.close();
98    }
99  }
100
```

src > modules > sections > TS sections.queue.consumer.ts > SectionQueueConsumer > deleteSectionTasks

```
1   import { OnQueueCompleted, OnQueueFailed, Process, Processor } from '@nestjs/bull';
2   import { Job } from 'bull';
3   import { TasksService } from '../tasks/services/tasks.service';
4   import { SectionsRepository } from './sections.repository';
5   import { EventEmitter2 } from '@nestjs/event-emitter';
6   import { EventSyncStatus } from '../tasks/dto/tasks.event';
7   import { EVENTS } from '../../shared/constants/evens.constants';
8   import { SectionsEvent } from './dto/sections.event';
9
10  @Processor('SECTIONS_QUEUE')
11  export class SectionQueueConsumer {
12    constructor(
13      private readonly tasksService: TasksService,
14      private readonly sectionRepository: SectionsRepository,
15      private readonly eventEmitter: EventEmitter2,
16    ) {}
17
18    @Process('DELETE_SECTION_TASKS')
19    async deleteSectionTasks(job: Job) {
20      console.log('DELETE_SECTION_TASKS Job started with job id: ', job.id);
21
22      try {
23        await this.tasksService.deleteAllSectionTasks(job.data.sectionIds, job.data.storeId);
24        for (const sectionId of job.data.sectionIds) {
25          await this.sectionRepository.delete(sectionId, job.data.storeId);
26        }
27        return;
28      } catch (error) {
29        throw error;
30      }
31    }
32
33    @OnQueueCompleted()
34    async OnCompleted(job: Job) {
35      const sectionEvent = {
36        storeId: job.data.storeId,
37        syncStatus: EventSyncStatus.DELETED,
38        sections: job.data.sectionIds,
39      } as SectionsEvent;
40
41      this.eventEmitter.emit(EVENTS.SECTIONS.DELETED, sectionEvent);
42      console.log('Job Completed with deleting the following section IDs:', job.data.sectionIds);
43    }
44
45    @OnQueueFailed()
46    handler(job: Job, error: Error) {
47      const sectionEvent = {
48        storeId: job.data.storeId,
49        syncStatus: EventSyncStatus.DELETE_FAILED,
50        sections: job.data.sectionIds,
51      } as SectionsEvent;
52
53      this.eventEmitter.emit(EVENTS.SECTIONS.DELETE_FAILED, sectionEvent);
54
55      console.log(
56        'Job Failed with deleting the following section IDs:',
57        job.data.sectionIds,
58        'having the errors: ',
59        error,
60      );
61    }
62  }
63
```

Categories Controller (NestJS Controller)

Annotations and Role Management:

- @ApiBearerAuth and @ApiTags are used for Swagger documentation to authenticate and categorize the endpoints.
- @Controller('sections') designates the base route for all endpoints within this controller.

Endpoint Definitions:

- Utilizes RESTful design principles (GET, POST, PUT, DELETE) to manage sections.
- Endpoints are protected with role-based access controls using the @HasRoles decorator, restricting certain operations to specific user roles like Role.Manager.
- Each method interacts with the SectionsService to perform operations like creating, editing, deleting, and updating the planning status of sections.

Categories Service (NestJS Service)

Business Logic:

- Handles all the logic required to manage sections, including CRUD operations and unique validations.
- Uses SectionsRepository for database interactions and TasksService for related tasks management.
- Utilizes EventEmitter2 for emitting events related to sections operations, which helps in maintaining the system reactive and extensible.

Complex Operations:

create, editSection, updatePlanningStatus, and delete are methods that perform respective operations and emit relevant events for actions like creation, update, or deletion.

Categories Repository (NestJS Repository)

Database Interaction:

- Provides methods for interacting with the database, such as findAll, findById, create, update, and delete.
- Uses a generic Repository service, which abstracts the database interaction, making the repository easier to manage and test.

Responsibilities:

- Manages CRUD operations on the sections stored in the database.
- Ensures the unique constraints and other validations at the data level.
- SectionQueueConsumer (NestJS Processor for Bull Queue)

Job Processing:

- Defined as a processor for a job queue named SECTIONS_QUEUE.
- @Process('DELETE_SECTION_TASKS') handles the deletion of sections and associated tasks asynchronously, ensuring that heavy tasks do not block the main application thread.

Event Handling:

- @OnQueueCompleted and @OnQueueFailed are event handlers that execute upon completion or failure of a job, respectively.
- These handlers emit events and log results or errors, which is crucial for monitoring and debugging.

2.3.4. Tasks create, delete, edit, view, prioritize, drag and drop

2.3.4.1. Next.js - Front end code

```
  loadingData: boolean;
}
export default function TaskClientComponent({
  sections,
  tasks,
  templateTasks,
  translations,
  queryFilter,
  date,
  months,
  loadingData,
}: Props) {
  const [isDraggedTaskArchive, setisDraggedTaskArchive] = useState(false);
  const [isDraggedTask, setIsDraggedTask] = useState(false);
  const [newTask, setNewTask] = useState<Task>({} as Task);
  const [deletedTask, setDeleteTask] = useState<Task>({} as Task);
  const [isArchive, setIsArchive] = useState<boolean>(false);
  const [data, setData] = useState(tasks);
  const [templateData, setTemplateData] = useState(templateTasks);
  const [sectionFilter, setSectionFilter] = useState<string[]>([queryFilter]);
  const [activeState, setActiveState] = useState<boolean>(true);
  const [deleteModal, setDeleteModal] = useState<boolean>(false);
  const [loading, setLoading] = useState<boolean>(false);

  const { setContextDate } = useContext(DateContext);
  const [isSortedYet, setIsSortedYet] = useState(false);

  const [draggableContextTask, setDraggableContextTask] = useState<Task>(
    {} as Task
  );
  const [taskExpanded, setTaskExpanded] = useState<boolean>(false);

  useEffect(() => {
    setContextDate(date);
  }, [date]);

  useEffect(() => {
    setData(tasks);
    setTemplateData(templateTasks);
    if (queryFilter !== 'allTasks') {
      setSectionFilter([queryFilter]);
    } else {
      setSectionFilter([]);
    }
    setLoading(false);
  }, [tasks, queryFilter, templateTasks]);

  useLayoutEffect(() => {
    setIsSortedYet(false);
    if (activeState) {
      setData(
        tasks
```

```
src > components > organisms > TaskClientComponent.tsx > TaskClientComponent
36  export default function TaskClientComponent({
70    useEffect(() => {
79    }, [tasks, queryFilter, templateTasks]);
80
81    useLayoutEffect(() => {
82      setIsSortedYet(false);
83      if (activeState) {
84        setData(
85          tasks
86            .filter((task) => {
87              if (task.status !== TaskStatus.DONE) {
88                if (sectionFilter.length === 0) {
89                  return true;
90                } else {
91                  return sectionFilter.includes(task.sectionId);
92                }
93              }
94            })
95            .filter((task) => {
96              if (!isArchive) {
97                return getSectionName(task.sectionId, sections) !== undefined;
98              }
99            })
100       );
101     } else {
102       setData(
103         tasks
104           .filter((task) => {
105             if (task.status === TaskStatus.DONE) {
106               if (sectionFilter.length === 0) {
107                 return true;
108               } else {
109                 return sectionFilter.includes(task.sectionId);
110               }
111             }
112           })
113           .filter((task) => {
114             if (!isArchive) {
115               return getSectionName(task.sectionId, sections) !== undefined;
116             }
117           })
118       );
119     }
120     setTemplateData(
121       templateTasks.filter((task) => {
122         if (sectionFilter.length === 0) {
123           return true;
124         } else {
125           return task.sectionIds?.some((sectionId) =>
126             sectionFilter.includes(sectionId)
127           );
128         }
129       })
```

```
src > components > organisms > TaskClientComponent.tsx > TaskClientComponent
36  export default function TaskClientComponent({
81    useLayoutEffect(() => {
130     );
131
132     setIsSortedYet(true);
133   }, [sectionFilter, activeState]);
134
135   const onDragEnd = ({ source, destination }: DropResult) => {
136     if (destination === undefined || destination === null) return null;
137     if (destination.droppableId === 'taskCreator') {
138       const draggedTaskIndex = source.index;
139       let draggedTaskData = {} as Task;
140       setIsDraggedTask(true);
141       if (!isArchive) {
142         draggedTaskData = data[draggedTaskIndex];
143       } else {
144         draggedTaskData = templateTasks[draggedTaskIndex];
145       }
146       clipboardCopy(JSON.stringify(draggedTaskData)).then(() => {
147         setNewTask && setNewTask(draggedTaskData);
148
149         if (isArchive) {
150           setisDraggedTaskArchive(true);
151         } else {
152           setisDraggedTaskArchive(false);
153         }
154       });
155     }
156   };
157   useEffect(() => {
158     setSectionFilter([]);
159   }, [isArchive]);
160   return (
161     <div className='flex flex-row px-14 pb-16 pt-3'>
162       {/*------ Modals ----------*/}
163       <ConfirmationModal
164         title={translations.modalTitle}
165         description={translations.modalDeleteDescription}
166         type='danger'
167         showModal={deleteModal}
168         confirmText={translations.modalConfirmText}
169         cancelText={translations.modalCancelText}
170         handleConfirmation={async () => {
171           if (!isArchive) {
172             const response = await deleteTask({
173               id: deletedTask.id,
174               date: date,
175               allEvents: deletedTask?.isRepeatable ?? false,
176             });
177             if (typeof response === 'boolean') {
178               toast(translations.FlashMessageDeletedTask, {
179                 type: 'success',
```

```
src > components > organisms > TaskClientComponent.tsx > TaskClientComponent
36  export default function TaskClientComponent({
170         handleConfirmation={async () => {
172           const response = await deleteTask({
173             id: deletedTask.id,
174             date: date,
175             allEvents: deletedTask?.isRepeatable ?? false,
176           });
177           if (typeof response === 'boolean') {
178             toast(translations.FlashMessageDeletedTask, {
179               type: 'success',
180             });
181           } else {
182             const errors = response.message.split('.');
183             errors.map((part: string) =>
184               toast(translations[part], { type: 'error' })
185             );
186           }
187         } else {
188           const response = await deleteTaskTemplate(deletedTask.id);
189           if (typeof response === 'boolean') {
190             toast(translations.FlashMessageDeleteTemplate, {
191               type: 'success',
192             });
193           } else {
194             const errors = response.message.split('.');
195             errors.map((part: string) => toast(part, { type: 'error' }));
196           }
197         }
198         setDeleteModal(false);
199         }}
200         handleCancellation={() => {
201           setDeleteTask({} as Task);
202           setDeleteModal(false);
203         }}
204       />
205
206       {/*----------------------------*/}
207       <DragDropContext onDragEnd={onDragEnd}>
208         {isSortedYet && (
209           <>
210             {/*Left Columns*/}
211
212             {!taskExpanded ? (
213               <div className='w-1/2 flex flex-col items-center relative overflow-y-scroll overflow-x-hidden h-screen pb-[16.5rem] scrollbar-none'>
214                 <div className='w-[0.07rem] h-[60%] absolute right-[-1.25rem] overflow-y-hidden bg-gradient-to-t from-taskBackground via-dark-80 to-taskBackground
215                 {!isArchive ? (
216                   <div className='flex w-full'>
217                     <div>
218                       <MonthDropDown months={months} setLoading={setLoading} />
219                     </div>
220                     <div class='flex w-full justify-center ml-[-6rem]'>
221                       <TaskViewerWeekChange
222                         translations={translations}
```

```
LoginForm.tsx   page.tsx .../account   PasswordChange.tsx   page.tsx .../department   page.tsx .../tasks   TaskClientData.tsx 1   TaskClientComponent.tsx 2
src > components > organisms > TaskClientComponent.tsx > TaskClientComponent
36  export default function TaskClientComponent({
218                       <MonthDropDown months={months} setLoading={setLoading} />
219                     </div>
220                     <div className='flex w-full justify-center ml-[-6rem]'>
221                       <TaskViewerWeekChange
222                         translations={translations}
223                         setLoading={setLoading}
224                       />
225                     </div>
226                   </div>
227                 ) : (
228                   <div className='flex w-full items-center'>
229                     <div
230                       className='cursor-pointer flex items-center'
231                       onClick={() => setIsArchive(false)}
232                     >
233                       <span>
234                         <BackIconSvg />
235                       </span>
236                       <p className=' text-white font-bold ml-2 text-sm'>
237                         {translations.backToPlanning}
238                       </p>
239                     </div>
240                     <div className='flex justify-center w-7/12'>
241                       <p className=' text-white text-lg font-bold'>
242                         {translations.taskArchive}
243                       </p>
244                     </div>
245                   </div>
246                 )}
247
248                 {!isArchive && (
249                   <>
250                     <div className=' mt-2'>
251                       <DateSwiper
252                         weekDaysTranslations={translations}
253                         setLoading={setLoading}
254                       />
255                     </div>
256                     <div className='w-full justify-center flex'>
257                       <Tab
258                         translations={translations}
259                         active={activeState}
260                         setActiveState={setActiveState}
261                       />
262                     </div>
263                   </>
264                 )}
265                 <div className='w-full'>
266                   <div class='w-full overflow-x-scroll scrollbar-none mt-3'>
267                     <div class='flex w-96 '>
268                       {sections.map((section: Section) => {
269                         return (
```

LoginForm.tsx | page.tsx .../account | PasswordChange.tsx | page.tsx .../department | page.tsx .../tasks | TaskClientData.tsx 1 | TaskClientComponent.tsx 2

src > components > organisms > TaskClientComponent.tsx > TaskClientComponent

```
36  export default function TaskClientComponent({
264                 )}
265                 <div className='w-full'>
266                   <div className='w-full overflow-x-scroll scrollbar-none mt-3'>
267                     <div className='flex w-96 '>
268                       {sections.map((section: Section) => {
269                         return (
270                           <div className='pe-2' key={section.id}>
271                             <CustomButton
272                               color={
273                                 sectionFilter.includes(section.id)
274                                   ? 'primary'
275                                   : 'default'
276                               }
277                               onClick={() => {
278                                 if (
279                                   sectionFilter.length === 1 &&
280                                   sectionFilter[0] === section.id
281                                 ) {
282                                   setSectionFilter([]);
283                                   return;
284                                 }
285
286                                 if (sectionFilter.includes(section.id)) {
287                                   const filter = sectionFilter.filter(
288                                     (id) => id !== section.id
289                                   );
290                                   setSectionFilter(filter);
291                                   return;
292                                 }
293                                 const filter = [...sectionFilter, section.id];
294                                 setSectionFilter(filter);
295                               }}
296                             >
297                               <p className='w-auto whitespace-nowrap text-xs text-white'>
298                                 {section.name}
299                               </p>
300                             </CustomButton>
301                           </div>
302                         );
303                       })}
304                     </div>
305                   </div>
306                 </div>
307
308                 {loading || loadingData ? (
309                   <div className='flex justify-center items-center mt-12 w-full'>
310                     <div className='animate-spin rounded-full h-32 w-32 border-b-2 border-primary'></div>
311                   </div>
312                 ) : (
313                   <div className='w-full mt-3'>
314                     {!isArchive ? (
315                       <>
```

LoginForm.tsx | page.tsx .../account | PasswordChange.tsx | page.tsx .../department | page.tsx .../tasks | TaskClientData.tsx 1 | TaskClientComponent.t

src > components > organisms > TaskClientComponent.tsx > TaskClientComponent

```
36  export default function TaskClientComponent({
312                 ) : (
313                   <div className='w-full mt-3'>
314                     {!isArchive ? (
315                       <>
316                         {data.length > 0 ? (
317                           <div className='w-full gap-x-4 '>
318                             <TaskItem
319                               tasks={data}
320                               setNewTask={setNewTask}
321                               setDeleteModal={setDeleteModal}
322                               setDeleteTask={setDeleteTask}
323                               isArchive={isArchive}
324                               translations={translations}
325                               sections={sections}
326                               setContextTask={setDraggableContextTask}
327                               setTaskExpanded={setTaskExpanded}
328                             />
329                           </div>
330                         ) : (
331                           <div className='flex justify-center items-center h-60'>
332                             <p className=' text-white text-base font-bold'>
333                               {translations['emptyTasksTitle']}
334                             </p>
335                           </div>
336                         )}
337                       </>
338                     ) : (
339                       <>
340                         {templateData.length > 0 ? (
341                           <div className='w-full gap-x-4 '>
342                             <TaskItem
343                               tasks={templateData}
344                               setNewTask={setNewTask}
345                               setDeleteModal={setDeleteModal}
346                               setDeleteTask={setDeleteTask}
347                               isArchive={isArchive}
348                               translations={translations}
349                               sections={sections}
350                               setContextTask={setDraggableContextTask}
351                               setTaskExpanded={setTaskExpanded}
352                             />
353                           </div>
354                         ) : (
355                           <div className='flex justify-center items-center h-60'>
356                             <p className=' text-white text-base font-bold'>
357                               {translations['emptyTasksTitle']}
358                             </p>
359                           </div>
360                         )}
361                       </>
362                     )}
363                   </div>
```

```
LoginForm.tsx   page.tsx .../account   PasswordChange.tsx   page.tsx .../department   page.tsx .../tasks   TaskClientData.tsx 1   TaskClientComponent.tsx 2
src > components > organisms > TaskClientComponent.tsx > TaskClientComponent
36  export default function TaskClientComponent({
359                                  </div>
360                                )}
361                              </>
362                            )}
363                          </div>
364                        )}
365                      </div>
366                    ) : (
367                      <div className='w-1/2'>
368                        <TaskExpandView
369                          task={draggableContextTask}
370                          sections={sections}
371                          translations={translations}
372                          setNewTask={setNewTask}
373                          setDeleteModal={setDeleteModal}
374                          setDeleteTask={setDeleteTask}
375                          isArchive={isArchive}
376                          setContextTask={setDraggableContextTask}
377                          setTaskExpanded={setTaskExpanded}
378                          setIsDraggedTask={setIsDraggedTask}
379                        />
380                      </div>
381                    )}
382
383                    <div className='w-[0.1rem] ml-10 mt-[-0.6rem] bg-gradient-to-t from-dark via-dark-80 to-dark' />
384
385                    {/*Right Columns*/}
386                    <div className=' w-1/2'>
387                      <TaskCreator
388                        translations={translations}
389                        sections={sections}
390                        editTask={newTask}
391                        setEditTask={setNewTask}
392                        isArchive={isArchive}
393                        setIsArchive={setIsArchive}
394                        isDraggedTaskArchive={isDraggedTaskArchive}
395                        setisDraggedTaskArchive={setisDraggedTaskArchive}
396                        setLoader={setLoading}
397                        isDraggedTask={isDraggedTask}
398                        taskExpanded={taskExpanded}
399                      />
400                    </div>
401                  </>
402                )}
403              </DragDropContext>
404            </div>
405          );
406        }
407
```

Props and State Management

- Props: The component receives various props such as sections, tasks, templateTasks, translations, and others related to date and loading state.
- State Variables:
- Task and UI State: Manages states related to task interaction, such as creating new tasks, tracking a deleted task, whether a task is being dragged, and whether the task viewer is expanded.
- Data Arrays: Maintains arrays for tasks and template tasks which can be manipulated locally.
- Modal States: Controls visibility of confirmation modals and other UI elements.
- Filter and Sorting States: Manages states related to filtering tasks by sections or other criteria and whether data has been sorted or filtered.

Hooks and Lifecycle

useEffect:

- Sets the context date to ensure synchronization with the selected date.
- Initializes or updates data based on changes in tasks, templates, or filters, and controls loading animations.
- useLayoutEffect: Ensures data is filtered and sorted based on user interactions like changing the active state or selecting filters, prior to browser paint.
- User Interaction Handlers
- Drag and Drop: Implements drag-and-drop functionalities for tasks using react-beautiful-dnd, handling end of drag events to potentially update tasks or create new ones based on the drop result.

Task Interaction:

- Editing and Deleting: Functions to toggle task editing mode, open modals for confirmation, and handle the actual deletion of tasks or templates via API calls.
- Filtering and Active State: Controls which tasks are displayed based on active state toggles and section filters, enhancing user experience by allowing them to focus on relevant tasks.

Components Usage

Task and Section Components:

- TaskItem and TaskCreator are used for displaying individual tasks and creating/editing tasks respectively.
- DateSwiper, Tab, MonthDropDown, and TaskViewerWeekChange provide interfaces for navigating and filtering tasks based on time frames and other criteria.

Modals and Icons:

- ConfirmationModal for confirming deletions and other critical actions.
- SVG components like BackIconSvg for navigation within the UI.

Rendering Logic

The component renders a complex UI that divides the screen into multiple sections:

- Left Column: Displays tasks based on current filters, with options to change views between archived tasks and active tasks. Includes UI elements to switch between different views and manage tasks.
- Right Column: Dedicated to task creation and editing, updated based on whether the task is being dragged or edited.
- Event Handling and Side Effects
- Modals and Toast Notifications: Uses react-toastify to provide feedback on operations like task deletions.
- Dynamic Task Loading: Handles conditional rendering and loading states to provide a responsive user experience.

2.3.4.2. Nest.js - Back-end code

src > modules > tasks > tasks.controller.ts > TasksController > findDailyByDate

```
18  import { UpdateTaskStatusDto } from './dto/update-task-status.dto';
19  import { RepeatableTask, Task } from './entity/task.entity';
20
21  @ApiBearerAuth()
22  @ApiTags('tasks')
23  @Controller('tasks')
24  export class TasksController {
25    constructor(private readonly tasksService: TasksService) {}
26
27    @ApiGetDailyTasksByDate()
28    @Get()
29    @HasRoles(Role.Manager, Role.HQ)
30    async findDailyByDate(@Query() { date }: QueryDateDto): Promise<Task[]> {
31      return await this.tasksService.findDailyByDate(date);
32    }
33
34    @ApiGetRepeatableTasksByDate()
35    @Get(`repeatable`)
36    @HasRoles(Role.Manager)
37    async findRepeatableByDate(@Query() { date }: QueryDateDto): Promise<RepeatableTask[]> {
38      return await this.tasksService.findRepeatableByDate(date);
39    }
40
41    @ApiCreateTask()
42    @Post()
43    @HasRoles(Role.Manager)
44    async create(@Body() taskDto: CreateTaskDto): Promise<boolean> {
45      return this.tasksService.create(taskDto);
46    }
47
48    @ApiUpdateTask()
49    @Put()
50    @HasRoles(Role.HQ, Role.Manager)
51    async update(@Body() taskDto: UpdateTaskDto): Promise<boolean> {
52      return this.tasksService.update(taskDto);
53    }
54
55    @ApiUpdateTaskStatus()
56    @Patch('status')
57    @HasRoles(Role.HQ, Role.Manager)
58    async updateStatus(@Body() taskDto: UpdateTaskStatusDto): Promise<boolean> {
59      return this.tasksService.updateStatus(taskDto);
60    }
61
62    @ApiDeleteTask()
63    @Delete()
64    @HasRoles(Role.Manager)
65    async delete(@Query() taskDto: DeleteTaskDto): Promise<boolean> {
66      return this.tasksService.delete(taskDto);
67    }
68  }
69
```

> modules > tasks > services > tasks.service.ts > TasksService > updateStatus

```
0
1   @Injectable()
2   export class TasksService {
3     constructor(
4       private readonly deleteTaskService: DeleteTaskService,
5       private readonly updateTaskService: UpdateTaskService,
6       private readonly tasksHelperService: TasksHelperService,
7       private readonly tasksValidationService: TasksValidationService,
8       private readonly repository: Repository,
9       private readonly cls: CellClsService,
0       private readonly eventEmitter: EventEmitter2,
1     ) {}
2
3     async findDailyByDate(date: string, storeId?: string): Promise<Task[]> {
4       const dailyTasks = (
5         await this.repository.findAll<Task>(DB_KEYS.TASKS.DAILY(storeId ?? this.cls.storeId, date))
6       )?.map((t) => {
7         return !t.repeatableTaskId ? { ...t, isRepeatable: false } : t;
8       });
9
0       if (date !== dateNowFormatted()) return dailyTasks;
1
2       const repeatableTasks = await this.repository.findAll<RepeatableTask>(
3         DB_KEYS.TASKS.REPEATS(storeId ?? this.cls.storeId),
4       );
5
6       const dailyRepeatableTasks = dailyTasks.filter((a) => a.repeatableTaskId);
7       for (const task of dailyRepeatableTasks) {
8         const repeatableTask = repeatableTasks.find((rTask) => rTask.id === task.repeatableTaskId);
9         if (repeatableTask) {
0           task.endDate = repeatableTask.endDate;
1           task.repeatDaysInWeek = repeatableTask.repeatDaysInWeek;
2           task.isRepeatable = isRepeatable(repeatableTask);
3           continue;
4         }
5         task.isRepeatable = false;
6       }
7
8       return dailyTasks;
9     }
0
1     async findRepeatableByDate(date: string): Promise<RepeatableTask[]> {
2       const repeatableTasks = await this.repository.findAll<RepeatableTask>(DB_KEYS.TASKS.REPEATS(this.cls.storeId));
3       const todayTasks = (
4         await this.repository.findAll<Task>(DB_KEYS.TASKS.DAILY(this.cls.storeId, dateNowFormatted()))
5       )?.filter((task) => task?.repeatableTaskId);
6
7       const dailyRepeatableTasks: RepeatableTask[] = [];
8       const formattedStartDate = formatDate(date);
9       const startDate = new Date(formattedStartDate);
0
1       for (const repeatableTask of repeatableTasks) {
2         const isRepeatableSingleInstance = isRepeatableTaskSingleInstance(date, repeatableTask);
```

Ln 131, Col 9 Spaces: 2 UTF-8 LF {} T

src > modules > tasks > services > TS tasks.service.ts > TasksService > updateStatus

```
32  export class TasksService {
71    async findRepeatableByDate(date: string): Promise<RepeatableTask[]> {
81      for (const repeatableTask of repeatableTasks) {
82        const isRepeatableSingleInstance = isRepeatableTaskSingleInstance(date, repeatableTask);
83
84        if (isRepeatableSingleInstance) {
85          findDailyRepeatableTasks(formattedStartDate, repeatableTask, todayTasks, dailyRepeatableTasks);
86          continue;
87        }
88        const isDayBetweenTaskDates = isDateBetween({
89          date: formattedStartDate,
90          startDate: repeatableTask.startDate,
91          endDate: repeatableTask.endDate,
92        });
93        const isTaskRepeatDaysMatchedWithDay = repeatableTask.repeatDaysInWeek.includes(startDate.getDay());
94        const isNotExcluded = !repeatableTask.excludeDays?.includes(formattedStartDate);
95
96        if (isDayBetweenTaskDates && isTaskRepeatDaysMatchedWithDay && isNotExcluded) {
97          findDailyRepeatableTasks(formattedStartDate, repeatableTask, todayTasks, dailyRepeatableTasks);
98        }
99      }
100
101     return dailyRepeatableTasks;
102   }
103
104   async create(taskDto: CreateTaskDto): Promise<boolean> {
105     if (taskDto.startDate === dateNowFormatted()) {
106       let newRepeatableTasks: RepeatableTask[] = [];
107       if (taskDto.repeatDaysInWeek.length) {
108         const newStartDate = addDaysWithFormat(taskDto.startDate, 1);
109         newRepeatableTasks = createRepeatableTasksFromSections({ ...taskDto, startDate: newStartDate });
110         await this.tasksHelperService.saveRepeatableTasks(newStartDate, newRepeatableTasks, true);
111       }
112       const newDailyTasks = createTasksFromSections(taskDto, newRepeatableTasks);
113       await this.tasksHelperService.saveDailyTasks(newDailyTasks);
114     } else {
115       const newRepeatableTasks = createRepeatableTasksFromSections(taskDto);
116       await this.tasksHelperService.saveRepeatableTasks(taskDto.startDate, newRepeatableTasks, false);
117     }
118
119     return true;
120   }
121
122   async update(taskDto: UpdateTaskDto): Promise<boolean> {
123     if (taskDto.date === dateNowFormatted()) {
124       await this.updateTaskService.updateTodayTask(taskDto);
125     } else {
126       await this.updateTaskService.updateRepeatableTask(taskDto);
127     }
128     return true;
129   }
130
```

ice.ts | TS users.repository.ts | TS users.controller.ts 4 | TS users.service.ts 1 | TS sections.controller.ts | TS tasks.controller.ts | TS tasks.service.ts × | TS update-task.ser

c > modules > tasks > services > TS tasks.service.ts > TasksService > updateStatus

```
32  export class TasksService {
29    }
30
31    async updateStatus(taskDto: UpdateTaskStatusDto): Promise<boolean> {
32      const dbKey = DB_KEYS.TASKS.DAILY(this.cls.storeId, formatDate(taskDto.currentDate));
33      const task = await this.repository.findOne<Task>(dbKey, taskDto.id);
34      this.tasksValidationService.validateTaskExists(task);
35      this.tasksValidationService.validateTaskUpdateStatus(task.status, taskDto.status);
36
37      task.status = taskDto.status;
38      if (taskDto.status === TaskStatus.INITIATED) {
39        task.actualStartTime = taskDto.currentTime;
40      }
41      if (taskDto.status === TaskStatus.DONE) {
42        task.endTime = taskDto.currentTime;
43      }
44      await this.repository.update<Task>(dbKey, task);
45
46      const taskEvent = {
47        storeId: this.cls.storeId,
48        syncStatus: EventSyncStatus.UPDATED,
49        tasks: [task],
50        taskIdForDelete: task.id,
51        date: formatDate(taskDto.currentDate),
52      } as TasksEvent;
53      this.eventEmitter.emit(EVENTS.TASKS.DAILY, taskEvent);
54
55      return true;
56    }
57
58    async delete(taskDto: DeleteTaskDto): Promise<boolean> {
59      if (taskDto.date === dateNowFormatted()) {
60        await this.deleteTaskService.deleteTodayTask(taskDto);
61      } else {
62        await this.deleteTaskService.deleteRepeatableTask(taskDto);
63      }
64      return true;
65    }
66
67    async deleteAllSectionTasks(sectionIds, storeId): Promise<boolean> {
68      console.time();
69      const dbKey = DB_KEYS.TASKS.ALL_BY_STORE(storeId);
70      const matchingKeys = await this.repository.findAllMatchingKeys(dbKey);
71
72      await Promise.all(
73        matchingKeys.map(async (taskKey) => {
74          const keyDate = taskKey.split('|')[2].split(':')[1];
75          const dailyTasks = await this.findDailyByDate(keyDate, storeId);
76          const deleteTasksArr: DeleteTask[] = [];
77          for (const task of dailyTasks) {
78            if (sectionIds.includes(task.sectionId)) {
79              const deleteTaskDto: DeleteTask = {
80                id: task.id,
```

```
src > modules > tasks > services > TS tasks.service.ts > TasksService > updateStatus
32   export class TasksService {
158    async delete(taskDto: DeleteTaskDto): Promise<boolean> {
160        await this.deleteTaskService.deleteTodayTask(taskDto);
161      } else {
162        await this.deleteTaskService.deleteRepeatableTask(taskDto);
163      }
164      return true;
165    }
166
167    async deleteAllSectionTasks(sectionIds, storeId): Promise<boolean> {
168      console.time();
169      const dbKey = DB_KEYS.TASKS.ALL_BY_STORE(storeId);
170      const matchingKeys = await this.repository.findAllMatchingKeys(dbKey);
171
172      await Promise.all(
173        matchingKeys.map(async (taskKey) => {
174          const keyDate = taskKey.split('|')[2].split(':')[1];
175          const dailyTasks = await this.findDailyByDate(keyDate, storeId);
176          const deleteTasksArr: DeleteTask[] = [];
177          for (const task of dailyTasks) {
178            if (sectionIds.includes(task.sectionId)) {
179              const deleteTaskDto: DeleteTask = {
180                id: task.id,
181                date: task.date,
182                allEvents: true,
183              };
184              deleteTasksArr.push(deleteTaskDto);
185            }
186          }
187          if (deleteTasksArr.length !== 0) await this.deleteTaskService.deleteManyTodayTasks(deleteTasksArr, storeId);
188        }),
189      );
190
191      console.timeEnd();
192      return true;
193    }
194  }
195
```

TasksController

- Annotations: The controller is decorated with Swagger annotations (@ApiBearerAuth and @ApiTags) for API documentation and security.
- Role-Based Access Control: The @HasRoles decorator is used to restrict access to different endpoints based on user roles.

Endpoints:

- GET /tasks: Retrieves daily tasks by date. It accepts a date query and returns a list of tasks for that date.
- GET /tasks/repeatable: Fetches repeatable tasks by date, similar to the daily tasks endpoint but specifically for tasks that repeat on a schedule.
- POST /tasks: Allows the creation of a new task. It requires task details in the request body and returns a boolean indicating success.
- PUT /tasks: Updates an existing task based on the details provided in the request body.
- PATCH /tasks/status: Specifically updates the status of a task, indicating operations like starting, pausing, or completing a task.
- DELETE /tasks: Deletes a task based on identifiers provided through query parameters.

TasksService

Functionality: This service handles the business logic associated with task operations, interfacing with a repository for data persistence.

Key Methods:

- findDailyByDate and findRepeatableByDate: These methods retrieve tasks based on their type and the specified date. They involve complex logic to filter tasks appropriately, considering attributes like repeat schedules and exclusions.
- create, update, updateStatus, delete: These methods manage the lifecycle of tasks, performing validations and ensuring data integrity. They interact with the repository to perform CRUD operations and handle special conditions like repeatable tasks.

Implementation Details

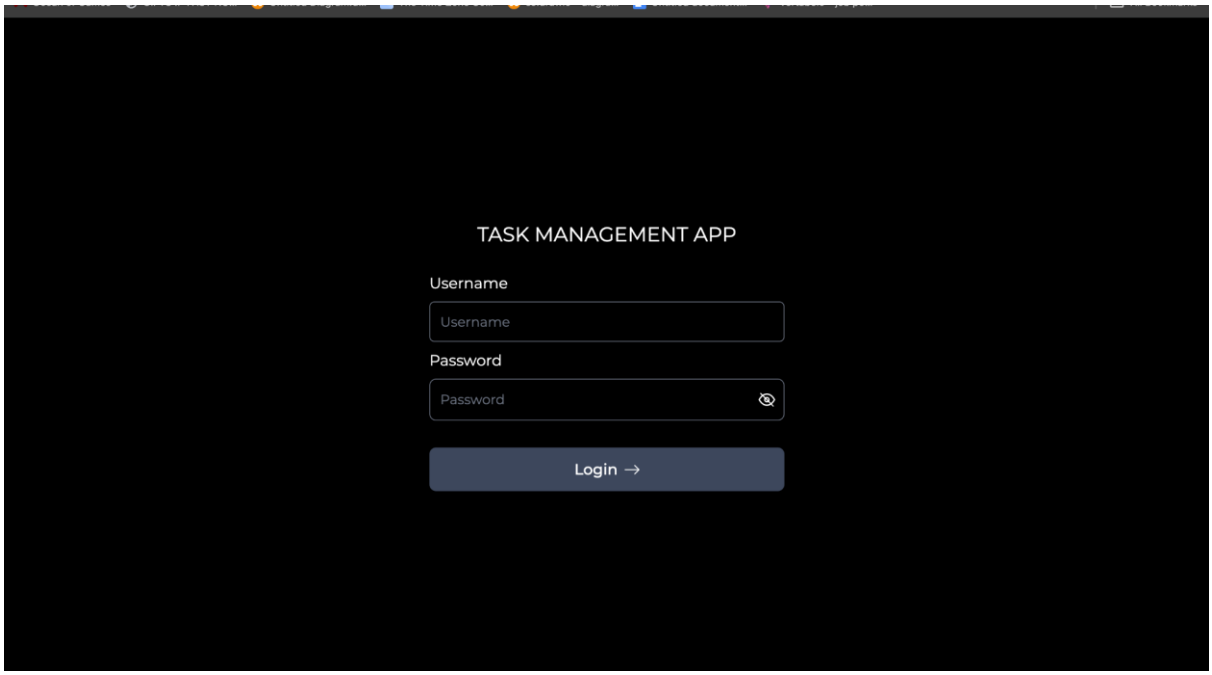
- Repository Interaction: The service uses a repository pattern to abstract database operations. This pattern helps in maintaining clean code and makes the system easier to test and maintain.
- Event Emitter: Utilizes EventEmitter2 for broadcasting events related to task changes, which can be used for logging, notifications, or triggering other asynchronous processes.
- Task Validation: Before performing operations like update or delete, tasks are validated using a dedicated validation service. This ensures that only valid and permissible operations are executed.
- Error Handling: Proper error handling is implemented to provide meaningful feedback for operations, which is crucial for maintaining a robust system.

Additional Features

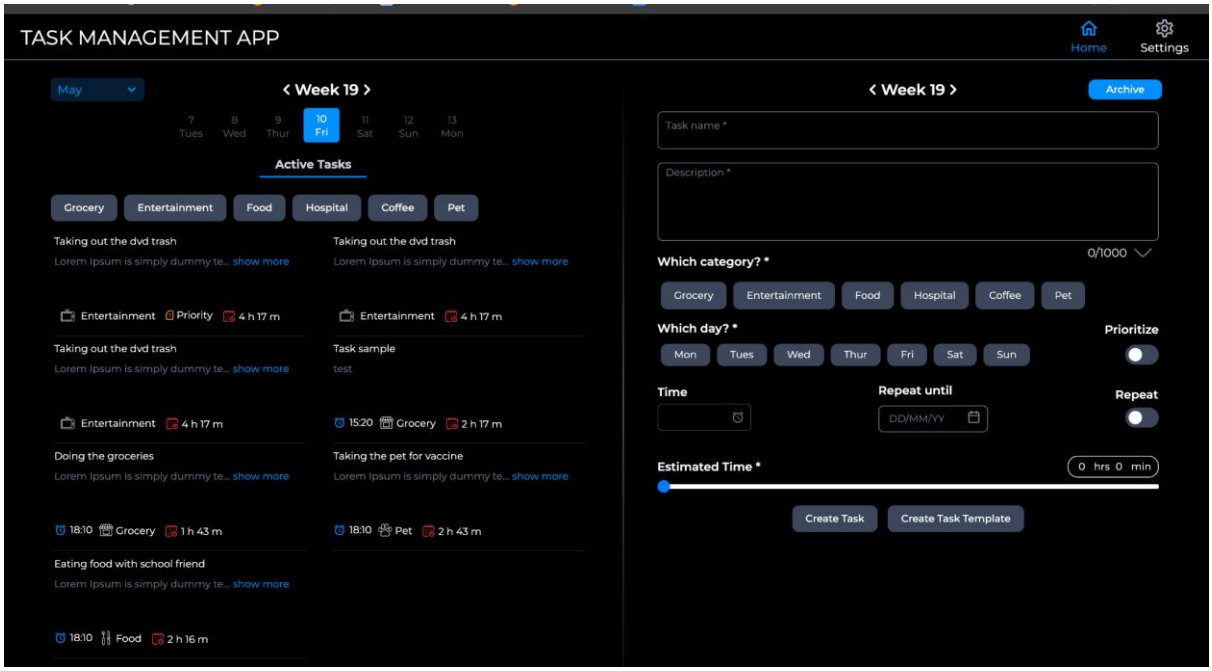
- Role-Based Access: Demonstrates how NestJS can be used to enforce security policies at the controller level, ensuring that endpoints are only accessible to users with appropriate roles.
- Queue Management: For operations that might require asynchronous execution or need to be performed in bulk without blocking the main thread, a queuing system (Bull) can be integrated, as hinted in the service with placeholders for such implementations.

2.4. Graphical User Interface (GUI)

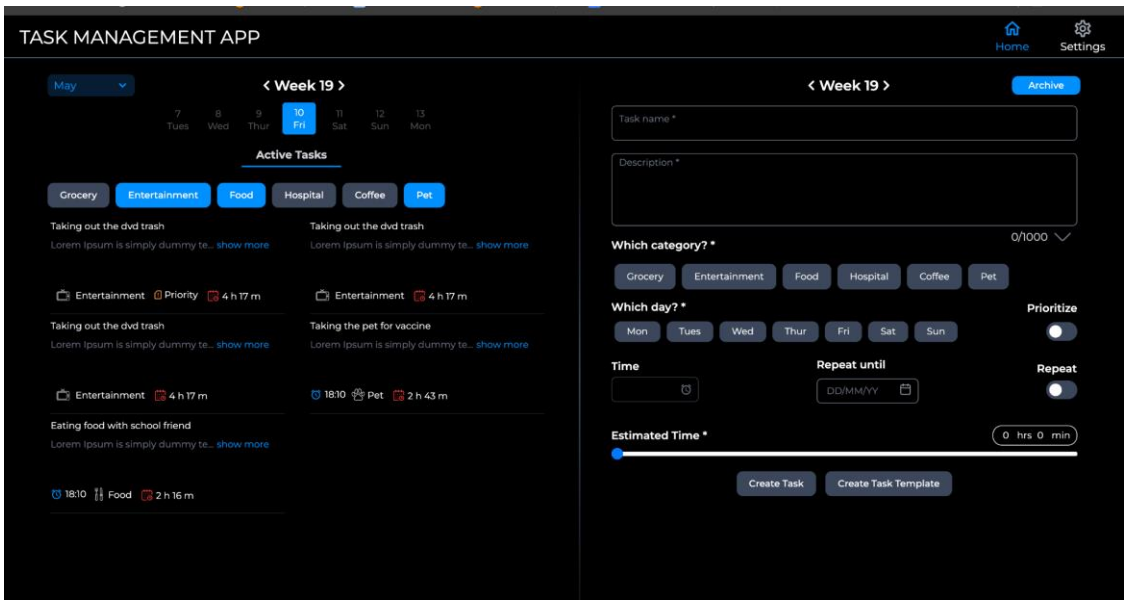
1. Sign In Page



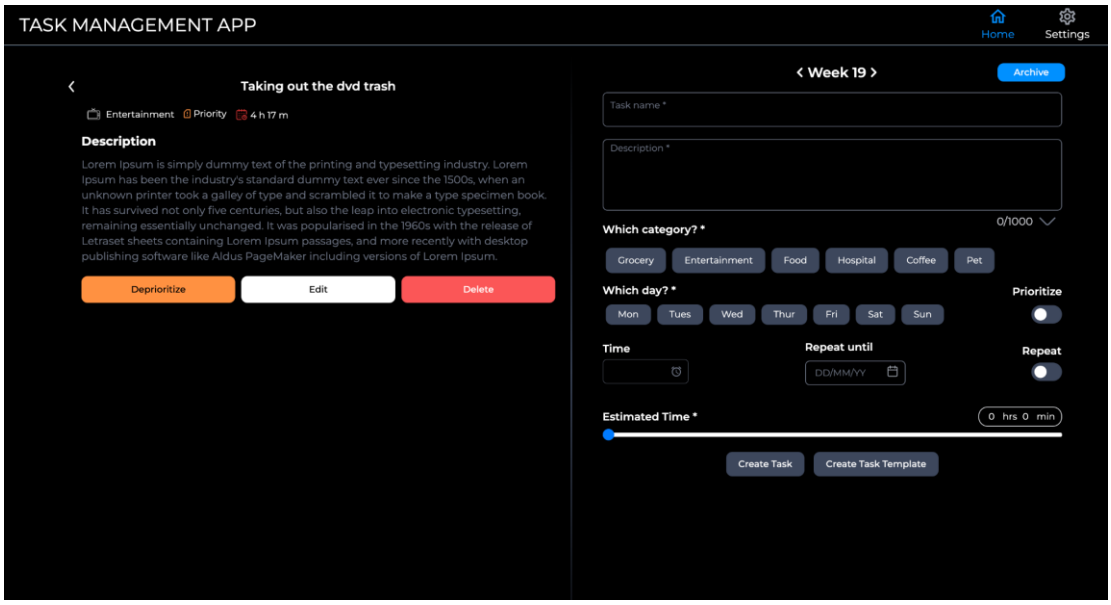
2. Home/Tasks Page



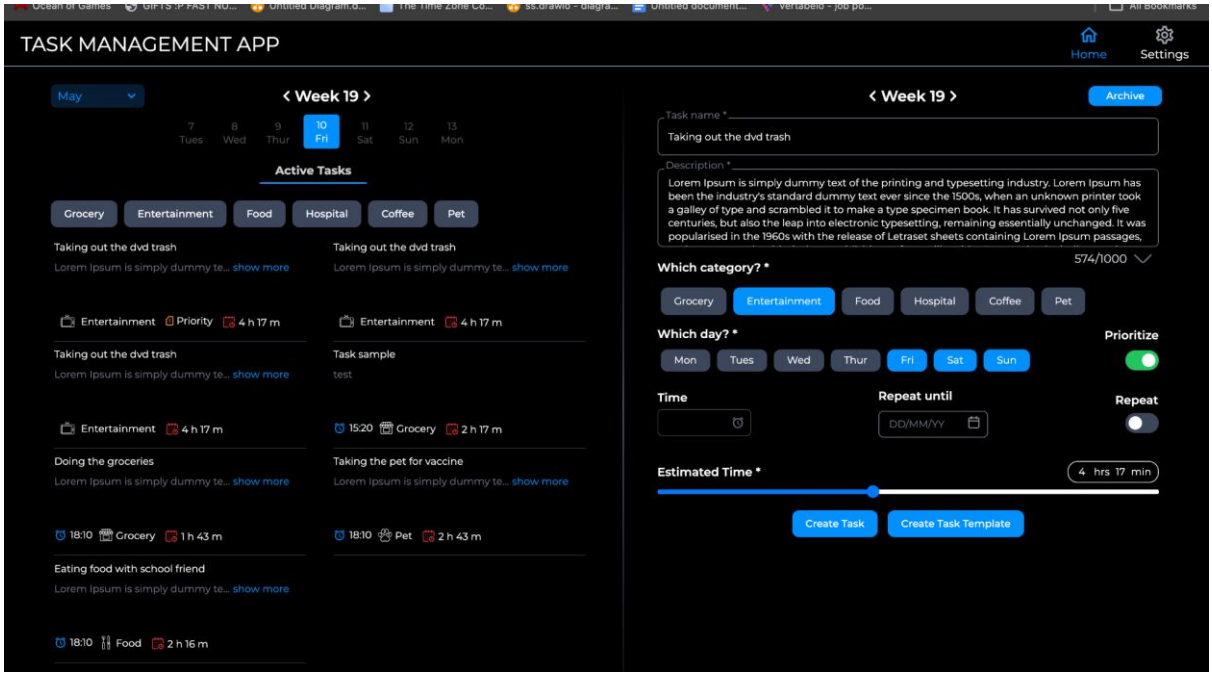
3. Tasks page when category filters are applied



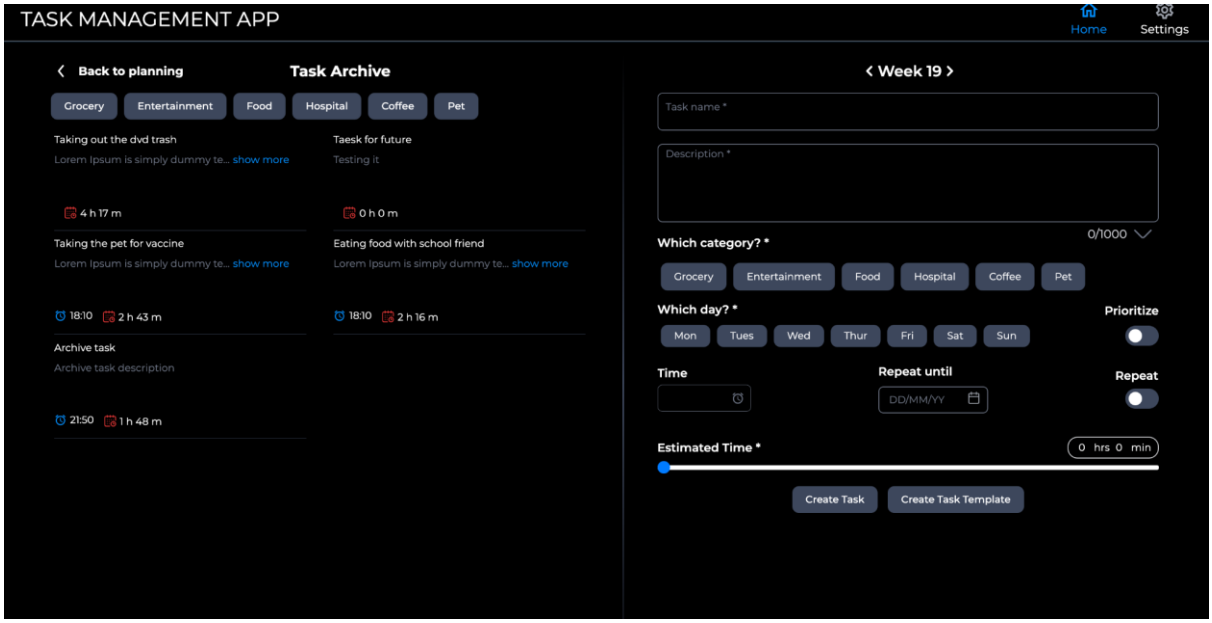
4. Home/Tasks page when a task is selected



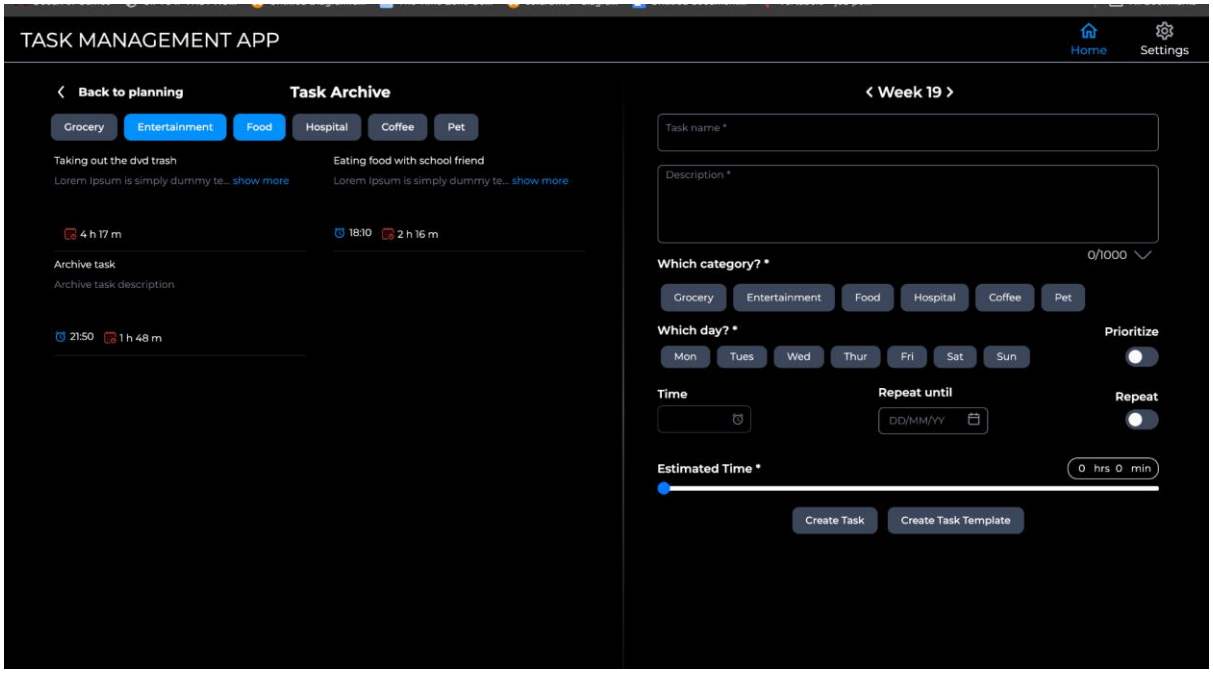
5. Home/Tasks page when a task drags and dropped to the right create task pane



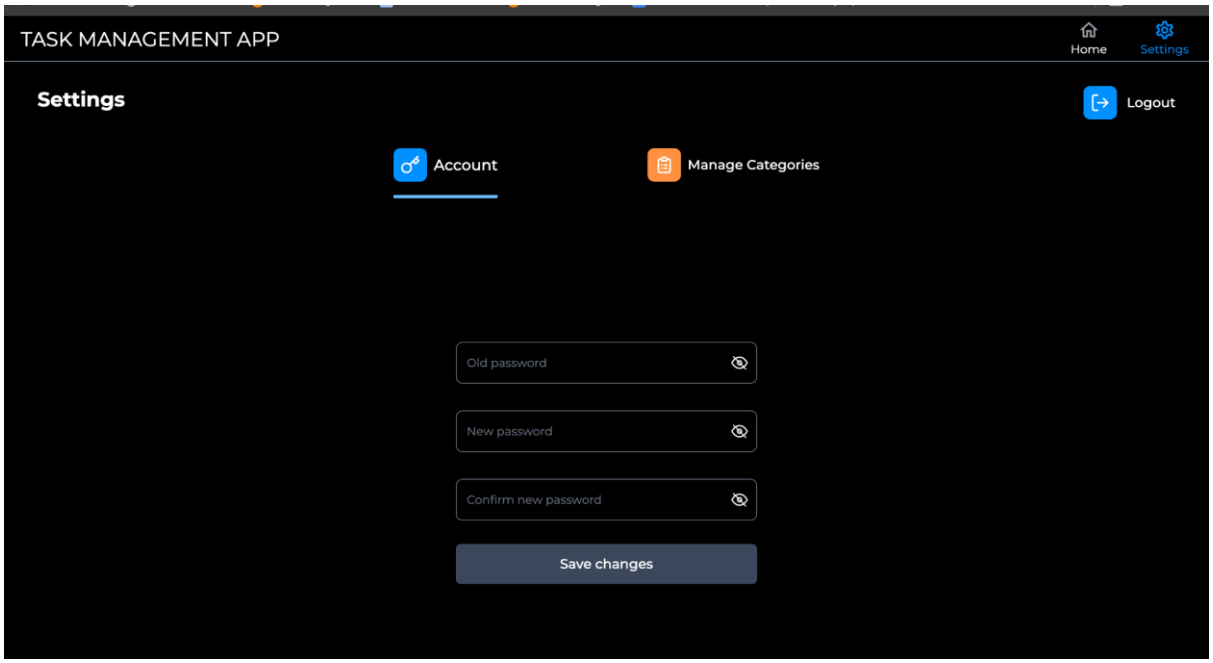
6. Archived/Template Tasks Page



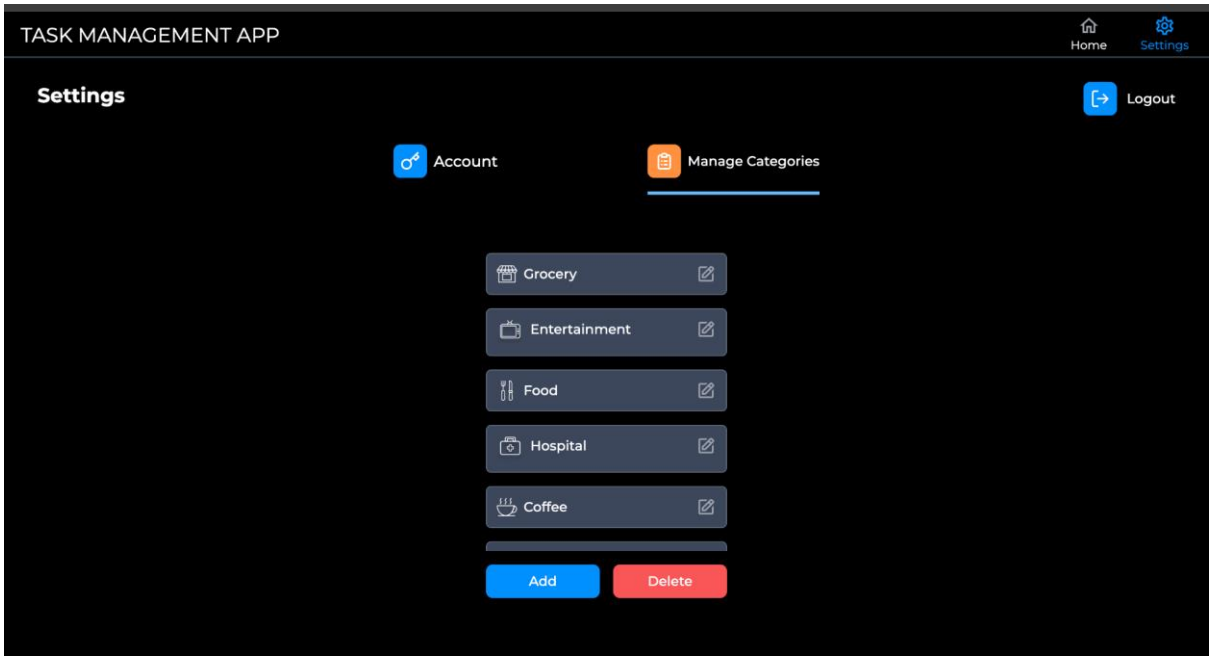
7. Archived Tasks page when category filters are applied



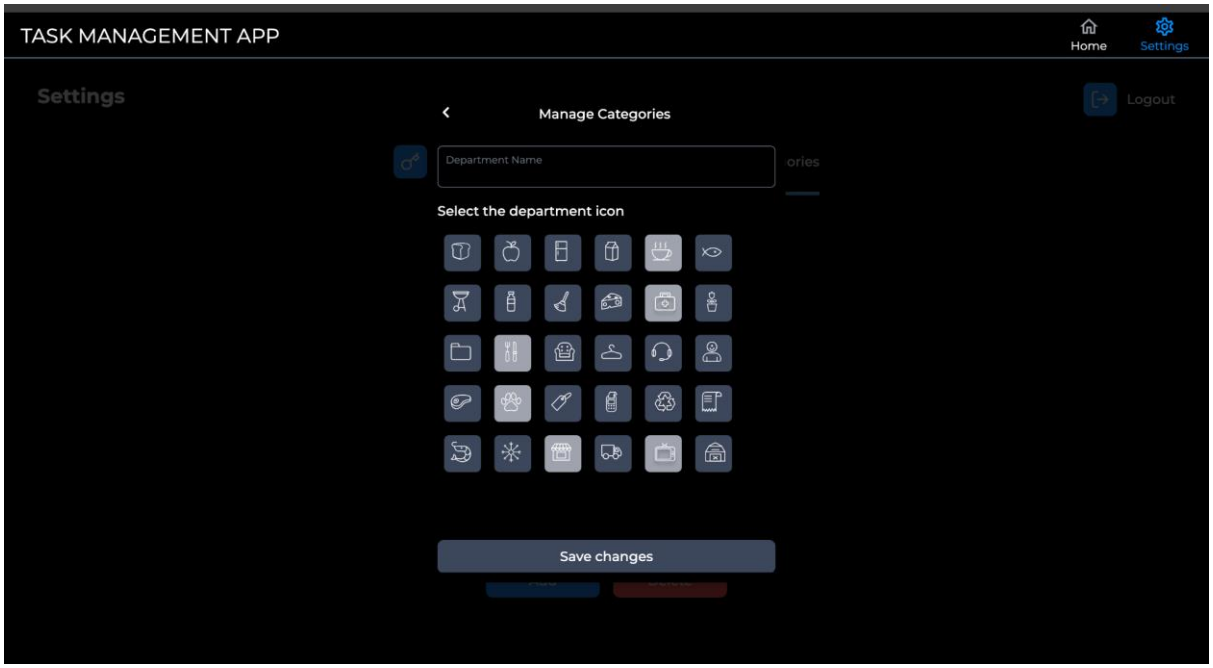
8. User profile page where user can update password



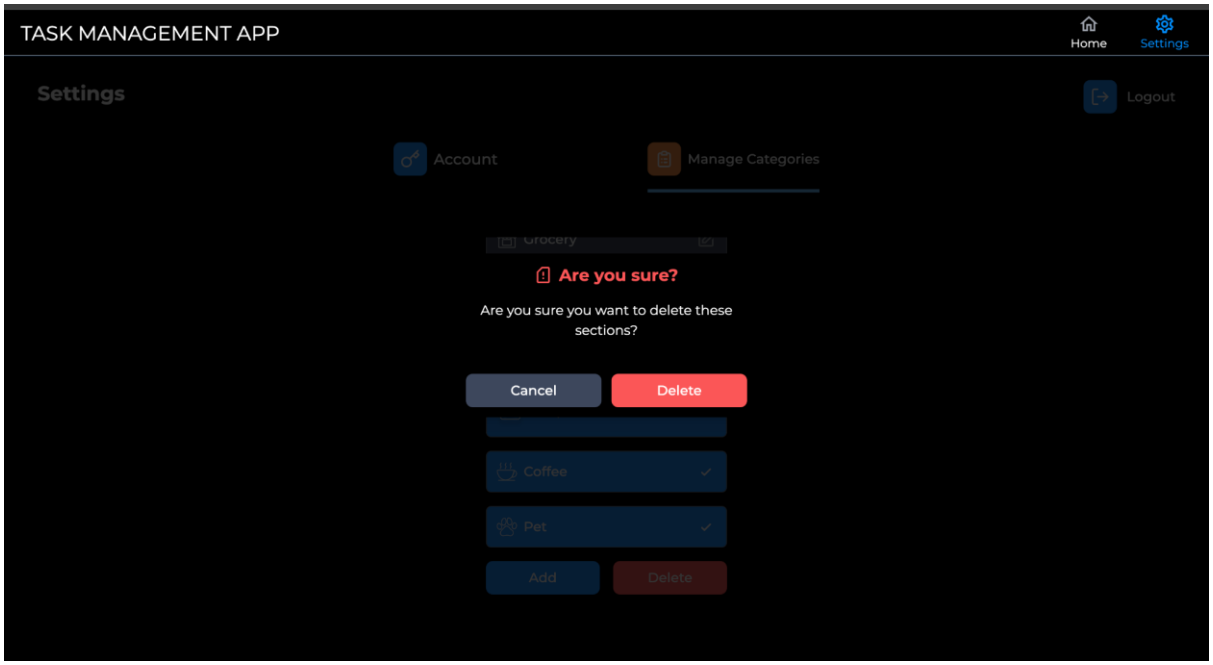
9. Categories page



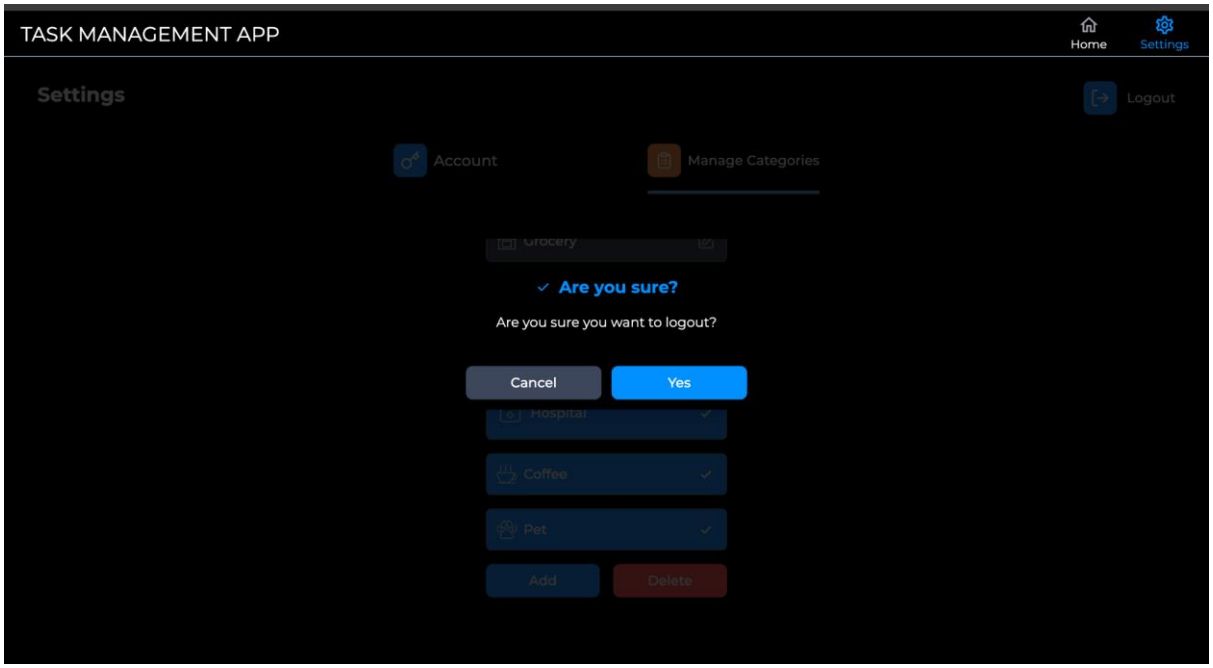
10. Add a new Category page



11. Delete selected Categories



12. Logout popup



2.5. Testing

Testing is an integral component of the software development lifecycle, crucial for validating the functionality, usability, and reliability of the Task Management Web Application. This section delves into the comprehensive testing approach adopted, which encompasses several testing methodologies to ensure a robust application.

Unit Testing

Unit tests were developed to test individual components and functions independently. Using Jest, components like TaskItem, LoginForm, and services such as AuthService were tested to ensure that they perform expected operations accurately under various conditions. Mocks and spies were utilized to simulate interactions with dependencies like APIs and context providers.

1. **Task Service test cases:**

```
jest.mock('uuid', () => ({ v4: () => savedTasksMock[0].id }));

describe('TasksService', () => {
  let service: TasksService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [
        TasksService,
        TasksValidationService,
        EventEmitter2,
        {
          provide: Repository,
          useValue: {
            findAll: async () => jest.fn(),
            findMany: async () => jest.fn(),
            findOne: async () => jest.fn(),
            update: async () => jest.fn(),
          },
        },
        {
          provide: DeleteTaskService,
          useValue: {
            deleteTodayTask: async () => jest.fn(),
            deleteRepeatableTask: async () => jest.fn(),
          },
        },
        {
          provide: UpdateTaskService,
          useValue: {
            updateTodayTask: async () => jest.fn(),
            updateRepeatableTask: async () => jest.fn(),
          },
        },
        {
          provide: TasksHelperService,
          useValue: {
            saveDailyTasks: async () => jest.fn(),
            saveRepeatableTasks: async () => jest.fn(),
          },
        },
        {
```

```
28  describe('TasksService', () => {
29    beforeEach(async () => {
32      const module: TestingModule = await Test.createTestingModule({
34        ],
35      }).compile();
36
37      service = module.get<TasksService>(TasksService);
38    });
39
40    it('should be defined', () => {
41      expect(service).toBeDefined();
42    });
43    describe('FindAll', () => {
44      it('should call and return findAll by StoreId and not date now', async () => {
45        jest.spyOn(service['repository'], 'findAll').mockResolvedValue([...savedTasksMock]);
46
47        const tasks = await service.findDailyByDate(subtractDaysWithFormat(createTaskMockDto.startDate, 1));
48        expect(tasks).toEqual(savedTasksMock);
49      });
50      it('should call and return findAll by StoreId and Date now', async () => {
51        jest
52          .spyOn(service['repository'], 'findAll')
53          .mockResolvedValueOnce([...savedTasksMock])
54          .mockResolvedValueOnce([...savedRepeatableTasksMock]);
55
56        const tasks = await service.findDailyByDate(createTaskMockDto.startDate);
57        tasks.map((a) => (a.isRepeatable = true));
58        const updatedDailyTasks = savedTasksMock.map((task) => {
59          return { ...task, isRepeatable: true };
60        });
61        expect(tasks).toEqual(updatedDailyTasks);
62      });
63    });
64    describe('FindAllRepeatable', () => {
65      it('should call and return all Repeatable Task by StoreId', async () => {
66        jest.spyOn(service['repository'], 'findAll').mockResolvedValue(savedRepeatableTasksMock);
67        const findAllRepeatableFn = jest.spyOn(service, 'findRepeatableByDate');
68
69        await service.findRepeatableByDate(createTaskMockDto.startDate);
70
71        expect(findAllRepeatableFn).toBeCalledWith(createTaskMockDto.startDate);
72      });
73    });
```

```
.service.ts   TS users.repository.ts   TS users.controller.ts 4   TS users.service.ts 1   TS sections.controller.ts   TS tasks.controller.ts   TS tasks.service.ts   tasks
src > modules > tasks > tests > tasks.service.spec.ts > describe('TasksService') callback
28  describe('TasksService', () => {
104   describe('FindAllRepeatable', () => {
113   });
114   describe('CreateNewTask', () => {
115     it('should create new daily and repeatable task', async () => {
116       const saveDailyTasksFn = jest.spyOn(service['tasksHelperService'], 'saveDailyTasks');
117       const saveRepeatableTasksFn = jest.spyOn(service['tasksHelperService'], 'saveRepeatableTasks');
118
119       const isCreated = await service.create(createTaskMockDto);
120
121       expect(saveDailyTasksFn).toBeCalled();
122       expect(saveRepeatableTasksFn).toBeCalled();
123       expect(isCreated).toBeTruthy();
124     });
125     it('should create new tasks for current day without repeatable tasks', async () => {
126       const saveDailyTasksFn = jest.spyOn(service['tasksHelperService'], 'saveDailyTasks');
127
128       const upsertTaskDto = {
129         ...createTaskMockDto,
130         startDate: dateNowFormatted(),
131         repeatDaysInWeek: [],
132       } as CreateTaskDto;
133       const savedTasks: Task[] = [
134         {
135           ...savedTasksMock[0],
136           repeatableTaskId: '',
137           isRepeatable: false,
138           repeatDaysInWeek: undefined,
139           endDate: undefined,
140         },
141       ];
142
143       const isCreated = await service.create(upsertTaskDto);
144
145       expect(saveDailyTasksFn).toBeCalledWith(savedTasks);
146       expect(isCreated).toBeTruthy();
147     });
148     it('should create new repeatable tasks', async () => {
149       const saveRepeatableTasksFn = jest.spyOn(service['tasksHelperService'], 'saveRepeatableTasks');
150
151       const startDate = '2023-03-04';
152       const endDate = '2023-03-10';
153       const upsertTaskDto = { ...createTaskMockDto, startDate, endDate } as CreateTaskDto;
154
155       const isCreated = await service.create(upsertTaskDto);
156
157       const savedRepeatableTask = { ...savedRepeatableTasksMock[0], startDate: '2023-03-05', endDate };
158
159       expect(saveRepeatableTasksFn).toBeCalledWith(startDate, [savedRepeatableTask], false);
160       expect(isCreated).toBeTruthy();
161     });
162   });
163   describe('UpdateTask', () => {
```

```
160        expect(isCreated).toBeTruthy();
161      });
162    });
163    describe('UpdateTask', () => {
164      it('should update daily task', async () => {
165        const updateTodayTaskFn = jest.spyOn(service['updateTaskService'], 'updateTodayTask');
166        const isUpdated = await service.update(updateTaskMockDto);
167        expect(updateTodayTaskFn).toBeCalled();
168        expect(isUpdated).toBeTruthy();
169      });
170      it('should update repeatable task', async () => {
171        const updateRepeatableTaskFn = jest.spyOn(service['updateTaskService'], 'updateRepeatableTask');
172
173        const taskDto = { ...updateTaskMockDto, date: addDaysWithFormat(dateNowFormatted(), 1) };
174
175        const isUpdated = await service.update(taskDto);
176        expect(updateRepeatableTaskFn).toBeCalled();
177        expect(isUpdated).toBeTruthy();
178      });
179    });
180
181    describe('DeleteTask', () => {
182      it('should delete daily task', async () => {
183        const deleteTodayTaskFn = jest.spyOn(service['deleteTaskService'], 'deleteTodayTask').mockResolvedValue();
184        const deleteTaskDto = {
185          id: savedTasksMock[0].id,
186          date: dateNowFormatted(),
187          allEvents: false,
188        } as DeleteTaskDto;
189        const isDeleted = await service.delete(deleteTaskDto);
190        expect(deleteTodayTaskFn).toBeCalled();
191        expect(isDeleted).toBeTruthy();
192      });
193      it('should delete repeatable task', async () => {
194        const deleteRepeatableTaskFn = jest
195          .spyOn(service['deleteTaskService'], 'deleteRepeatableTask')
196          .mockResolvedValue();
197        const deleteTaskDto = {
198          id: savedTasksMock[0].id,
199          date: addDaysWithFormat(new Date(), 3),
200          allEvents: false,
201        } as DeleteTaskDto;
202        const isDeleted = await service.delete(deleteTaskDto);
203        expect(deleteRepeatableTaskFn).toBeCalled();
204        expect(isDeleted).toBeTruthy();
```

Ln 180, Col 3 Spaces: 2 UTF-8 LF {}

Task Service Test Cases:

A simple test checks if the service is defined, which ensures that the NestJS framework correctly initializes the service with all its dependencies.

Find Operations:

- findDailyByDate: Tests the service's ability to retrieve tasks based on a specific date. It checks two scenarios:
- Retrieving tasks for a date that is not today, expecting a straightforward retrieval.
- Retrieving tasks for today's date, expecting a combination of tasks including repeatable tasks marked accordingly.
- findRepeatableByDate: Tests retrieval of repeatable tasks by a specific date, ensuring the service can filter tasks that are meant to repeat on given days.

Create Operation:

- Tests creating new tasks both for a specific date and for repeatable schedules. It checks that:
- Daily tasks are saved when provided with a date.
- Repeatable tasks are created when applicable.
- No repeatable tasks are created if not specified.

Update Operations:

- update: Tests updating tasks for both daily and repeatable tasks, ensuring the correct service methods are called based on the task's date context.
- updateStatus: Focuses on updating the status of a task, handling different task statuses and ensuring the operation fails gracefully with proper exceptions when a task cannot be updated due to logical constraints (e.g., updating a completed task).

Delete Operations:

Tests the deletion of tasks, both daily and repeatable. It verifies that the appropriate service functions are called and handle task deletions correctly.

Testing Techniques

Mocking and Spying:

- External interactions, like database access, are mocked using Jest functions (jest.fn()), allowing tests to verify interactions without actually performing I/O operations.
- jest.spyOn() is used to spy on the calls to these mocked functions, which helps in asserting that the service methods interact correctly with their dependencies.

Behaviour Verification:

The tests check not only the return values of the service methods but also the side effects, such as the correct handling of database operations and the proper emission of events.

Error Handling:

The service's resilience is tested by simulating conditions where operations fail, such as not finding a task or trying to perform invalid updates. This ensures that the service can gracefully handle errors and respond with appropriate exceptions.

2.6. Evaluation

The evaluation of the Task Management Web Application focuses on assessing the system against its initial objectives and user requirements outlined in the design phase. This section presents the findings from various evaluation methods including user feedback, performance metrics, and functional adequacy.

User Satisfaction and Feedback

Surveys and interviews with beta testers provided invaluable insights into user satisfaction. The majority of users appreciated the simplicity and responsiveness of the interface but some reported a desire for more customizable features such as theme changes. Feedback on real-time updates and notification systems was overwhelmingly positive, emphasizing the application's effectiveness in enhancing task management.

Performance Metrics

Performance evaluation involved measuring the response times for task operations and the load times of various pages. Metrics collected indicated that task retrieval and display were efficient, even under load. However, the task creation process showed a slight delay when simultaneous requests were made, which was subsequently optimized.

Functional Adequacy

Functionality tests verified that all specified requirements were met, including task creation, editing, deletion, and filtering. The drag-and-drop functionality for task management was particularly well-received, enhancing user engagement and productivity. Every use case defined in the project scope was tested to ensure complete coverage and functionality.

Accessibility Compliance

The application was tested against WCAG (Web Content Accessibility Guidelines) to ensure accessibility for users with disabilities. Tools like Axe and manual testing sessions identified some areas for improvement, such as colour contrast ratios and keyboard navigability, which were promptly addressed.

3.0 Conclusions

The development of the Task Management Web Application has been a journey of addressing the nuanced needs of personal task management through a robust, intuitive, and efficient platform. The application stands as a testament to the effectiveness of integrating cutting-edge technologies and user-cantered design principles in software development. The final product not only meets the initial specifications laid out at the project's inception but also provides a solid foundation for personal productivity enhancement.

Through rigorous testing and evaluation, the application demonstrated high performance, reliability, and user satisfaction. The integration of technologies such as Next.js for the frontend and Nest.js for the backend, complemented by real-time functionalities using WebSocket and data management using Redis, has ensured that the application is fast, responsive, and capable of handling real-time data efficiently. The application supports essential features such as task creation, editing, deletion, and filtering with both ease and speed, which are critical in a task management tool.

User feedback has been positive, highlighting the application’s user-friendly interface and the simplicity of navigating its features. This feedback is a crucial indicator of the application's success in achieving its aim of simplicity and ease of use. However, during the lifecycle of the project, several opportunities for further enhancements were identified, which could address some of the users' advanced needs and preferences that have evolved during the testing phase.

Moreover, the project’s approach to agile development allowed for continuous integration and deployment, which facilitated the timely delivery of features and quick adaptation to user feedback and testing outcomes. This methodology proved invaluable in maintaining high standards of quality and adaptability in the fast-paced environment of software development.

4.0 Further Development or Research

Looking ahead, the Task Management Web Application holds substantial potential for further development and exploration. The landscape of personal productivity tools is rapidly evolving, driven by advances in technology and changing user expectations. To stay ahead and make the application even more versatile and useful, several enhancements are proposed:

- **Artificial Intelligence and Machine Learning Integration:**

  Predictive Task Management: Implement AI algorithms to analyze users' task completion patterns and predict future tasks. This could help in automatically suggesting the best times and methods for tackling certain types of tasks.

- **Smart Suggestions:**

  AI could be used to offer smart suggestions for task prioritization and categorization based on the user's past activity and preferences.

- **Mobile Application Development:**

  Given the increasing reliance on mobile devices, developing a mobile version of the application could significantly boost its accessibility and usability. A mobile app would allow users to manage their tasks on the go, with features optimized for mobile use, including offline access and mobile notifications.

- **Enhanced Customization and Personalization:**

  User-Defined Themes and Layouts: Allowing users to customize the interface according to their personal preferences can significantly enhance user satisfaction and productivity.

- **Adaptive Interfaces:**

  The application could adapt its interface based on the user's behavior and preferred workflow, learning from their interactions to streamline task management processes.

- **Integration with Other Tools and Platforms:**

  Developing integrations with calendars, email clients, and other productivity tools could provide a unified platform for managing all aspects of personal organization. This would make the application a central hub for all personal productivity needs.

- **Advanced Analytics Dashboard:**

  Incorporating an analytics dashboard that provides insights into task performance, productivity trends, and potential bottlenecks could empower users to optimize their workflows more effectively.

- **Research on User Interaction and Task Management:**

  Conducting detailed user experience research to explore how different demographics manage tasks and incorporate feedback mechanisms to tailor the application more closely to varying needs and preferences.

5.0 References

- https://nextjs.org/docs
- https://docs.nestjs.com/
- https://redis.io/docs/latest/develop/
- https://jwt.io/introduction
- https://docs.nestjs.com/techniques/events
- https://docs.nestjs.com/techniques/queues
- https://dev.to/alfism1/build-complete-rest-api-feature-with-nest-js-using-prisma-and-postgresql-from-scratch-beginner-friendly-part-7-6me
- https://socket.io/docs/v4/

6.0 Appendices

6.1. Project Proposal

National College of Ireland

National College of Ireland

Project Proposal

Task Management Web Application

29th- October 2023

Bachelor of Science (Honours) in Computing

Software Development

Academic Year i.e., 2023/2024

Satyam Sehgal

X19104464

X19104464@student.ncirl.ie

Contents

1.0 Objectives 70
2.0 Background 70
3.0 State of the Art 70
4.0 Technical Approach 71
5.0 Technical Details 71
6.0 Special Resources Required 71
7.0 Project Plan 72
8.0 Testing 73

Objectives

The main goal of this project is to make a user-friendly **Task Management Web Application** that helps people keep track of their day-to-day tasks. It will be easy to use and meant mainly for personal use. The website will let users make, sort, and organize their tasks easily. It will also show updates in real time, which will help users get things done faster. Additionally, for automation calendar sync which will be an Integration for Calendar Apps, Task prioritization Algorithms, and Automation of Recurring Tasks.

Background

I wanted to create a tool to help people manage their tasks. Most tools out there are made for teams and work life, which can be confusing for individuals. So, I decided to make a simple and easy-to-use tool just for personal use with a vast number of features. I will pay a lot of attention to making it user-friendly and adding features like real-time updates to make sure it meets the goals mentioned in Section 1.0.

State of the Art

Similar task management applications which are popular ones such as Trello, Asana, and Todoist currently dominate the market. However, these platforms are mainly made for teams and might have more features than one person needs. My app is just for individuals, and it's really easy to use. Plus, it lets you see updates in real time, which makes it special and gives you a more interactive experience.

Technical Approach

I'll use an Agile approach for development, which means I'll stress talking often and making progress bit by bit. At first, I'll dig deep into existing task apps to understand them better. I'll learn from the best practices and what users like. This will guide me in setting specific requirements and then splitting these requirements into smaller tasks, activities, and goals. I'll plan regularly to make sure the project steadily moves forward. This approach lets me adjust easily if new info or changes pop up. I'll use version control to make sure the code stays reliable, to work together with others on development, and to keep track of any changes easily.

Technical Details

I will use JavaScript to build the web app's front part. To make it interactive and user-friendly, I'll use a tool called React. For the back part, we'll use Node.js with Express to handle things efficiently on the server side. To manage data effectively, we'll use a powerful system called PostgreSQL/ MongoDB for the database. Additionally, we'll use WebSocket technology to provide updates in real-time.

Special Resources Required

The project won't need any special tools, just the usual ones for development purposes. Already have access to the necessary software, hardware, and development environment.

Project Plan

The project plan includes the following milestones:

1. **Requirements Definition, Analysis, and Task Decomposition**

- Define project requirements based on outlined objectives.
- Analyze gathered data to define detailed requirements.
- Break down features into development tasks.

2. **Design and Wireframing**

- Create wireframes and mock-ups based on gathered requirements.
- Review and finalize UI/UX designs.

3. **Backend Development**

- Set up backend infrastructure using Node.js and Express.
- Implement database functionality using PostgreSQL/MongoDB.

4. **Frontend Development**

- Develop interactive UI components using React/ HTML/ CSS/ JavaScript
- Integrate frontend with backend using APIs.

5. **Real-time Updates Implementation / Automation**

- Incorporate WebSocket technology for real-time task updates.

6. **Testing and Quality Assurance**

- Conduct unit, integration, and user acceptance testing.
- Address and resolve any identified issues.

7. **Deployment and User Training**

- Deploy the web app on a hosting platform (e.g., AWS, Heroku, or any other hosting platform).
- Provide user documentation and training resources.

8. **Improving the Project Plan with feedback from the Project Supervisor**

- Gathering, and incorporating feedback in the project plan.
- Refining and Updating the Project Plan
- Communicating the Revised Plan.

This project plan will be further clarified with detailed tasks in the mid-point documentation.

Testing

I will set up a thorough testing plan to make sure the Task Management Web App works well and is dependable.

System Tests:

- **Functional Testing:** I'll check if all the features work properly, like making, changing, sorting tasks, setting priority, automation, and getting updates in real time.
- **Performance Testing:** I'll see how fast and stable the system is when different numbers of people are using it to make sure it works well even when many people use it at once.
- **Security Testing:** I'll test the app to find and fix any security problems just to make sure that it's safe to use.

Integration Tests:

- Backend-frontend Integration: I'll make sure that the front part of the app and the back part work together smoothly and that the information stays correct and the app works well.

User Acceptance Testing (UAT):

- I'll ask the testers who will use the app to try it out and tell me what they think about how easy it is to use and what their experience is like.
- I'll listen to their suggestions on how to make it better or fix any issues.

Doing all these tests will help me understand how well the app works, how fast it is, and if people like using it. If I find any problems, I'll fix them before finishing up the final deployment.

1.1. Reflective Journals

November

2.0 Supervision & Reflection Template

Student Name	Satyam Sehgal
Student Number	X19104464
Course	Bachelor of Science (Honours) in Computing
Supervisor	Adriana Chis

3.0

4.0 Month:

What?

Reflect on what has happened in your project this month?

This month I spent a lot of time on my project. My main goal was to design the user interface, for the task app management. I made plans for how the app will look and work, to make it easy for users to use and show which pages will lead to another. I also started building the front part of the app. I turned my starting plans into code and added basic functionality for my app. Furthermore, I met with my supervisor to discuss the next steps for my project and also had discussion about my core functionality of my project.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

So, reflecting on the progress of my task management web app, I've achieved some success and faced several challenges. As follows:

UI Design: I focused on creating an appealing and user-friendly interface. This involved designing the look and user experience (UX) while generating creative ideas for the UI. A visually attractive interface sets a strong foundation for the overall user experience.

Frontend Development: I worked on converting the UX design into code, implementing it in the frontend of the application.

However, challenges still remain. I am currently figuring out what gonna be complex and unique features of my application that will set it apart from other available products.

Now What? What can you do to address outstanding challenges? Doing intensive research about other products and come up something complex functionality for my web application.

Student Signature	

December

5.0 Supervision & Reflection Template

Student Name	Satyam Sehgal
Student Number	X19104464
Course	Bachelor of Science (Honours) in Computing
Supervisor	Adriana Chis

6.0

7.0 Month:

What? Reflect on what has happened in your project this month? This month I was more focused on finding other ways to look upon basically on other ways to implement backend developments rather than using node js.
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? Upgrading UI Structure and using a different kind css structure (tailwind css) which is more robust and easier to manage

However, challenges still remain. I am currently figuring out how to add and use the bull queues and rest api's	
Now What? Researching for other ways	
Student Signature	

March

8.0 Supervision & Reflection Template

Student Name	Satyam Sehgal
Student Number	X19104464
Course	Bachelor of Science (Honours) in Computing
Supervisor	Adriana Chis

9.0

10.0 Month:

What? Reflect on what has happened in your project this month? Finalizing the project database using nest.js and redis, used a technology called websocket to keep frontend stuff happening in real time. Implementation was done for rest api, jwt and swagger but still more thing was done to make the development process manageable.
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? Finish up the project itself and testing before moving on to project documentation
Now What? Learning & practicing ways to deploy this pending stuff.

Student Signature	

April

11.0 Supervision & Reflection Template

Student Name	Satyam Sehgal
Student Number	X19104464
Course	Bachelor of Science (Honours) in Computing
Supervisor	Adriana Chis

12.0

13.0 Month:

What? Reflect on what has happened in your project this month? Updated my project technical report up to the date and filled/tested out the testing part conducted with the unit testing • Task service test cases • Mocking and spying	
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? Managed to implement working database using next.js and swagger	
Now What? Finishing touches to the new ui and updating documentation as it is	
Student Signature	

Signature