Software Requirements Specification

for

Deadline Management System

Version 1.0 approved

Prepared by Yeshwanthraj G

Coimbatore Institute of Technology

07-10-2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document describes the functional and non-functional requirements for the Deadline Management System, version 1.0.

Clockyn - A **Deadline Management System** is a secure, web-based platform designed to help organizations and individuals manage, track, and meet deadlines for projects, tasks, and events. It enables users to create deadlines, assign tasks, receive automated reminders, and monitor completion status.

The purpose of this document is to define the scope and expectations of the system and to serve as a reference for developers, testers, project managers, and all stakeholders involved in deadline tracking and compliance.

This SRS focuses on the core features of deadline management and does not include extended features such as integration with external calendars or third-party productivity tools, which may be considered for future releases.

1.2 Document Conventions

- Bold text is used to highlight important headings and terminology.
- Links to future enhancements are presented in italics.
- Placeholder tags are included where further content may be added or refined.
- Requirements are labeled with unique identifiers for easy traceability.
- Priority levels are clearly defined for each requirement. There is no assumption that child requirements inherit priorities from parent ones
- All diagrams follow UML Unified Modeling Language standards and will be presented where appropriate

1.3 Intended Audience and Reading Suggestions

This document is intended for the following audiences:

- **Project managers**: To plan and monitor deadline driven activities.
- **Developers**: To implement and maintain the system based on stated requirements.
- **Testers**: To validate system functionalities and ensure compliance.
- **End Users** (team members, individuals): To understand the platform's capabilities related to deadline management and compliance.

Suggested Reading Sequence:

1. Begin with Section 1: Introduction, for an overview of the project.
2. Proceed to Section 2: Overall Description, to understand the system's context and environment.
3. Move to Section 3: Functional Requirements, for detailed software functionality.

4. Refer to Section 4: Non-Functional Requirements, for information on performance, security, and usability.
5. Use the Appendices for diagrams, glossary, and reference materials.

1.4 Product Scope

Clockyn is intended to digitize and streamline the traditionally manual process of setting, monitoring, and meeting deadlines across organizations. It serves both individuals and administrators through a centralized web interface.

Key system objectives:

- Enable users to set deadlines for various tasks and projects.
- Allow administrators to oversee deadlines, monitor compliance, and generate comprehensive reports.
- Maintain records of past deadlines and completion status.
- Ensure secure handling of sensitive project and user data.
- Generate progress status updates, and analytical reports.

This system supports a paperless, transparent, and efficient approach to deadline tracking, aligning with organizational digital transformation goals

1.5 References

The following documents and sources were consulted in preparing this SRS for the Deadline Management System:

- IEEE 830-1998 SRS Documentation Standard
- Industry best practices in project and time management
- UML Diagrams and Notation Guidelines
- Security guidelines for organizational data management
- Personal experience and standard workflow patterns in modern organizations

2. Overall Description

2.1 Product Perspective

Clockyn - An Intelligent Deadline Management System is a modern, cloud-ready web application built using the MERN stack. It is designed as a centralized platform for individuals and teams to efficiently track, manage, and meet deadlines within projects or personal schedules. The system integrates seamlessly with existing workflows and is accessible through standard web browsers, supporting dynamic, real-time interaction across devices. The application operates as an independent module but supports future integration with third-party productivity tools and calendar services through RESTful APIs.

2.2 Product Functions

Key functions of the Deadline Management System include:

- User registration and authentication.
- Task and deadline creation, editing, and deletion.
- Visual dashboards to display pending, in-progress, completed, and missed deadlines.
- Progress tracking and completion status updates.
- Advanced filtering for tasks and deadlines.
- Dynamic Motivational Quotes based on Task Completion Rate.
- *Planned integration with calendar apps and team collaboration tools (future enhancement)*

All requirements are labeled as **REQ-n** and prioritized as High, Medium, or Low for traceability.

2.3 User Classes and Characteristics

- **Administrator:** Oversees all system data, manages users, and runs analytics. Requires advanced digital literacy.
- **User:** The primary user who creates, edits, and completes deadlines/tasks. Requires basic computer skills.
- **Collaborator:** Can be assigned to shared tasks, can view and update progress for collaborative deadlines. Needs familiarity with team collaboration features.

2.4 Operating Environment

- **Hardware**: Standard desktop or laptop computer with at least 4GB RAM and 2GHz processor, mobile devices for limited access.
- **Software**
  - **Client tier**: Browser-based React JS application compatible with Chrome, Firefox, Edge, and Safari on Windows, Linux, macOS.
  - **Server tier**: Node.js (Express.js) backend API.
  - **Database**: MongoDB (NoSQL), cloud- or local-hosted.
  - **Protocols**: All server-client communication operates over secure HTTPS.
  - **Deployment**: Supports cloud, on-premises, or hybrid deployment as per organizational policy

2.5 Design and Implementation Constraints

- Application must follow secure authentication and authorization standards.
- Database schema should be scalable for adding modules (e.g., advanced analytics or notifications).
- Only web browser access is supported; no native mobile or desktop app included in current scope.
- React and Node version compatibility must be ensured.
- Deployment must support scalability for up to 1,000 concurrent users.
- Follows responsive web design for accessibility on desktops, laptops, and tablets.

2.6 User Documentation

The system will include the following user documentation

- **User Guide**: Instructions for account setup, deadline creation, and dashboard usage.
- **Administrator Manual**: Managing users, deadlines, and generating reports.
- **Online Help/FAQ**: Contextual help integrated within the application

2.7 Assumptions & Dependencies

- Users have internet connectivity and modern web browsers.
- Organization provides infrastructure for hosting database and backend.
- Email service or push notification provider is available for sending reminders.
- Future enhancements may depend on integration APIs for calendar or collaboration tools.

3. External Interface Requirements

3.1 User Interfaces

The system will provide a web-based user interface accessible through standard browsers.

- **Login Page**: Secure entry point for users, collaborators, and administrators with form fields for email/username and password.
- **Register Page**: Allows new users to sign up by providing personal details and assigning a user role.
- **Dashboard**: Personalized summary displaying upcoming, overdue, and completed deadlines. Users can view, create, or update tasks and deadlines.
- **Task/Deadline Management Screens**: Enable creation, editing, and deletion of tasks with due dates, responsible users, and tagging.
- **Responsive Design**: All UI elements adapt for usability across desktop and tablet browsers.

The interface will be designed for ease of use with consistent navigation, buttons, and forms.

3.2 Hardware Interfaces

- Standard desktop or laptop computers running modern browsers.
- Tablets for on-the-go management (optional).
- No special hardware requirements; relies only on web access and optional peripherals (keyboard, mouse).

3.3 Software Interfaces

- **Operating System**: Compatible with Windows, macOS, Linux.
- **Web Browser**: Supports latest versions of Chrome, Firefox, Edge, and Safari.
- **Backend**: REST API endpoints served by Express.js (Node.js), using HTTPS.
- **Database**: MongoDB for all structured and unstructured data storage.
- **Authentication/Authorization**: JWT-based user authentication system.
- *Optional future APIs for integration with external calendar, notification, or productivity tools.*

3.4 Software Interfaces

- HTTPS protocol for all client-server data exchanges.
- Email service or third-party provider for sending deadline reminders and notifications.
- API gateway for secure, authenticated future integrations.
- WebSocket support for real-time notifications and updates (optional, for scalability).

4. System Features

4.1 User Authentication and Authorization

- **Description:** The system provides secure login and registration for users, collaborators, and administrators with role-based access control.
- **Inputs:** User credentials - username/email and password.
- **Outputs:** Authenticated user session with access rights assigned based on role.
- **Preconditions:** User must be registered in the system.
- **Postconditions:** User session is initiated; dashboard access is granted according to role.
- **Normal Flow:**
    1. User navigates to the login page.
    2. Enters valid credentials.
    3. System verifies credentials against the database.
    4. System creates a session and redirects to role-specific dashboard.
- **Alternative Flow:** On invalid login, an error message is shown, and reattempt is allowed.

4.2 Task and Deadline Management

- **Description:** Users can create, edit, assign, and delete tasks and deadlines.
- **Inputs:** Task details including title, description, due date, assignees, and priority.
- **Outputs:** Updated task list reflecting changes.
- **Preconditions:** User must be authenticated.
- **Postconditions:** Task is stored or updated in the database and visible in user dashboards.
- **Normal Flow:**
    1. User accesses task management interface.

2. Inputs task details or selects existing task.
3. System validates and saves the data.
4. Updates are reflected in dashboards and notifications.

4.3 Progress Tracking and Status Updates

- **Description:** Users mark tasks as in-progress, completed, or delayed, and track overall deadline compliance.
- **Inputs:** Status changes on tasks and deadlines.
- **Outputs:** Updated status displayed in the dashboard.
- **Preconditions:** Tasks exist and are assigned.
- **Postconditions:** Status changes are persisted and visible to relevant users.
- **Normal Flow:**
    1. User updates task status.
    2. System records the change with timestamps.
    3. Dashboards update to show current progress.

4.4 Administrative Reporting

- **Description:** The system generates reports on tasks, deadlines, and user compliance for administrators.
- **Inputs:** Report criteria such as date range, project, or user filters.
- **Outputs:** Dashboard Overview (Progress Track).
- **Preconditions:** Administrative access is required.
- **Postconditions:** Report generation logs are recorded.
- **Normal Flow:**
    1. Admin selects report parameters.
    2. System compiles data from the database.

4.5 Use Cases

This section describes simplified use cases of the Deadline Management System for SRS documentation. These use cases reflect the key interactions between users (students and admins) and the system to demonstrate core functionality.

UC 1: Create Task

- Primary Actor: User
- Secondary Actor: System (Admin)
- Preconditions: User is authenticated and authorized to create tasks.
- Trigger: User selects the "Create Task" option.
- Main Success Scenario
    1. User inputs task details (title, description, due date, priority, etc.).
    2. System validates inputs.

3. Task is stored in the database.
4. System displays confirmation.

- Exceptions
  1. Invalid/missing input - system prompts for correction.
  2. Database failure - user is notified of the error.

UC 2: Edit Task

- Primary Actor: Task Owner(User).
- Secondary Actor: Admin
- Preconditions: Task exists; user has permission to edit.
- Trigger: User selects a task and clicks "Edit".
- Main Success Scenario
  1. System displays editable fields.
  2. User modifies desired fields.
  3. System validates and saves changes.
  4. Confirmation is shown.
- Exceptions
  1. Task not found - user is notified.

UC 3: Delete Task

- Primary Actor: User
- Secondary Actor: System(Admin)
- Preconditions: Task exists; user is the owner.
- Trigger: User clicks "Delete" on a task.
- Main Success Scenario
  1. System prompts for confirmation.
  2. User confirms deletion.
  3. Task is deleted from the database.
  4. System notifies collaborators (if any).
- Exceptions
  1. User cancels deletion - task remains.

UC 4: View Task

- Primary Actor: User
- Secondary Actor: Admin
- Preconditions: User has access to the task.
- Trigger: User navigates to the task list or selects a task.
- Main Success Scenario
  1. System fetches task details.
  2. Task is displayed with all information (status, priority, assignees).

- Exceptions
  1. Task doesn't exist or access denied - error message.

UC 5: Manage Notifications

- Primary Actor: User(Task Admin)
- Secondary Actor: System
- Preconditions: User has an account and at least one task
- Main Success Scenario
  1. System sends notification on task events (create,delete,edit,complete).
- Exceptions
  1. Email delivery failure - retry mechanism or log error.
  2. Blocked by System - In-app messages to notify the cause to the user.

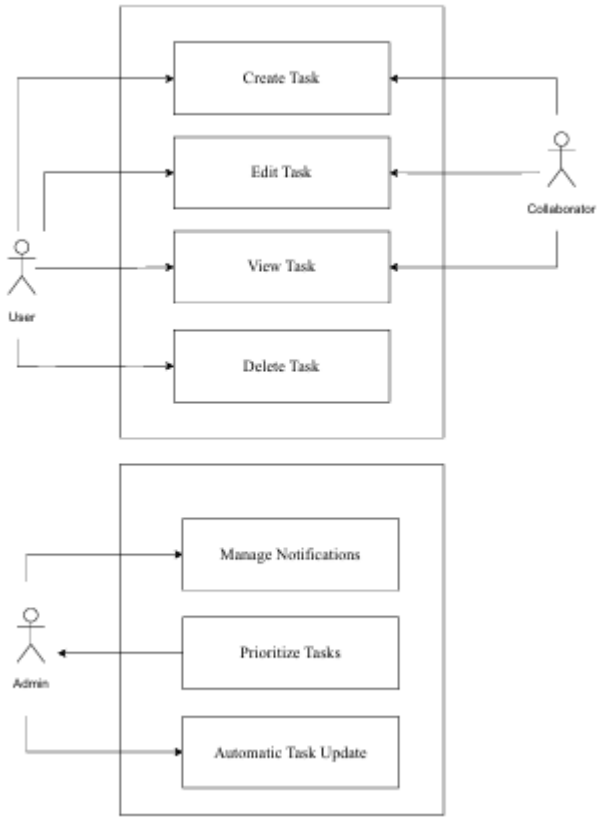
UC 6: Prioritize Task

- Primary Actor: Task admin
- Secondary Actor: System
- Preconditions: Task list is accessible.
- Trigger: User opens priority settings or task list view.
- Main Success Scenario
  1. User selects a task to reprioritize.
  2. System allows reordering or changing priority level.
  3. System saves new order/priority.
- Exceptions
  1. Invalid priority level - system rejects.
  2. Access denied - action blocked.

UC 7: Automatic Task Update

- Primary Actor: Admin
- Secondary Actor: Task User
- Preconditions: Task has a start and end time; current time matches condition.
- Trigger: Time reaches task start date or conditions change.
- Main Success Scenario
  1. System detects state change trigger (e.g., current time = task start time).
  2. Task status is automatically updated (e.g., "future" -> "active").
  3. Notification is sent to stakeholders.
- Exceptions
  1. Conflicting manual edits - notify user of conflict.

Use Case Diagram



5. Other Non-Functional Requirements

5.1 Performance Requirements

- The system shall process task creation, updates, and retrieval requests within 3 seconds under normal load.
- The system shall support at least 500 simultaneous active users without performance degradation.
- The database queries shall execute within 2 seconds for 10,000+ records.
- System downtime for maintenance shall not exceed 2 hours per month.

5.2 Safety Requirements

- The system shall maintain daily backups of all task and user data to prevent data loss.
- Data recovery mechanisms must restore system state within 4 hours after a failure.
- Only administrators have rights to modify user roles and permissions.
- Operation failures such as interrupted uploads must not corrupt existing data.

5.3 Security Requirements

- All communications between client and server shall use HTTPS encryption.
- Passwords must be securely hashed and salted before storage.
- Role-based access control (RBAC) shall be implemented for users, collaborators, and administrators.
- The system shall log all login attempts and critical actions for audit purposes.
- Uploaded files shall be scanned for malware before processing.

5.4 Software Quality Attributes

- **Usability**: The user interface shall be intuitive, requiring minimal training.
- **Reliability**: The system shall achieve 99.9% uptime availability during operational hours.
- **Scalability**: The architecture shall support scaling to accommodate more users and integrations.
- **Maintainability**: The codebase shall be modular and documented for easier updates and bug fixes.
- **Portability**: The application shall be compatible with major browsers such as Chrome, Firefox, Edge, and Safari on Windows, macOS, and Linux.

5.5 Business Rules

- Only registered users with valid roles may create and manage deadlines.
- Each task deadline must have a unique identifier and cannot overlap with critical system maintenance windows.
- Administrators hold the authority to approve, modify, or delete tasks as needed.
- Reminder schedules are configurable but must adhere to organizational policies.

6. Other Requirements

6.1 Database Requirements

- The system shall use MongoDB with a fully normalized schema (3NF or higher) to ensure data integrity and avoid redundancy.
- Daily backups shall be implemented with a disaster recovery plan to prevent data loss.
- The database must support efficient queries for tasks, deadlines, user management, notifications, and reports.

6.2 Internationalization Requirements

- English shall be the primary language supported.
- The design architecture shall allow future addition of multiple languages without major code refactoring.

6.3 Legal Requirements

- The system shall comply with applicable data protection laws relevant to personal and organizational data.
- User consent must be obtained prior to storing or processing personal data.
- Copyright and licensing rules shall be followed for any shared materials or uploaded content.

6.4 Reuse Objectives

- Core modules such as authentication, task management, notification, and reporting shall be developed modularly to support reuse in future projects
- The architecture shall support easy extension or integration with other systems (e.g., calendar apps or team collaboration tools).

6.5 Documentation Requirements

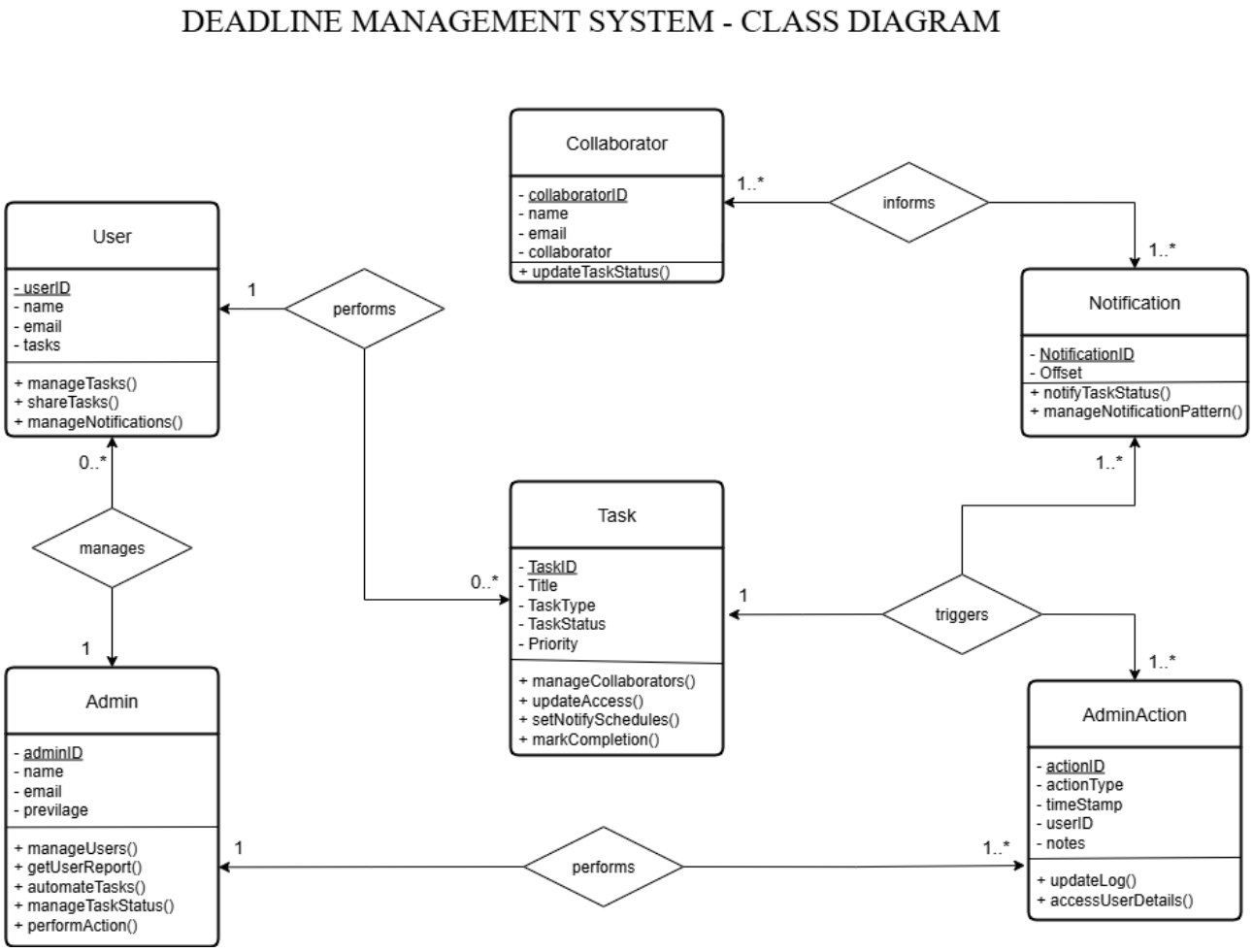
- User manuals shall be maintained for different user roles (administrator, user, collaborator).
- API documentation shall be provided with tools for backend endpoints.
- All documentation shall be updated with each major release, including tutorials, FAQs, troubleshooting guides, and workflow diagrams.

6.6 Future Enhancements

- Integration of AI-based recommendations for task prioritization and deadline management.
- Gamification features to encourage timely submissions and engagement.
- Offline access for web app users to track and update status.
- Integration with third-party Learning Management Systems (LMS) and productivity platforms for seamless workflow.

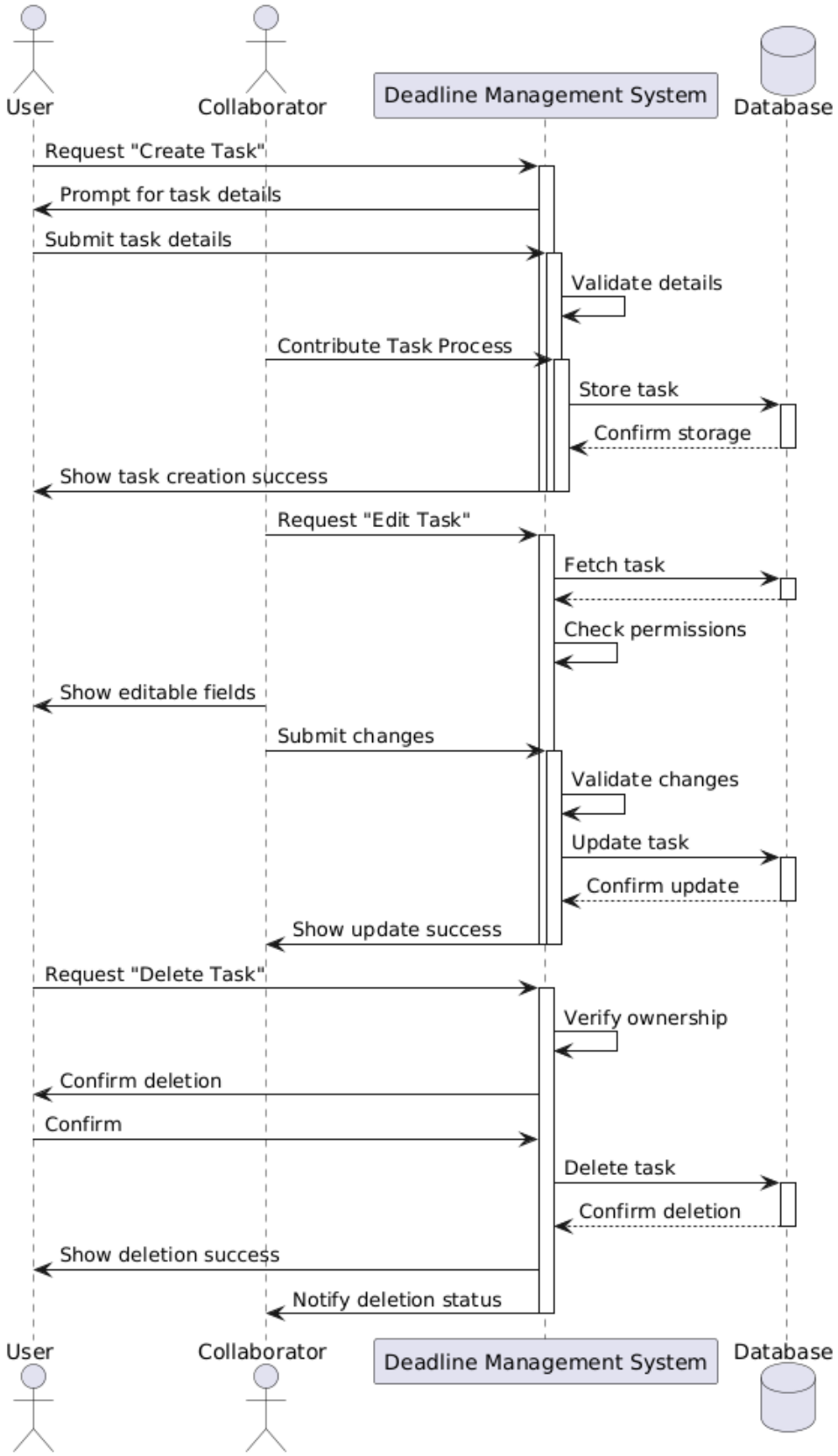
Appendix A: Analysis Models

CLASS DIAGRAM

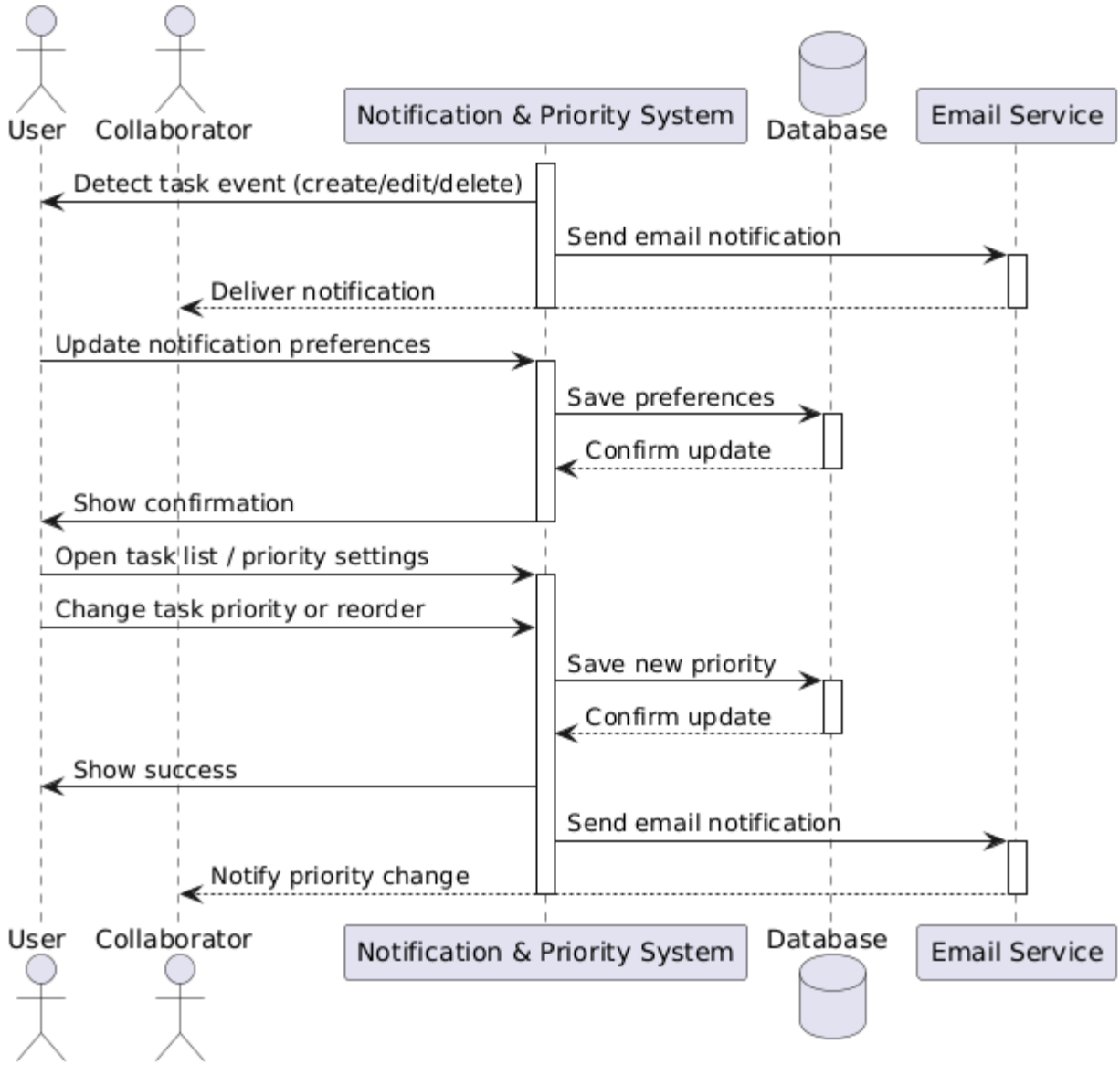


SEQUENCE DIAGRAM

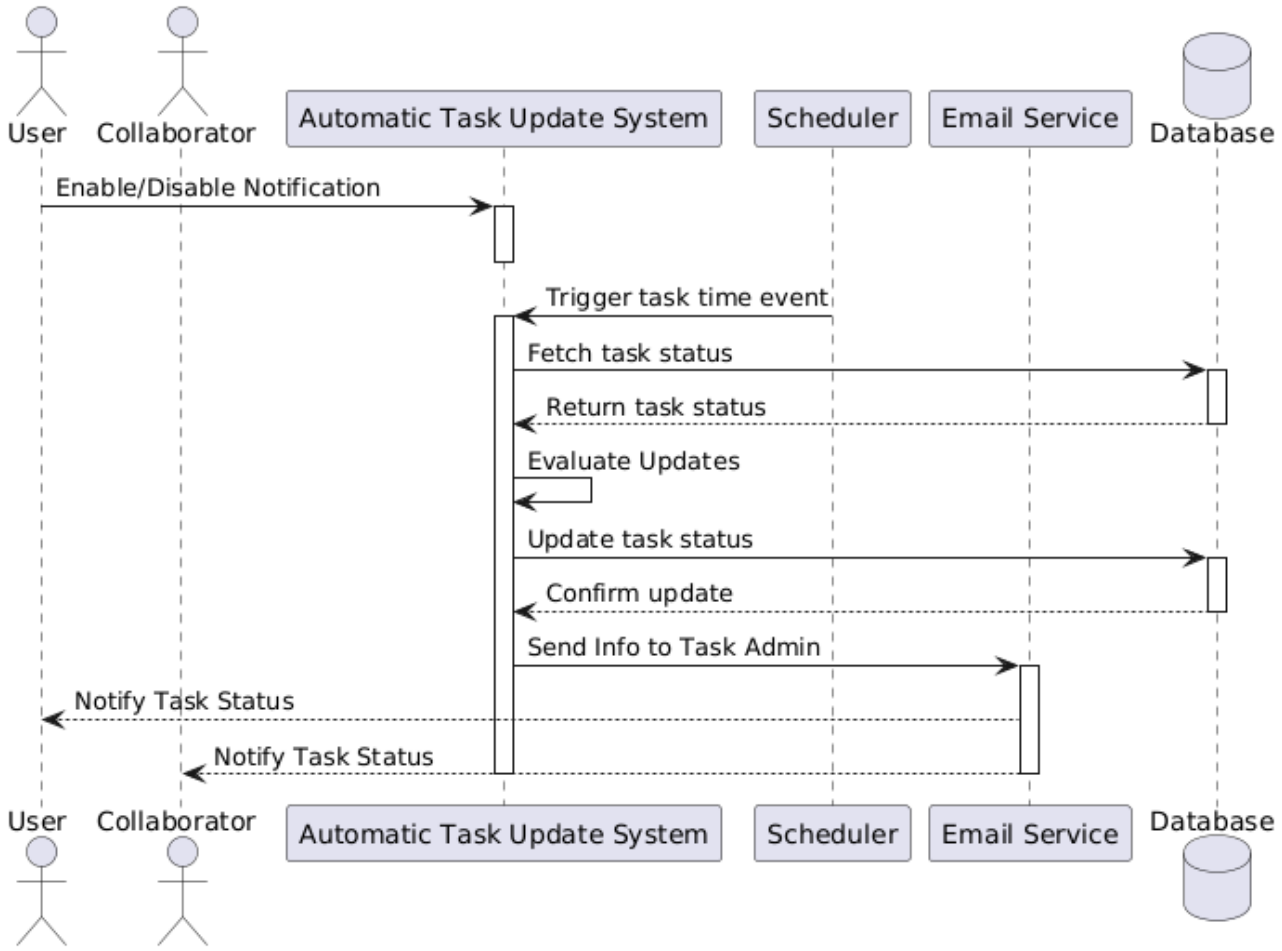
SEQUENCE DIAGRAM - 1



SEQUENCE DIAGRAM – 2



SEQUENCE DIAGRAM -3

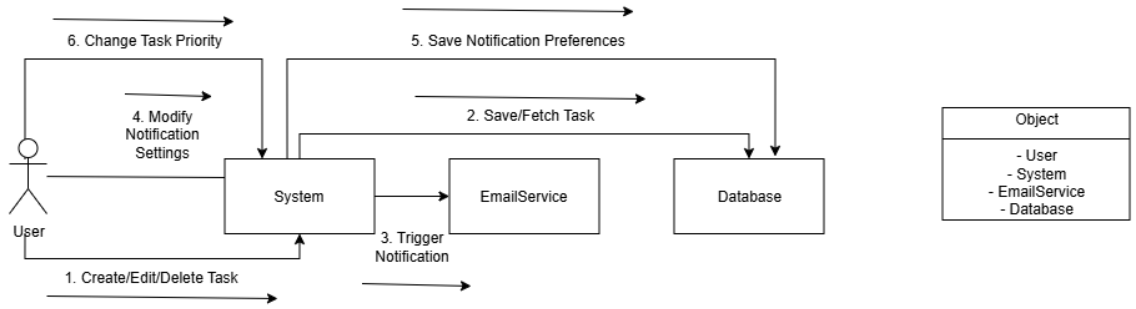


COLLABORATION DIAGRAM

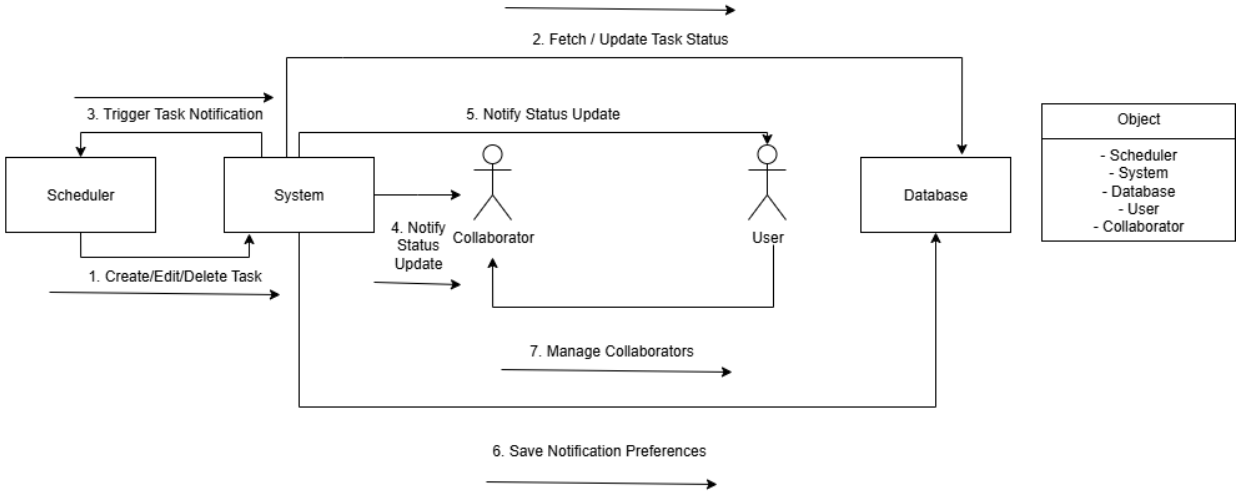
Collaboration Diagram for Deadline Management System

- Yeshwanthraj G

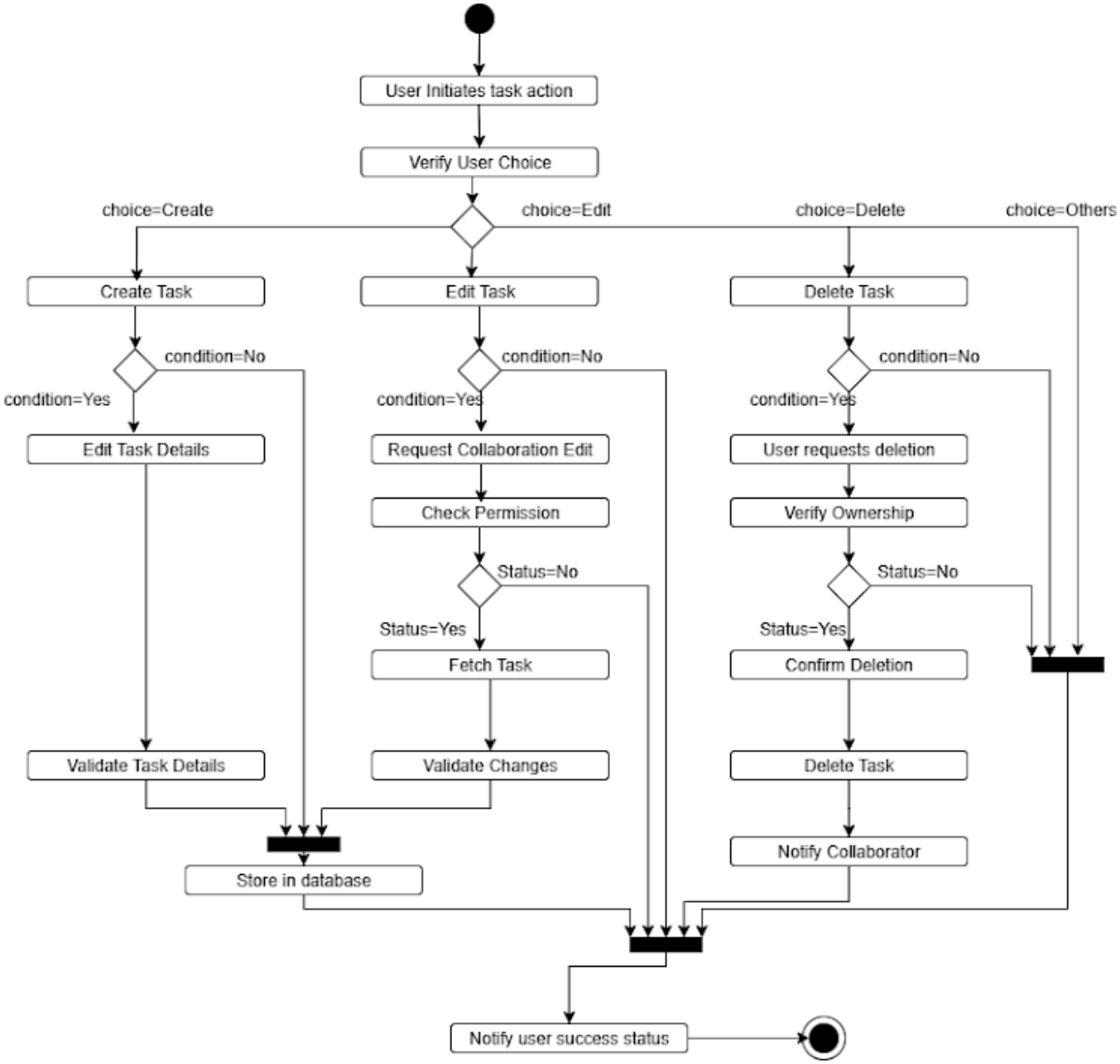
Task Creation and Management System



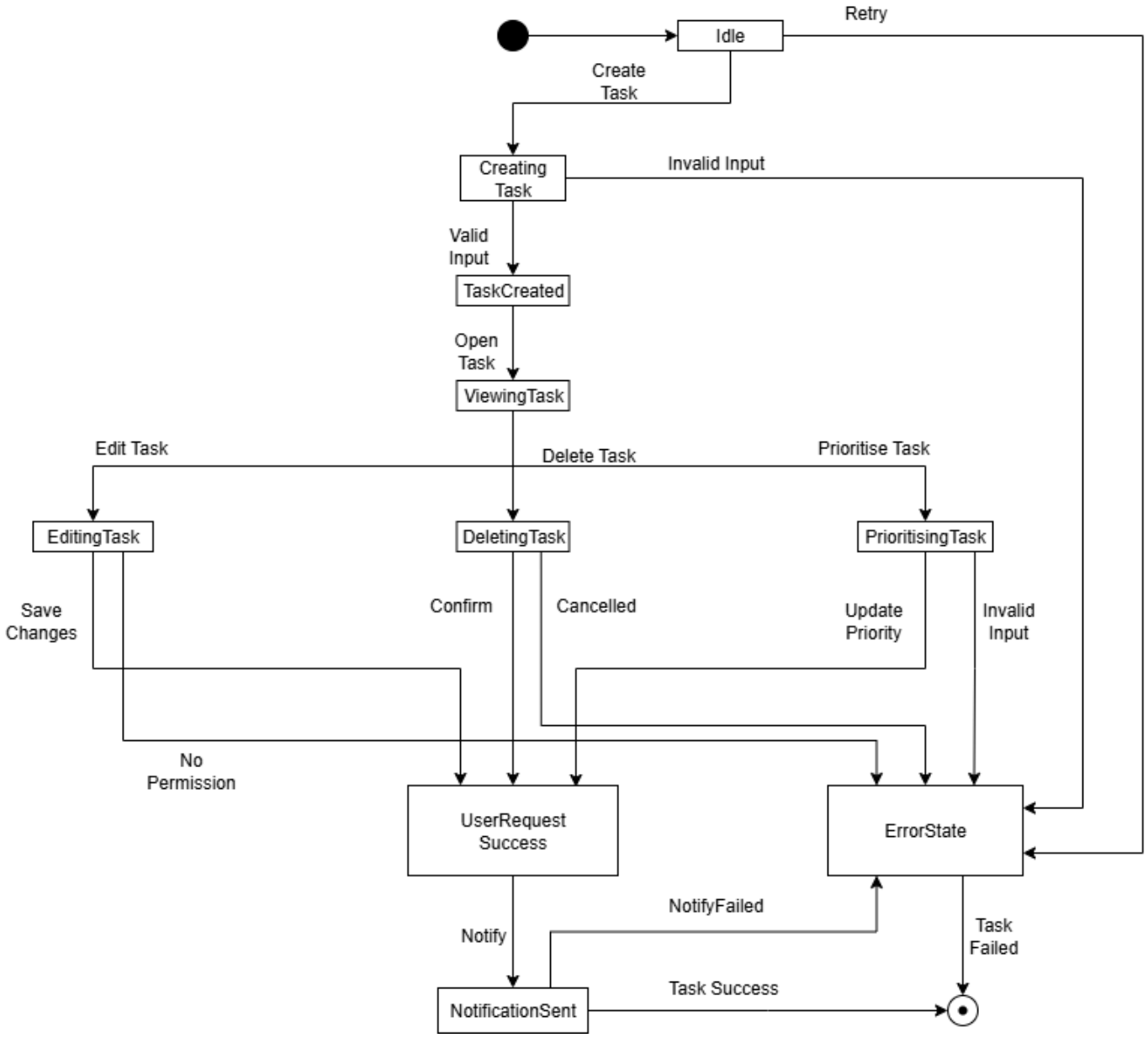
Automatic Notification Update System



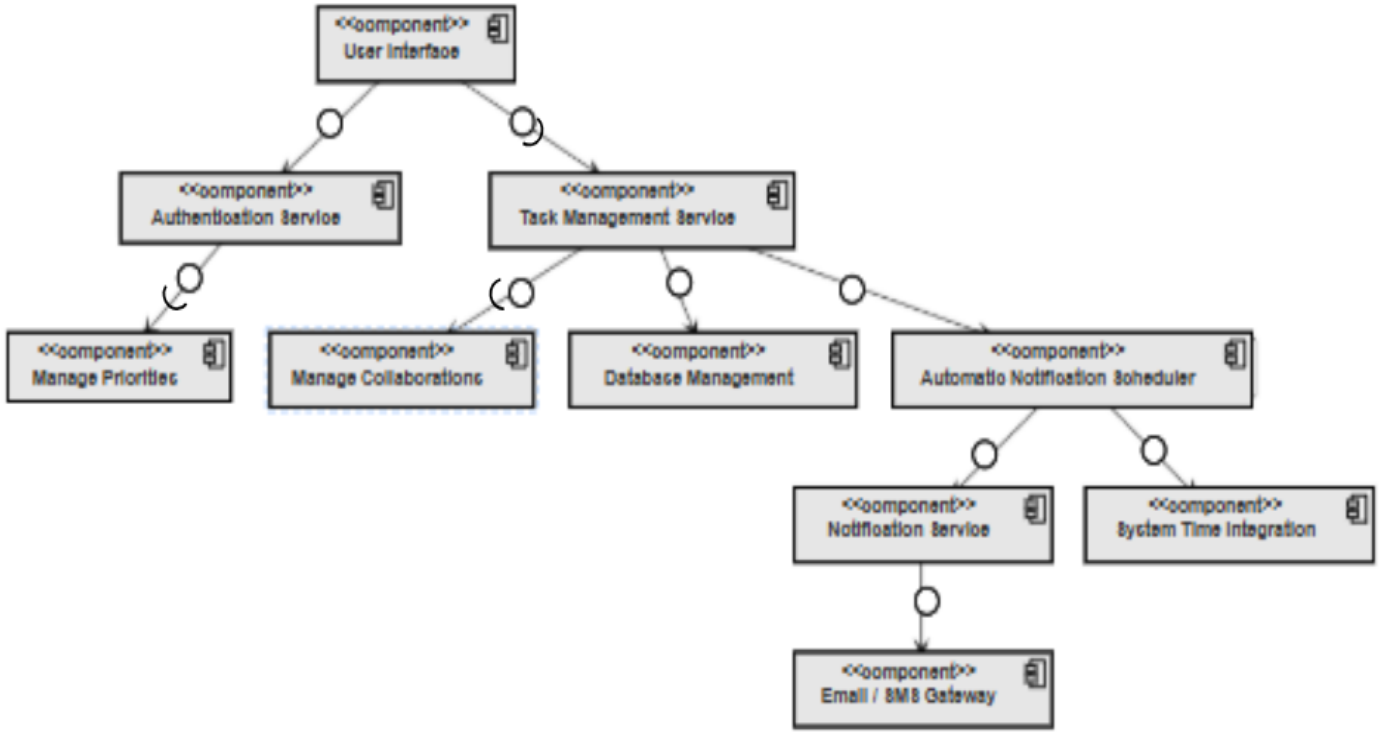
ACTIVITY DIAGRAM



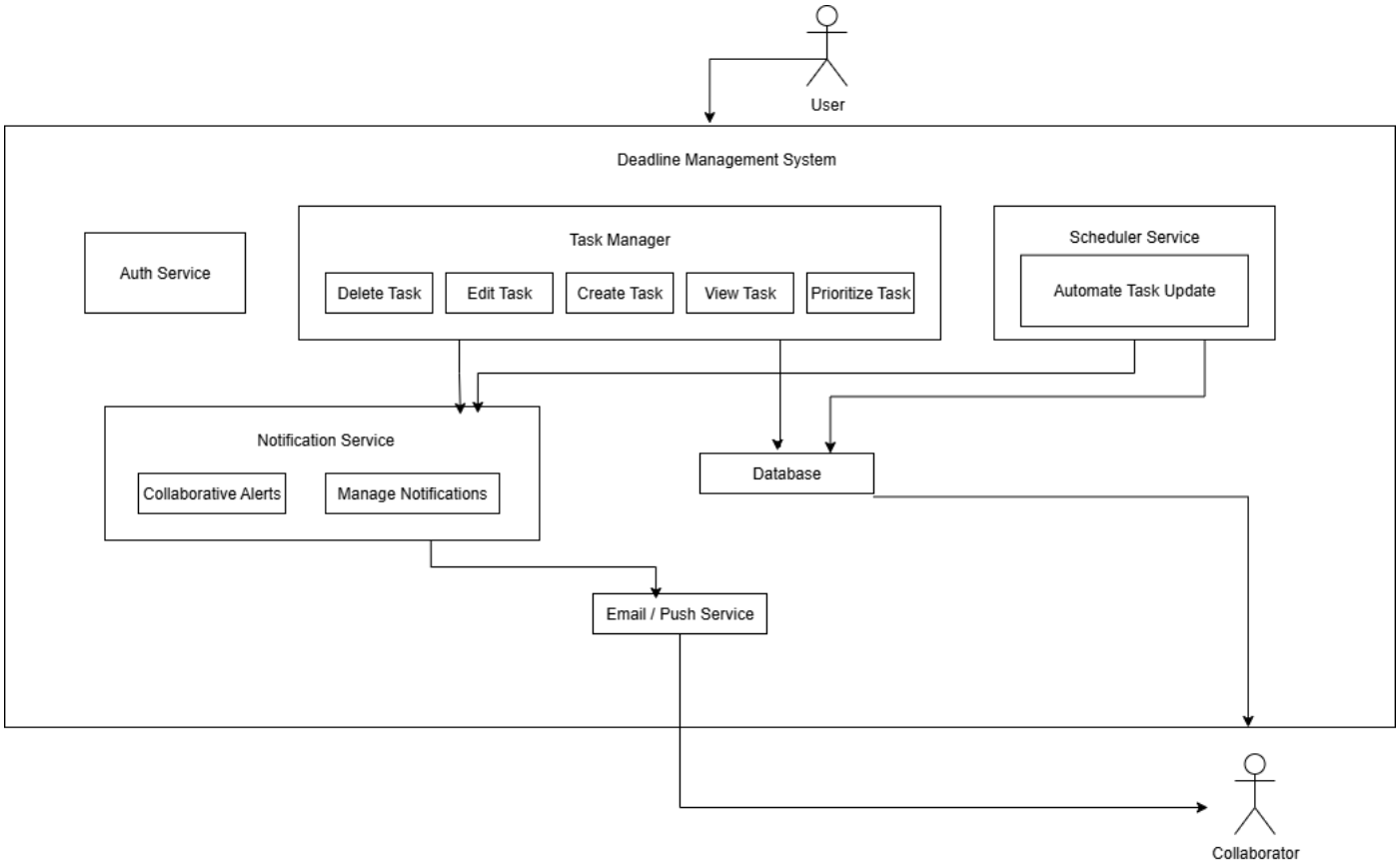
STATE DIAGRAM



COMPONENT DIAGRAM



DEPLOYMENT DIAGRAM



Appendix B: Code & Demo

SOURCE CODE

Backend/config/database.js

```
const mongoose = require('mongoose');
const connectDB = async () => {
  try {
                    const conn = await mongoose.connect(process.env.MONGO_URI ||
'mongodb://localhost:27017/taskmanager', {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log(`MongoDB connected: ${conn.connection.host}`);
  } catch (error) {
    console.error(error);
    process.exit(1);
  }
};
module.exports = connectDB;
```

Backend/controllers/taskController.js

```
const Task = require('../models/Task');
const { validationResult } = require('express-validator');
const convertToISTDate = (input) => {
  const [date, time] = input.split('T');
  const [year, month, day] = date.split('-').map(Number);
  const [hour, minute] = time.split(':').map(Number);
  return new Date(Date.UTC(year, month - 1, day, hour - 5, minute - 30));
};
const shouldTaskBeActiveToday = (task) => {
  const today = new Date().toLocaleDateString('en-US', { weekday: 'long' }).toLowerCase();
  if (!task.recurrence || !task.recurrence.type) return true;
  if (task.recurrence.type === 'one-time') return true;
  if (task.recurrence.type === 'daily') return true;
  if (task.recurrence.type === 'custom') {
    return task.recurrence.customDays.includes(today);
  }
  return true;
};
const resetRecurringTasks = async (userId) => {
  const now = new Date();
  const startOfDay = new Date(now.getFullYear(), now.getMonth(), now.getDate());
  const tasks = await Task.find({
    user: userId,
    'recurrence.type': { $in: ['daily', 'custom'] },
    status: { $in: ['completed', 'missed'] },
    updatedAt: { $lt: startOfDay },
  });
  for (const task of tasks) {
    if (shouldTaskBeActiveToday(task)) {
      task.status = 'pending';
```

```
      await task.save();
    }
  }
};
const getTasks = async (req, res) => {
  try {
    const { status, priority, sortBy, sortOrder } = req.query;
    await resetRecurringTasks(req.user._id);
    const filter = { user: req.user._id };
    if (status) filter.status = status;
    if (priority) filter.priority = priority;
    let tasks = await Task.find(filter);
    const now = new Date();
    const endOfDay = new Date(now.getFullYear(), now.getMonth(), now.getDate(), 23, 59, 59);
    const toMiss = tasks.filter(task => {
      if (['completed', 'missed'].includes(task.status)) return false;
      if (!task.dueDate) return false;
      const due = new Date(task.dueDate);
      if (!task.recurrence || task.recurrence.type === 'one-time') {
        return due < now;
      }
      return now > endOfDay && shouldTaskBeActiveToday(task);
    })
    await Promise.all(toMiss.map(task => {
      task.status = 'missed';
      return task.save();
    }));
    tasks = await Task.find(filter);
    tasks = tasks.filter(task => {
      if (!task.recurrence) return true;
      if (task.recurrence.type === 'one-time') return true;
      return shouldTaskBeActiveToday(task);
    });

    if (sortBy === 'priority') {
      const prioritySortMap = { high: 1, medium: 2, low: 3 };
      const direction = sortOrder === 'desc' ? -1 : 1;
      tasks.sort((a, b) => {
        const diff = (prioritySortMap[a.priority || 'medium'] - prioritySortMap[b.priority || 'medium']) *
direction;
        if (diff !== 0) return diff;
        return new Date(b.createdAt) - new Date(a.createdAt);
      });
    } else if (sortBy === 'dueDate') {
      const direction = sortOrder === 'desc' ? -1 : 1;
      tasks.sort((a, b) => (new Date(a.dueDate) - new Date(b.dueDate)) * direction);
    } else if (sortBy === 'createdAt') {
      const direction = sortOrder === 'desc' ? -1 : 1;
      tasks.sort((a, b) => (new Date(a.createdAt) - new Date(b.createdAt)) * direction);
    }
    res.json({
      success: true,
      count: tasks.length,
      data: tasks,
```

```
    });
  } catch (error) {
    console.error('Error in getTasks:', error);
    res.status(500).json({ message: 'Server error while fetching tasks' });
  }
};
const createTask = async (req, res) => {
  try {
    const errors = validationResult(req);
    if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });
    let { title, description, priority, dueDate, meetingLink, recurrenceType, customDays } = req.body;
    const imageUrl = req.file ? `/uploads/${req.file.filename}` : '';
    if (dueDate) {
      dueDate = convertToISTDate(dueDate);
    }
    const recurrence = {
      type: recurrenceType || 'one-time',
      customDays: recurrenceType === 'custom' ? JSON.parse(customDays || '[]') : []
    };
    const task = await Task.create({
      title,
      description,
      priority,
      dueDate,
      meetingLink,
      imageUrl,
      recurrence,
      user: req.user._id,
    });
    res.status(201).json({ success: true, data: task });
  } catch (error) {
    console.error('Error in createTask:', error);
    res.status(500).json({ message: 'Server error creating task' });
  }
};
const updateTask = async (req, res) => {
  try {
    const task = await Task.findById(req.params.id);
    if (!task) return res.status(404).json({ message: 'Task not found' });
    if (task.user.toString() !== req.user._id.toString())
      return res.status(401).json({ message: 'Not authorized' });
    if (req.file) req.body.imageUrl = `/uploads/${req.file.filename}`;
    if (req.body.dueDate) {
      req.body.dueDate = convertToISTDate(req.body.dueDate);
    }
    if (req.body.recurrenceType) {
      req.body.recurrence = {
        type: req.body.recurrenceType,
    customDays: req.body.recurrenceType === 'custom' ? JSON.parse(req.body.customDays || '[]') : []
      };
      delete req.body.recurrenceType;
      delete req.body.customDays;
    }
    const updatedTask = await Task.findByIdAndUpdate(req.params.id, req.body, {
```

```
      new: true,
      runValidators: true,
    });
    res.json({ success: true, data: updatedTask });
  } catch (error) {
    console.error('Error in updateTask:', error);
    res.status(500).json({ message: 'Server error updating task' });
  }
};
const deleteTask = async (req, res) => {
  try {
    const task = await Task.findById(req.params.id);
    if (!task) return res.status(404).json({ message: 'Task not found' });
    if (task.user.toString() !== req.user._id.toString())
      return res.status(401).json({ message: 'Not authorized' });

    await Task.findByIdAndDelete(req.params.id);

    res.json({ success: true, message: 'Task deleted successfully' });
  } catch (error) {
    console.error('Error in deleteTask:', error);
    res.status(500).json({ message: 'Server error deleting task' });
  }
};
module.exports = { getTasks, createTask, updateTask, deleteTask };
```

backend/middleware/auth.js

```
const jwt = require('jsonwebtoken');
const User = require('../models/User');
const protect = async (req, res, next) => {
  let token;
  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith('Bearer')
  ) {
    try {
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = await User.findById(decoded.id).select('-password');
      next();
    } catch (error) {
      console.error(error);
      res.status(401).json({ message: 'Not authorized, token failed' });
    }
  }
  if (!token) {
    res.status(401).json({ message: 'Not authorized, no token' });
  }
};

module.exports = { protect };
```

backend/models/Task.js

```
const mongoose = require('mongoose');
const taskSchema = new mongoose.Schema({
  title: { type: String, required: true, maxlength: 100, trim: true },
  description: { type: String, required: true, maxlength: 500 },
  status: { type: String, enum: ['pending', 'in-progress', 'completed', 'missed'], default: 'pending' },
  priority: { type: String, enum: ['low', 'medium', 'high'], default: 'medium' },
  dueDate: { type: Date, required: true },
  meetingLink: { type: String, default: '' },
  imageUrl: { type: String, default: '' },
    recurrence: {
    type: { type: String, enum: ['one-time', 'daily', 'custom'], default: 'one-time' },
    customDays: [{ type: String, enum: ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday',
'sunday'] }]
  },
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
});
taskSchema.pre('save', function (next) {
  this.updatedAt = Date.now();
  next();
});
module.exports = mongoose.model('Task', taskSchema);
```

frontend/src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import './tailwind-output.css';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

frontend/src/app.js

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { AuthProvider } from './context/AuthContext';
import { TaskProvider } from './context/TaskContext';
import Login from './components/auth/Login';
```

```
import Register from './components/auth/Register';
import Dashboard from './components/dashboard/Dashboard';
import TaskList from './components/tasks/TaskList';
import ProtectedRoute from './components/common/ProtectedRoute';
import Header from './components/layout/Header';
import { Toaster } from 'react-hot-toast';
function App() {
 return (
  <div className="min-h-screen bg-gradient-to-br from-slate-900 via-purple-900 to-slate-900">
   <Router>
    <AuthProvider>
     <TaskProvider>
      <Toaster
       position="top-right"
       toastOptions={{
        duration: 4000,
        style: {
         background: '#1e293b',
         color: '#f1f5f9',
         border: '1px solid #475569',
        },
        success: {
         style: {
          background: '#064e3b',
          color: '#d1fae5',
          border: '1px solid #047857',
         },
        },
        error: {
         style: {
          background: '#7f1d1d',
          color: '#fecaca',
          border: '1px solid #dc2626',
         },
        },
       }}
      />
      <div className="relative min-h-screen">
       <div
        className="absolute inset-0 opacity-20"
```

```
            style={{
              backgroundImage: `url("data:image/svg+xml,%3Csvg width='60' height='60'
viewBox='0 0 60 60' xmlns='http://www.w3.org/2000/svg'%3E%3Cg fill='none' fill-
rule='evenodd'%3E%3Cg fill='%239C92AC' fill-opacity='0.05'%3E%3Ccircle cx='30' cy='30'
r='4'/%3E%3C/g%3E%3C/g%3E%3C/svg%3E")`,
            }}
          ></div>
          <Routes>
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
            <Route path="/" element={
              <ProtectedRoute>
                <>
                  <Header />
                  <main className="relative z-10">
                    <Dashboard />
                  </main>
                </>
              </ProtectedRoute>
            } />
            <Route path="/tasks" element={
              <ProtectedRoute>
                <>
                  <Header />
                  <main className="relative z-10">
                    <TaskList />
                  </main>
                </>
              </ProtectedRoute>
            } />
            <Route path="*" element={<Navigate to="/" replace />} />
          </Routes>
        </div>
      </TaskProvider>
    </AuthProvider>
  </Router>
 </div>
 );
}
export default App;
```

frontend/src/App.css
```
.App {
  text-align: center;
}
.App-logo {
  height: 40vmin;
  pointer-events: none;
}
@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}
.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}
.App-link {
  color: #61dafb;
}
@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

frontend/src/components/auth/login.jsx
```
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { toast } from 'react-hot-toast';
import { useAuth } from '../../context/AuthContext';
import Button from '../common/Button';
import Input from '../common/Input';
import { EnvelopeIcon, LockClosedIcon, EyeIcon, EyeSlashIcon } from
'@heroicons/react/24/outline';
const Login = () => {
  const [formData, setFormData] = useState({ email: '', password: '' });
  const [showPassword, setShowPassword] = useState(false);
  const [loading, setLoading] = useState(false);
  const { login } = useAuth();
  const navigate = useNavigate();
  const handleChange = (e) => setFormData({ ...formData, [e.target.name]: e.target.value });
  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);
```

```
    try {
      await login(formData);
      toast.success(' Login successful!');
      navigate('/tasks');
    } catch (error) {
      toast.error(error.response?.data?.message || error.message || ' Login failed');
    } finally {
      setLoading(false);
    }
  };
  return (
    <div className="min-h-screen flex items-center justify-center px-4 relative">
      <div className="absolute inset-0 bg-gradient-to-br from-indigo-900/20 via-purple-900/20 to-
pink-900/20 pointer-events-none"></div>
      <div className="absolute top-0 left-0 w-full h-full pointer-events-none">
        <div className="absolute top-20 left-20 w-72 h-72 bg-purple-500/10 rounded-full blur-
3xl"></div>
        <div className="absolute bottom-20 right-20 w-96 h-96 bg-indigo-500/10 rounded-full blur-
3xl"></div>
      </div>
      <div className="relative z-10 w-full max-w-md bg-white/5 backdrop-blur-xl border border-
white/10 rounded-2xl shadow-2xl p-8">
        <div className="text-center mb-8">
          <div className="inline-flex items-center justify-center w-16 h-16 bg-gradient-to-r from-
indigo-500 to-purple-600 rounded-xl mb-4 shadow-lg">
            <svg className="w-8 h-8 text-white" fill="none" stroke="currentColor" viewBox="0 0 24
24">
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M9 5H7a2 2 0
00-2 2v12a2 2 0 002 2h10a2 2 0 002-2V7a2 2 0 00-2-2h-2M9 5a2 2 0 002 2h2a2 2 0 002-2M9 5a2
2 0 012-2h2a2 2 0 012 2" />
            </svg>
          </div>
          <h2 className="text-3xl font-bold bg-gradient-to-r from-white to-gray-300 bg-clip-text text-
transparent mb-2">
            Welcome Back!!!
          </h2>
          <p className="text-gray-400">Sign in to continue to TaskFlow</p>
        </div>
        <form className="space-y-6" onSubmit={handleSubmit}>
          <div>
            <label className="block text-sm font-medium text-gray-200 mb-2">Email
Address</label>
            <div className="relative">
              <div className="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
                <EnvelopeIcon className="h-5 w-5 text-gray-400" />
              </div>
              <Input
                type="email"
                name="email"
                value={formData.email}
                onChange={handleChange}
                required
                placeholder="Enter your email"
```

```
            className="w-full pl-10 pr-4 py-3 bg-white/5 border border-white/10 rounded-xl text-
white placeholder-gray-400 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-
transparent transition-all"
           />
          </div>
        </div>
        <div>
          <label className="block text-sm font-medium text-gray-200 mb-2">Password</label>
          <div className="relative">
            <div className="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
              <LockClosedIcon className="h-5 w-5 text-gray-400" />
            </div>
            <Input
              type={showPassword ? 'text' : 'password'}
              name="password"
              value={formData.password}
              onChange={handleChange}
              required
              placeholder="Enter your password"
              className="w-full pl-10 pr-12 py-3 bg-white/5 border border-white/10 rounded-xl text-
white placeholder-gray-400 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:border-
transparent transition-all"
            />
            <button
              type="button"
              onClick={() => setShowPassword((show) => !show)}
              className="absolute inset-y-0 right-0 pr-3 flex items-center text-gray-400 hover:text-
gray-200 transition-colors"
            >
              {showPassword ? <EyeSlashIcon className="h-5 w-5" /> : <EyeIcon className="h-5
w-5" />}
            </button>
          </div>
        </div>
        <Button
          type="submit"
          isLoading={loading}
          className="w-full bg-gradient-to-r from-indigo-500 to-purple-600 hover:from-indigo-600
hover:to-purple-700 text-white font-semibold py-3 rounded-xl shadow-lg hover:shadow-xl
transform hover:scale-[1.02] transition-all duration-200"
        >
          {loading ? 'Signing in...' : 'Sign In'}
        </Button>
      </form>
      <div className="mt-6 text-center">
        <p className="text-gray-400">
          Don't have an account?{' '}
          <Link to="/register" className="text-indigo-400 hover:text-indigo-300 font-medium
transition-colors">
            Create one here
          </Link>
        </p>
      </div>
    </div>
```

```
    </div>
  );
};
export default Login;
```

frontend/src/components/tasks/TaskFilter.jsx

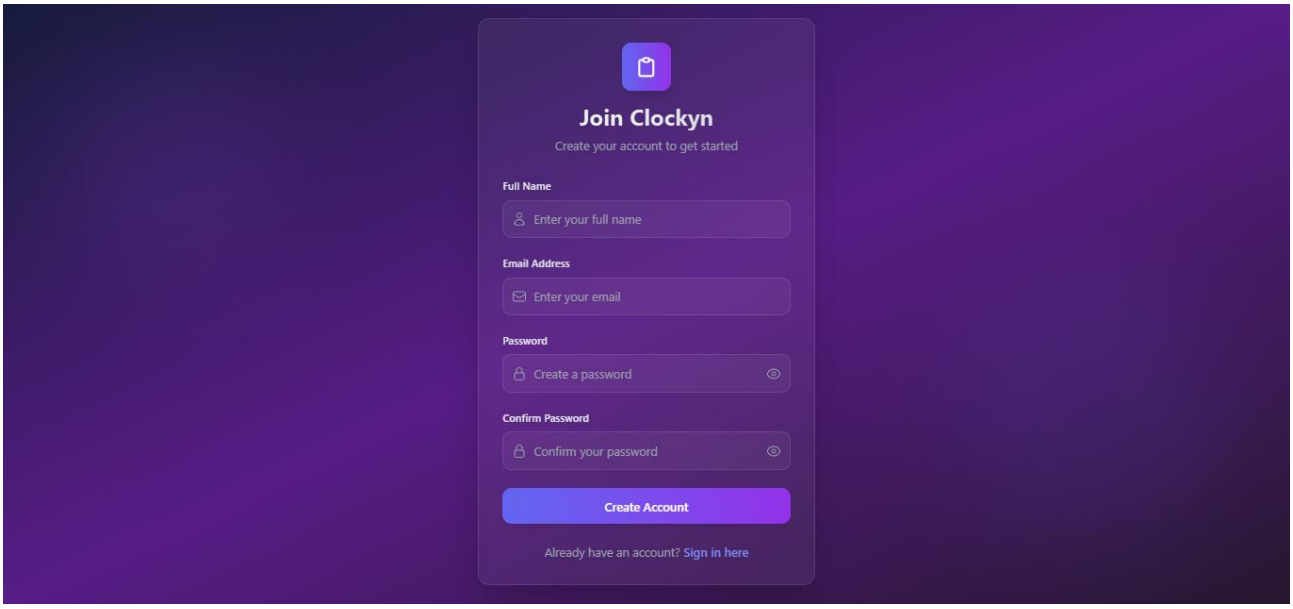
```
import React from 'react';
const TaskFilter = ({ filters, setFilters }) => (
  <div className="bg-slate-700/40 rounded-xl p-5 shadow-lg border border-slate-600 mb-5">
    <div className="flex flex-wrap gap-6">
      <div>
        <label className="block text-gray-300 mb-1 font-semibold">Status</label>
        <select
          value={filters.status || ''}
          onChange={e => setFilters({ ...filters, status: e.target.value })}
          className="bg-slate-800 border border-slate-600 text-white px-3 py-2 rounded-lg"
        >
          <option value="">All</option>
          <option value="pending">Pending</option>
          <option value="in-progress">In Progress</option>
          <option value="completed">Completed</option>
        </select>
      </div>
      <div>
        <label className="block text-gray-300 mb-1 font-semibold">Priority</label>
        <select
          value={filters.priority || ''}
          onChange={e => setFilters({ ...filters, priority: e.target.value })}
          className="bg-slate-800 border border-slate-600 text-white px-3 py-2 rounded-lg"
        >
          <option value="">All</option>
          <option value="high">High</option>
          <option value="medium">Medium</option>
          <option value="low">Low</option>
        </select>
      </div>
    </div>
  </div>
);
export default TaskFilter;
```

frontend/src/App.test.js

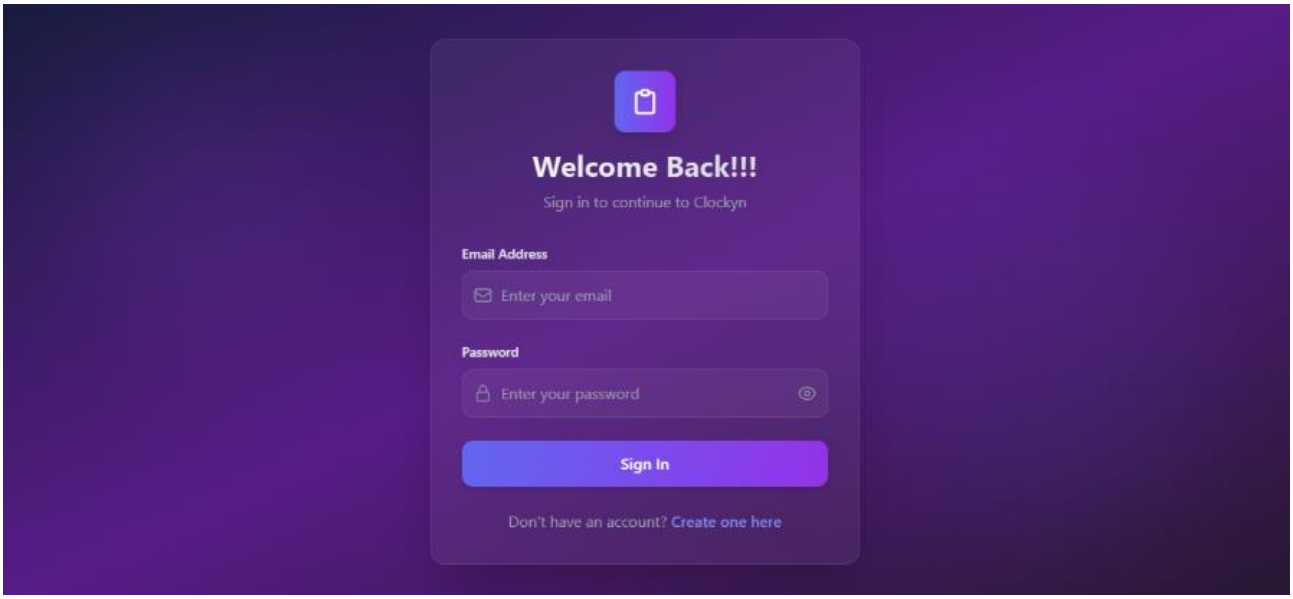
```
import { render, screen } from '@testing-library/react';
import App from './App';
test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

DEMONSTRATION

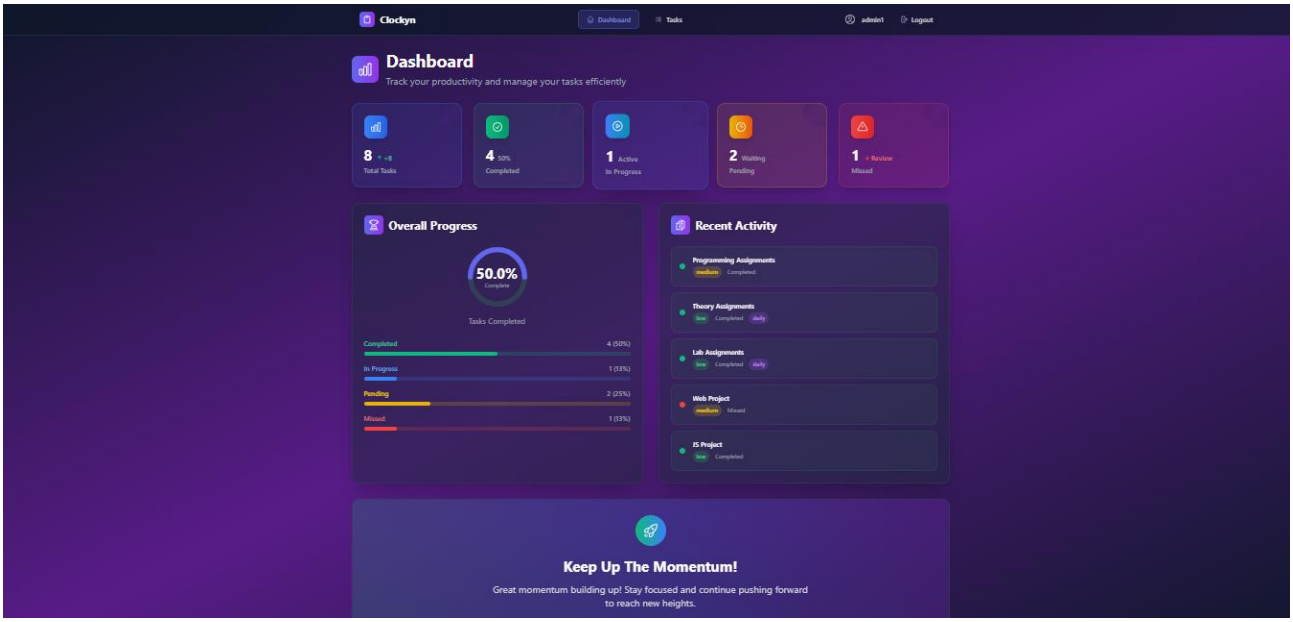
Login page - User authentication screen with glassmorphic UI and email/password fields.



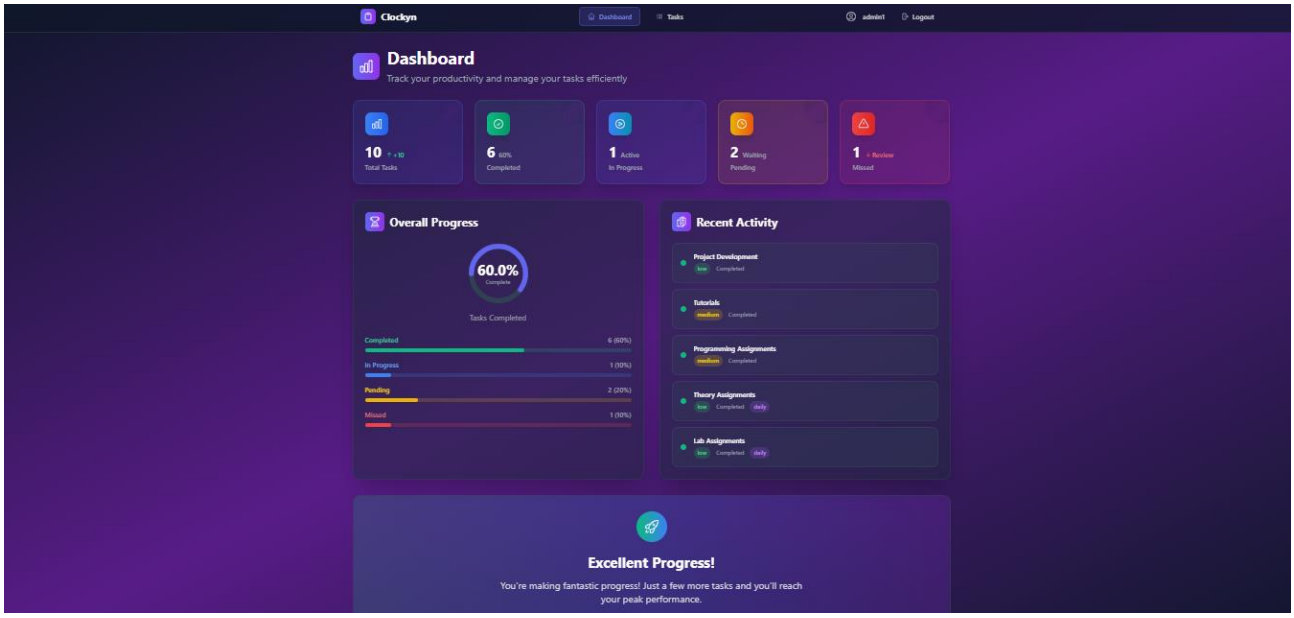
Register Page – Account creation form with full name, email, password, and confirm password in a modern card.



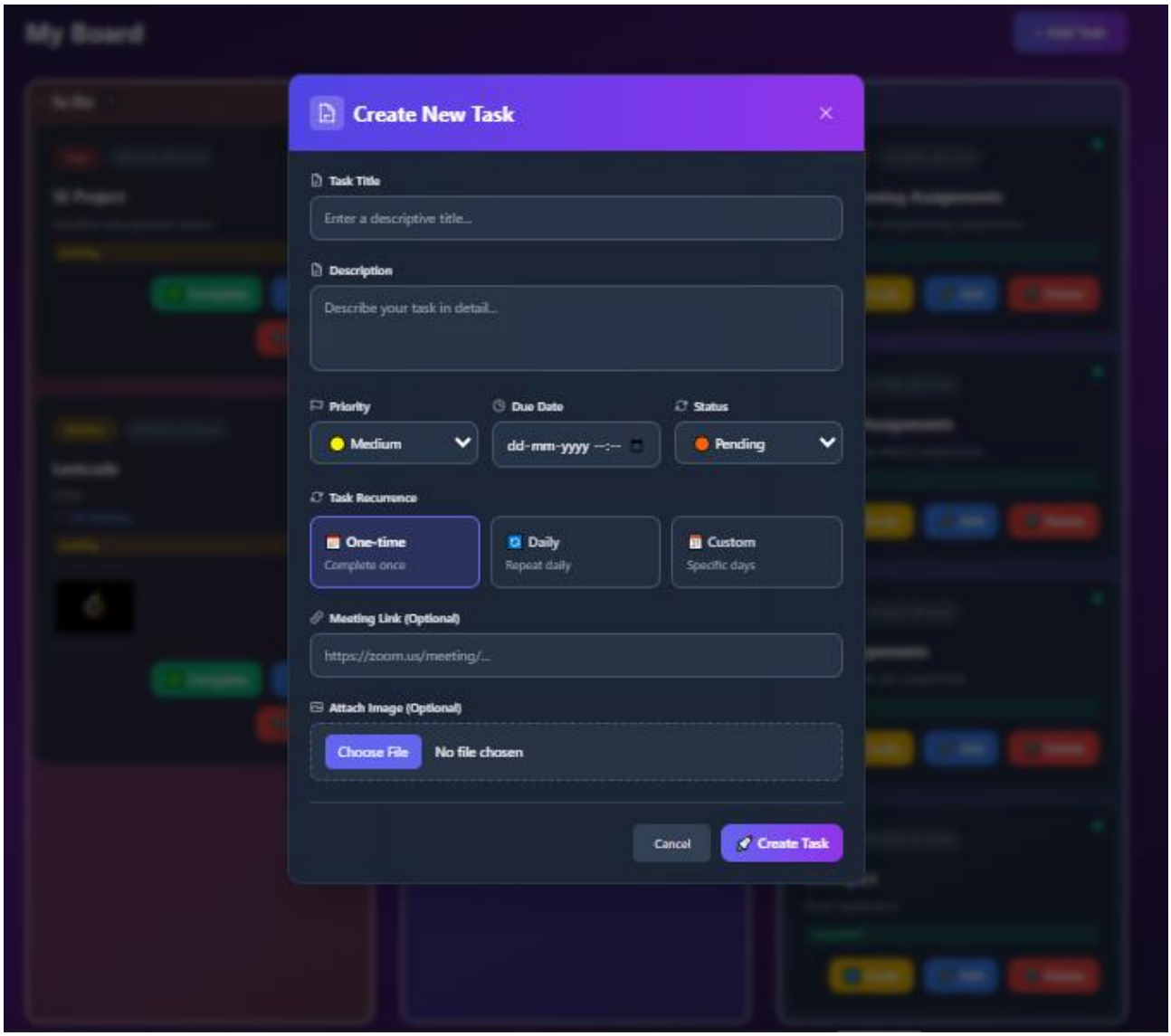
Dashboard (high progress) - Progress metrics, activity and motivation (>= 40% completion of total tasks).



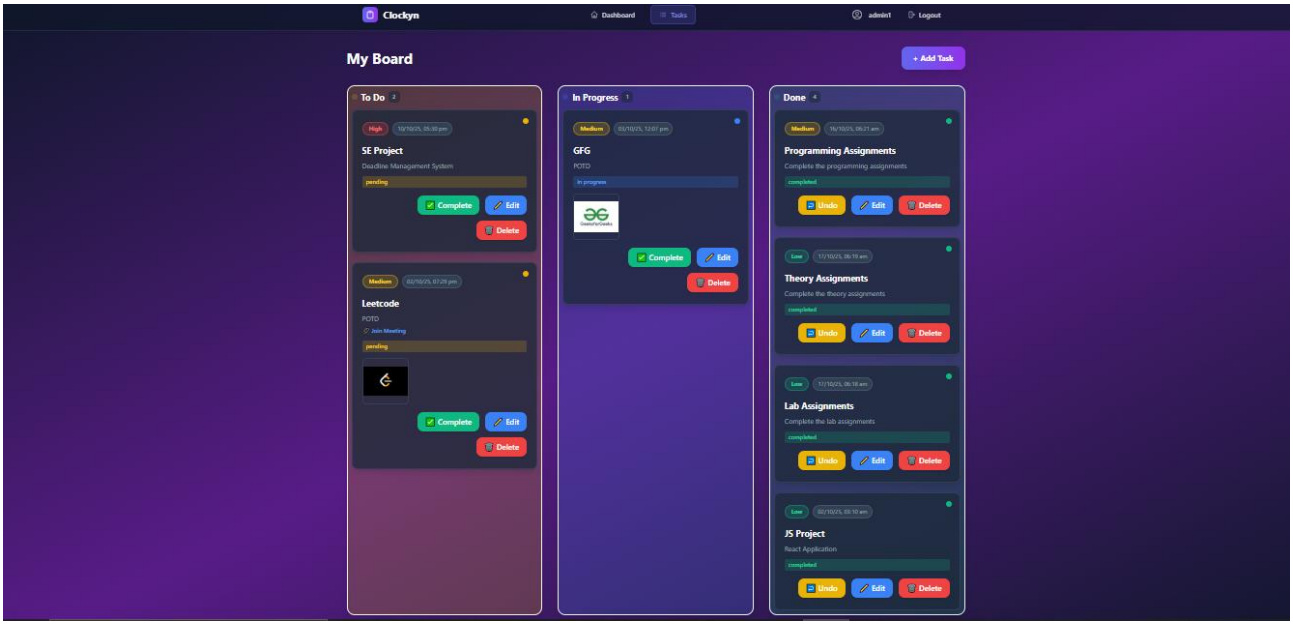
Dashboard (excellent progress) - Progress metrics, activity and motivation (>= 60% completion of total tasks).



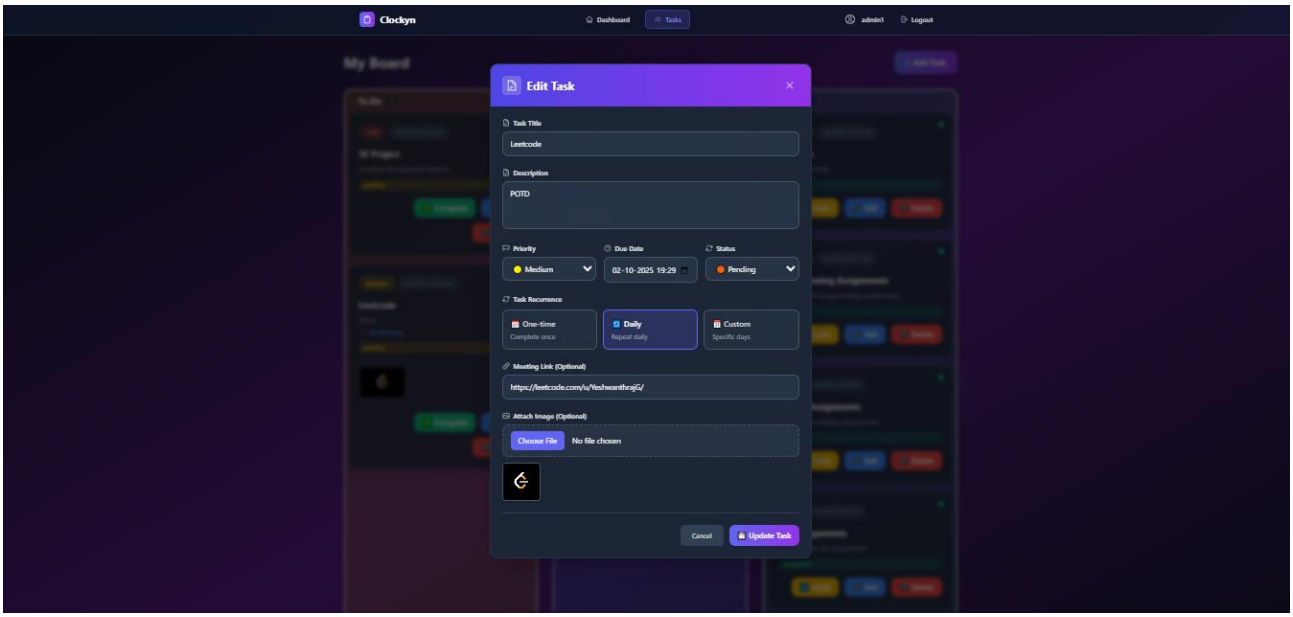
Task form (create) - for creating a new task, with priority, due date, status, recurrence, link, and file upload options.



Kanban board (many tasks) - Expanded board showing detailed cards under each column, sortable by priority/status.



Kanban board (active editing) - Board in background while editing an existing task in the popup.



GITHUB REPO

https://github.com/YeshwanthrajG/Deadline-Management-System-Using-MERN.git