Final Report

Level 02

Employee Performance Review System (RevX)

Team CodeNex

Department of Information Technology
Faculty of Information Technology
University of Moratuwa

July 2024

Final Report

Level 02

Employee Performance Review System (RevX)

Team CodeNex

Team ‘CodeNex’ Group Members

Index No.	Name
224195D	**Thilshath SM (Leader)**
224051J	Fadhil MFM
224127A	Muadh MRM
224257V	Haneef MNAR
224251X	Faskath MHM

This report was submitted to the Department of Information Technology, Faculty of Information Technology, University of Moratuwa, Sri Lanka, for the partial fulfilment of the requirements of the Degree of BSc Hons (IT) degree program.
July 2024

Declaration

I declare that this report does not incorporate, without acknowledgement, any material previously submitted for a degree or a Diploma in any University and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby consent to my report, if accepted, being made available for photocopying and interlibrary loans and for the title and summary to be made available to outside organizations.

Index No	Name	Signature
224195D	Thilshath SM	Thilshath
224051J	Fadhil MFM	[illegible]
224127A	Muadh MRM	[illegible]
224257V	Haneef MNAR	[illegible]
224251X	Faskath MHM	Faskath

Date	Name Of Supervisor	Signature
14th of July 2025	Ms. M N Chandimali	Nipuni Chandimali Digitally signed by Nipuni Chandimali Date: 2025.07.18 22:03:58 +05'30'
13th of July 2025	Ms. Nimni Senarathna	[illegible]

Acknowledgements

We would like to express my deepest gratitude to all those who have contributed to the successful completion of this project.

First and foremost, we would like to thank our supervisor at university,
Ms. M N Chandimali, for her invaluable guidance, insightful feedback, and constant encouragement throughout this journey. Your expertise and support have been instrumental in shaping this project.

Secondly, we thank **Ms. Nimni Senarathna**, our Industry Supervisor . Your support and advice have helped us reach the deadlines and finish the project with the best outputs.

We express our gratitude to the Department of Information Technology, Faculty of Information Technology, for all the guidance. We are also immensely grateful to my friends and colleagues, whose support and encouragement have been a source of great strength. Special thanks to our project team members for their dedication, hard work, and collaborative spirit.

Thank you all for making this journey possible.

Abstract

The ***Employee Performance Management and Review System*** **(RevX)** is a web-based application developed to modernize and optimize the employee performance review process within organizations. The system offers a comprehensive platform that enables HR administrators, managers, and employees to collaborate effectively in managing performance-related activities.

In this system, HR personnel **create and manage user accounts for managers and employees,** defining secure access roles and permissions. HR administrators set up review cycles that apply to both managers and employees, establishing clear timelines and automated notifications to ensure timely participation in reviews.

Managers can define performance **goals and assign specific tasks** to their team members, providing clarity and accountability throughout the review period. Employees have the ability to **track progress on goals** and **submit self-assessments to their managers**, documenting achievements and areas for development. Managers, in turn, **provide structured feedback** to employees based on predefined performance criteria.

After the completion of reviews and feedback submission, **HR administrators analyze performance data** across individuals and teams, generating detailed reports and insights to support strategic decision-making and organizational development.

The system's capabilities include role-based access control, goal and task management, review scheduling, feedback workflows, self-assessment submission, performance dashboards, and reporting tools.

Technologically, the application is built with a modern stack: **Next.js** and **NextAuth** for a secure, responsive frontend, and **Node.js**, **Express**, and **MongoDB** on the backend to ensure scalability, flexibility, and robust data management.

By streamlining every stage of the performance review process—from goal setting and self-assessment to feedback and reporting—this system enhances employee engagement, supports continuous improvement, and empowers organizations to make data-driven decisions.

Contents

Appendices

List of Figures

List of Tables

Chapter 1 – Introduction

1.1 Introduction

Performance reviews are critical for organizational success, yet many businesses struggle with manual, inefficient processes that lead to missed opportunities for employee development and engagement. The absence of streamlined systems often results in unclear goal setting, delayed feedback, and poor alignment between employee performance and organizational objectives.

To address this challenge, we propose an Employee Performance Review System, a web-based application designed to simplify and enhance the performance review process. By leveraging modern technologies such as Next.js, Node.js, and MongoDB, this system will enable HR teams, managers, and employees to seamlessly collaborate on setting and tracking performance goals, providing constructive feedback, and making data-driven decisions.

The proposed solution offers key features like role-based access control, automated review scheduling, performance dashboards, and analytics reporting. These functionalities aim to foster continuous improvement, enhance transparency, and support strategic workforce planning. Ultimately, this solution will empower organizations to achieve better productivity, engagement, and growth.

1.2 Problem in Brief

Employee performance management is a critical process for organizations to achieve their goals, yet many businesses rely on outdated or manual systems that are inefficient and prone to errors. Traditional performance review methods often lack transparency, fail to provide timely feedback, and do not adequately align employee objectives with organizational goals. This results in decreased employee engagement, poor productivity, and limited opportunities for development.

In today's competitive landscape, organizations need a structured and data-driven approach to manage employee performance. Without such systems, it becomes difficult to identify high-performing employees, address skill gaps, and make informed decisions about promotions, training, or workforce planning.

The importance of solving this problem lies in its impact on both employees and the organization. Employees benefit from clear expectations, constructive feedback, and opportunities for growth, while organizations achieve better alignment, improved morale, and higher retention rates. A scalable and efficient solution that automates and enhances the performance review process is essential for modern businesses to thrive.

1.3 Aim and Objectives

Aim

- ➢ The aim of this project is to develop a web-based Employee Performance Review System to streamline the performance review process using modern technologies like Next.js, Node.js, and MongoDB, ensuring efficient goal setting, feedback, and performance tracking.

Objective

• To provide secure role-based access control for employees, managers, and HR administrators.

• To facilitate goal setting and tracking for employees, ensuring alignment with organizational objectives.

• To automate the scheduling of performance review cycles with notifications and reminders. • To enable managers to deliver structured feedback and employees to submit self-assessments.

• To develop performance dashboards for tracking key performance indicators (KPIs).

• To generate comprehensive reports and analytics for HR teams to make data driven decisions.

• To ensure a user-friendly interface that promotes engagement and seamless interaction for all users.

1.4 Background and Motivation

Organizations today face significant challenges in managing employee performance effectively. Traditional performance review methods often involve cumbersome manual processes, lack of timely feedback, and inconsistent evaluation criteria. These issues can result in disengaged employees, unclear expectations, and a lack of alignment between individual and organizational goals.

The need for a more structured, transparent, and efficient performance review system has grown as businesses increasingly rely on data-driven decision-making and employee development strategies to stay competitive in dynamic markets. A well-designed performance management system not only enhances organizational productivity but also fosters a culture of continuous improvement and employee engagement.

The motivation behind developing the **Employee Performance Review System** stems from these challenges. By providing a web-based platform, this project aims to bridge the gap between employees, managers, and HR teams, ensuring a seamless and collaborative performance review process. Key features like goal-setting, constructive feedback mechanisms, and automated review scheduling aim to empower organizations to:

1. **Promote Employee Growth:** Enable employees to track their goals, receive constructive feedback, and improve their performance.
2. **Support Managerial Effectiveness:** Provide tools for managers to set expectations, assess performance, and offer actionable feedback in a structured manner.
3. **Enhance HR Efficiency:** Equip HR teams with data analytics and reporting tools to make informed decisions and align employee performance with organizational objectives.
4. **Encourage Transparency and Fairness:** Establish clear criteria for evaluation, ensuring consistency and fairness in the review process.

This system is motivated by the vision of creating an integrated platform that not only simplifies the review process but also drives a culture of accountability, collaboration, and growth across the organization.

1.5 Proposed Solution

The proposed solution is a web-based Employee Performance Review System designed to streamline the performance review process, enhance transparency, and foster a culture of continuous improvement within organizations.

The system will cater to three primary user roles :

1. **HR Admins** – Responsible for managing review cycles, user accounts, and generating reports.

2. **Managers** – Tasked with setting performance goals, providing feedback, and assessing employee performance.

3. **Employees** – Able to view and update goals, submit self-assessments, and track progress.

The system will enable HR admins to create and manage review cycles with deadlines and notifications, ensuring that reviews are completed on time. Managers will use the platform to set **SMART** (Specific, Measurable, Achievable, Relevant, Time-bound) goals for their team members, while employees can monitor and update their goals to stay aligned with organizational expectations.

Chapter 2 - Existing Systems and Limitations

2.1 Introduction

Chapter 2 focuses on documenting the various approaches and systems currently used to manage employee performance and reviews, addressing the same problem for which this application aims to provide a solution.

In today's digital age, Employee Performance Review Systems (RevX) are essential tools for organizations to monitor and improve employee performance, set goals, and foster growth. These systems often include goal-setting modules, feedback mechanisms, and reporting features. However, many existing solutions fail to provide the flexibility and user-centric design required by diverse organizations.

This chapter highlights similar systems in the market, analyzing their features, functionalities, and limitations compared to the proposed solution. By understanding the gaps in current applications, this project aims to address the challenges and shortcomings of existing systems while providing a tailored and efficient solution for performance management.

2.2 Existing Solutions

There are several Employee Performance Management and Review Systems currently available in the market, each designed to streamline and enhance the performance review process. These solutions aim to provide tools for goal setting, performance evaluation, feedback sharing, and reporting. Below are a few notable examples:

1. BambooHR

BambooHR is a widely used human resource management system that includes a performance management module. It allows managers to set goals, track employee progress, and conduct performance reviews. While it offers robust features, it often lacks customization options for specific organizational needs.

2. Workday Performance Management

Workday provides a comprehensive suite of performance management tools, including continuous feedback, goal management, and employee self-assessments. However, its complexity and cost can make it less accessible for smaller organizations.

3. SAP SuccessFactors

SAP SuccessFactors is a leading solution for managing employee performance and development. It offers advanced analytics and integration capabilities but may require significant expertise to implement and manage effectively.

4. Lattice

Lattice focuses on performance management and employee engagement. It provides features for real-time feedback, 1-on-1 meetings, and progress tracking. Despite its user-friendly interface, it may not be scalable for larger enterprises.

2.3 Limitations

While the above systems provide valuable features, they often share the following limitations

- ❖ **Lack of Customization**: Many systems are rigid and do not allow organizations to tailor features according to their specific needs.
- ❖ **High Cost**: Advanced performance management tools are often expensive, making them inaccessible for small and medium-sized enterprises.
- ❖ **Complexity**: The learning curve for some platforms can be steep, requiring extensive training for HR teams and employees.
- ❖ **Limited Integration**: Some solutions do not integrate seamlessly with other organizational tools, reducing overall efficiency.

2.4 Summary

The analysis of these existing solutions underscores the need for a more tailored and accessible approach to employee performance management. The proposed system addresses these limitations by offering a user-friendly interface, flexible goal setting and review options, cost-effective deployment, and seamless integration capabilities. This positions the application as a practical and innovative solution for organizations seeking to improve their performance management processes.

Chapter 3 - Technologies To Be Used

3.1 Frontend Technologies

3.1.1 Next.js

Next.js is a popular React framework used for building dynamic and high-performing web applications. Its server-side rendering (SSR) and static site generation (SSG) capabilities enhance speed and SEO, ensuring a smooth user experience.

3.1.2 NextAuth

NextAuth is an open-source authentication library for Next.js. It provides secure, flexible, and easy-to-implement authentication, including support for OAuth, email/password login, and single sign-on (SSO).

3.2 Backend Technologies

3.2.1 Node.js with Express

Node.js is a runtime environment that enables the development of scalable, event-driven applications. Combined with the Express.js framework, it facilitates efficient routing, middleware integration, and a robust backend for handling server-side logic.

3.2.2 MongoDB

MongoDB is a NoSQL database designed for flexibility and scalability. Its document-oriented structure is well-suited for managing user roles, performance data, and review cycles, enabling rapid data retrieval and storage.

These technologies collectively ensure the system's performance, reliability, and adaptability to meet organizational needs.

Chapter 4 - Proposed Solution

4.1 Introduction

The **Employee Performance Management and Review System** aims to provide a seamless and effective solution for organizations to assess employee performance, set goals, and provide feedback. This system leverages modern technologies to enhance the user experience for HR administrators, managers, and employees. The goal is to create a user-friendly platform that promotes continuous feedback and performance tracking.

The project will be developed using the **Agile Scrum model**, which allows for iterative development, frequent feedback, and flexibility to adapt to changing requirements. This model is ideal for building an application that needs to be refined and improved over time to meet the evolving needs of the organization.

4.2 Software Model

The development of the **Employee Performance Management and Review System** will follow the **Agile Scrum methodology**, breaking the project into manageable tasks for continuous progress tracking. This iterative approach enables timely delivery of key features and incorporates user feedback through regular iterations. The development process includes **Sprint Planning**, where tasks are prioritized, **Development and Testing** to ensure quality, and daily **Standups** for team communication.

At the end of each sprint, a **Sprint Review and Demo** will be conducted to showcase completed work, gather feedback, and plan improvements. Additionally, a **Sprint Retrospective** will reflect on the process to identify strengths and areas for improvement, ensuring continuous refinement of the system. This approach ensures flexibility and responsiveness to evolving project needs.

4.3 Users, Activities, Inputs, and Outputs

User	Activities	Inputs	Outputs
HR Admin	Create and manage user accounts	Employee details, role assignment	Secure user accounts with role-based access
	Schedule performance review cycles	Review cycle details (dates, deadlines)	Notifications sent to managers and employees
	Generating performance reports	Filter criteria (e.g., department, review cycle)	Performance analysis reports

Table 01 : HR activities, inputs and Outputs

User	Activities	Inputs	Outputs
Manager	Set and edit performance goals for employees	Employee goals, measurable criteria	Assigned performance goals visible to employees
	Provide performance feedback	Employee performance metrics, written comments	Feedback submitted and stored in the system
	Submit review to HR	Team performance & project outputs in that scrum.	Completed performance reviews

Table 02 : Manager activities, inputs and Outputs

User	Activities	Inputs	Outputs
Employee	View and update personal performance goals	Personal goals, progress updates	Updated goal progress visible on the dashboard
	Submit self-assessment before the review	Achievements, challenges, areas for growth	Self-assessment submitted for manager review
	Provide Review to HR	Feedback from managers, performance review summary	Review Submitted to HR

Table 03 : Employee activities, inputs and Outputs

4.4 System Features and Modules

4.4.1 Role-Based Access Control

- Overview:
  This module ensures secure and role-specific access to the system. It is designed to manage different user roles (e.g., employees, managers, HR admins), with each role granted specific permissions.

- Key Features:
  - User authentication via secure login credentials.
  - Authorization policies to control access to features like goal management, reviews, and reports.
  - Ability to add, edit, or deactivate user accounts by HR admins.

- Benefits:
  - Enhances data security by restricting access based on roles.
  - Simplifies the management of user roles and responsibilities.

4.4.2 Goal Management

- Overview:
  Enables the setting, tracking, and updating of performance goals for employees. Both managers and employees can interact with this feature to ensure alignment with organizational objectives.

- Key Features:
  - Managers can create, assign, and modify specific goals for employees.
  - Employees can view their goals, provide updates, and track progress.
  - Automated reminders for goal deadlines and progress reviews.

- Benefits:
  - Promotes clarity and accountability in achieving targets.
  - Facilitates continuous performance monitoring and improvement.

4.4.3 Review Scheduling and Notifications

- Overview:
  This module supports the scheduling of review cycles and automated notifications to ensure timely completion of performance evaluations.

- Key Features:
  - HR admins can set review periods and deadlines for submission.
  - Notifications sent to employees and managers for upcoming or pending reviews.
  - Calendar integration for tracking review schedules.

- Benefits:
  - Reduces manual follow-ups by HR.
  - Encourages adherence to review timelines.

4.4.4 Performance Feedback and Self-Assessment

- Overview:
  Provides a structured mechanism for managers to offer constructive feedback and for employees to reflect on their performance through self-assessments.

- Key Features:
  - Predefined templates for feedback covering various performance criteria.
  - Self-assessment forms for employees to highlight achievements and challenges.
  - Feedback history for tracking progress over time.

- Benefits:
  - Encourages open communication between employees and managers.
  - Aids in identifying strengths and areas for development.

4.4.5 Reporting and Analytics

- Overview:
  Advanced reporting tools that provide insights into employee performance and review trends, assisting HR in making data-driven decisions.

- Key Features:
  - Dashboard for visualizing performance metrics (e.g., KPIs).
  - Customizable reports by department, team, or individual performance.
  - Data export functionality for further analysis using external tools.

- **Benefits**:
  - Helps HR identify patterns and areas requiring intervention.
  - Supports strategic planning and decision-making based on performance data.

4.5 System Architecture

The system architecture of the Employee Performance Management and Review System is designed to ensure scalability, security, and seamless performance. It adopts a multi-tier structure comprising distinct layers for the frontend, backend, and database. The frontend, developed using Next.js, provides an intuitive user interface catering to employees, managers, and HR admins. The backend is powered by secure servers with REST APIs to handle communication and implement business logic. User authentication and role-based access control are managed via NextAuth, ensuring secure and streamlined login processes. A relational database serves as the backbone, storing user data, goals, reviews, and performance metrics to facilitate efficient data retrieval and analysis. This architecture enables modular development, easy maintenance, and robust data management.

4.6 Summary

The Employee Performance Management and Review System offers a comprehensive solution to streamline the performance review process for organizations. By integrating features such as role-based access control, goal management, review scheduling, feedback mechanisms, and reporting tools, the system enhances employee engagement and fosters continuous performance improvement. Its robust architecture ensures a secure, user-friendly, and scalable platform, supporting data-driven decision-making and organizational development.

Chapter 5 - Analysis and Design

5.1 Use Case Diagram

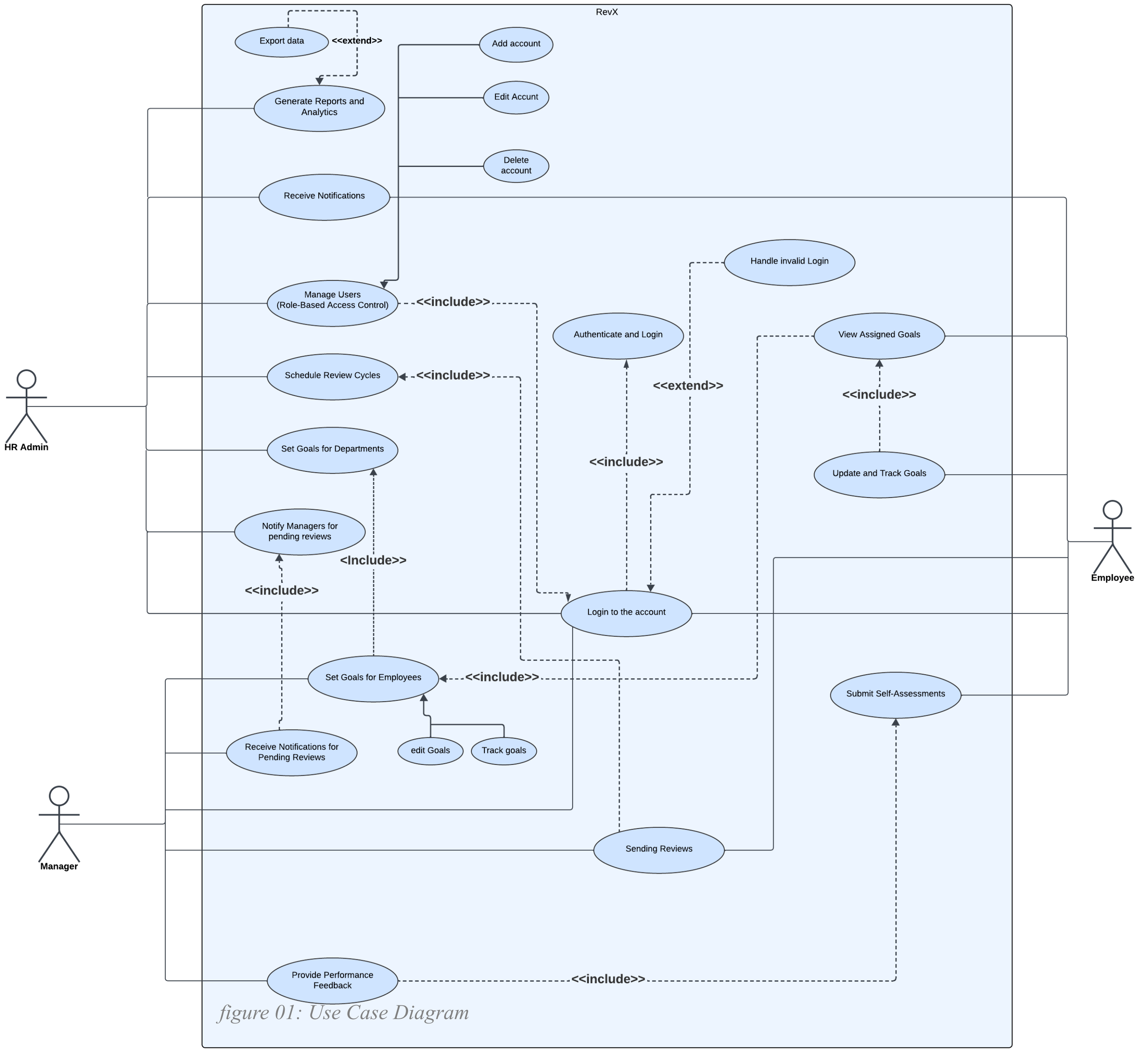


figure 01: Use Case Diagram

5.2 Activity Diagrams

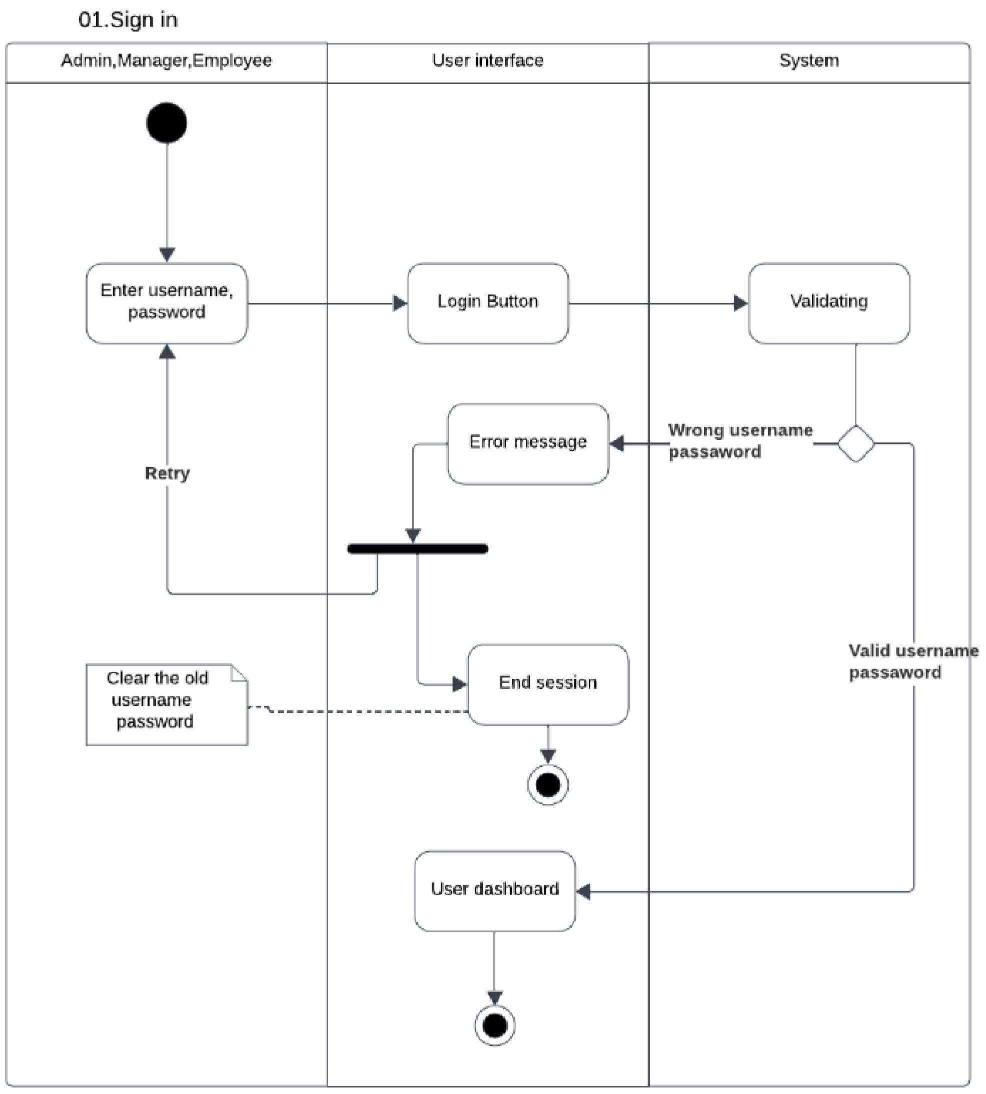


figure 02: Activity Diagram - Login

02.Manage Users (Role-Based Access Control)

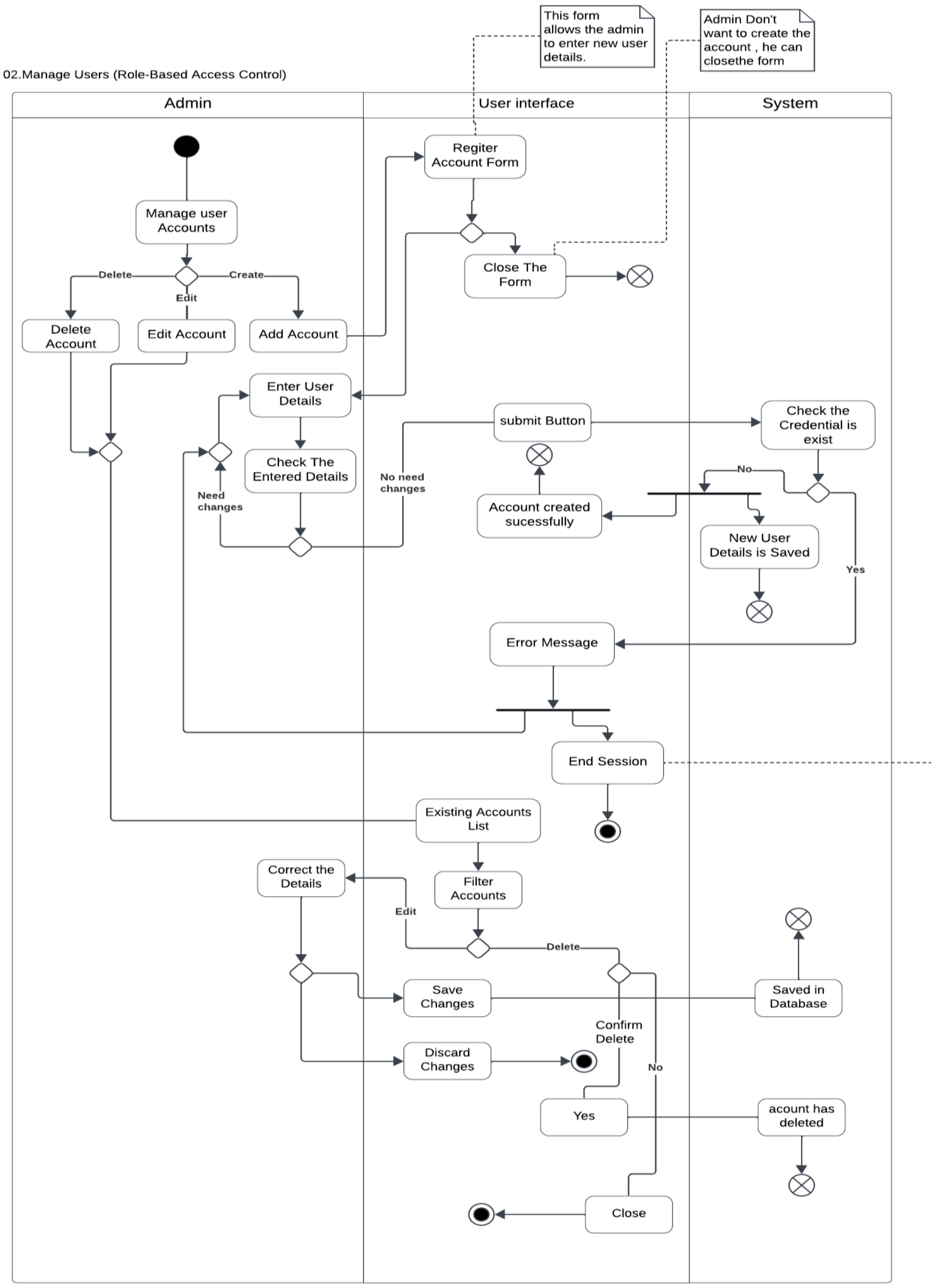


figure 03: Activity Diagram – HR Admin Mange User Account

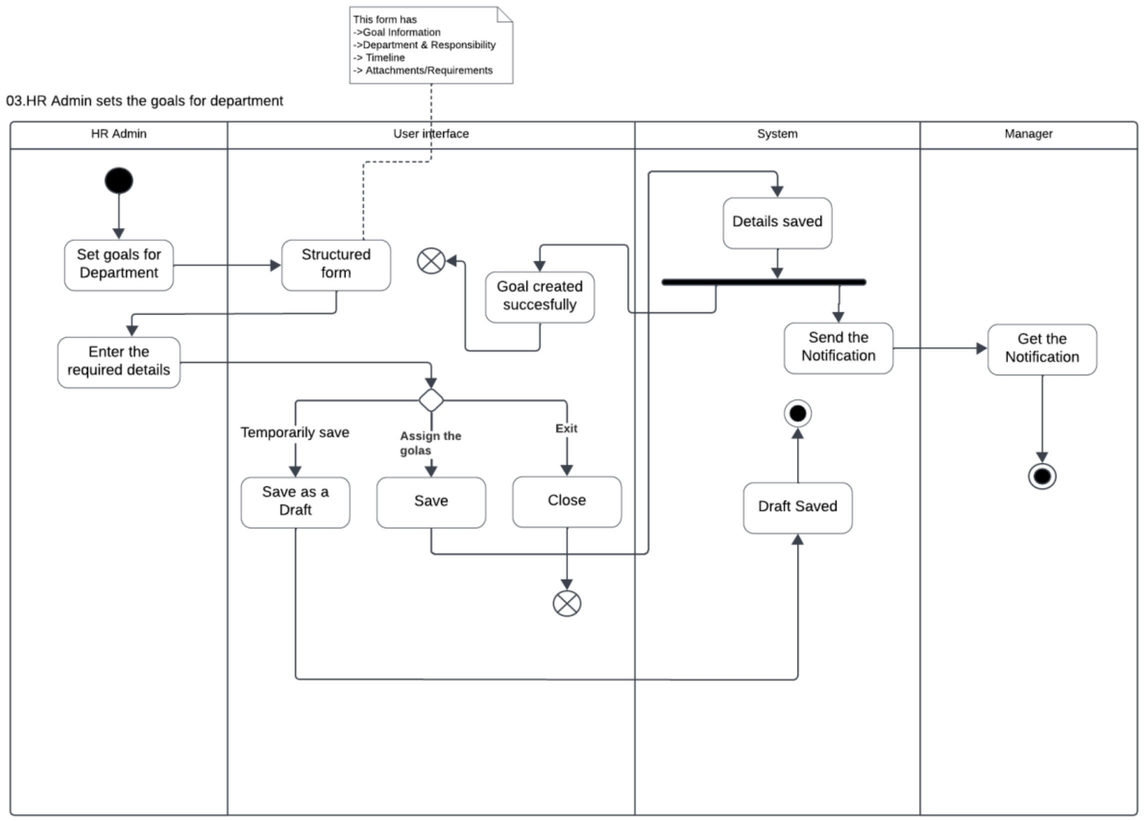


figure 04: Activity Diagram – HR Admin Sets Goals

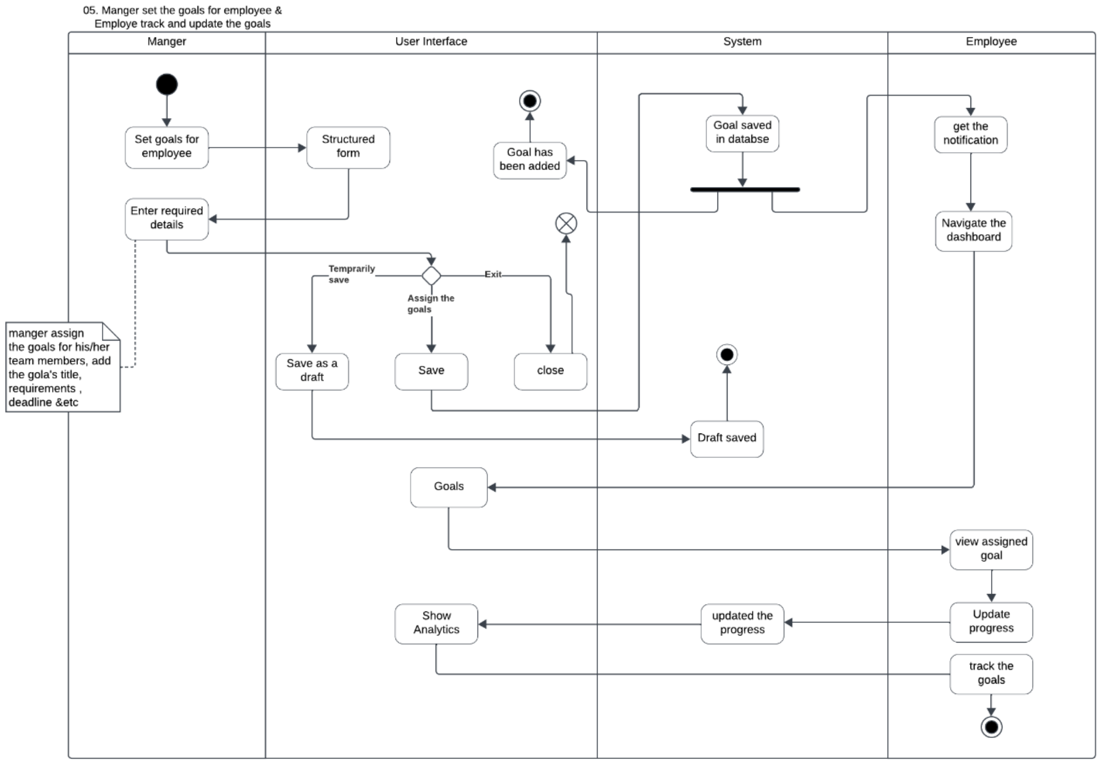


figure 05: Activity Diagram – Manger Sets Goals for Employee

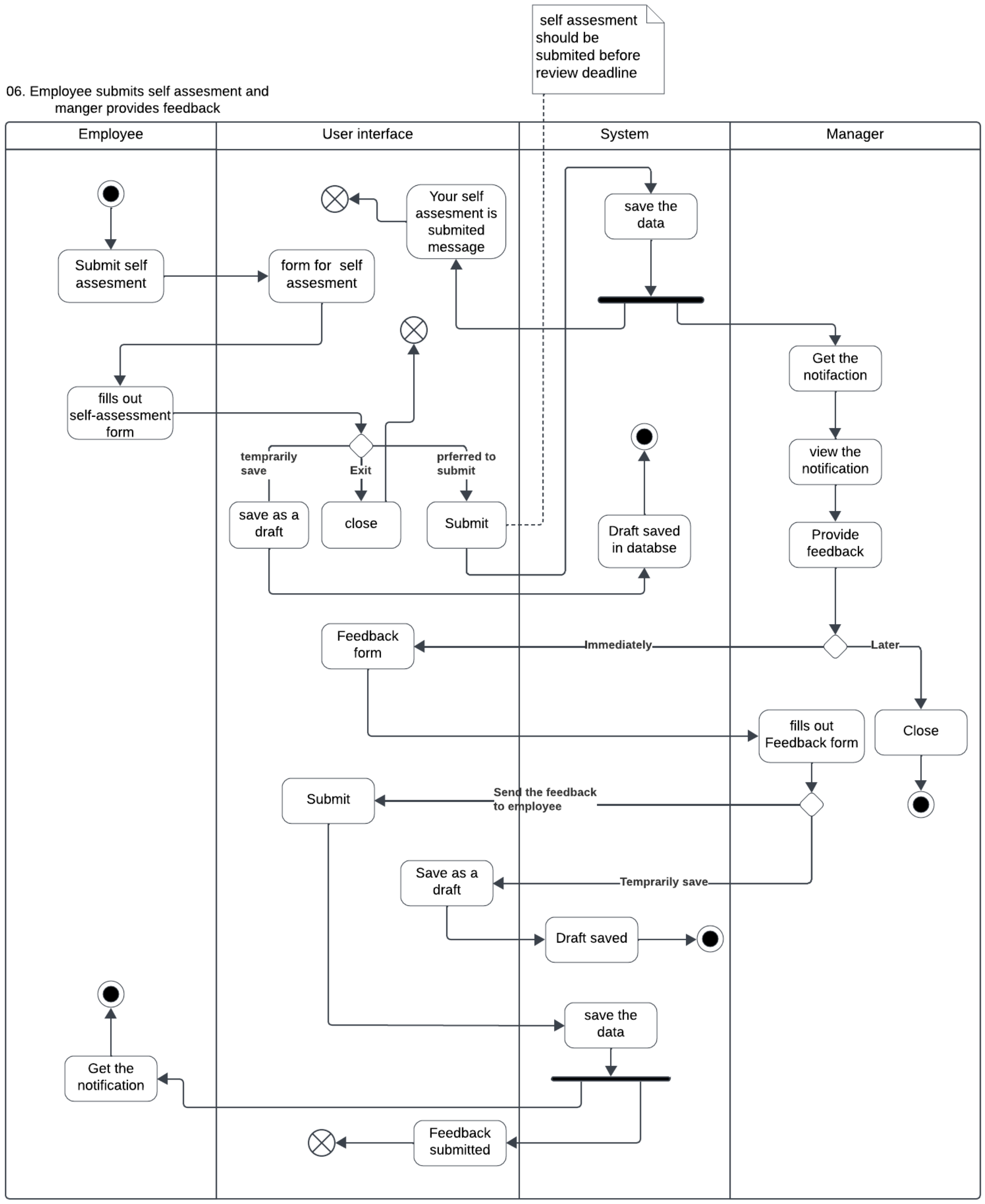


figure 06,07: Activity Diagram – Feedback and Self-Assessment

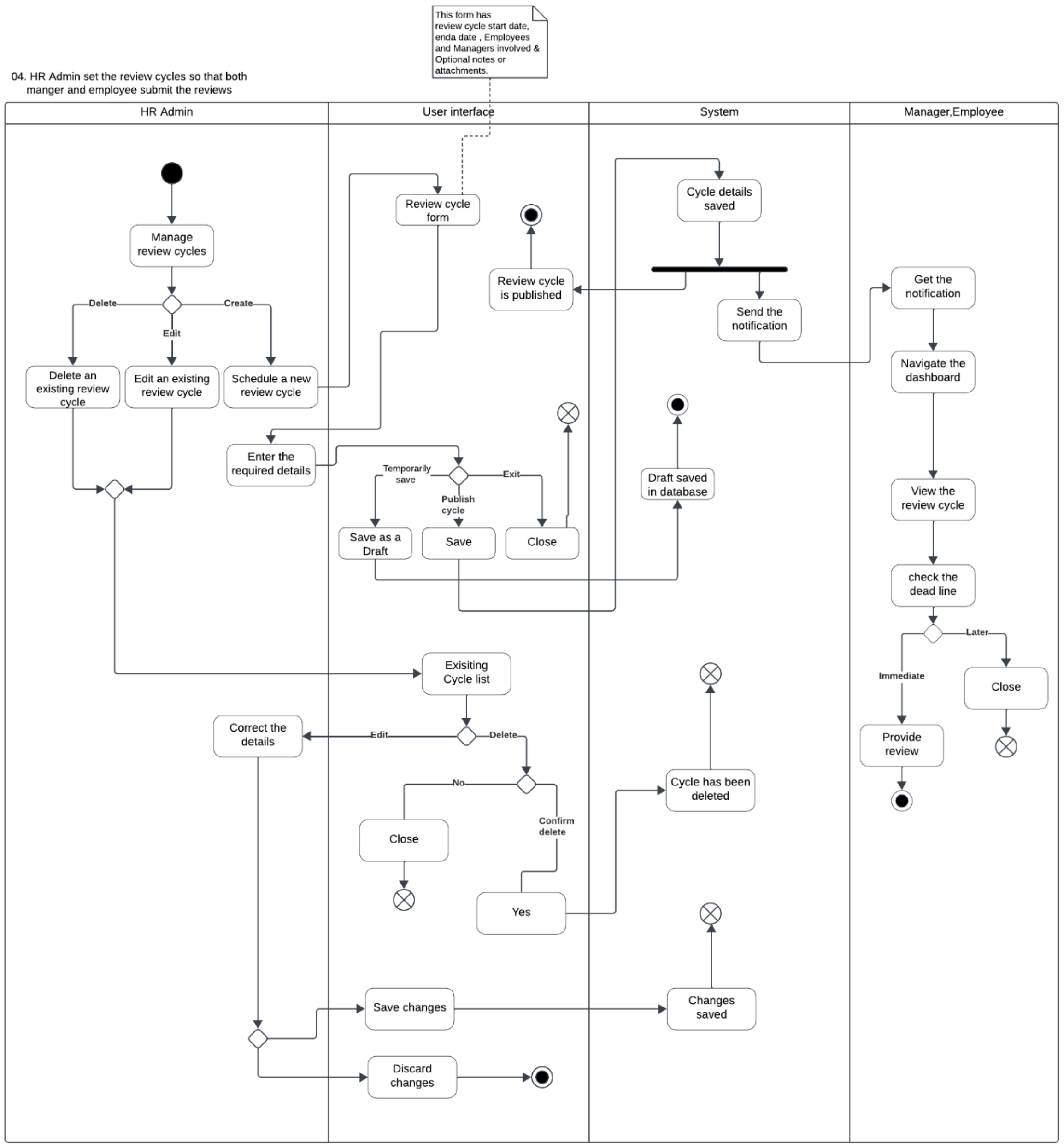


figure 08: Activity Diagram – HR Admin Sets Review Cycle ; Employee and Manager submit Review

07. HR Admin generate the report

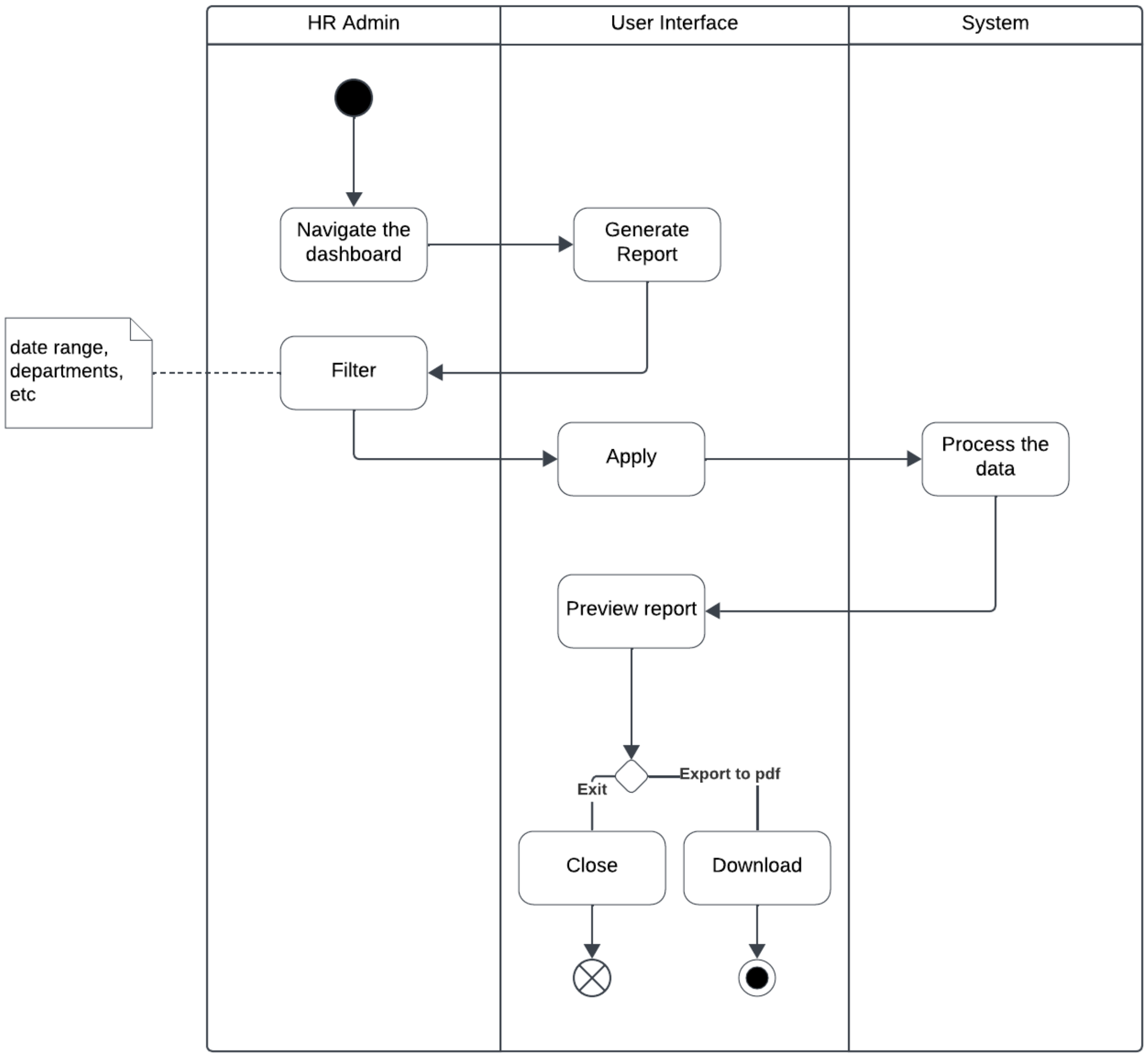


figure 09: Activity Diagram – HR Generate Reports

5.3 Class Diagram

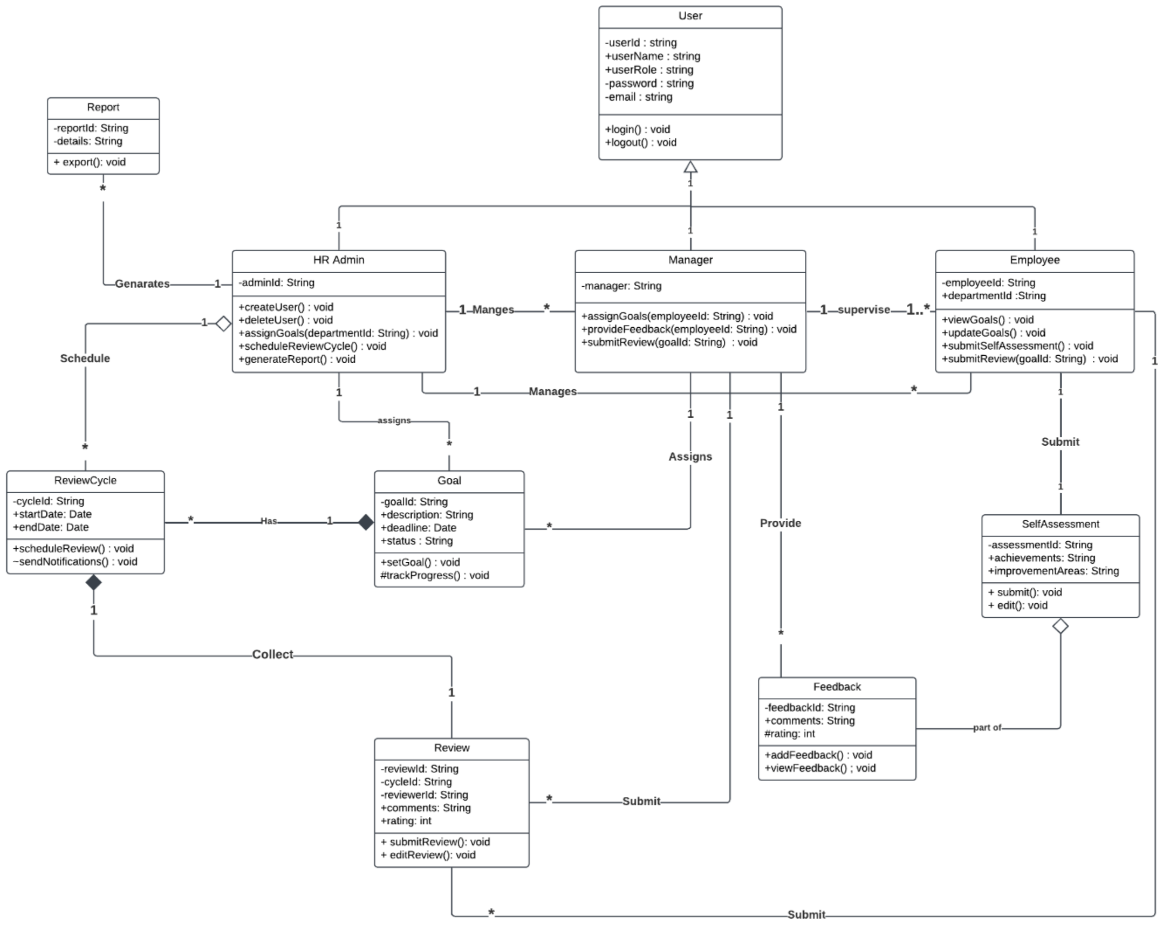


Figure 10: Class Diagram

5.4 Sequence Diagrams

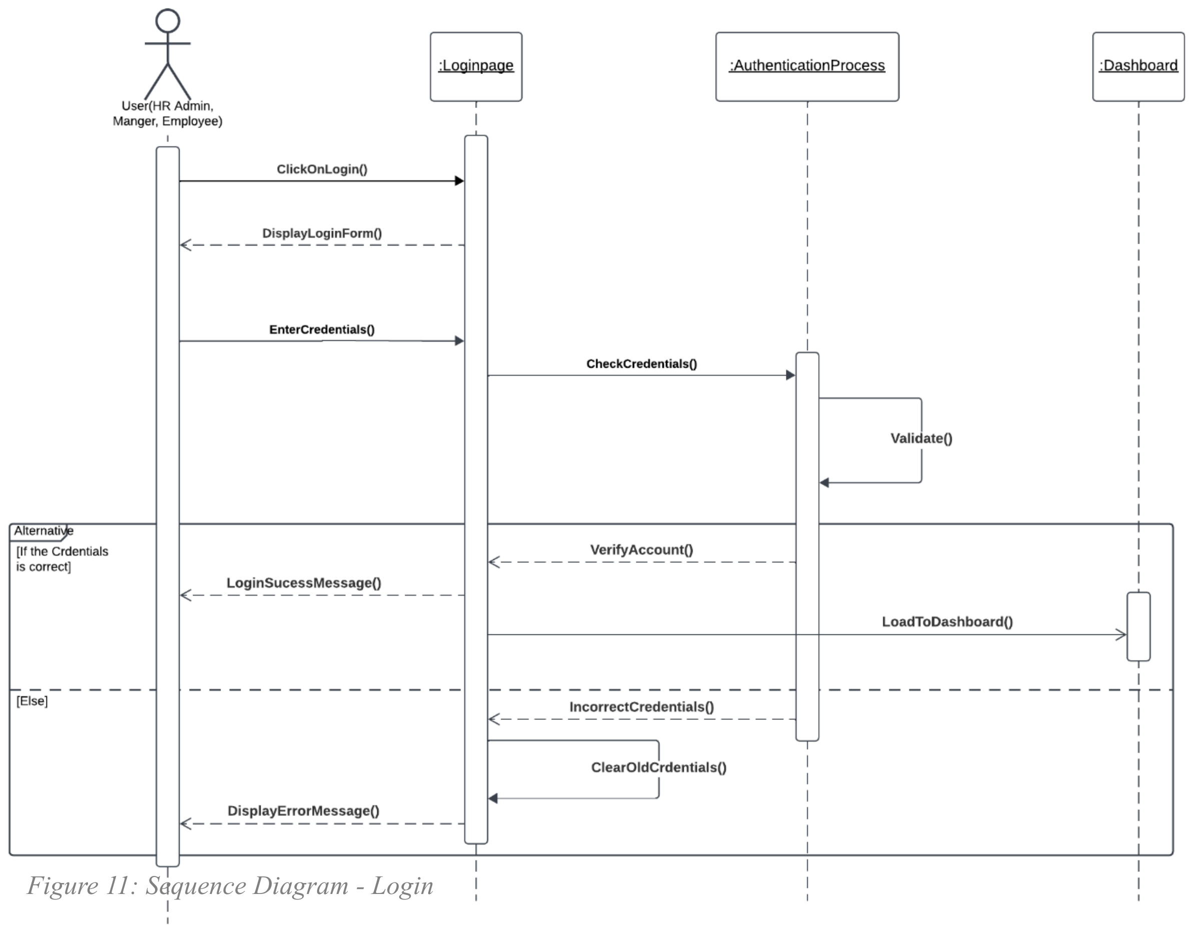


Figure 11: Sequence Diagram - Login

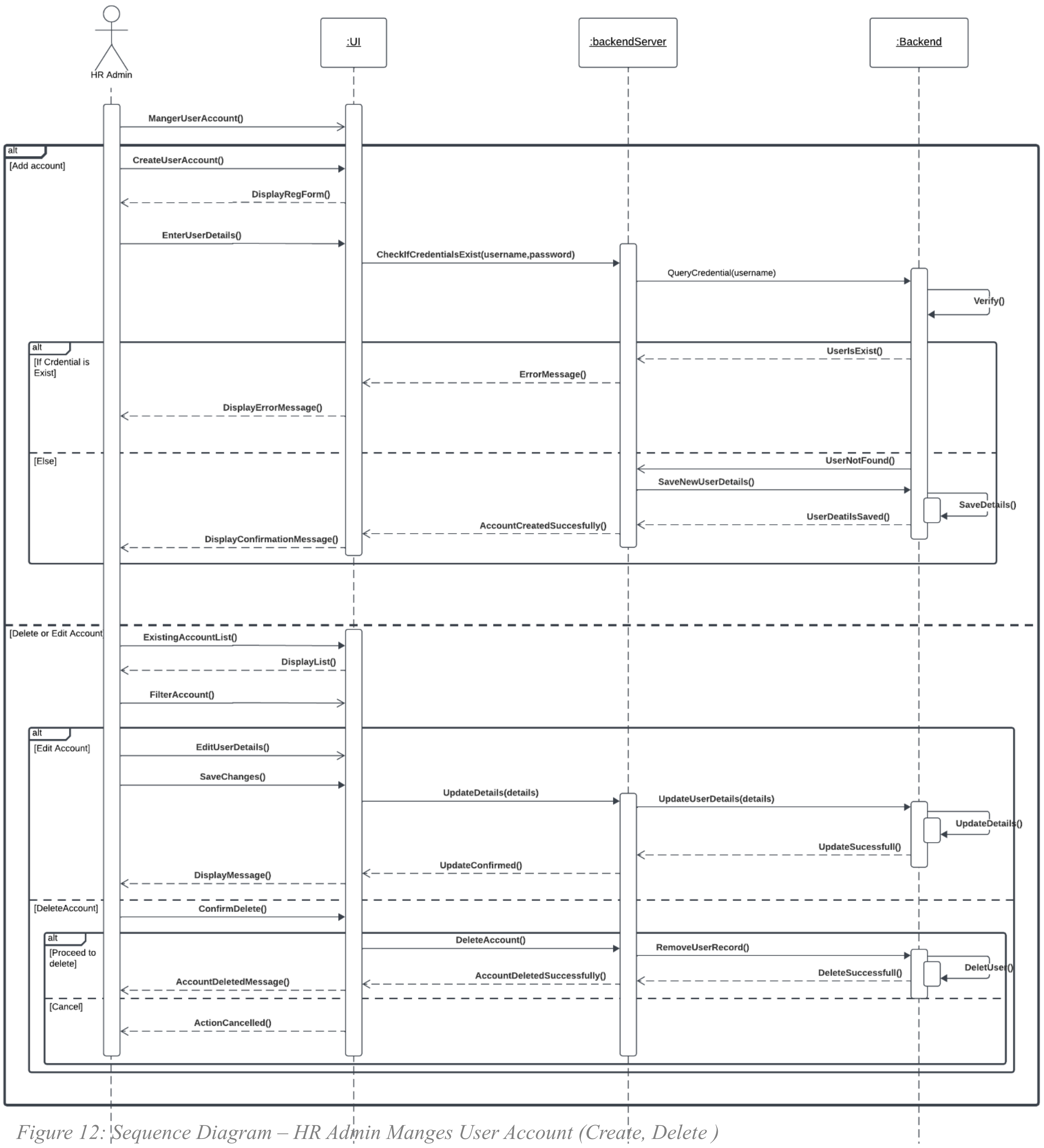


Figure 12: Sequence Diagram – HR Admin Manges User Account (Create, Delete)

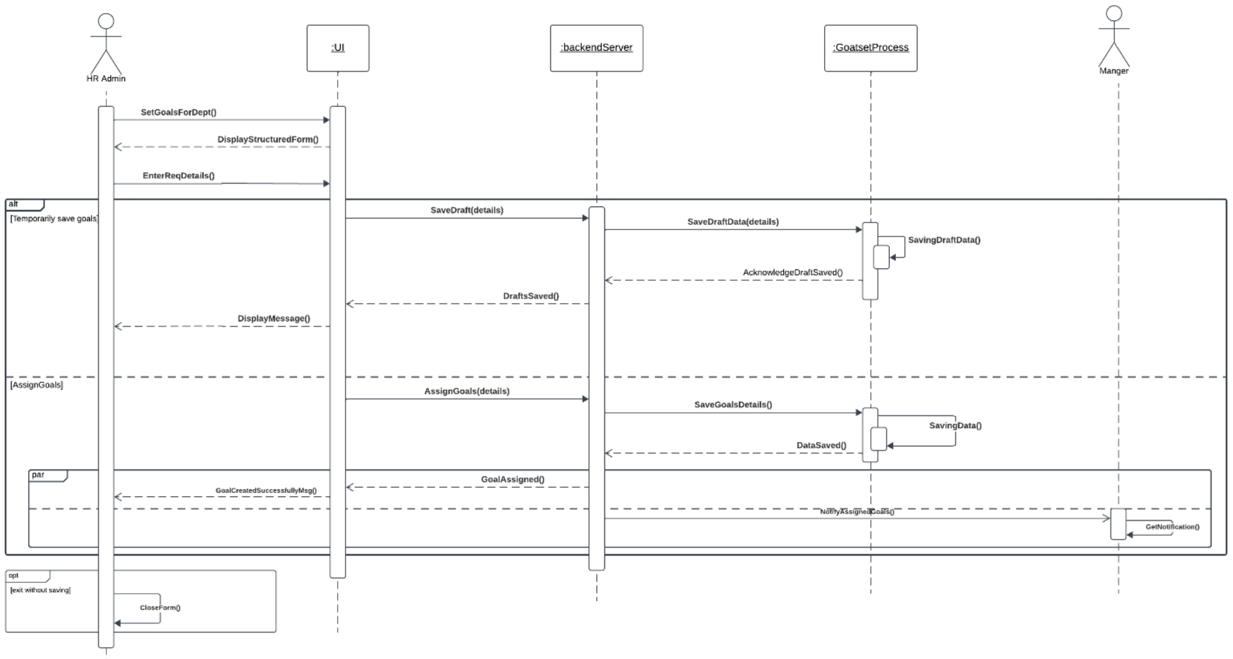


Figure 13: Sequence Diagram – HR Admin Sets Goals

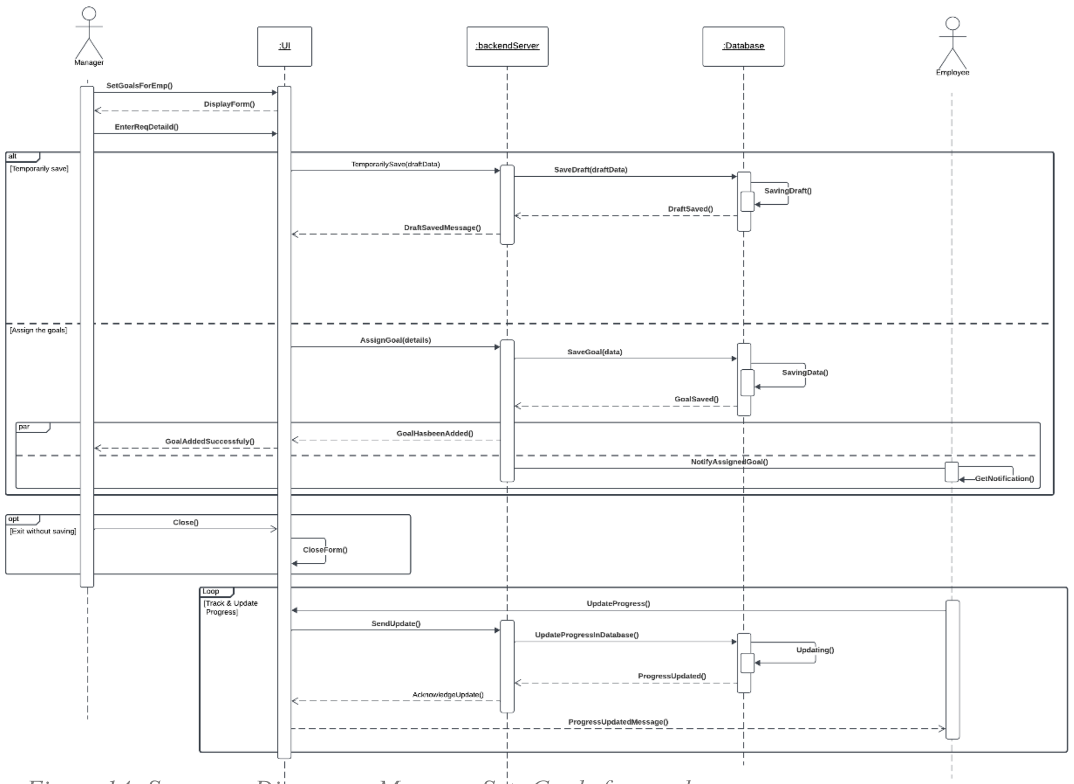


Figure 14: Sequence Diagram – Manager Sets Goals for employee

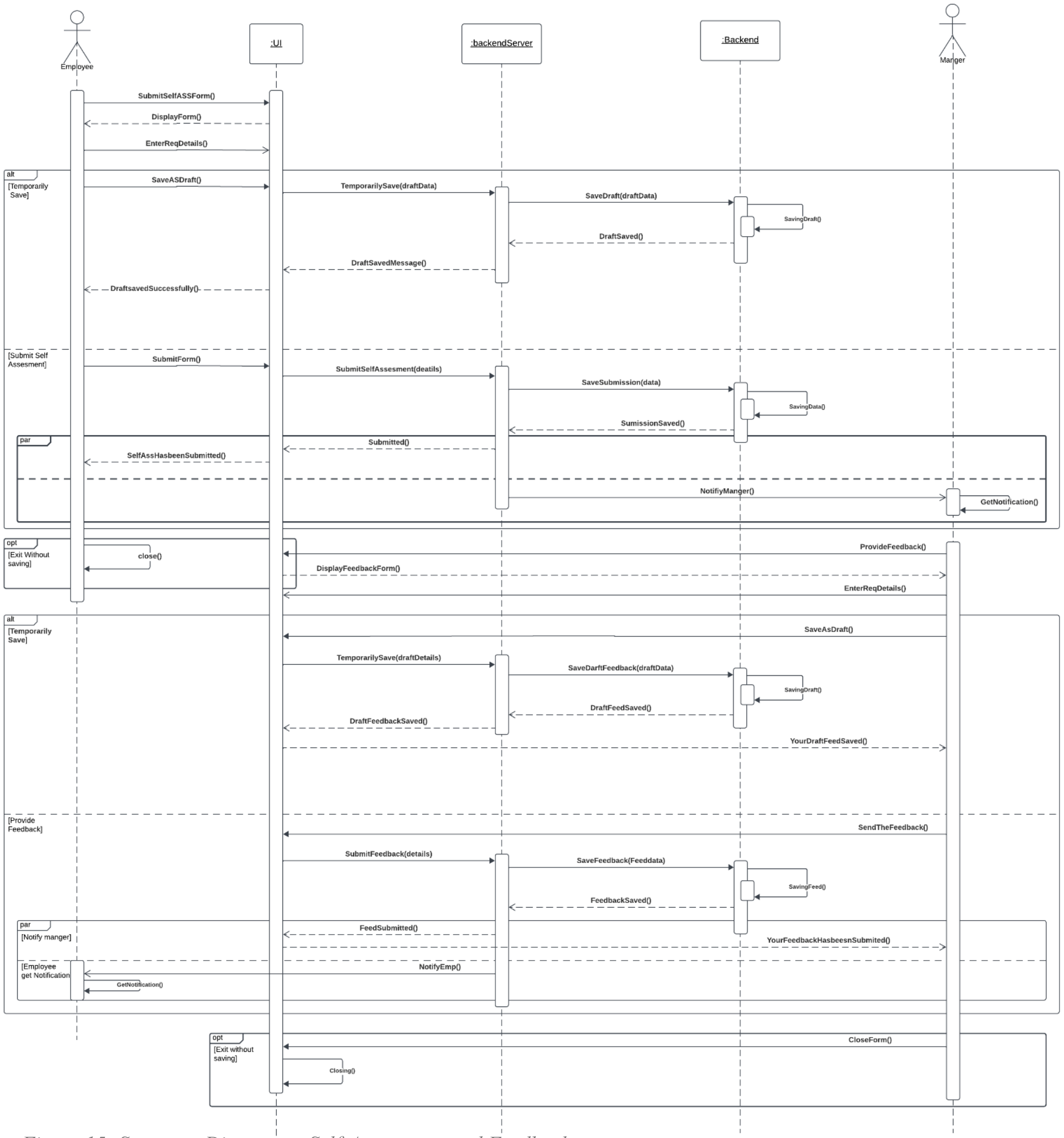


Figure 15: Sequence Diagram – Self-Assessment and Feedback

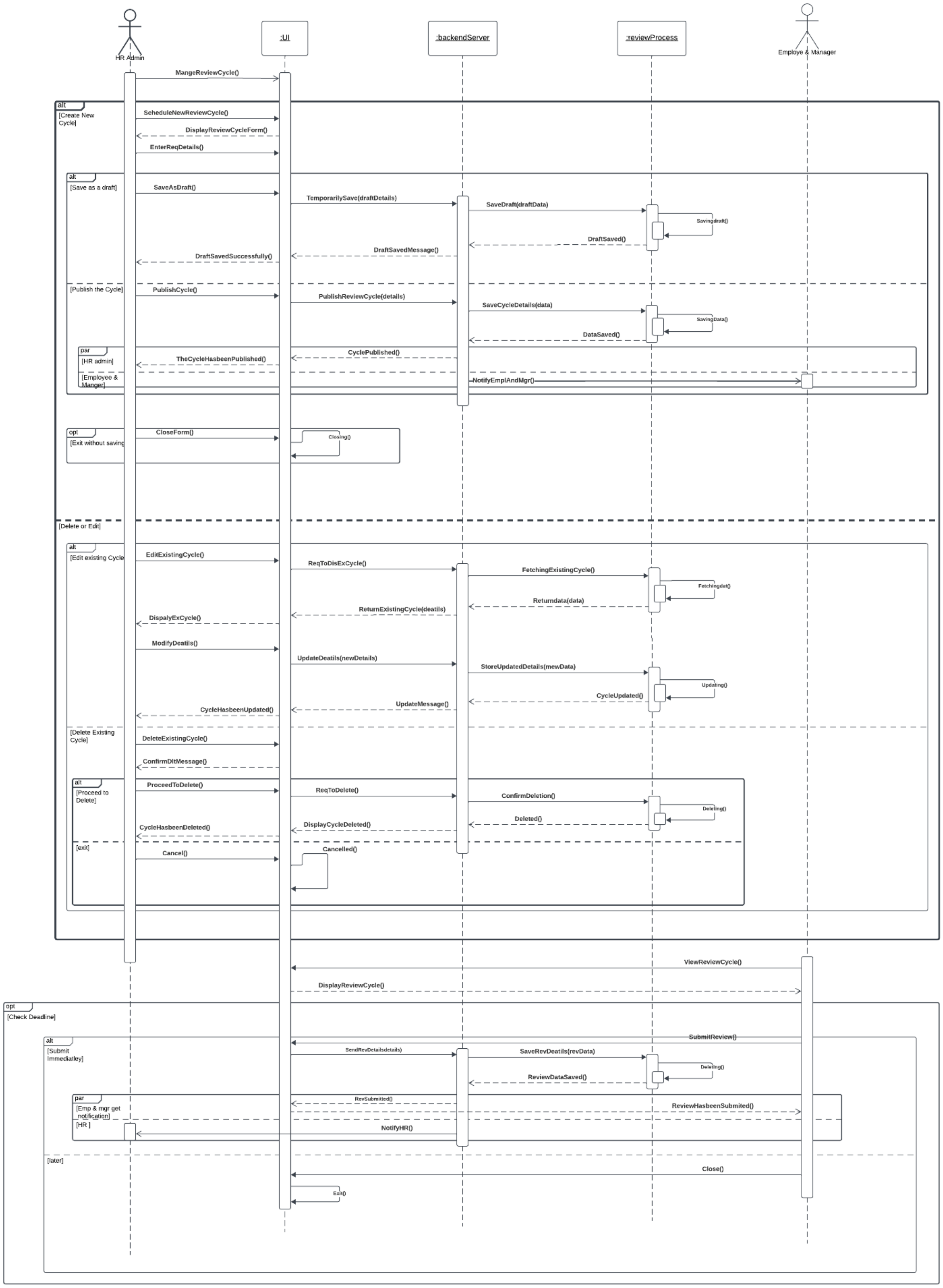
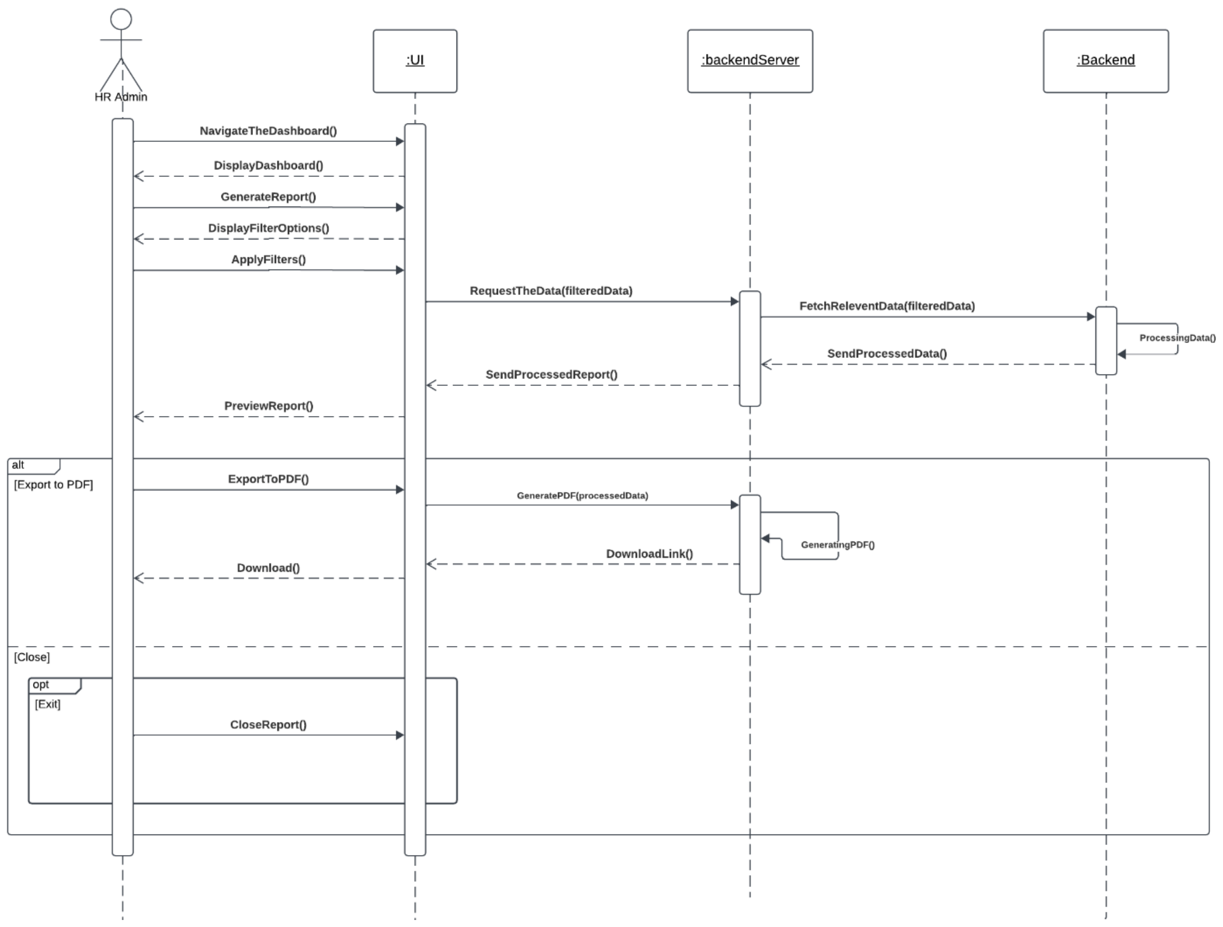
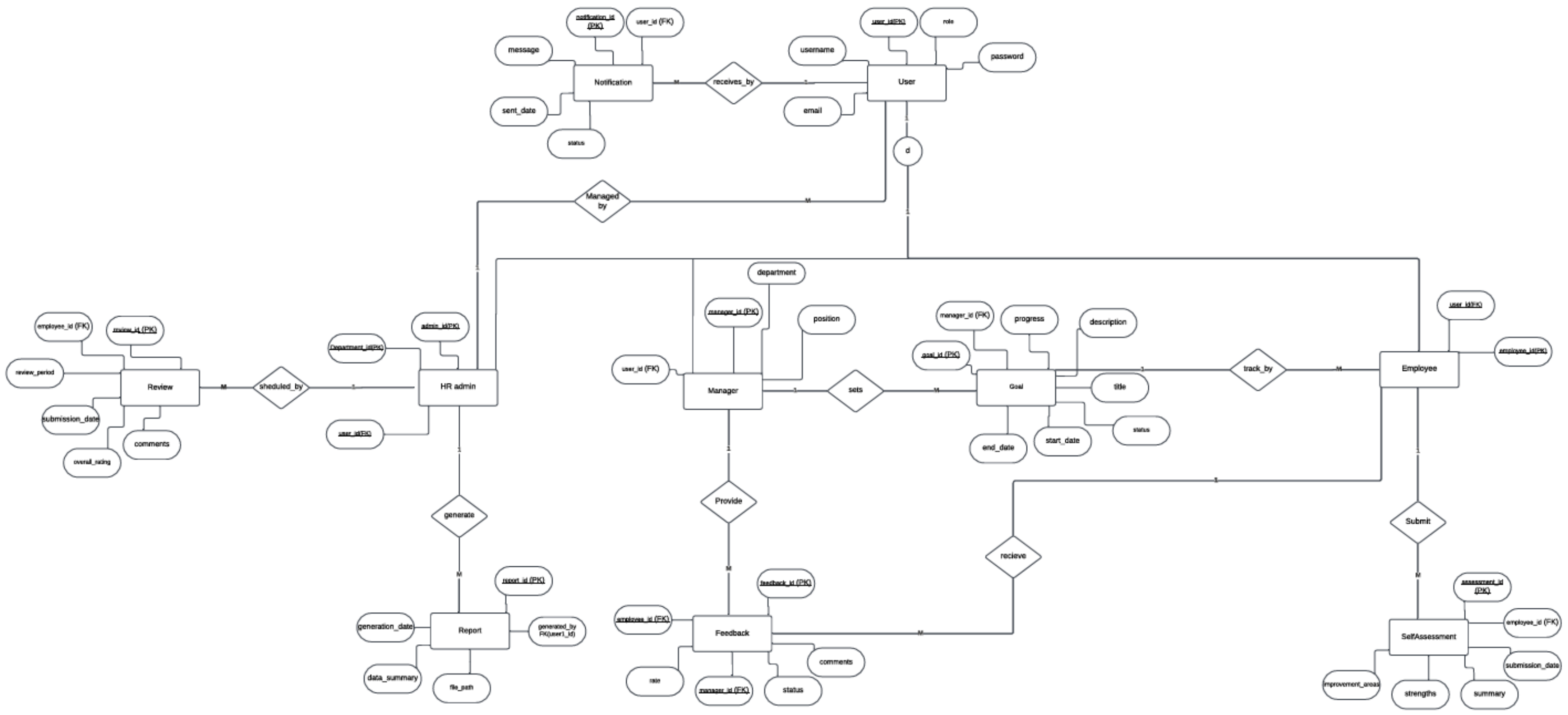


Figure 16 & 17: Sequence Diagram – Review Cycle

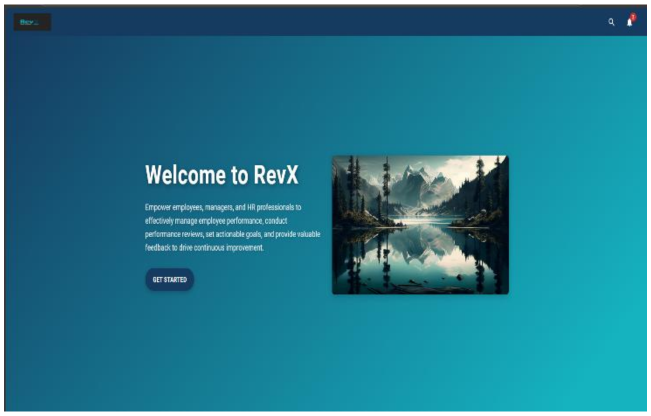

HR Admin
:UI
:backendServer
:Backend
NavigateTheDashboard()
DisplayDashboard()
GenerateReport()
DisplayFilterOptions()
ApplyFilters()
RequestTheData(filteredData)
FetchReleventData(filteredData)
ProcessingData()
SendProcessedData()
SendProcessedReport()
PreviewReport()
alt
[Export to PDF]
ExportToPDF()
GeneratePDF(processedData)
GeneratingPDF()
DownloadLink()
Download()
[Close]
opt
[Exit]
CloseReport()

Employee Management System

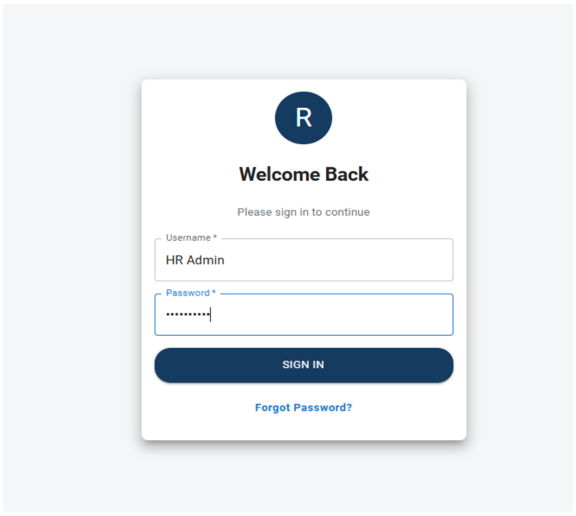


Chapter 6 – Implementation

6.1 Login Page


Welcome to RevX

Empower employees, managers, and HR professionals to effectively manage employee performance, conduct performance reviews, set actionable goals, and provide valuable feedback to drive continuous improvement.

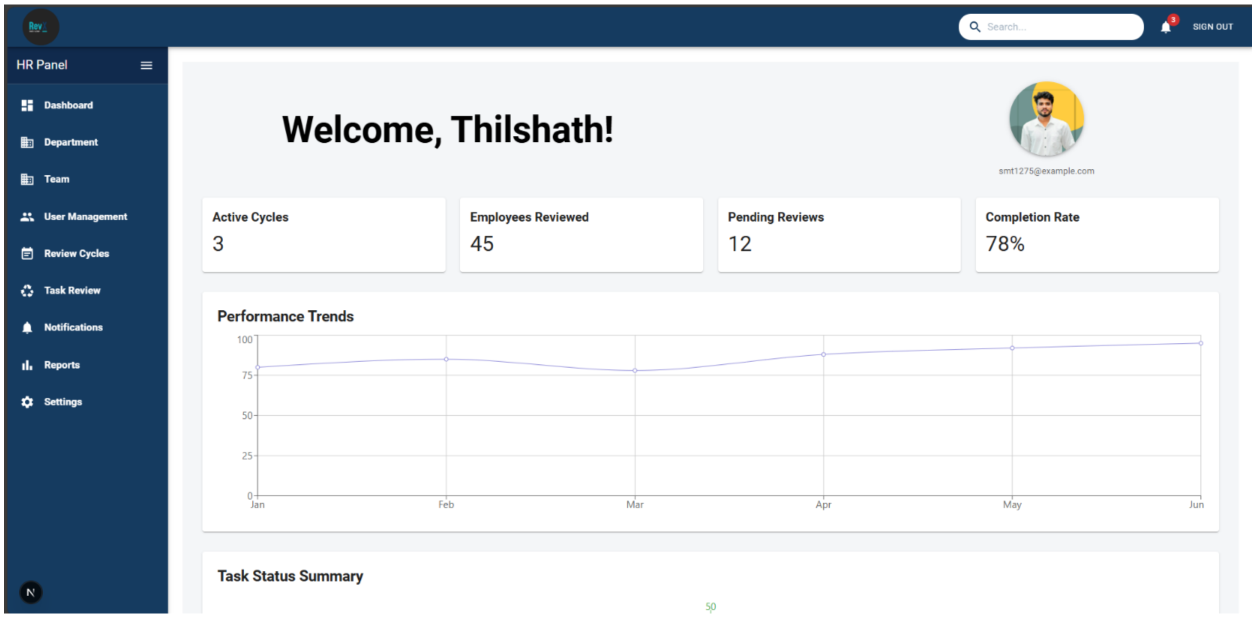


The system includes a secure **sign-in page** to authenticate users and manage their access based on predefined roles. Only **HR personnel** can register new users (employees and managers) through the sign-up interface, ensuring controlled access to the system. During sign-in, the application uses **role-based authentication** to redirect users to their respective dashboards based on their roles (HR, Manager, or Employee). This structure ensures that each user only interacts with features relevant to their role, maintaining system security and usability.

Front-end code snippet

```
import NextAuth from "next-auth";
import CredentialsProvider from "next-auth/providers/credentials";
import axios from "axios";

export default NextAuth({
  providers: [
    CredentialsProvider({
      async authorize(credentials) {
        const response = await axios.post(`${process.env.NEXT_PUBLIC_BACKEND_URL}/api/auth/login`, {
          username: credentials.username,
          password: credentials.password,
        });

        const { data } = response;

        if (data?.token && data?.role) {
          return {
            id: data.id,
            username: data.username,
            role: data.role,
            token: data.token,
          };
        }

        throw new Error("Invalid credentials");
      },
    }),
  ],
```

6.2 HR Admin's Layout

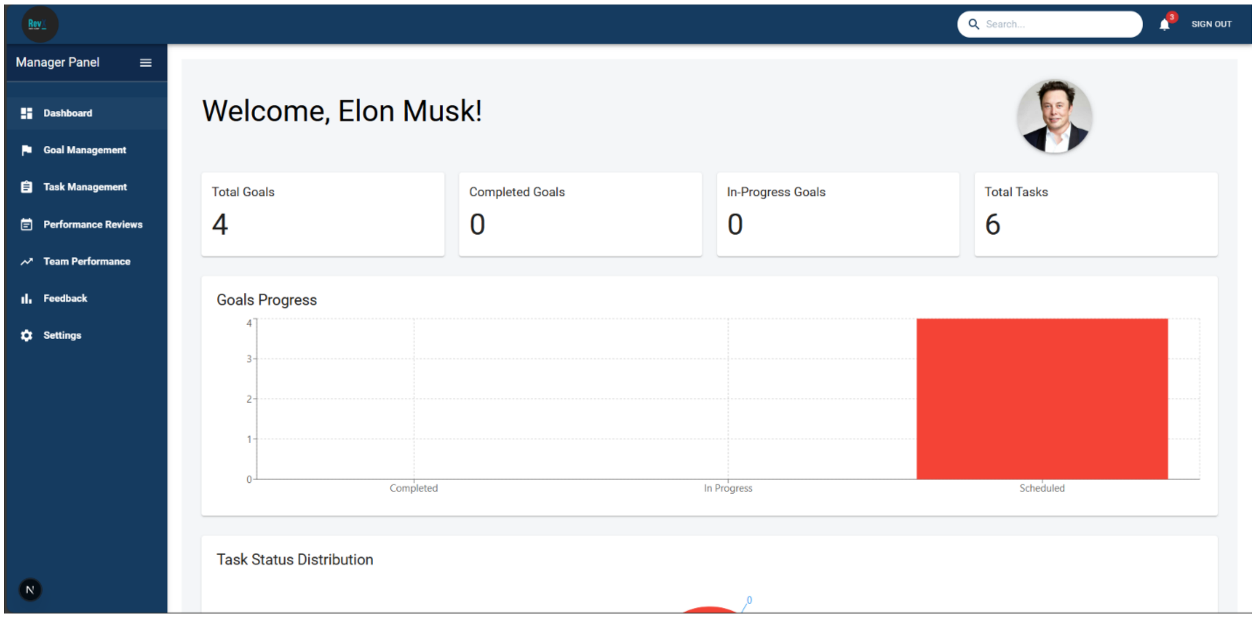


The HR's Layout serves as the central interface for Human Resource administrators, providing easy access to all the core functionalities required to manage and monitor the performance review process across the organization. From this dashboard, HR personnel can navigate through various modules such as Dashboard, Department, Teams, Review Cycles, Task Review, Notifications, Reports, Analytics, User Management, and Settings. These menus collectively enable HR to organize the company structure, schedule and manage review periods, handle user roles, track task progress, send system notifications, view reports, analyze performance data, and manage their own profile settings. Each of these sections plays a critical role in ensuring a smooth, structured, and efficient performance management workflow. Detailed descriptions of these features are provided in the following sections.

Front-end code snippet

```
const menuItems = [
  { text: "Dashboard", icon: <DashboardIcon />, path: "/hr" },
  { text: "Department", icon: <DomainIcon />, path: "/hr/department" },
  { text: "Team", icon: <DomainIcon />, path: "/hr/team" },
  { text: "User Management", icon: <PeopleIcon />, path: "/hr/user-management" },
  { text: "Review Cycles", icon: <EventNoteIcon />, path: "/hr/goalReviews" },
  { text: "Task Review", icon: <RecyclingIcon />, path: "/hr/taskReviews" },
  { text: "Notifications", icon: <NotificationsIcon />, path: "/hr/notification  " },
  { text: "Reports", icon: <BarChartIcon />, path: "/hr/reports" },
  { text: "Settings", icon: <SettingsIcon />, path: "/profile/profile" },
];

const HRSidebar = () => {
  const router = useRouter();
  const [open, setOpen] = useState(true);

  const handleNavigation = (path) => {
    router.push(path);
  };
```

6.3 Manager's Layout

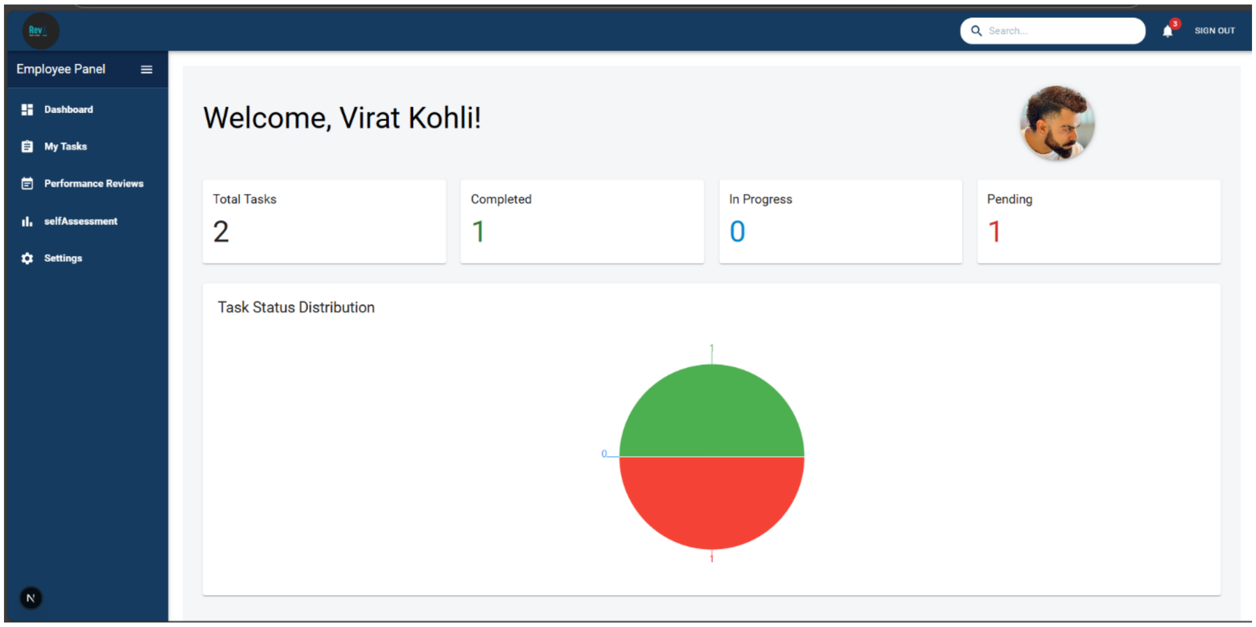


Manager Layout serves as a dedicated workspace for managers, providing access to all essential features required for managing their team's performance within the system. The layout includes navigation menus such as Dashboard, Goal Management, Task Management, Performance Reviews, Team Performance, Feedback, and Settings. This structured layout enables managers to seamlessly switch between different functions like setting goals, assigning tasks, reviewing employee performance, and monitoring team progress. The intuitive design ensures that managers can efficiently carry out their responsibilities and maintain effective communication and feedback with their team members. A detailed breakdown of each menu item is presented in the upcoming sections.

Front-end code snippet

```
const menuItems = [
  { text: "Dashboard", icon: <DashboardIcon />, path: "/manager" },
  { text: "Goal Management", icon: <FlagIcon />, path: "/manager/goals" },
  { text: "Task Management", icon: <AssignmentIcon />, path: "/manager/tasks" },
  { text: "Performance Reviews", icon: <EventNoteIcon />, path: "/manager/goalReviews" },
  { text: "Team Performance", icon: <TrendingUpIcon />, path: "/manager/performance" },
  { text: "Feedback", icon: <BarChartIcon />, path: "/manager/feedback" },
  { text: "Settings", icon: <SettingsIcon />, path: "/profile/profile" },
];

const ManagerSidebar = () => {
  const router = useRouter();
  const [open, setOpen] = useState(true);

  const handleNavigation = (path) => {
    router.push(path);
  };
```

6.4 Employee's Layout



Welcome, Virat Kohli!

Total Tasks 2

Completed 1

In Progress 0

Pending 1

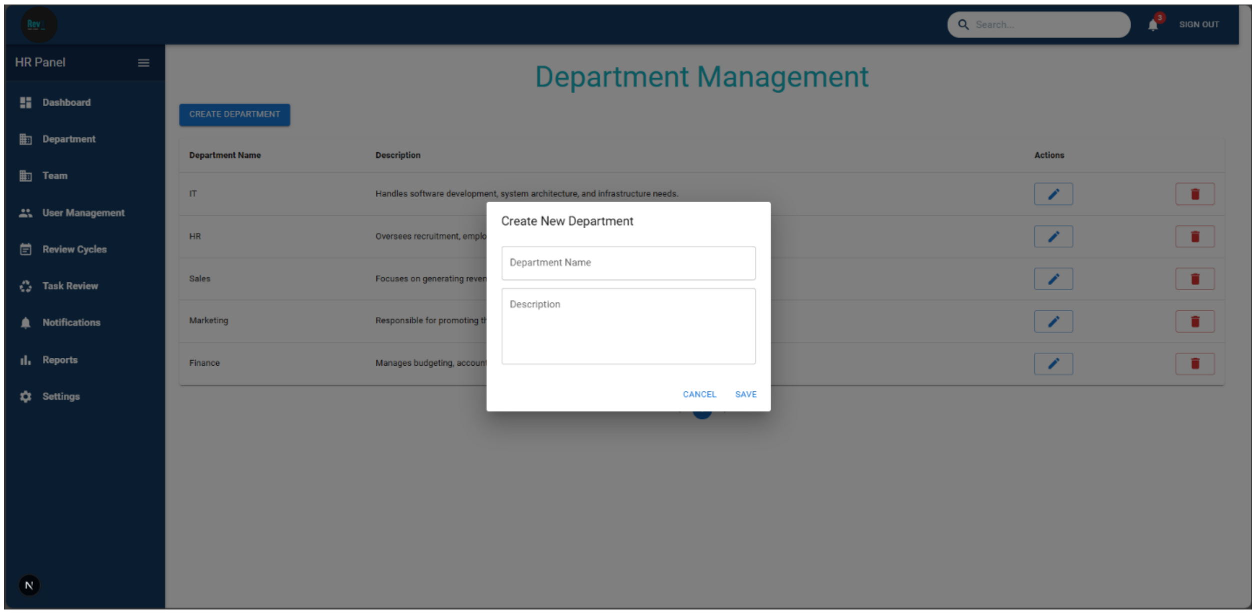
Task Status Distribution

The **Employee Layout** provides a user-friendly interface tailored for employees to manage and track their individual performance-related activities. The layout includes key menu items such as **Dashboard**, **My Tasks**, **Performance Reviews**, **Self-Assessment**, and **Settings**. This layout allows employees to view assigned tasks, submit self-assessments, track their performance review status, and stay informed about their progress. The clean and structured design helps employees remain focused on their goals and actively participate in the performance management process. Further details of each feature are described in the following sections.

Front-end code snippet

```
const menuItems = [
  { text: "Dashboard", icon: <DashboardIcon />, path: "/employee" },
  { text: "My Tasks", icon: <AssignmentIcon />, path: "/employee/tasks" },
  { text: "Performance Reviews", icon: <EventNoteIcon />, path: "/employee/taskReviews" },
  { text: "selfAssessment", icon: <BarChartIcon />, path: "/employee/SelfAssessment" },
  { text: "Settings", icon: <SettingsIcon />, path: "/profile/profile" },
];

const EmployeeSidebar = () => {
  const router = useRouter();
  const [open, setOpen] = useState(true);

  const handleNavigation = (path) => {
    router.push(path);
  };
```

6.5 HR Creates Departments



Department Name	Description	Actions
IT	Handles software development, system architecture, and infrastructure needs.	
HR	Oversees recruitment, emplo	
Sales	Focuses on generating reven	
Marketing	Responsible for promoting th	
Finance	Manages budgeting, accoun	

Create New Department

Department Name

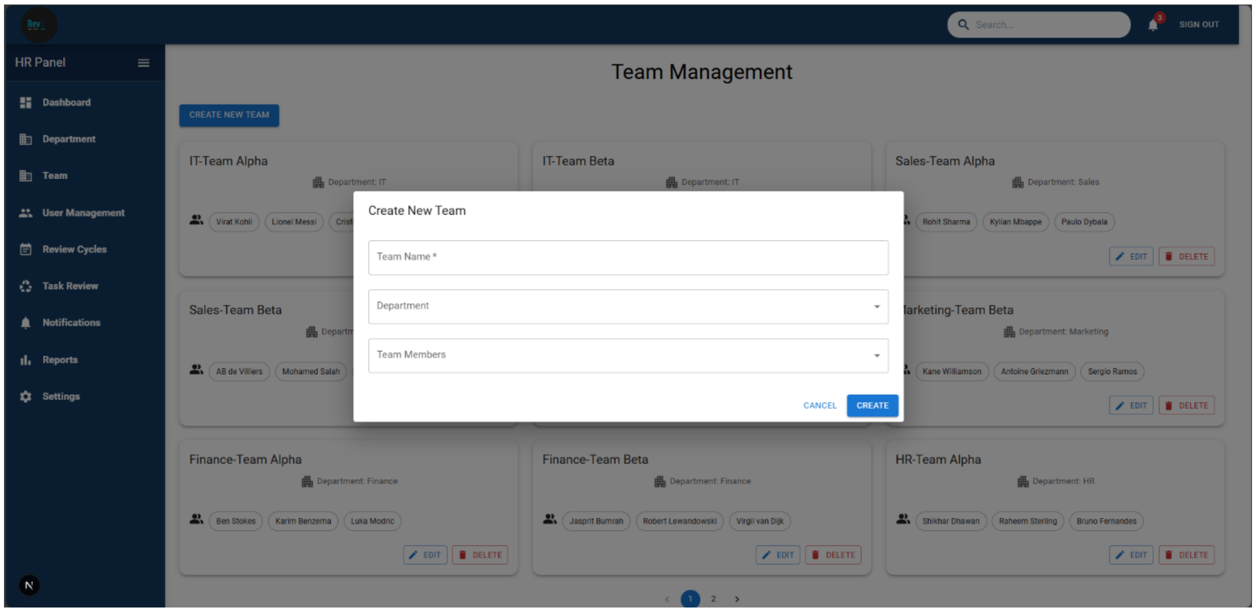
Description

HR personnel have the ability to create and manage departments within the system. Each department serves as a structural unit of the organization, allowing HR to assign managers and group employees accordingly. This helps maintain organizational hierarchy and simplifies performance tracking by department.

back-end code snippet

```
const Department = require('../models/Department');

// Create a new department
exports.createDepartment = async (req, res) => {
  const { departmentName, description } = req.body;

  try {
    const newDepartment = new Department({
      departmentName,
      description,
      createdBy: req.user.id, // User (likely HR) who creates the department
    });

    await newDepartment.save();
    res.status(201).json({ message: 'Department created successfully', department: newDepartment });
  } catch (error) {
    res.status(500).json({ message: 'Error creating department', error });
  }
};
```

6.6 HR Creates Teams

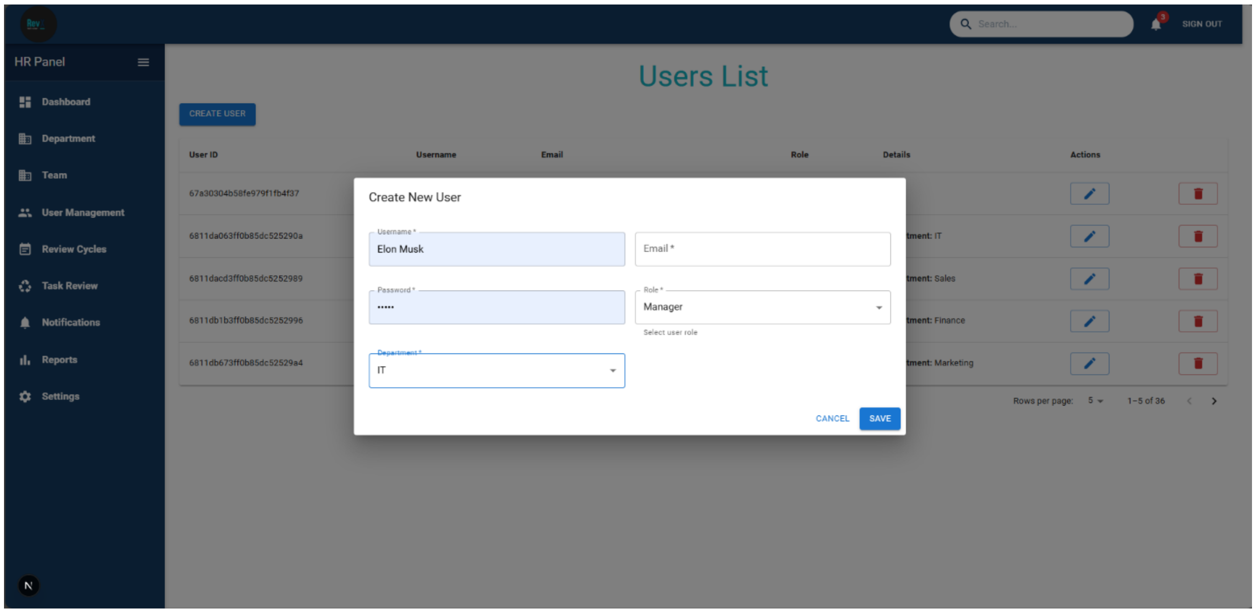


HR can create multiple teams under each department to organize employees **based on roles or project needs**. while creating a team, HR selects relevant employees to be part of the team. This structure ensures clear reporting lines and helps in managing performance at the team level.

back-end code snippet

```
const Team = require('../models/Team');
const User = require('../models/User');
const Department = require('../models/Department');
const mongoose = require('mongoose');

// Create new Team with members
exports.createTeam = async (req, res) => {
    const { teamName, members = [], departmentId } = req.body; // Default to empty array

    try {
        // Validate members only if provided
        if (members.length > 0) {
            const validMembers = await User.find({ _id: { $in: members } });
            if (validMembers.length !== members.length) {
                return res.status(400).json({ message: 'Invalid member IDs' });
            }
        }
```

6.7 HR Creates User – Accounts



Users List

User ID	Username	Email	Role	Details	Actions
67a30304b58fe979f1fb4f37					
6811da063ff0b85dc525290a				tment: IT	
6811dacd3ff0b85dc5252989				tment: Sales	
6811db1b3ff0b85dc5252996				tment: Finance	
6811db673ff0b85dc52529a4				tment: Marketing	

Rows per page: 5 1–5 of 36

Create New User

Username * Elon Musk

Email *

Password * •••••

Role * Manager

Select user role

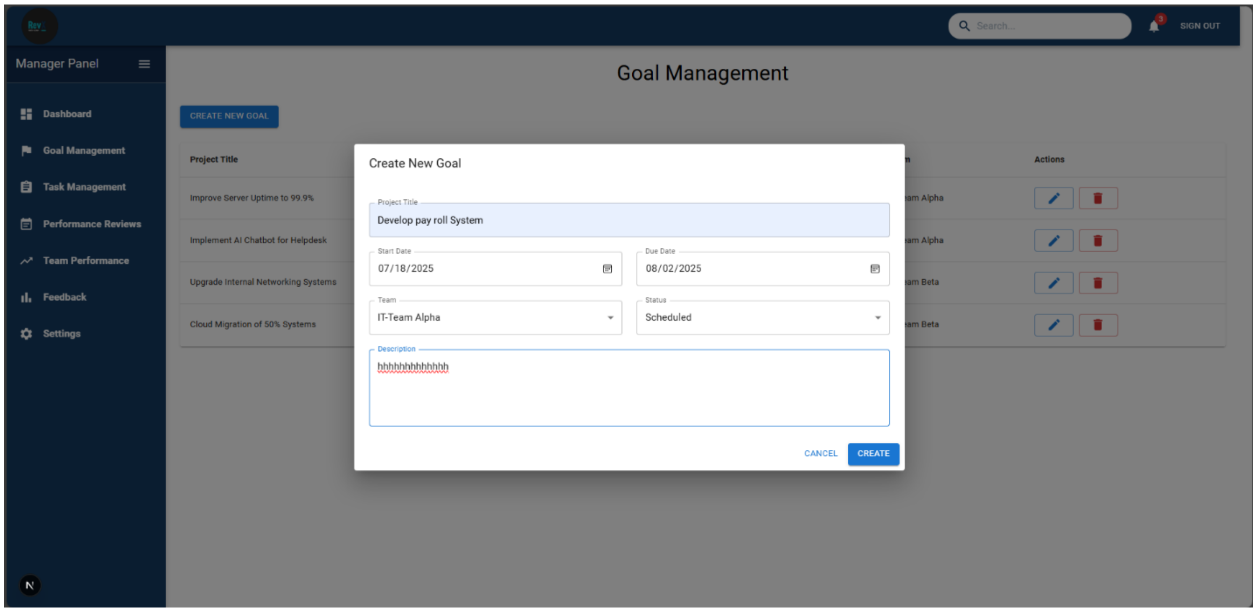
Department * IT

HR is responsible for creating user accounts for all system users, including managers, employees, and other HR personnel. During account creation, HR inputs the user's name, email, sets a password, and selects the appropriate role (Manager, Employee, or HR). Based on the selected role, additional fields appear , for example, when assigning the Manager role, HR can also select the department and team the manager will lead. This role-based account creation ensures each user receives proper access and responsibilities within the system.

back-end code snippet

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const nodemailer = require('nodemailer');
const User = require('../models/User');
const { sendPasswordResetEmail } = require('../utils/mailer'); // Optional if you're using your own
mailer
//Register a new user
exports.register = async (req, res) => {
  const { username, email, password, role, employeeDetails, managerDetails, hrDetails } = req.body;
    try {
    const existingUser = await User.findOne({ $or: [{ username }, { email }] });
    if (existingUser) {
      return res.status(400).json({ message: 'Username or Email already exists' });
    }

    if (!['employee', 'manager', 'hr'].includes(role)) {
      return res.status(400).json({ message: 'Invalid role specified' });
    }

    if (role === 'employee' && !employeeDetails) {
      return res.status(400).json({ message: 'Employee details are required for the employee role' });
    }
    if (role === 'manager' && !managerDetails) {
      return res.status(400).json({ message: 'Manager details are required for the manager role' });
    }
    if (role === 'hr' && !hrDetails) {
      return res.status(400).json({ message: 'HR details are required for the HR role' }); }
```

6.8 Manager Creates Goals for Teams



Goal Management

Project Title		Actions
Improve Server Uptime to 99.9%	am Alpha	
Implement AI Chatbot for Helpdesk	am Alpha	
Upgrade Internal Networking Systems	am Beta	
Cloud Migration of 50% Systems	am Beta	

Create New Goal

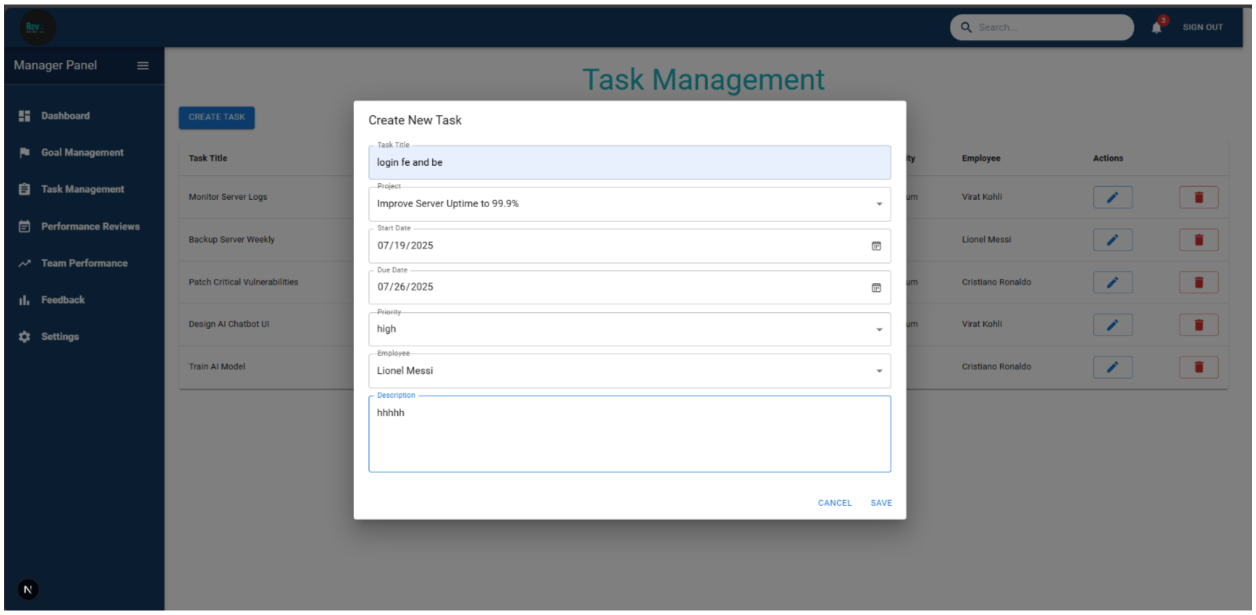
Project Title: Develop pay roll System
Start Date: 07/18/2025
Due Date: 08/02/2025
Team: IT-Team Alpha
Status: Scheduled
Description: hhhhhhhhhhhhh

Managers can create performance goals for their teams to align work efforts with organizational objectives. While creating a goal, the manager provides details such as the **project title**, **start date**, **due date**, selects the relevant **team**, and adds a **description** outlining the goal or project scope. These goals help guide team members' tasks and provide measurable targets for performance evaluation.

back-end code snippet

```
const Goal = require('../models/Goal');
const User = require('../models/User');
const Team = require('../models/Team');

// Manager creates goal for their team
exports.createGoal = async (req, res) => {
  const { projectTitle, startDate, dueDate, description, teamId } = req.body;

  try {
    // Validate manager exists
    const manager = await User.findById(req.user.id);
    if (!manager || manager.role !== 'manager') {
      return res.status(403).json({ message: 'Manager access required' });
    }

    // Validate team exists and manager is assigned
    const team = await Team.findOne({
      _id: teamId
    });
    if (!team) {
      return res.status(404).json({ message: 'Team not found under your management' });
    }
```

6.9 Manager Assigns tasks for employees



Create New Task

Task Title: login fe and be
Project: Improve Server Uptime to 99.9%
Start Date: 07/19/2025
Due Date: 07/26/2025
Priority: high
Employee: Lionel Messi
Description: hhhhh

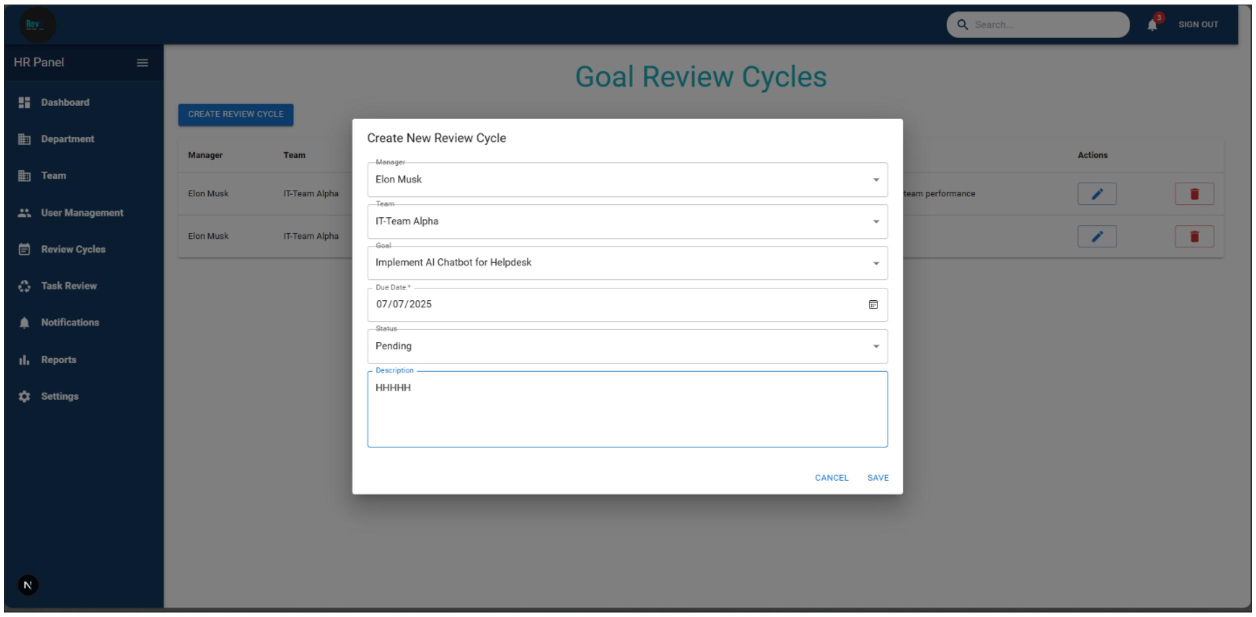
Task Title	Employee	Actions
Monitor Server Logs	Virat Kohli	
Backup Server Weekly	Lionel Messi	
Patch Critical Vulnerabilities	Cristiano Ronaldo	
Design AI Chatbot UI	Virat Kohli	
Train AI Model	Cristiano Ronaldo	

After creating goals, the manager can assign specific tasks to employees under those goals. While creating a task, the manager provides a **task title**, due date, selects a previously created **goal**, and chooses the relevant **team**. Based on the selected team, a **dropdown list of employees** appears, allowing the manager to assign the task to a specific employee. The manager can also include a **description** to provide additional context or instructions for the task. This structured approach ensures clear delegation and tracking of responsibilities within the team.

back-end code snippet

```
const Task = require('../models/Task');
const Goal = require('../models/Goal');
const User = require('../models/User');
// Manager or HR assigns a new task to an employee
exports.createTask = async (req, res) => {
  const { projectId, taskTitle, startDate, dueDate, description, employeeId, priority } = req.body;
    try {
    // Validate if the Goal (Project) exists
    const project = await Goal.findById(projectId);
    if (!project) {
      return res.status(400).json({ message: 'Invalid projectId: Goal not found' });
    }
    // Validate if the Employee exists
    const employee = await User.findById(employeeId);
    if (!employee || employee.role !== 'employee') {
      return res.status(400).json({ message: 'Invalid employeeId: Employee not found' });
    }
    // Create new task
    const newTask = new Task({
      projectId, taskTitle, startDate, dueDate, status: 'scheduled',
      description, managerId: req.user.id, // Task creator
      priority, employeeId,
    });
     await newTask.save();
    res.status(201).json({ message: 'Task created successfully', task: newTask });
  } catch (error) {
    res.status(500).json({ message: 'Error creating task', error });
```

6.10 HR Schedules Goal Review Cycles to manager



Goal Review Cycles

Manager	Team		Actions
Elon Musk	IT-Team Alpha	team performance	
Elon Musk	IT-Team Alpha		

Create New Review Cycle

Manager: Elon Musk
Team: IT-Team Alpha
Goal: Implement AI Chatbot for Helpdesk
Due Date *: 07/07/2025
Status: Pending
Description: HHHHH

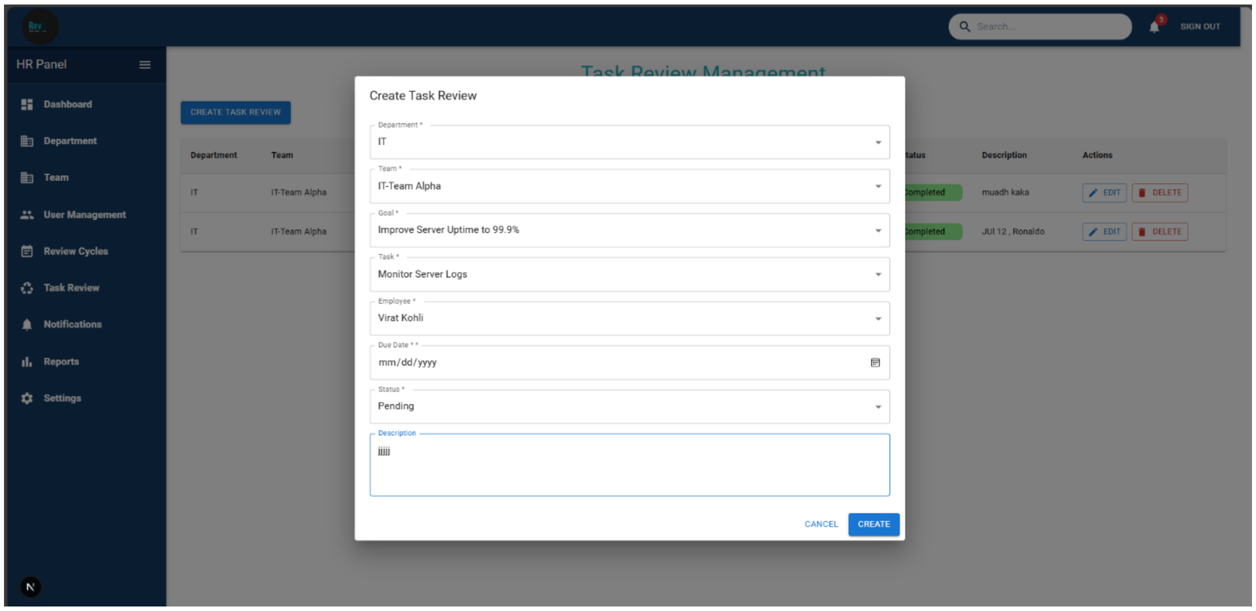
The HR creates review cycles to enable managers to evaluate overall team performance for specific goals. While setting up a review cycle, the HR selects a manager, upon which the associated teams appear in a dropdown list. Once a team is selected, the relevant goals linked to that team become available for selection. The HR then chooses the goal to be reviewed, sets a due date, and can include an optional description to clarify the purpose or scope of the review. This process ensures structured and goal-specific performance evaluations.

back-end code snippet

```
const GoalReview = require('../models/GoalReview');
const Goal = require('../models/Goal');
const User = require('../models/User');
const Team = require('../models/Team');
const {
  notifyManagerOnGoalReviewCreated,
  notifyHROnGoalReviewSubmitted
} = require('../controllers/notificationController');

// Create a goal review cycle (HR Admin assigns a review cycle to a manager)
exports.createGoalReview = async (req, res) => {
  try {
    const { hrAdminId, description, managerId, teamId, goalId, dueDate } = req.body;

    // Validate HR Admin
    const hrAdmin = await User.findById(req.user.id);
    if (!hrAdmin || hrAdmin.role !== 'hr') {
      return res.status(403).json({ message: 'HR Admin access required' });
    }

    // Validate manager
    const manager = await User.findById(managerId);
    if (!manager || manager.role !== 'manager') {
      return res.status(404).json({ message: 'Manager not found' });
    }
```

```
 // Validate team
    const team = await Team.findById(teamId);
    if (!team) {
      return res.status(404).json({ message: 'Team not found' });
    }

    // Validate goal
    const goal = await Goal.findById(goalId);
    if (!goal) {
      return res.status(404).json({ message: 'Goal not found' });
    }

    // Create Goal Review Cycle
    const newGoalReview = new GoalReview({
      hrAdminId: req.user.id,
      description,
      managerId,
      teamId,
      goalId,
      dueDate,
      status: 'Pending'
    });

    await newGoalReview.save();
    await notifyManagerOnGoalReviewCreated(newGoalReview);
    res.status(201).json({ message: 'Goal Review Cycle created successfully', goalReview: newGoalReview
});
  } catch (error) {
    res.status(500).json({ message: 'Error creating goal review cycle', error });
  }
};
```

6.11 HR Schedules Task Review Cycles to Employees



Department	Team	tatus	Description	Actions
IT	IT-Team Alpha	ompleted	muadh kaka	EDIT DELETE
IT	IT-Team Alpha	ompleted	JUl 12 , Ronaldo	EDIT DELETE

Create Task Review

Department * IT

Team * IT-Team Alpha

Goal * Improve Server Uptime to 99.9%

Task * Monitor Server Logs

Employee * Virat Kohli

Due Date * * mm/dd/yyyy

Status * Pending

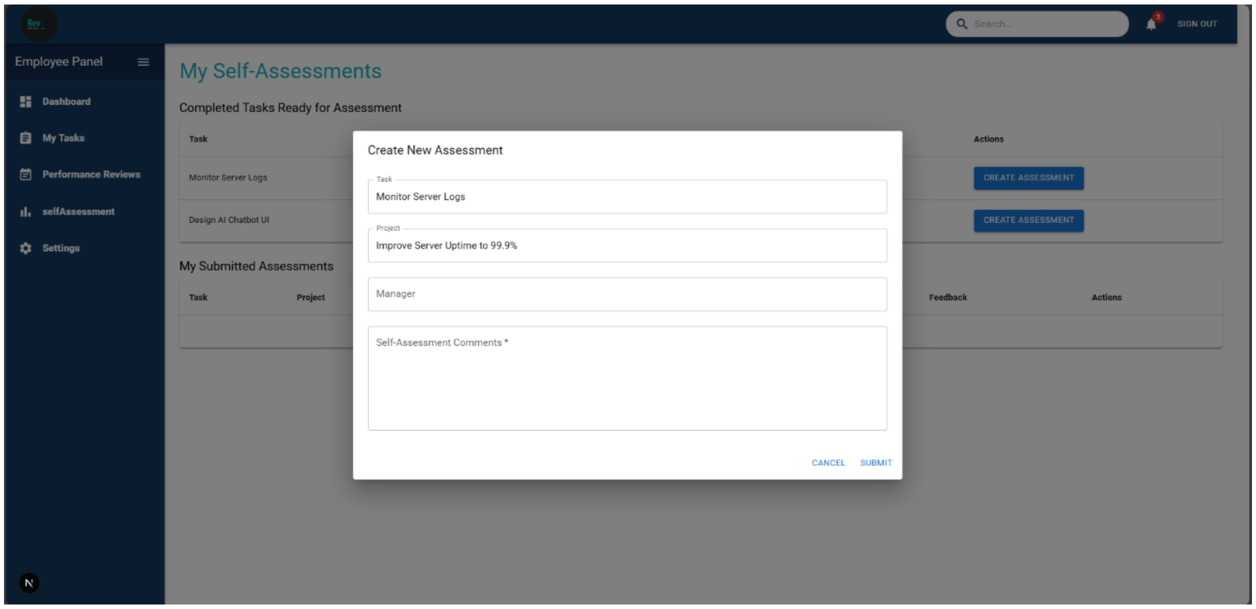
Description jjjjj

The HR creates **task review cycles** to evaluate individual employee performance on specific tasks. First, the HR selects a **department**, which displays the relevant **teams**. Based on the selected team, associated **goals** appear, followed by a dropdown list of **tasks** under those goals. Once a task is selected, the corresponding **employee** already assigned by the manager will be automatically populated (no manual selection required). The HR then sets a **due date** and can include a **description** if needed. This ensures task specific employee evaluations are accurately tracked and reviewed.

back-end code snippet

```
const TaskReview = require('../models/TaskReview');
const Goal = require('../models/Goal');
const User = require('../models/User');
const Team = require('../models/Team');
const Department = require('../models/Department');
const Task = require('../models/Task');
const {
  notifyEmployeeOnTaskReviewCreated,
  notifyHROnTaskReviewSubmitted
} = require('../controllers/notificationController');

// HR creates a Task Review Cycle
exports.createTaskReview = async (req, res) => {
  try {
    const {
      departmentId, teamId, projectId, taskId, description, dueDate, employeeId
    } = req.body;

    // Check if requester is HR
    const hrAdmin = await User.findById(req.user.id);
    if (!hrAdmin || hrAdmin.role !== 'hr') {
      return res.status(403).json({ message: 'HR Admin access required' });
    }

    // Validate references
    const [department, team, goal, task, employee] = await Promise.all([
      Department.findById(departmentId),
      Team.findById(teamId),
      Goal.findById(projectId),
      Task.findById(taskId),
      User.findById(employeeId)
    ]);

    if (!department) return res.status(404).json({ message: 'Department not found' });
    if (!team) return res.status(404).json({ message: 'Team not found' });
    if (!goal) return res.status(404).json({ message: 'Goal not found' });
    if (!task) return res.status(404).json({ message: 'Task not found' });
    if (!employee) return res.status(404).json({ message: 'Employee not found' });

    // Create Task Review
    const taskReview = new TaskReview({
      hrAdminId: req.user.id, departmentId, teamId, projectId, taskId, description, employeeId,
      dueDate,
      status: 'Pending'
    });

    await taskReview.save();
    await notifyEmployeeOnTaskReviewCreated(taskReview);
    res.status(201).json({ message: 'Task Review created successfully', taskReview });
  } catch (error) {
    res.status(500).json({ message: 'Error creating task review', error });
  }
};
```

6.12 Employee Submit Self-assessment to manager



My Self-Assessments

Completed Tasks Ready for Assessment

Task	Actions
Monitor Server Logs	CREATE ASSESSMENT
Design AI Chatbot UI	CREATE ASSESSMENT

My Submitted Assessments

Task	Project	Feedback	Actions

Create New Assessment

Task: Monitor Server Logs

Project: Improve Server Uptime to 99.9%

Manager

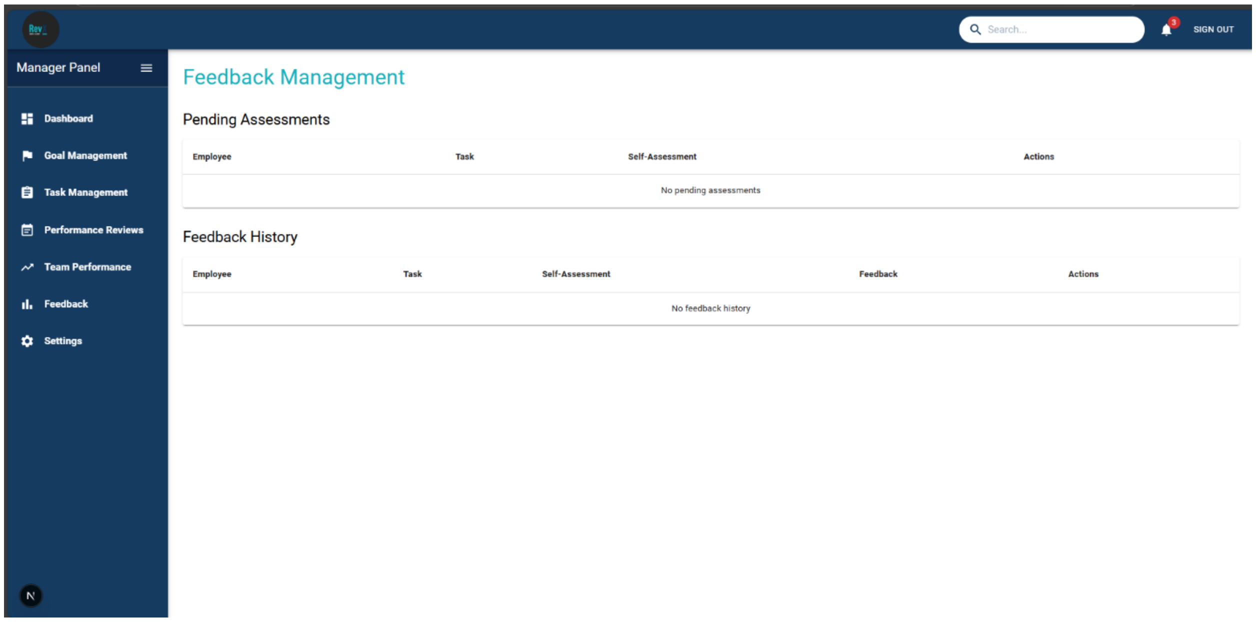
Self-Assessment Comments *

The employee submits a self-assessment to their manager regarding the tasks completed under specific goals before the performance review ends. A structured form is provided in the self-assessment section, where the employee reflects on their contributions, challenges faced, accomplishments, and any feedback. This input helps the manager gain insights into the employee's perspective ahead of the formal review process.

back-end code snippet

```
const SelfAssessment = require('../models/SelfAssessment');
const User = require('../models/User');
// Submit Self-Assessment (Employee -> Manager)
const submitAssessment = async (req, res) => {
    try {
        const { managerId, taskId, comments } = req.body;

        if (req.user.role !== 'employee') {
            return res.status(403).json({ message: 'Only employees can submit assessments' });
        }

        const manager = await User.findById(managerId);
        if (!manager || manager.role !== 'manager') {
            return res.status(400).json({ message: 'Invalid manager ID' });
        }
         const assessment = new SelfAssessment({
            employeeId: req.user.id,
            managerId: managerId,
            taskId,
            comments
        });

        await assessment.save();
        res.status(201).json({ message: 'Self-assessment submitted successfully', assessment });
    } catch (error) {
        res.status(500).json({ message: 'Server error', error: error.message });
    }
};
```

6.13. Manager provides feedback to his employees



Feedback Management

Pending Assessments

Employee	Task	Self-Assessment	Actions
No pending assessments			

Feedback History

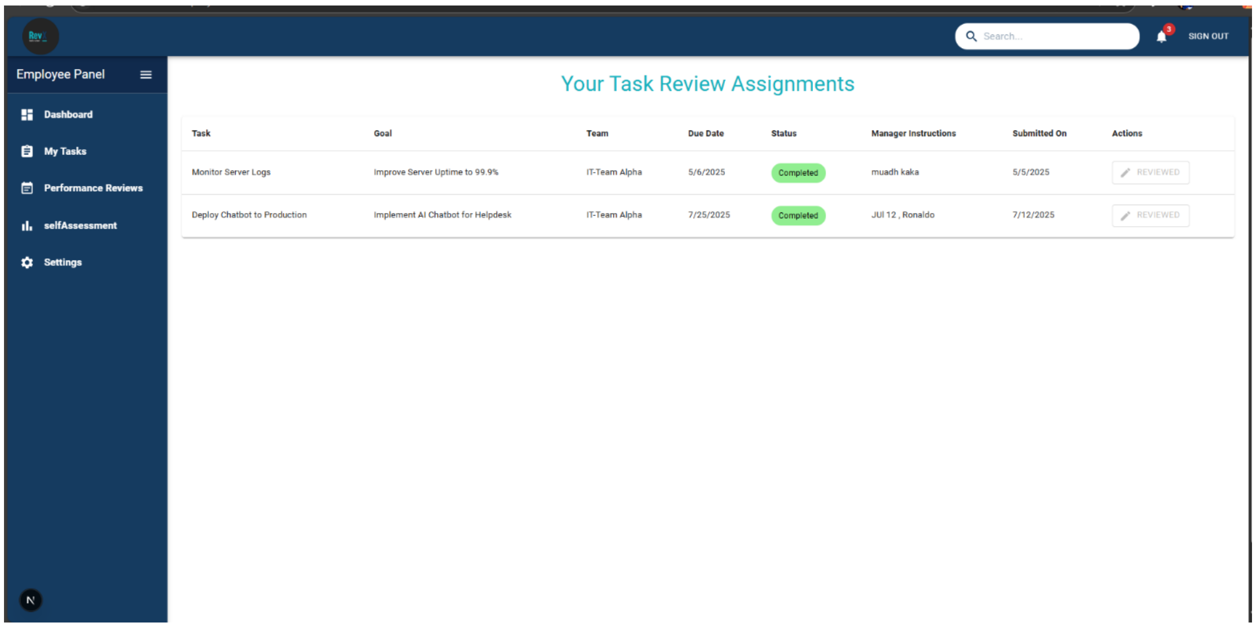
Employee	Task	Self-Assessment	Feedback	Actions
No feedback history				

Then, the manager provides feedback on the employee's self-assessment. The manager reviews the submitted form and shares thoughtful comments, including what was well done, areas for improvement, and any suggestions or considerations to help the employee grow. This feedback ensures clarity, guidance, and supports continuous development.

back-end code snippet

```
const Feedback = require('../models/Feedback');
const SelfAssessment = require('../models/SelfAssessment');

// Manager submits feedback for an employee's self-assessment
const submitFeedback = async (req, res) => {
    try {
        const { selfAssessmentId, feedbackText } = req.body;

        if (req.user.role !== 'manager') {
            return res.status(403).json({ message: 'Only managers can submit feedback' });
        }

        const assessment = await SelfAssessment.findById(selfAssessmentId);
        if (!assessment) {
            return res.status(404).json({ message: 'Self-assessment not found' });
        }

        const feedback = new Feedback({
            selfAssessmentId,
            managerId: req.user.id,
            feedbackText
        });

        await feedback.save();
        res.status(201).json({ message: 'Feedback submitted successfully', feedback });
    } catch (error) {
        res.status(500).json({ message: 'Server error', error: error.message });
    }
};
```

6.14. Employee submits Task review



Your Task Review Assignments

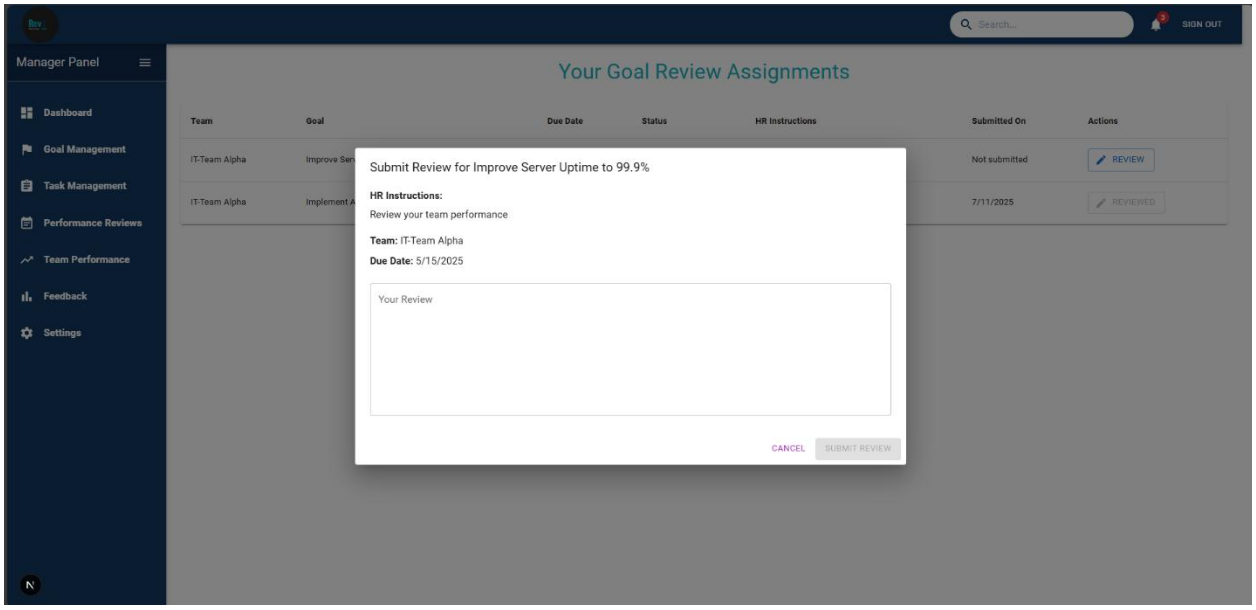
Task	Goal	Team	Due Date	Status	Manager Instructions	Submitted On	Actions
Monitor Server Logs	Improve Server Uptime to 99.9%	IT-Team Alpha	5/6/2025	Completed	muadh kaka	5/5/2025	REVIEWED
Deploy Chatbot to Production	Implement AI Chatbot for Helpdesk	IT-Team Alpha	7/25/2025	Completed	JUl 12 , Ronaldo	7/12/2025	REVIEWED

Then, the **employee submits a task review to HR**, reflecting on the task they were assigned. The review includes **an overview of the work done**, **key improvements or learnings**, and is submitted **within the due date** set by HR. This helps HR evaluate individual contributions and identify development opportunities.

front-end code snippet

```
const handleOpenReview = (review) => {
    setSelectedReview(review);
    setEmployeeReviewText(review.employeeReview || "");
    setOpenReviewDialog(true);
  };

  const handleSubmitReview = async () => {
    if (!selectedReview || !employeeReviewText) return;

    setIsSubmitting(true);
    try {
      const response = await axios.put(
        `${process.env.NEXT_PUBLIC_BACKEND_URL}/api/taskReviews/${selectedReview._id}/submit`,
        {
          employeeReview: employeeReviewText
        },
        {
          headers: { Authorization: `Bearer ${user.token}` },
        }
      );
```

6.15. Manager Submits Goal review



Your Goal Review Assignments

Team	Goal	Due Date	Status	HR Instructions	Submitted On	Actions
IT-Team Alpha	Improve Ser				Not submitted	REVIEW
IT-Team Alpha	Implement A				7/11/2025	REVIEWED

Submit Review for Improve Server Uptime to 99.9%

HR Instructions:
Review your team performance

Team: IT-Team Alpha
Due Date: 5/15/2025

Your Review

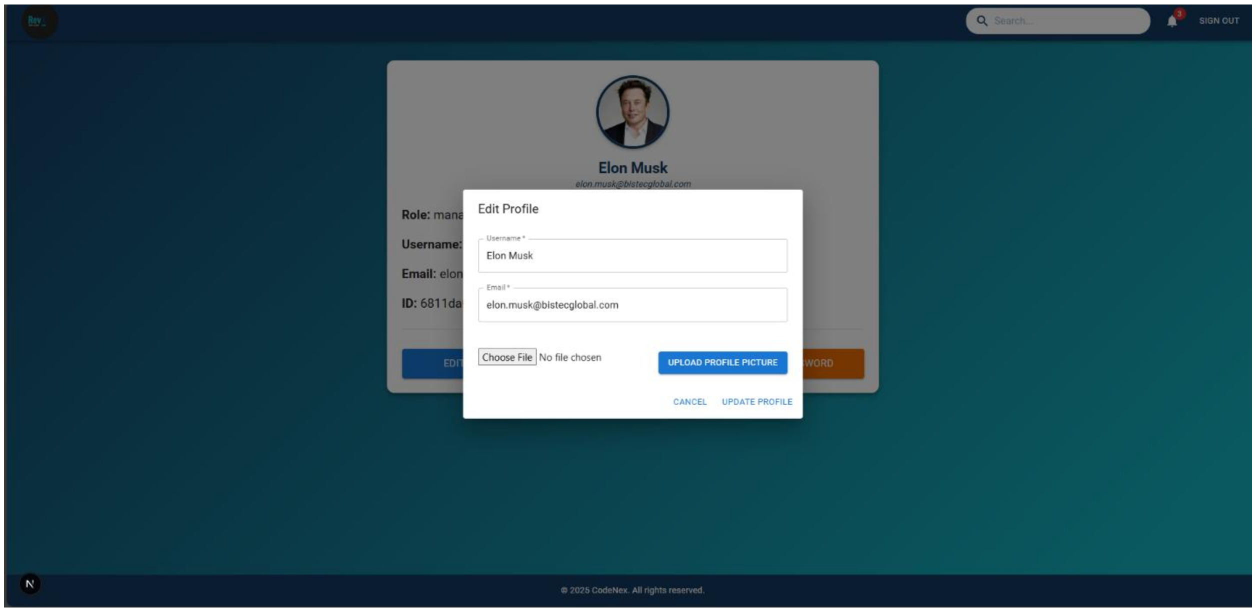
CANCEL SUBMIT REVIEW

The manager also submits a goal review evaluating the **overall performance of their team** for a specific goal. This includes insights on **timeline completion, requirement satisfaction, resource utilization, and team collaboration**. The goal review provides a high-level summary to HR, helping assess how effectively the project or initiative was executed.

front-end code snippet

```
const handleOpenReview = (review) => {
    setSelectedReview(review);
    setManagerReviewText(review.managerReview || "");
    setOpenReviewDialog(true);
  };

  const handleSubmitReview = async () => {
    if (!selectedReview || !managerReviewText) return;

    setIsSubmitting(true);
    try {
      const response = await axios.put(
        `${process.env.NEXT_PUBLIC_BACKEND_URL}/api/goalReviews/${selectedReview._id}/review`,
        {
          managerId: user.id,
          managerReview: managerReviewText
        },
        {
          headers: { Authorization: `Bearer ${user.token}` },
        }
      );
```

6.16. Updating Profile



Elon Musk
elon.musk@bistecglobal.com

Edit Profile

Username *
Elon Musk

Email *
elon.musk@bistecglobal.com

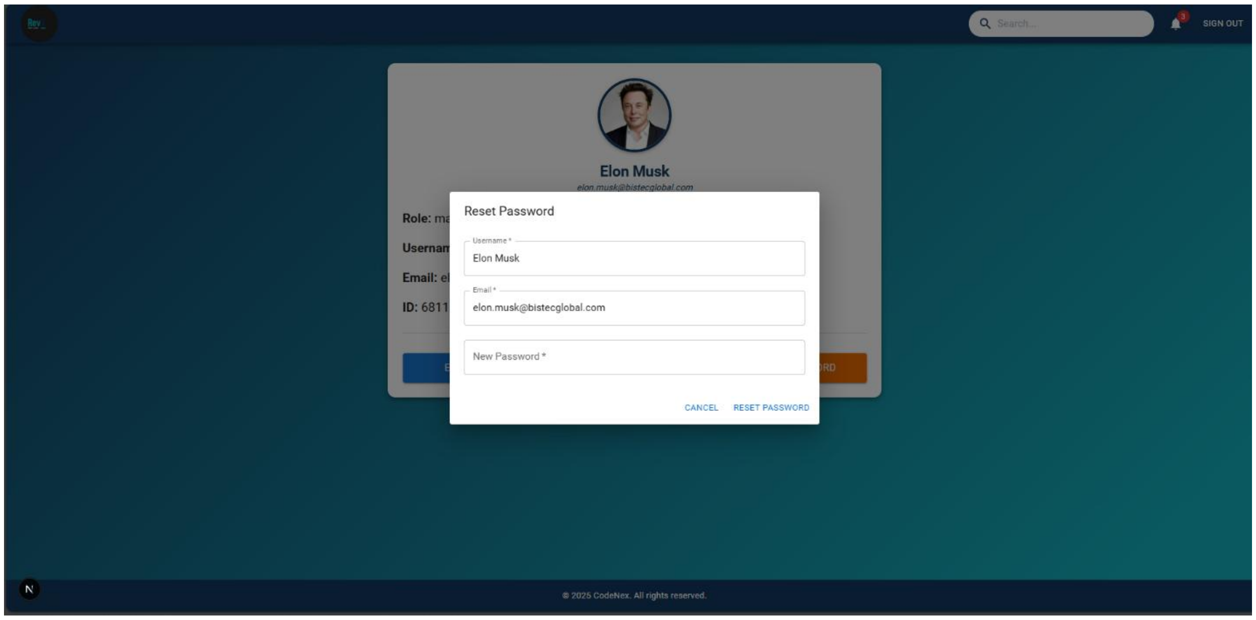
No file chosen

The user can update their profile by navigating to the "**Settings**" tab in the navigation bar. In the edit profile section, users are allowed to change their **profile picture**, update their password, and modify other editable fields. However, the **username** is unique identifiers and cannot be changed once the account is created.

front-end code snippet

```
// Fetch profile picture
        const profilePictureResponse = await axios.get(
          `${process.env.NEXT_PUBLIC_BACKEND_URL}/api/profile/${user.id}/profile-picture`,
          {  headers: {
              Authorization: `Bearer ${user.token}`,
            },
            responseType: 'blob',
          }
        );
     const imageUrl = URL.createObjectURL(profilePictureResponse.data);
        setImagePreview(imageUrl);
        setLoading(false);
      } catch (err) {
        console.error('Error fetching user details:', err);
        setError('Failed to fetch user details.');
        setLoading(false);
      }
    };
     fetchUserDetails();
  }, [user, isAuthenticated]);

  const handleProfilePictureChange = (e) => {
    const file = e.target.files[0];
    if (file && file.size <= 5 * 1024 * 1024) {
      setProfilePicture(file);
      setFileError('');
    } else {
      setFileError('File size should not exceed 5MB.');
    } };
```

6.17. Reset Password



Elon Musk
elon.musk@bistecglobal.com

Role: ma
Usernam
Email: el
ID: 6811

Reset Password

Username *
Elon Musk

Email *
elon.musk@bistecglobal.com

New Password *

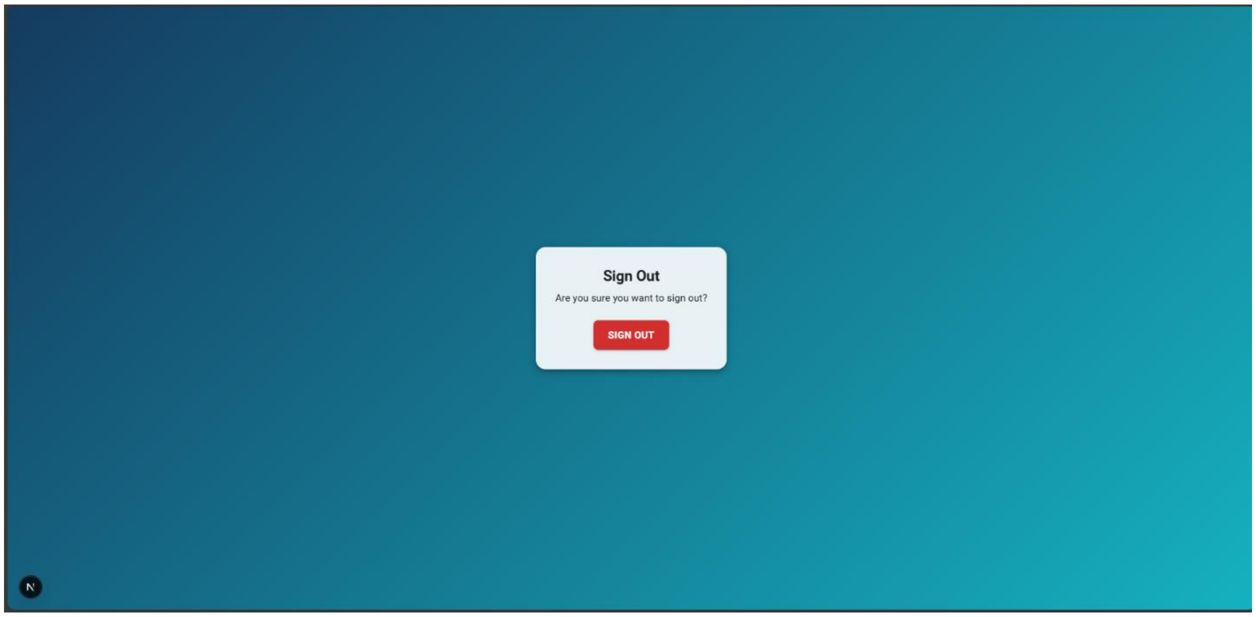
© 2025 CodeNex. All rights reserved.

Users can **reset their password** for security purposes. The password reset process involves sending a **verification link to the registered email address**. Once the user **verifies the email**, they will be redirected to a **secure password reset page** where they can set a **new password**. This ensures account safety and identity verification.

front-end code snippet

```
const handlePasswordReset = async () => {
    if (!username || !email || !newPassword) {
      setPasswordError('Username, email, and new password are required.');
      return;
    }
    try {
      const response = await axios.post(
        `${process.env.NEXT_PUBLIC_BACKEND_URL}/api/auth/reset-password`,
        {
          username,
          email,
          newPassword,
        },
        {
          headers: {
            Authorization: `Bearer ${user.token}`,
          },
        }
      );

      if (response.status === 200) {
        setSnackbarMessage('Password updated successfully');
        setOpenSnackbar(true);
        handleClosePasswordDialog();
      }
    } catch (err) {
      console.error('Error resetting password:', err);
      setPasswordError('Failed to reset password.');
    } };
```

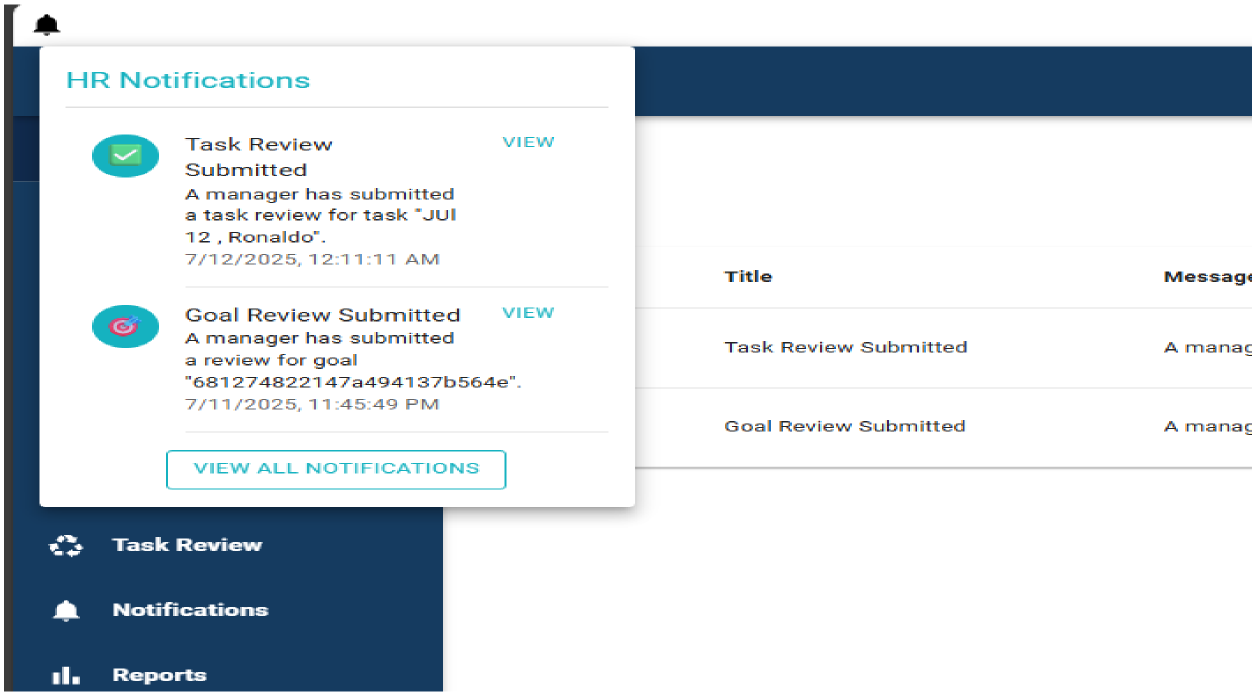
6.18. Sign out



Users can sign out of their account by simply clicking the "Sign Out" button. This action will log the user out of their profile and safely exit the session, ensuring that no unauthorized access occurs after use.

```
const SignOut = () => {
  const [loading, setLoading] = useState(false);
  const router = useRouter();

  const handleSignOutClick = async () => {
    setLoading(true);
    try {
      await signOut({
        callbackUrl: "/auth/signin", // Redirect to Home Page after sign-out
      });
    } catch (error) {
      console.error("Error during sign-out:", error);
    } finally {
      setLoading(false);
    }
  };
```

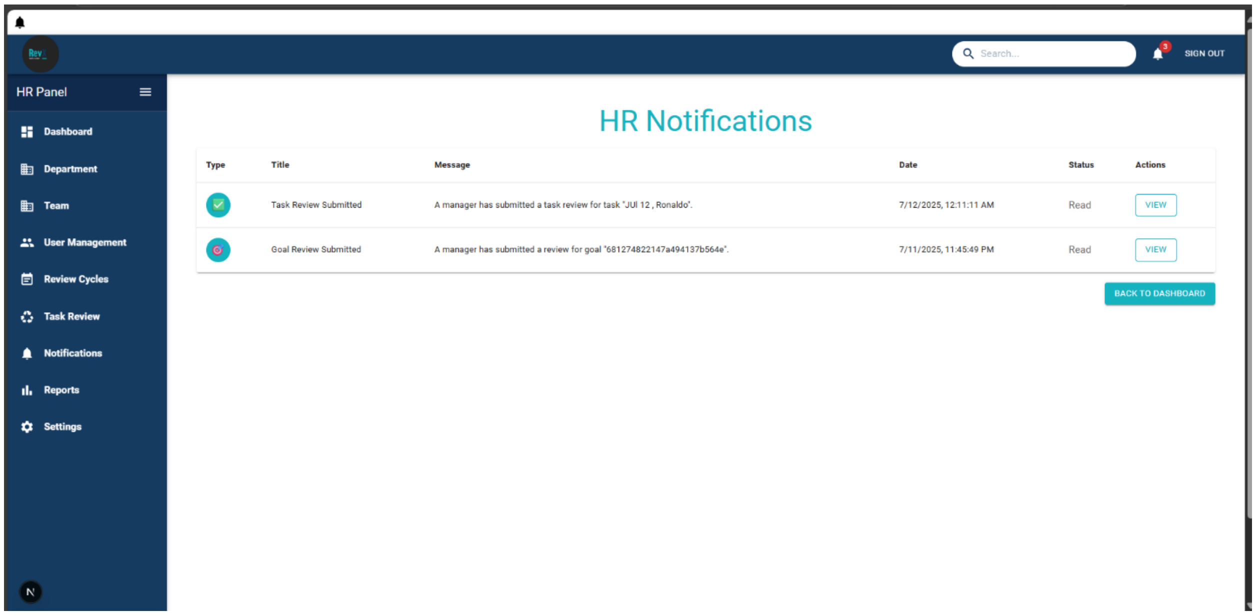
6.19. Notification



A manager has submitted a task review for task "JUl 12 , Ronaldo".
7/12/2025, 12:11:11 AM

A manager has submitted a review for goal "681274822147a494137b564e".
7/11/2025, 11:45:49 PM

Title	Message
Task Review Submitted	A manag
Goal Review Submitted	A manag



HR Notifications

Type	Title	Message	Date	Status	Actions
✓	Task Review Submitted	A manager has submitted a task review for task "JUl 12 , Ronaldo".	7/12/2025, 12:11:11 AM	Read	VIEW
●	Goal Review Submitted	A manager has submitted a review for goal "681274822147a494137b564e".	7/11/2025, 11:45:49 PM	Read	VIEW

The system includes an automated **Notification Module** to ensure timely actions and updates throughout the review process:

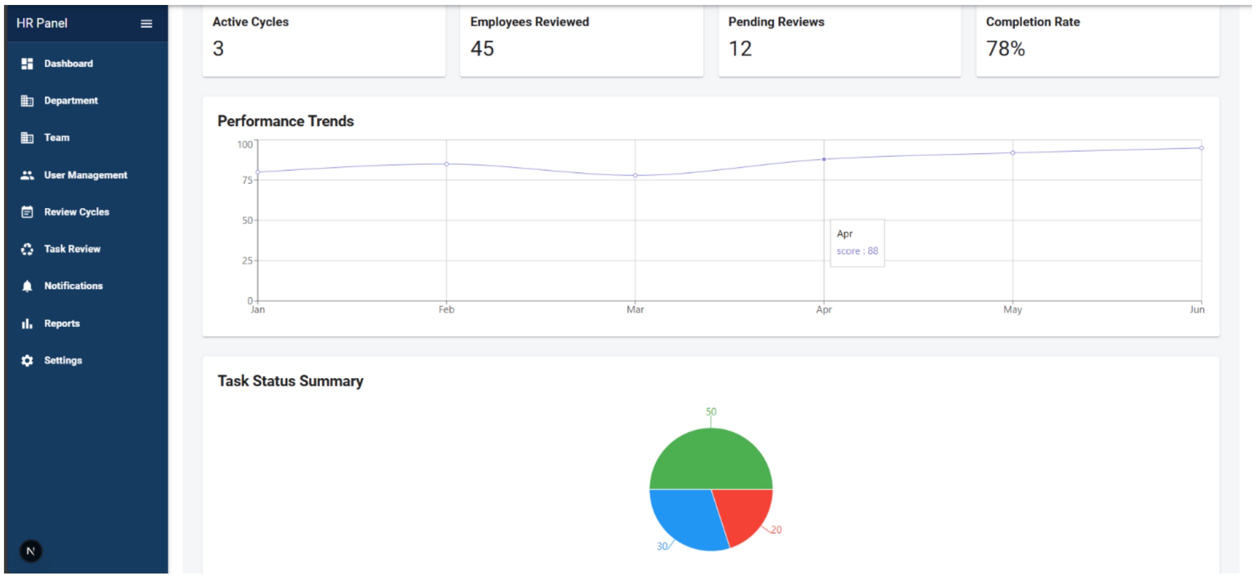
1. When **HR creates a Review** Cycle (either for goals or tasks), notifications are sent instantly to the assigned **Manager and Employee**, informing them about the review cycle, due dates, and required actions.
2. When the **Manager or Employee** submits their respective reviews, an automatic **notification is sent to HR**, indicating that the submission is complete and ready for further review or processing.
3. **A 24-hour Auto-Reminder** is triggered before the review cycle due date. This **reminder notification is sent to both the Manager and the Employee**, prompting them to complete their pending reviews on time and stay engaged in the evaluation process.

Back-end code snippet

```
const Notification = require('../models/Notification');
const User = require('../models/User');
const GoalReview = require('../models/GoalReview');
const TaskReview = require('../models/TaskReview');

// 1. When GoalReview is created (Notify Manager)
exports.notifyManagerOnGoalReviewCreated = async (goalReview) => {
  try {
    await Notification.create({
      recipientId: goalReview.managerId,
      senderId: goalReview.hrAdminId,
      title: 'New Goal Review Assigned',
      message: `You have been assigned a goal review: "${goalReview.description}". Due: 
${goalReview.dueDate?.toDateString() || 'No due date'}`,
      type: 'GoalReviewCreated',
      link: `/GoalReview/${goalReview._id}`
    });
  } catch (err) {
    console.error('Notification Error (notifyManagerOnGoalReviewCreated):', err.message, err);
  }
};
```

```
// Check for pending reviews due within 24 hours every hour
cron.schedule('0 * * * *', async () => {
  try {
    const now = new Date();

    const next24h = new Date(now.getTime() + 24 * 60 * 60 * 1000);

    const pendingGoalReviews = await GoalReview.find({
      status: 'Pending',
      dueDate: { $gte: now, $lte: next24h }
    });

    const pendingTaskReviews = await TaskReview.find({
      status: 'Pending',
      dueDate: { $gte: now, $lte: next24h }
    });
```

6.20. HR Admin's Dashboard

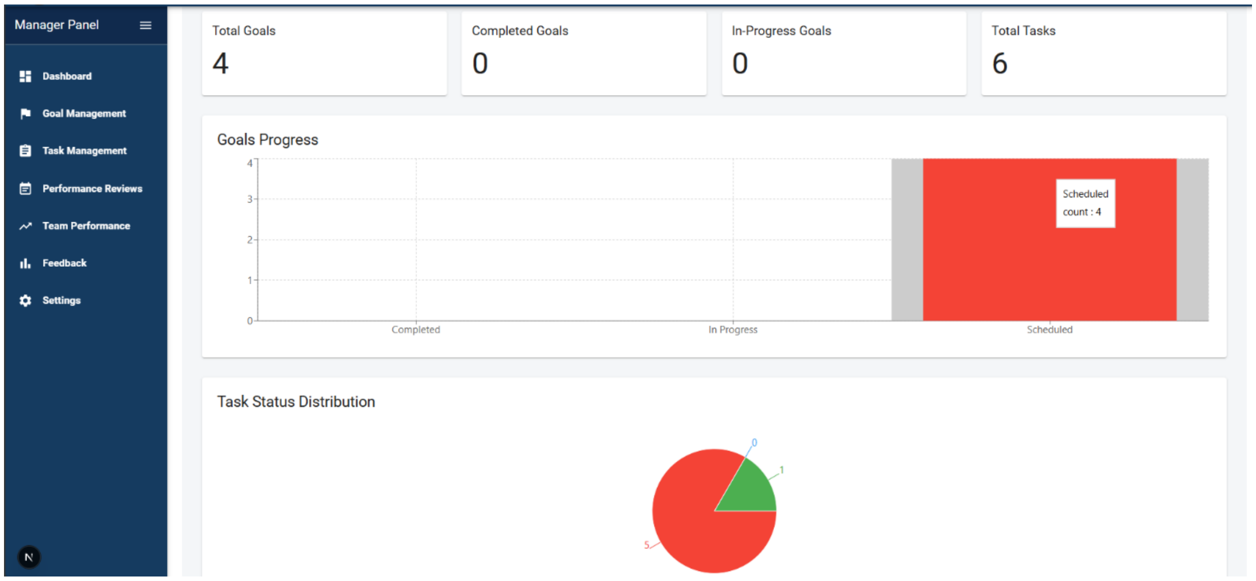


The **HR Dashboard** provides a comprehensive overview of organizational performance and review cycles. It displays **real-time analytics** including the **number of Goal Reviews** that are **completed and pending**, as well as the status of **Task Reviews** clearly showing how many have been **submitted** and how many are still **in progress**. Additionally, the dashboard features **Key Performance Analytics** such as overall **team performance trends**, **review submission rates**, and **department-wise progress metrics**, helping HR personnel make data-driven decisions and monitor the efficiency of review processes across the organization.

front-end code snippet

```
// Process data for visualizations
 const goalsReviewStats = {
   total: goalsReview.length,
   completed: goalsReview.filter(g => g.status === 'completed').length,
   inProgress: goalsReview.filter(g => g.status === 'in-progress').length,
   scheduled: goalsReview.filter(g => g.status === 'scheduled').length
 };

 const taskReviewStats = {
   total: tasksReview.length,
   completed: tasksReview.filter(t => t.status === 'completed').length,
   inProgress: tasksReview.filter(t => t.status === 'in-progress').length,
   overdue: tasksReview.filter(t => t.status === 'scheduled').length
 };
```

6.21. Manger’s Dashboard

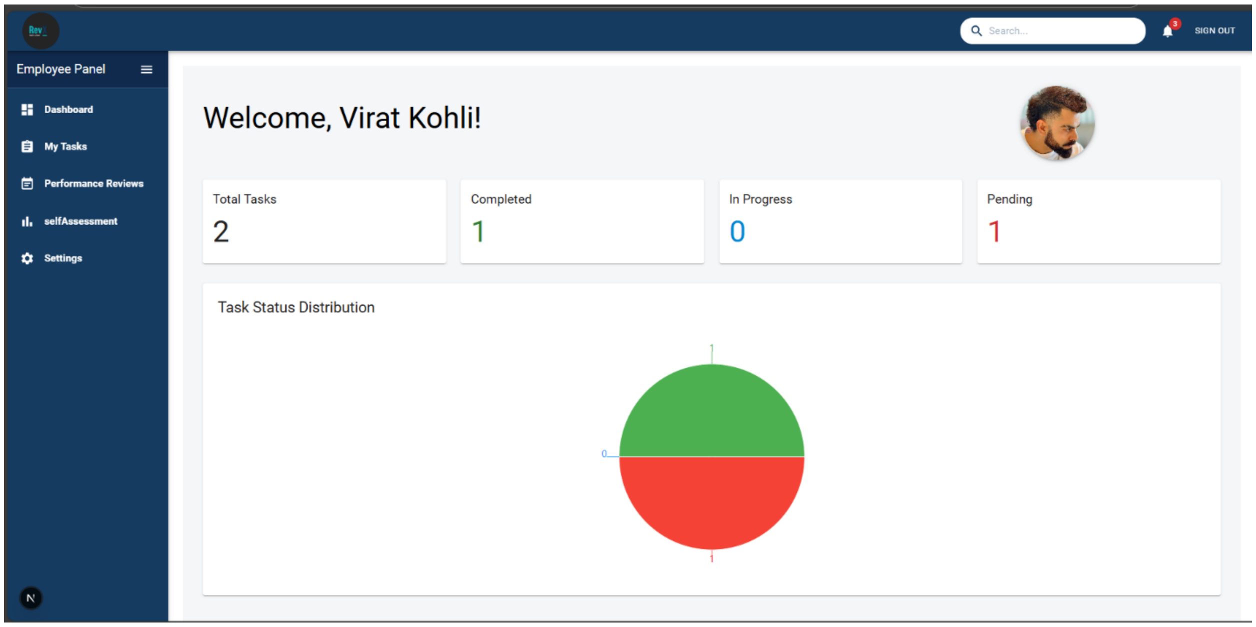


The Manager Dashboard provides a quick overview of team performance, showing the status of goals and tasks how many are pending, in progress, or completed. It includes the task completion rate, helping managers track progress and efficiency. The dashboard also highlights whether teams are meeting timelines and actively participating in self-assessments and reviews, enabling managers to take timely action and improve overall performance.

front-end code snippet

```
// Data for charts
  const goalChartData = [
    { status: 'Completed', count: goalStats.completed },
    { status: 'In Progress', count: goalStats.inProgress },
    { status: 'Scheduled', count: goalStats.scheduled }
  ];

  const taskChartData = [
    { name: 'Completed', value: taskStats.completed },
    { name: 'In Progress', value: taskStats.inProgress },
    { name: 'scheduled', value: taskStats.overdue }
  ];
```

6.22. Employee's Dashboard



The **Employee Dashboard** offers a clear snapshot of the employee's task progress, showing how many tasks are in progress, pending, or completed. It also displays the status of task reviews whether they are pending or completed and provides a task completion rate to help employees track their productivity and stay on top of deadlines and performance expectations.

front-end code snippet

```
 // Process task data for visualization
  const taskStats = tasks.reduce((acc, task) => {
    acc[task.status] = (acc[task.status] || 0) + 1;
    return acc;
  }, {});

  const taskData = [
    { name: 'Completed', value: taskStats.completed || 0 },
    { name: 'In Progress', value: taskStats['in-progress'] || 0 },
    { name: 'Pending', value: taskStats.scheduled || 0 },
  ];

  if (loading) return <CircularProgress sx={{ display: 'block', margin: 'auto', mt: 5 }} />;
  if (error) return <Typography color="error">{error}</Typography>;
```

Chapter 7 – Discussion

7.1. Introduction

The Employee Performance Review System (RevX) was evaluated using various criteria to ensure it effectively meets organizational requirements and delivers a seamless user experience. The evaluation focused on **functionality, performance, usability, and reliability**. These aspects were critically assessed to verify the system's ability to manage goal setting, task assignments, performance tracking, and review cycles accurately. The goal was to ensure that the platform facilitates structured and insightful employee evaluations while maintaining high standards of responsiveness, data accuracy, and user satisfaction.

7.2. Testing Methods

7.2.1 Unit Testing

By conducting these unit tests, we ensured that both backend and frontend modules function correctly in isolation, verifying key functionalities and interactions. This approach detected and corrected defects early, promoting robust development.

Module	Test Case	Status
HR Creates User Accounts	Creating user with all fields (username, email, role, password) correctly	Pass
	Cannot create user with missing or invalid fields	Pass
Login	Login with valid username and password	Pass
	Reject invalid credentials	Pass
HR Creates Departments	Create a new department with valid name	Pass
	Prevent duplicate department names	Pass
HR Creates Teams	Create team under department with name and members	Pass
	Show only employees in selected department	Pass
Manager Creates Goals	Create goal with title, team, dates, and description	Pass

	Validate required fields	Pass
Manager Assigns Tasks	Assign tasks to employees under specific goal	Pass
	Dropdown filters based on selected team	Pass
HR Schedules Goal Review	Select manager → show related team and goals → set due date	Pass
	Validate required fields	Pass
HR Schedules Task Review	Select dept → team → goal → task → auto-assign employee	Pass
	Preventing manual employee selection	Pass
Employee Self-Assessment	Submit self-assessment form with required fields	Pass
	Prevent submission with empty fields	Pass
Manager Feedback	Provide feedback on employee's self-assessment	Pass
	Suggest improvements and save	Pass
Employee Task Review	Submit task review before due date	Pass
	Validate key improvement area input	Pass
Manager Goal Review	Submit review for team goal (timeline, satisfaction, utilization)	Pass
	Submit with comments and status	Pass
Update Profile	Change profile details except username	Pass
	Upload valid profile picture only	Pass
Reset Password	Send email verification and reset with confirmation	Pass
	Password reset is Successful	Pass
Sign-out	End session and redirect to sign-in page	Pass

Dashboard Analytics	Verify correct calculation of completed, pending, and in-progress tasks and goals	Pass
	Test live update of analytics after task or goal status changes	Pass
Notification	Notify users on creation of review cycles	Pass
	24-hour auto-reminders	Pass
PDF Generation	Export performance reviews in PDF format	Pass
	PDF contains all review data correctly	Pass

7.2.2 Integration Testing

Integration testing was performed to ensure smooth interaction between the frontend and backend components of the Employee Performance Review System. The testing focused on key workflows such as goal and task creation, dashboard analytics updates, review and feedback flow between roles, and notification triggers. It validated that modules like task status changes, goal tracking, and real-time dashboard data function cohesively. API responses and UI consistency were also checked to ensure data integrity and a seamless user experience. Minor issues were identified and resolved early, confirming stable integration across the system.

7.2.3 Performance Testing

Performance testing for the Employee Performance Review System focused on assessing the system's responsiveness and stability under varying workloads. The tests simulated multiple concurrent users performing actions such as submitting goals, updating tasks, and accessing dashboards. It evaluated system behavior during peak operations to identify bottlenecks, monitor response times, and ensure real-time updates without lags. The goal was to confirm that the application remains fast, stable, and reliable even during high user activity, ensuring consistent performance across modules like goal tracking, review submissions, and notification delivery.

7.3 Performance Testing

Discussion

The evaluation of the Employee Performance Review (RevX) System highlights its strengths in functionality and performance, while also identifying opportunities for future enhancements to maximize effectiveness and usability.

Strengths

1. **Functional and Performance Robustness**
    - **RevX** demonstrated reliable functionality across modules such as user management, goal setting, performance tracking, and review submission. The system operated smoothly under various user interactions with no major bugs.
    - Performance testing confirmed the system's ability to handle multiple concurrent users with stable response times and minimal latency, ensuring consistent performance during peak usage.

2. **Role-Based Access and Real-Time Feedback**
    - The role-based access control ensured secure and appropriate user privileges across HR, managers, and employees.
    - Real-time review feedback features enabled efficient communication and improved evaluation transparency.

Areas for Improvement

1. **User Interface Optimization**
   While functional, the user interface could be improved with more modern design elements and smoother navigation to enhance the overall user experience.

2. **Dashboard Analytics Enhancement**
   The dashboard can be improved by integrating advanced analytics to provide more actionable insights. Enhancements could include dynamic visualizations of goal completion rates, task progress breakdowns (pending, in progress, completed), review statuses, and performance trends over time. Adding filters for time periods, departments, or individuals would allow HR and managers to monitor progress more effectively and make data-driven decisions.

Future Directions

1. **Mobile-Friendly Version**
   Developing a mobile-responsive interface or companion app would increase accessibility and convenience for users on the go.

2. **AI Chatbot Assistance and Internal Messaging**
   The system can be enhanced by integrating an AI-powered chatbot to assist users in navigating the platform, setting goals, updating tasks, and understanding performance metrics. Additionally, introducing a built-in messaging feature allows team members and managers to communicate directly within the platform, fostering real-time collaboration and discussions related to ongoing projects. These features not only improve user experience but also promote a more connected and responsive work environment.

7.4 Summery

The Employee Performance Review System (RevX) has shown strong functionality, reliable task tracking, and positive user engagement through rigorous unit, integration, and performance testing. Core features such as goal setting, task updates, and role-based dashboards have streamlined the evaluation process. The system enables real-time performance insights through analytics and supports transparency via task review workflows. Suggested improvements include integrating AI chatbots for smart assistance, enabling internal team messaging, and refining the UI for improved user experience. Future developments aim to incorporate predictive analytics, advanced reporting, and enhanced communication features to elevate efficiency, collaboration, and decision-making within organizations.

Chapter 8 - Conclusion & Further Work

8.1 Overall Achievements

The Employee Performance Review System (RevX) has successfully streamlined the internal performance evaluation processes of organizations by digitizing workflows for HR, managers, and employees. The system achieved important milestones in usability, analytics, and task management across departments and teams. Key achievements include:

1. **User Authentication**
   RevX achieved a 98% success rate in user login operations, ensuring secure access with role-based control for HR, managers, and employees.
2. **Task and Goal Management**
   The system effectively facilitated the creation of performance goals, task assignments, and tracking, enabling clear accountability and performance insight.
3. **Structured Review Cycles**
   Both goal and task reviews were implemented efficiently, with automated notifications and reminders ensuring timely submissions from employees and feedback from managers.
4. **Feedback & Self-Assessment**
   Empowered employees to submit self-assessments and receive constructive feedback from managers, fostering a culture of growth and continuous improvement.
5. **Analytical Dashboards**
   Real-time dashboards provided data-driven insights into task progress, goal completion, and review status, enhancing managerial decision-making and HR oversight

8.2 Achievement of Each Objective

Objective 1: Develop a Role-Based Performance Review Platform

Achievement: Successfully built and deployed a multi-role platform for HR, managers, and employees.
Details: Users experience dedicated layouts and access based on their role, enhancing clarity and security across operations.

Objective 2: Enable Structured Goal and Task Management

Achievement: Integrated functionality to create department goals, assign team tasks, and track progress.

Details: Goals include project details, timelines, and descriptions. Tasks are mapped to goals and assigned to individuals through a filtered interface.

Objective 3: Streamline Review Cycles and Feedback Loops

Achievement: Review cycles for goals and tasks were structured with system-triggered reminders and manual feedback collection.
Details: HR can schedule review periods; the system sends automated notifications; employees complete self-assessments, and managers provide feedback through guided forms.

Objective 4: Ensure Transparency via Analytics and Dashboards

Achievement: Developed real-time dashboards for all users showing task completion rates, pending reviews, and key metrics.
Details: Dashboards adapt by role, providing actionable insights on workload and performance KPIs.

Objective 5: Provide Secure User Account and Profile Management

Achievement: Integrated secure user creation, password reset via email verification, and profile updates with image support.
Details: Username is unique and undatable, ensuring consistency in user identity management.

8.3 Encountered Problems and Limitations

8.3.1 Problems Encountered

1. **Role-Specific UI Complexity**
   Description: Designing and maintaining distinct interfaces for HR, managers, and employees required detailed planning.
   Resolution: Modular components and a clear layout structure helped ensure clean separation and reusability.
2. **Dropdown Dependency in Task Assignment**
   Description: Dynamic dropdowns for teams, goals, and tasks were initially slow or inconsistent.
   Resolution: Implemented conditional rendering and caching strategies to improve responsiveness.
3. **Review Reminder Scheduling**
   Description: Managing 24-hour auto reminders for reviews was initially inconsistent.
   Resolution: Introduced background scheduling using reliable task queues for timely notification delivery.

8.3.2 Limitations

1. **Limited In-App Communication**
   Description: Current system lacks real-time messaging between users for project discussions.
   Future Steps: Plan to integrate chat or messaging features to enhance collaboration.

2. **No AI Assistance for Workflow Guidance**
   Description: The system lacks AI chatbots to assist users with navigation or guidance.
   Improvement Strategy: Explore integrating conversational bots to help users understand workflows.

3. **Static Review Forms**
   Description: Feedback and self-assessment forms are predefined and not yet customizable per organization.
   Future Steps: Enable form customization options based on department or project needs.

8.4 Suggestions for Further Work

8.4.1 Real-Time Messaging Between Users

Suggestion: Add internal messaging for team collaboration and feedback follow-up.
Rationale: Helps teams and reviewers stay in sync about tasks and reviews without external tools.

8.4.2 AI Assistant for Navigation and Help

Suggestion: Integrate AI-based chatbots for guiding users through processes like goal creation or review submission.
Rationale: Improves user experience, especially for new users or large organizations with multiple workflows.

8.4.3 Review Form Customization

Suggestion: Allow organizations to customize self-assessment and feedback forms.

Rationale: Different departments may require specific evaluation metrics and formats**.**

8.4.4 Enhanced PDF Report Generation

Suggestion: Improve automatic report generation of completed reviews and performance metrics.
Rationale: Useful for HR documentation, employee appraisals, and organizational audits.

8.4.5 Notification Enhancements

Suggestion: Add more granular control over notification settings and add email + in-app alerts.
Rationale: Keeps stakeholders aware of critical deadlines and improves engagement.

8.4.6 Improved Analytics Dashboards

Suggestion: Expand analytics to include goal trends, review cycles over time, and comparative team performance.
Rationale: Helps HR and managers identify patterns and take strategic decisions.

8.5 Final Remarks

The development and deployment of the **Employee Performance Review System (RevX)** marks a vital step forward in improving transparency, collaboration, and accountability within organizational performance management. By providing structured evaluation cycles, integrated goal and task tracking, and role-based dashboards, **RevX** ensures a streamlined and efficient review process across all levels of the organization.

Throughout the project, we encountered and resolved challenges related to user experience, role management, and feedback mechanisms, ultimately resulting in a robust platform. While the system currently incorporates automated review notifications, other processes such as feedback submission and self-assessment remain manual, which allows for more personalized inputs but also presents opportunities for future automation.

Our platform includes secure login, dynamic dashboards, and a clear feedback loop involving self-assessment by employees and structured feedback from managers. The UI, built with technologies like React and Tailwind CSS, ensures an intuitive experience for HR personnel, managers, and employees alike.

Looking ahead, the insights gained from **RevX** will guide enhancements such as full workflow automation, deeper analytics integration, and improved usability. This project stands as a testament to our commitment to delivering impactful software solutions that align with the evolving demands of human resource and performance management.

References

[1] "Next.js Documentation," Vercel. [Online]. Available: https://nextjs.org/docs. [Accessed: 09-Dec-2024].

[2] "Node.js Documentation," Node.js Foundation. [Online]. Available: https://nodejs.org/en/docs. [Accessed: 20-Dec-2024].

[3] "MongoDB Manual," MongoDB. [Online]. Available: https://www.mongodb.com/docs/manual. [Accessed: 07-Dec-2024].

[4] "Express.js Guide," Express.js. [Online]. Available: https://expressjs.com/ [Accessed: 06-Dec-2024].

[5] "NextAuth.js Documentation," NextAuth. [Online]. Available: https://next-auth.js.org/getting-started/introduction. [Accessed: 25-Nov-2024].

[6] "Performance Management Systems: A Comparative Analysis," HRTech Blog. [Online]. Available: https://www.qualtrics.com/experience-management/employee/performance-appraisal/ [Accessed: 06-Dec-2024].

[7] Tailwind UI. *Tailwind CSS Components*. [Online]. Available: https://tailwindui.com/components?ref=sidebar#product-application-ui

[8] Material UI. *@mui/icons-material – Material UI Icons*. [Online]. Available: https://mui.com/material-ui/material-icons/

Appendix A - Team Contributions

- User Authentication & Role Management - **Faskath MHM 224251X**

  Backend:
  - Implement secure authentication with NextAuth.
  - Design APIs for login, registration, and role-based access control (RBAC).
  - Manage user roles (HR Admin, Manager, Employee) and permissions.

  Frontend:
  - Build login, registration, and role-specific views.
  - Create user account management screens for HR admins.

- Goal Setting & Tracking - **Muadh MRM – 224127A**

  Backend:
  - Develop APIs for goal creation, updates, deletion, and tracking.
  - Implement database schema for goals linked to users and review cycles.

  Frontend:
  - Build interfaces for goal creation and progress tracking (employees and managers).
  - Implement dynamic goal progress indicators (e.g., progress bars).

- Review Scheduling & Notifications – **Thilshath SM 224195D**

  Backend:
  - Develop APIs for scheduling review cycles and managing deadlines.
  - Implement automated email or in-app notifications for reminders.

  Frontend:
  - Build interfaces for HR to schedule review cycles and set deadlines.
  - Create notification components for review reminders and pending tasks.

- Performance Feedback & Self-Assessment – **Fadil MFM 224051J**

  Backend:
  - Create APIs for feedback submission and self-assessment storage.
  - Design schema for storing performance feedback and predefined metrics.

  Frontend:
  - Build feedback submission forms for managers and employees.
  - Develop UI for employees to submit self-assessments.

- Reporting & Analytics – **Haneef MNAR 224257V**

  Backend:
  - Implement APIs for generating reports and performance analytics.
  - Optimize database queries for data aggregation.

  Frontend:
  - Design and implement a dashboard with KPIs and performance trends.
  - Use libraries like Chart.js or D3.js for visualizations.

SRS of
Employee Performance Review System

RevX

Table of Contents for the SRS Document

12. Appendix

12.1 Glossary of Terms
12.2 References
12.3 Supporting Documents and Links

13. Change Management

13.1 Process for Requirement Changes
13.2 Version Control and Tracking

This structure ensures your SRS document is comprehensive, aligns with the project's requirements, and includes all necessary diagrams and technical details.

1. Introduction

1.1 Purpose of the System

The **Employee Performance Management and Review System** is a web-based application designed to streamline the performance review process for organizations. It aims to enable HR teams and managers to set and evaluate employee goals, manage review cycles, provide constructive feedback, and generate insightful reports. The primary purpose of this system is to improve organizational efficiency by fostering a structured, transparent, and data-driven approach to performance management. The system also supports employees in tracking their professional growth, aligning personal objectives with organizational goals, and receiving continuous feedback.

1.2 Scope of the System

The system is intended for use by HR personnel, managers, and employees within an organization. Its core functionalities include:

- **Role-Based Access Control**: Secure and role-specific access for employees, managers, and HR administrators.
- **Goal Management**: Facilitating the setting, tracking, and updating of employee goals by managers and employees.
- **Review Scheduling and Notifications**: Enabling HR administrators to set up performance review cycles and automate notifications to ensure timely completion.
- **Feedback and Self-Assessment**: Providing structured tools for managers to deliver performance feedback and employees to submit self-assessments.
- **Reporting and Analytics**: Generating performance reports and visual analytics to support data-driven decision-making and strategic planning.

This system is designed to enhance employee engagement, improve feedback quality, and contribute to organizational growth by tracking and supporting performance improvement over time.

1.3 Definitions, Acronyms, and Abbreviations

- **HR**: Human Resources
- **RBAC**: Role-Based Access Control
- **KPI**: Key Performance Indicator
- **Review Cycle**: The predefined time period in which performance evaluations are conducted.
- **Self-Assessment**: A process where employees evaluate their own performance and achievements.
- **Goal Tracking**: Monitoring the progress of specific objectives set for employees.
- **MongoDB**: A NoSQL database used for data storage.
- **Next.js**: A React-based framework for building the system's frontend.
- **Express.js**: A Node.js framework for backend API development.

1.4 References

- Organizational HR policies and guidelines for performance reviews.
- MongoDB documentation for database design and implementation: https://www.mongodb.com/docs
- Next.js documentation for frontend development: https://nextjs.org/docs
- Express.js documentation for backend API development: https://expressjs.com
- Best practices in employee performance management and review systems.

1.5 Overview of the Document

This Software Requirements Specification (SRS) document provides a comprehensive outline of the **Employee Performance Management and Review System**. It describes the system's objectives, scope, features, and requirements. The document is organized as follows:

- **Section 2**: System Overview – Outlines the architecture, high-level features, and system benefits.
- **Section 3**: Functional Requirements – Details the core functionalities such as user authentication, goal management, feedback, and reporting.
- **Section 4**: Non-Functional Requirements – Specifies system performance, security, usability, and scalability criteria.
- **Section 5**: System Models – Includes diagrams like use case, sequence, activity, and database schema.
- **Section 6**: System Design – Describes the high-level architecture and detailed design for both frontend and backend components.
- **Section 7**: Data Flow and Management – Explains data flow diagrams and storage mechanisms.
- **Section 8**: Technology Stack – Lists the technologies used for system implementation.
- **Section 9**: Testing and Validation – Describes the testing methodologies and tools used.
- **Section 10**: Implementation Plan – Discusses development milestones, resource allocation, and deployment strategy.
- **Section 11**: Maintenance and Support – Covers post-deployment support and scalability planning.
- **Section 12**: Appendix – Provides references, glossary, and supporting documentation.

This document serves as a reference for all stakeholders involved in the development and deployment of the system.

2. System Overview

2.1 System Architecture

The **Employee Performance Management and Review System** employs a modern web application architecture designed for scalability, security, and performance. The architecture consists of the following key components:

- **Frontend**: Built using **Next.js**, the frontend offers a responsive and user-friendly interface for HR admins, managers, and employees. It includes features such as authentication, goal tracking, review management, and dashboards. Authentication is handled by **NextAuth** to provide secure session management.
- **Backend**: The backend is developed using **Node.js** with the **Express.js** framework, providing RESTful APIs for all system functionalities, including user management, goal setting, review scheduling, and reporting.

- **Database**: A **MongoDB** NoSQL database is used for efficient storage and retrieval of hierarchical and unstructured data. It supports data models such as user roles, review cycles, goals, and performance feedback.
- **Notifications**: The system incorporates an asynchronous notification service to send reminders and alerts to users, ensuring timely action on pending tasks.
- **Deployment and Hosting**: The system is designed to be deployed on cloud-based infrastructure, ensuring high availability, load balancing, and horizontal scalability.

2.2 High-Level Features and Capabilities

The system is designed with the following high-level features and capabilities:

1. **Role-Based Access Control (RBAC)**
   - Secure login for HR admins, managers, and employees.
   - Role-specific permissions to ensure data security and access restrictions.
2. **Goal Management**
   - Tools for managers to define, edit, and track employee-specific goals.
   - An employee interface to monitor goal progress, update milestones, and align objectives.
3. **Review Scheduling and Notifications**
   - HR admins can set up review cycles with start and end dates.
   - Automatic reminders and notifications for managers and employees to ensure timely reviews.
4. **Feedback and Self-Assessment Tools**
   - Predefined performance criteria and metrics for structured feedback.
   - Self-assessment forms for employees to highlight achievements and areas for improvement.
5. **Performance Dashboard**
   - Visual dashboards showing key performance indicators (KPIs) and trends for employees and managers.
   - Real-time insights into goal completion, review status, and performance analytics.
6. **Reporting and Analytics**
   - HR admins can generate reports on individual and team performance.
   - Data export capabilities for further analysis and strategic planning.

2.3 Benefits and Justification

The **Employee Performance Management and Review System** offers several benefits to organizations, enhancing the overall performance review process:

1. **Streamlined Performance Management**
   - By automating key processes like goal tracking, review scheduling, and feedback collection, the system reduces manual effort and administrative overhead.
2. **Improved Feedback Quality**
   - Structured feedback forms and predefined criteria ensure that managers deliver constructive and actionable feedback.
3. **Enhanced Employee Engagement**
   - Employees are actively involved in their growth through goal tracking and self-assessments, leading to increased motivation and productivity.
4. **Data-Driven Decision Making**
   - Real-time analytics and comprehensive reports provide HR teams with insights to make informed decisions on promotions, training, and organizational development.
5. **Scalability and Flexibility**
   - The system can adapt to the needs of organizations of any size, supporting a growing workforce without compromising performance or usability.
6. **Increased Accountability**
   - Clear deadlines, notifications, and goal tracking tools promote accountability among employees, managers, and HR personnel.

The system's features and benefits collectively support the organization's goal of fostering a performance-driven culture while simplifying the performance management lifecycle.

3. Functional Requirements

3.1 User Authentication and Role Management

3.1.1 User Roles and Permissions

- The system must support three primary user roles: **HR Admin**, **Manager**, and **Employee**.
- Each role must have distinct permissions:
  - **HR Admin**: Full access to create and manage user accounts, configure review cycles, and generate reports.
  - **Manager**: Access to set goals for team members, provide feedback, and review employee performance.
  - **Employee**: Access to view and update personal goals, submit self-assessments, and view feedback.

3.1.2 Secure Login Mechanism

- Users must authenticate using a secure login system based on **NextAuth** with options for multi-factor authentication (MFA).

- Passwords must adhere to strong security policies (minimum length, special characters, etc.), and sensitive data must be encrypted during transmission and storage.

3.1.3 Role-Based Access Control (RBAC)

- The system must enforce RBAC to ensure users only access features and data relevant to their roles.
- Unauthorized access to restricted areas or operations must be logged and reported for security purposes.

3.2 Goal Setting and Tracking

3.2.1 Manager's Goal-Setting Workflow

- Managers must be able to set performance goals for their team members, specifying objectives, deadlines, and measurable success criteria.
- Goals must include descriptions, priority levels, and milestones for tracking progress.

3.2.2 Employee's Goal Management Interface

- Employees must have a dedicated interface to:
    - View assigned goals and their details.
    - Update progress on goals with comments or evidence of completion.
    - Track milestones and deadlines in real-time.

3.2.3 Notifications and Updates

- Notifications must be sent to employees when new goals are assigned, updated, or nearing deadlines.
- Managers must receive alerts when employees update their goals or when deadlines are at risk of being missed.

3.3 Review Scheduling and Notifications

3.3.1 Setting Review Cycles

- HR admins must be able to configure review cycles with:
    - Start and end dates.
    - Target groups (departments, teams, or individuals).
- Review cycles must support customization, such as quarterly, semi-annual, or annual cycles.

3.3.2 Automated Notification Triggers

- The system must send automated reminders to managers and employees based on review deadlines.
- Notifications must be configurable in terms of timing and content by HR admins.

3.3.3 Deadline Management

- Managers and employees must be able to view all upcoming deadlines in a centralized dashboard.
- HR admins must receive alerts for overdue reviews to ensure compliance.

3.4 Performance Feedback and Self-Assessment

3.4.1 Feedback Submission Forms for Managers

- Managers must have structured feedback forms to evaluate employee performance against predefined metrics, such as:
    - Quality of work.
    - Team collaboration.
    - Achievement of goals.
- Forms must allow comments and provide ratings on a standardized scale.

3.4.2 Self-Assessment Tools for Employees

- Employees must have tools to submit self-assessments, including:
    - A summary of achievements.
    - Reflection on areas for improvement.
    - Suggestions for professional development.

3.4.3 Performance Metrics and Criteria

- The system must include predefined performance criteria, with options for customization by HR admins to align with organizational standards.
- Feedback and self-assessments must be stored for future reference and reporting.

3.5 Reporting and Analytics

3.5.1 Data Visualization Tools

- The system must provide a dashboard with visual charts and graphs, displaying:
    - Individual and team performance trends.
    - Goal completion rates.
    - Review status summaries.

3.5.2 Performance Trends and Insights

- HR admins and managers must have access to insights, such as:
  - Top-performing employees or teams.
  - Areas needing improvement across departments.
  - Historical performance trends over multiple review cycles.

3.5.3 Report Export Functionality

- Reports must be exportable in formats such as PDF or CSV for further analysis.
- HR admins must be able to generate reports based on filters like department, job role, or review cycle.

This comprehensive functional requirements section ensures that all stakeholders have the tools and features necessary to streamline the performance management process while fostering transparency and accountability.

4. Non-Functional Requirements

4.1 Scalability

- The system must handle a growing number of users, including HR admins, managers, and employees, without significant degradation in performance.
- The architecture must support horizontal scaling to accommodate large organizations and future expansions.
- The database must efficiently manage increasing data volumes, including goals, feedback records, and historical performance data.

4.2 Security Requirements

- **Authentication and Authorization**:
  - User authentication must use secure protocols like OAuth 2.0 and implement multi-factor authentication (MFA).
  - All APIs must enforce Role-Based Access Control (RBAC) to prevent unauthorized access.
- **Data Protection**:
  - Sensitive data, such as user credentials and performance reviews, must be encrypted both at rest and in transit using industry-standard encryption algorithms (e.g., AES-256, TLS 1.2/1.3).
- **Audit and Logging**:
  - The system must maintain detailed logs of user activities, including login attempts, data access, and updates.

    - Logs must be stored securely and regularly reviewed to detect potential security breaches.
- **Vulnerability Management**:
    - Regular penetration testing and security audits must be performed to identify and mitigate vulnerabilities.
    - The system must ensure protection against common threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

4.3 Performance and Availability

- **Performance**:
    - The system must respond to user requests within 1-2 seconds under normal load conditions.
    - Batch operations, such as generating reports or exporting data, must be completed within a reasonable timeframe (e.g., < 10 seconds for standard reports).
- **Availability**:
    - The system must have an uptime of at least **99.9%**, ensuring minimal disruption for users.
    - A failover mechanism must be implemented to maintain availability during server or database outages.
- **Concurrency**:
    - The system must support concurrent usage by at least **500 active users** in its initial deployment and scale to accommodate **5,000+ users** as needed.

4.4 Usability and User Experience

- **Intuitive Design**:
    - The interface must be intuitive, requiring minimal training for employees, managers, and HR admins to perform tasks such as goal tracking, feedback submission, and review scheduling.
- **Accessibility**:
    - The system must comply with **WCAG 2.1** guidelines to ensure accessibility for users with disabilities, including keyboard navigation and screen reader compatibility.
- **Multilingual Support**:
    - The interface must support multiple languages to cater to diverse user bases, with the ability to easily switch between languages.
- **Mobile Responsiveness**:

    - The application must be fully responsive, providing seamless user experience across desktops, tablets, and smartphones.

4.5 Compliance and Standards

- **Data Privacy**:
    - The system must comply with global data protection regulations such as **GDPR** and **CCPA**, ensuring user data is handled lawfully and transparently.
- **Industry Standards**:
    - Development must follow industry best practices, such as OWASP guidelines for secure coding and ISO/IEC 27001 standards for information security management.
- **Auditability**:
    - The system must maintain records of all changes to critical data (e.g., goals, feedback, and review cycles) to meet compliance and audit requirements.

This section outlines the essential non-functional requirements necessary to ensure the system's scalability, security, performance, usability, and compliance with relevant standards.

6. System Design

6.1 High-Level Architecture Diagram

The **Employee Performance Management and Review System** follows a multi-tier architecture designed to ensure modularity, scalability, and maintainability. The architecture consists of the following layers:

1. **Client Layer (Frontend)**:
   - Users interact with the system through a web interface built with **Next.js**. The frontend provides responsive and dynamic user interfaces for different roles (HR admins, managers, and employees).
   - It communicates with the backend via **RESTful APIs**.
2. **API Layer (Backend)**:
   - The backend, developed using **Node.js** and **Express.js**, hosts the system's core business logic. It exposes APIs to interact with data models like user management, goals, reviews, and feedback.
   - API endpoints are secured with **JWT (JSON Web Tokens)** for authentication and authorization.
3. **Database Layer (Data Storage)**:
   - **MongoDB** serves as the NoSQL database to store user data, performance reviews, goals, feedback, and other organizational data.
   - The database is designed for scalability to handle increasing data volumes over time.
4. **Notification System**:
   - The notification service handles sending automated reminders for goal updates, review cycles, and feedback deadlines.
   - It integrates with external email and messaging services for notifications.
5. **External Integrations**:
   - The system may integrate with third-party services such as external HR tools, Single Sign-On (SSO) providers, or analytics platforms.

Diagram: A high-level diagram showing the flow of requests from the client layer to the backend, interacting with the database and notification services.

6.2 Backend System Design

6.2.1 API Endpoints and Functionalities

The backend exposes several key API endpoints to perform various functions:

1. **User Authentication**:
   - **POST /api/auth/login –** Allows users to log in using their credentials.
   - **POST /api/auth/register** – Registers new users (HR Admins can register managers and employees).
   - **GET /api/auth/me** – Retrieves the currently logged-in user's details.
2. **Goal Management**:
   - **POST /api/goals** – Creates a new goal for an employee.
   - **GET /api/goals** – Retrieves all goals for the logged-in user.
   - **PUT /api/goals/{goal_id}** – Updates an existing goal.
   - **DELETE /api/goals/{goal_id}** – Deletes a goal.
3. **Review Scheduling**:
   - **POST /api/reviews** – Sets up a new review cycle.
   - **GET /api/reviews** – Retrieves review cycle details.
   - **PUT /api/reviews/{review_id}** – Modifies an existing review cycle.
   - **DELETE /api/reviews/{review_id}** – Deletes a review cycle.
4. **Feedback and Self-Assessment**:
   - **POST /api/feedback** – Allows managers to submit feedback on employee performance.
   - **POST /api/self-assessment** – Allows employees to submit self-assessments.
5. **Reporting and Analytics**:
   - **GET /api/reports** – Generates and retrieves reports based on filter parameters (e.g., department, performance trend).
   - **GET /api/reports/{report_id}/export –** Exports reports in CSV or PDF formats.

Middleware:

- **Authentication Middleware**: Ensures that every API request is authenticated using JWT.
- **Authorization Middleware**: Checks whether the user has the necessary permissions to access the requested resources (RBAC).
- **Error Handling Middleware**: Catches and returns appropriate error messages for invalid or failed requests.

6.2.2 Middleware and Services

- **Authentication Middleware**:
    - Verifies the validity of JWT tokens in incoming requests, ensuring that users are logged in before they access protected routes.
    - Implements session expiration and refresh token mechanisms for persistent login states.
- **Authorization Middleware**:
    - Enforces role-based access control (RBAC), restricting actions to users based on their roles (HR admin, manager, employee). For example, only HR admins can create users, and only managers can assign goals to employees.
- **Notification Service**:
    - Sends automated notifications and reminders for review deadlines, goal updates, and feedback submissions.
    - The notification service integrates with third-party email and messaging APIs (e.g., SendGrid, Twilio).
- **Reporting Service**:
    - Responsible for generating performance reports based on goal completion, feedback, and review cycle data. It processes large datasets, aggregates relevant data, and generates visualizations.

6.3 Frontend System Design

6.3.1 Component Hierarchy

The frontend is designed using **Next.js**, with a component-based architecture. The components are divided into the following layers:

1. **Pages**:
    - The main pages include **Dashboard**, **Goals**, **Reviews**, **Feedback**, and **Reports**.
    - Each page is responsible for rendering different views based on user roles (HR admin, manager, employee).
2. **Components**:
    - **UI Components**: Includes reusable components such as buttons, forms, modals, and tables.
    - **Form Components**: Components for managing goal submissions, feedback forms, and self-assessments.
    - **Charts and Graphs**: Data visualization components displaying performance metrics and review cycle trends.
3. **Containers**:

  - Containers are higher-level components that handle state management and data fetching. For example, the **GoalContainer** fetches goal data and passes it down to the **GoalList** component.

4. **State Management**:
   - The application uses **React Context API** or **Redux** for managing global application state, such as user authentication status and data for goal tracking or review cycles.

6.3.2 User Interface Design Principles

- **Consistency**: All pages will have a consistent layout, typography, and design elements. Common UI patterns like card views, tables, and modal dialogs will be used across the application.
- **Responsiveness**: The user interface will be fully responsive to ensure that it works on different screen sizes, from desktop to mobile devices.
- **Simplicity**: The design will prioritize simplicity, ensuring that users can easily navigate through their tasks without unnecessary complexity.
- **Accessibility**: The UI will adhere to WCAG 2.1 guidelines, providing features like keyboard navigation and screen reader support for users with disabilities.

6.4 Integration Points and External Systems

1. **Authentication Providers**:
   - Integration with **NextAuth** for handling authentication, supporting third-party authentication services like Google, Facebook, and SSO if required.
2. **Notification Services**:
   - Integration with **SendGrid** for email notifications and **Twilio** for SMS alerts. These services will send reminders for upcoming review cycles, goal updates, and feedback deadlines.
3. **Analytics Platforms**:
   - Integration with **Google Analytics** or custom reporting tools to track user engagement, goal completion rates, and feedback trends.
4. **External HR Tools**:
   - The system may integrate with other HR management tools for employee data synchronization, such as employee databases or payroll systems.

This system design aims to create a flexible, scalable, and secure platform for managing employee performance reviews, ensuring a seamless user experience, and supporting integration with other organizational systems.

7. Data Flow and Management

7.1 Data Flow Diagrams

Data Flow Diagrams (DFDs) are used to visually represent the flow of data within the system. They outline how data moves between different components and processes, as well as the interactions between external entities. Below is an overview of the primary DFDs for the **Employee Performance Management and Review System**:

1. **Level 0 DFD (Context Diagram)**
    - **Entities**:
        - **HR Admin**: Manages user accounts, review cycles, and generates reports.
        - **Manager**: Sets goals, provides feedback, and reviews employee performance.
        - **Employee**: Submits self-assessments and views goals and feedback.
        - **System**: The application itself that processes data and performs tasks based on user interactions.
    - **Data Flow**:
        - HR Admin and Managers interact with the system to set goals, manage reviews, and generate reports.
        - Employees interact with the system to submit self-assessments and view their performance.
2. **Level 1 DFD (Process Diagram)**
    - **Processes**:
        - **Goal Management**: The system allows managers to set and track employee goals, and employees can view and update goals.
        - **Review Cycle Management**: The system allows HR Admins to schedule review cycles and send reminders to managers and employees.
        - **Performance Feedback**: Managers provide feedback on employee performance, and employees submit self-assessments.
        - **Reporting**: The system generates reports on performance trends, goals, and review cycles.
    - **Data Stores**:
        - **User Data**: Contains information about employees, managers, and HR personnel (e.g., name, role, authentication credentials).
        - **Goal Data**: Stores employee goals, their progress, and associated metadata.
        - **Review Data**: Contains review cycles, deadlines, and feedback.
        - **Performance Data**: Stores feedback, self-assessments, and evaluation criteria.
    - **External Entities**:

        - **Email System**: Sends notifications about review cycles and deadlines.
        - **Analytics Tools**: Receives data for generating performance insights.

3. **Level 2 DFD (Detailed Process Diagram)**
    - This DFD will dive into specific processes like the **Goal Management** system, breaking down its functionality into more detailed steps, such as goal creation, status updates, and progress tracking.
    - Similarly, **Review Cycle Management** will include detailed processes like setting deadlines, sending notifications, and tracking review completion.

7.2 Data Storage Requirements

Data storage is a crucial aspect of the **Employee Performance Management and Review System**, as it ensures that all performance data, user details, goals, and reviews are securely stored and easily accessible. The system will use **MongoDB** for persistent data storage, leveraging the following requirements:

1. **User Data Storage**:
    - Store information related to HR admins, managers, and employees. This includes personal details (name, email, role) and credentials (hashed passwords).
    - **Collection**: users
    - **Fields**: user_id, first_name, last_name, email, password_hash, role, tenant_id, created_at, updated_at.
2. **Goal Data Storage**:
    - Store goals associated with each employee, including the goal title, description, status, and completion percentage.
    - **Collection**: goals
    - **Fields**: goal_id, user_id (employee), goal_title, goal_description, status, target_date, completion_percentage, created_at, updated_at.
3. **Review Cycle Data Storage**:
    - Store information about the review cycle, including start and end dates, feedback deadlines, and associated review process details.
    - **Collection**: reviews
    - **Fields**: review_id, cycle_name, start_date, end_date, review_deadline, manager_id, employee_id, status, created_at, updated_at.
4. **Performance Feedback Data Storage**:
    - Store feedback provided by managers, including performance ratings and comments.

- **Collection**: feedback
- **Fields**: feedback_id, review_id, employee_id, manager_id, performance_rating, comments, created_at, updated_at.

5. **Self-Assessment Data Storage**:
   - Store employee-submitted self-assessments, including responses to predefined performance metrics and personal achievements.
   - **Collection**: self_assessments
   - **Fields**: assessment_id, employee_id, review_id, self_assessment_text, rating, created_at, updated_at.
6. **Audit and Log Data Storage**:
   - Log all significant events, including user login attempts, data changes (e.g., goal updates, feedback submission), and report generation.
   - **Collection**: logs
   - **Fields**: log_id, user_id, action_type, action_details, timestamp.

7.3 Data Integrity and Validation

Data integrity and validation are critical for ensuring the accuracy and consistency of data across the system. The following strategies will be employed to maintain data integrity and perform validation:

1. **Data Validation**:
   - **Input Validation**: All user inputs (e.g., goal details, feedback, self-assessments) will be validated for correctness. For example, goals will have constraints such as a non-empty title, completion percentage between 0-100%, and a valid target date.
   - **Field Validation**: All critical fields (e.g., user email, review cycle dates) will be checked to ensure they meet predefined formats (e.g., valid email format, date in ISO 8601 format).
   - **Required Fields**: The system will ensure that all mandatory fields are populated (e.g., feedback must have a rating, self-assessment must have text).
2. **Data Consistency**:
   - **Database Constraints**: MongoDB will use schema definitions with required fields, data types, and validation rules to ensure consistency.
   - **Atomic Operations**: Operations like creating or updating goals, reviews, and feedback will be atomic, ensuring that changes are fully committed to the database or not applied at all in case of failure.

- **Foreign Key References**: While MongoDB is a NoSQL database, references between collections (e.g., user_id in goals and feedback) will be used to maintain consistency between related data.

3. **Data Accuracy**:
   - **Data Synchronization**: The system will synchronize data between the frontend and backend to ensure that changes made by users (e.g., goal updates, feedback submission) are accurately reflected in the database.
   - **Audit Logs**: Audit logs will track changes made to critical data and provide an accurate history of modifications, helping to identify and resolve discrepancies.
4. **Data Backup and Recovery**:
   - **Backup Strategy**: Regular backups will be taken to ensure the recovery of data in case of system failure. This will be scheduled daily for essential data (e.g., goals, feedback, self-assessments).
   - **Recovery Plan**: A recovery process will be in place to restore data from backups in case of accidental data loss or corruption.

This data management approach ensures the accuracy, consistency, and reliability of the **Employee Performance Management and Review System** data, which is crucial for providing meaningful insights and maintaining a smooth operation of the application.

8. Technology Stack

The **Employee Performance Management and Review System** leverages modern technologies to ensure efficiency, scalability, and a seamless user experience. Below is a detailed description of the technology stack used for different components of the system:

8.1 Frontend Technologies

The frontend of the application is responsible for delivering an intuitive and responsive user interface, providing access to features such as goal setting, review management, and reporting. The following technologies are used:

1. **Next.js**
   - **Description**: Next.js is a React-based framework used for building modern web applications with server-side rendering and static site generation capabilities. It allows for faster page loading times and better SEO performance.
   - **Usage**: Next.js is used for building the dynamic user interface, handling routing, and rendering the application efficiently on both the client-side and server-side.
2. **NextAuth**
   - **Description**: NextAuth is an authentication library for Next.js, designed to handle user authentication and session management. It supports various authentication strategies like JWT, OAuth, and credentials-based login.

- **Usage**: NextAuth is used for managing secure user login, session handling, and role-based access control, ensuring that only authorized users can access specific features based on their roles.

3. **React**
   - **Description**: React is a JavaScript library for building user interfaces, particularly for single-page applications where data changes dynamically.
   - **Usage**: React components are used to build the various UI elements of the application, such as forms for goal setting, feedback submission, and performance tracking.
4. **CSS Modules / Tailwind CSS**
   - **Description**: Tailwind CSS is a utility-first CSS framework for creating custom designs quickly and responsively.
   - **Usage**: Tailwind CSS is used to style the frontend of the application, ensuring a consistent design and responsive layout across devices. CSS Modules may also be used for scoped styles within React components.
5. **Chart.js / D3.js**
   - **Description**: These are JavaScript libraries for data visualization.
   - **Usage**: These libraries are used to display performance metrics, goal progress, and other insights in the form of charts and graphs on the performance dashboard.

8.2 Backend Technologies

The backend of the application manages the logic for user authentication, data storage, and application functionality. The following technologies are used:

1. **Node.js**
   - **Description**: Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine, known for its non-blocking, event-driven architecture. It is commonly used for building scalable and fast server-side applications.
   - **Usage**: Node.js is used to build the server-side API for the application, handling HTTP requests, interacting with the database, and providing data to the frontend.
2. **Express.js**
   - **Description**: Express.js is a minimalist web framework for Node.js, designed for building RESTful APIs.
   - **Usage**: Express.js is used to create the REST API endpoints for the system, handling CRUD operations, user authentication, and other business logic.
3. **NextAuth (Backend Integration)**
   - **Description**: NextAuth can be integrated with the backend for advanced authentication features.

- **Usage**: It handles authentication and session management on the server side, ensuring secure user login and access control.

4. **JWT (JSON Web Tokens)**
   - **Description**: JWT is an open standard for securely transmitting information between parties as a JSON object, typically used for authentication.
   - **Usage**: JWT is used for secure user authentication, where the backend generates a token after successful login, which the frontend uses to authenticate subsequent requests.
5. **GraphQL (optional)**
   - **Description**: GraphQL is a query language for APIs that allows clients to request only the data they need.
   - **Usage**: If required, GraphQL can be integrated into the backend to provide flexible, efficient data querying.

8.3 Database and Storage Solutions

The application stores and manages data related to users, goals, reviews, feedback, and performance metrics. The following technologies are used:

1. **MongoDB**
   - **Description**: MongoDB is a NoSQL document-based database that allows for flexible, scalable storage of data.
   - **Usage**: MongoDB is used for storing user data, goals, review cycles, feedback, self-assessments, and performance-related information. Its flexible schema allows easy handling of diverse and evolving data structures.
2. **Mongoose**
   - **Description**: Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js that provides a straightforward API for interacting with MongoDB.
   - **Usage**: Mongoose is used to model and validate MongoDB data in the backend, making it easier to interact with the database and perform CRUD operations.
3. **Cloud Storage (Amazon S3, Google Cloud Storage, etc.)**
   - **Description**: Cloud storage solutions provide scalable storage for large files like reports, documents, and other attachments.
   - **Usage**: Used for storing and managing large files such as performance reports, review documents, and feedback attachments. The system integrates with cloud storage solutions like Amazon S3 or Google Cloud Storage.
4. **Redis (Optional)**
   - **Description**: Redis is an in-memory data structure store, used as a cache or message broker.

- **Usage**: Redis can be used for caching frequently accessed data, such as user session information, to improve performance and reduce database load.

8.4 Deployment and Hosting Infrastructure

The deployment and hosting infrastructure are critical for ensuring the application is available, scalable, and secure. The following solutions are used:

1. **Cloud Platforms (AWS, Google Cloud, or Azure)**
   - **Description**: Cloud platforms provide scalable and reliable infrastructure for hosting the backend and frontend services.
   - **Usage**: The application is deployed to a cloud platform like AWS, Google Cloud, or Azure to ensure scalability, high availability, and redundancy. Services such as compute instances (EC2), managed databases (MongoDB Atlas), and object storage (S3) are used for the deployment.
2. **Docker and Kubernetes**
   - **Description**: Docker is a containerization platform that allows packaging applications and dependencies into portable containers. Kubernetes is an orchestration platform for automating container deployment and management.
   - **Usage**: Docker is used to containerize the application components (frontend, backend, database) for portability. Kubernetes is used for automating the deployment, scaling, and management of containers across cloud infrastructure.
3. **CI/CD (Continuous Integration/Continuous Deployment)**
   - **Description**: CI/CD pipelines are used for automating the testing, building, and deployment of the application.
   - **Usage**: Tools like GitHub Actions, Jenkins, or CircleCI are used to automate the deployment process, ensuring that the latest version of the application is always available in production with minimal downtime.
4. **Load Balancer**
   - **Description**: Load balancers distribute incoming traffic across multiple instances of the application to ensure high availability and performance.
   - **Usage**: A load balancer is used to manage traffic between application instances, ensuring that no single server is overwhelmed by requests and maintaining the system's responsiveness.
5. **Content Delivery Network (CDN)**
   - **Description**: CDNs are used to distribute static assets (e.g., images, CSS, JavaScript) across multiple geographical locations to speed up loading times.
   - **Usage**: CDNs such as Cloudflare or AWS CloudFront are used to ensure fast delivery of static content to users, improving the overall performance of the frontend.

6. **Monitoring and Logging (Prometheus, Grafana, ELK Stack)**
    - **Description**: Monitoring tools are used to track system performance, while logging tools are used to capture and analyze logs from the application.
    - **Usage**: Tools like Prometheus and Grafana are used for real-time monitoring of the application's performance. The ELK (Elasticsearch, Logstash, Kibana) stack is used for logging and visualizing application logs to detect and troubleshoot issues quickly.

This technology stack ensures that the **Employee Performance Management and Review System** is scalable, secure, and capable of providing a high-quality user experience across all user roles.

9. Testing and Validation

Testing and validation are critical components in ensuring that the **Employee Performance Management and Review System** meets its functional, security, usability, and performance requirements. This section outlines the types of testing, methodologies, and tools used to validate the system.

9.1 Functional Testing

Functional testing ensures that the system's features and functionalities work as expected according to the requirements. It involves verifying that all user stories are properly implemented and the system behaves as intended in various scenarios.

Key Areas to Test:

- **User Authentication and Role Management**: Ensure that login, registration, and role-based access control work correctly.
- **Goal Setting and Tracking**: Test the creation, updating, and tracking of employee goals by both managers and employees.
- **Review Scheduling and Notifications**: Validate that review cycles are created, and notifications are sent on time.
- **Feedback and Self-Assessment**: Test the functionality for submitting feedback and self-assessments, ensuring all fields and criteria are correctly captured.
- **Reporting and Analytics**: Verify the generation of performance reports, ensuring that data is accurate and presented in the correct format.

Methods:

- **Manual Testing**: Test individual features manually by interacting with the system.
- **Automated Testing**: Use automated tests to validate the functionality of the backend API and frontend UI.

9.2 Security Testing

Security testing ensures that the system is protected against unauthorized access, data breaches, and other security threats. It focuses on identifying vulnerabilities and ensuring data integrity, confidentiality, and availability.

Key Areas to Test:

- **Authentication**: Ensure that the login process is secure, with proper encryption for passwords and sensitive data.
- **Role-Based Access Control (RBAC)**: Validate that users can only access the features and data appropriate for their roles.
- **Data Encryption**: Test the encryption of sensitive data both at rest and in transit (e.g., using HTTPS, data storage encryption).
- **Injection Attacks**: Check for vulnerabilities like SQL injection or NoSQL injection, ensuring proper sanitization and validation of user inputs.
- **Session Management**: Test the security of session tokens (JWT) and verify that sessions are terminated after logout or inactivity.
- **Authorization**: Ensure that users cannot access restricted resources without proper permissions.

Methods:

- **Penetration Testing**: Simulate attacks to find vulnerabilities.
- **Static Code Analysis**: Use tools like SonarQube to identify security flaws in the codebase.
- **Security Audits**: Conduct regular audits of the application's security policies and configurations.

9.3 Usability Testing

Usability testing ensures that the system is user-friendly, intuitive, and provides a smooth experience for all users. It evaluates the system's interface, navigation, and interaction flow to ensure that users can complete tasks efficiently.

Key Areas to Test:

- **User Interface (UI) Design**: Test if the design is visually appealing, consistent, and easy to navigate.
- **User Flow**: Ensure that users can perform tasks with minimal steps and without confusion (e.g., goal setting, feedback submission).
- **Error Handling**: Test how the system handles user errors (e.g., invalid input, incomplete forms).
- **Accessibility**: Ensure the system meets accessibility standards, making it usable by individuals with disabilities.

Methods:

- **User Testing**: Gather feedback from actual users (employees, managers, HR personnel) on the usability of the system.
- **Heuristic Evaluation**: A usability expert reviews the system based on established design principles.
- **A/B Testing**: Compare different designs or features to see which one performs better in terms of user satisfaction and efficiency.

9.4 Performance Testing

Performance testing evaluates the system's responsiveness, scalability, and stability under various loads. This type of testing is crucial to ensure that the system can handle a high number of users and large amounts of data without degradation in performance.

Key Areas to Test:

- **Load Testing**: Measure the system's response time and behavior under normal and peak load conditions.
- **Stress Testing**: Test the system's ability to handle extreme conditions (e.g., a large number of concurrent users or high data volumes).
- **Scalability Testing**: Ensure that the system can scale horizontally or vertically to accommodate growth (e.g., more users, more data).
- **Latency Testing**: Test the time taken for a request to travel through the system (e.g., API response times).

Methods:

- **Load Testing Tools**: Use tools like Apache JMeter or Artillery to simulate traffic and measure performance under load.
- **Stress Testing**: Push the system to its limits and monitor performance using tools like LoadRunner.
- **Profiling and Monitoring**: Use tools like New Relic or Prometheus to monitor system performance and identify bottlenecks.

9.5 Testing Tools and Frameworks

Various testing tools and frameworks will be used to automate and streamline the testing process for the **Employee Performance Management and Review System**.

Functional Testing Tools:

- **Jest**: A JavaScript testing framework used for unit and integration testing of both backend and frontend components.
- **Mocha/Chai**: A combination of testing frameworks used for backend API testing, allowing for assertion-based validation and asynchronous testing.
- **Cypress**: A front-end testing tool for automating interaction testing with the UI and ensuring all user-facing features work correctly.

Security Testing Tools:

- **OWASP ZAP (Zed Attack Proxy)**: A security testing tool for finding vulnerabilities in web applications, including injection flaws and cross-site scripting (XSS).
- **Burp Suite**: A suite of tools for web application security testing, including scanning for vulnerabilities like SQL injection and XSS.
- **SonarQube**: A tool for static code analysis that identifies potential security vulnerabilities, bugs, and code quality issues.

Usability Testing Tools:

- **Lookback**: A usability testing tool that allows for recording user sessions and gathering feedback during real-time testing.
- **Optimal Workshop**: A tool for conducting surveys, card sorting, and tree testing to evaluate the information architecture and usability of the system.
- **Hotjar**: A tool that tracks user behavior on the website, including heatmaps and session recordings, to identify areas for improvement.

Performance Testing Tools:

- **Apache JMeter**: A popular tool for load and performance testing of web applications, APIs, and other services.
- **Artillery**: A modern, powerful load testing toolkit for testing the performance and scalability of web applications.
- **New Relic**: A real-time monitoring tool that provides performance analytics for applications, including response times, error rates, and resource utilization.

By combining these tools and methods, the system will undergo comprehensive testing to ensure it meets the functional, security, usability, and performance requirements, delivering a high-quality product that meets the needs of all users.

10. Implementation Plan

The implementation of the **Employee Performance Management and Review System** will follow a structured approach to ensure efficient development, testing, deployment, and maintenance. The implementation plan outlines the development phases, resource allocation, risk management strategies, and deployment strategies.

10.1 Development Phases and Milestones

The development of the system will be carried out in multiple phases to ensure iterative progress and timely delivery. Each phase will have specific milestones to measure progress and ensure alignment with project objectives.

Phase 1: Project Initiation and Planning

- **Milestone 1.1**: Project requirements gathering and finalization.
- **Milestone 1.2**: Creation of a detailed project timeline and assignment of roles.
- **Milestone 1.3**: Finalizing the project scope and deliverables.

Phase 2: System Design

- **Milestone 2.1**: Completion of the system architecture and database design.
- **Milestone 2.2**: Finalization of UI/UX design and wireframes.
- **Milestone 2.3**: Creation of detailed system design documentation.

Phase 3: Backend Development

- **Milestone 3.1**: Development of user authentication, role management, and goal management modules.
- **Milestone 3.2**: Implementation of review scheduling, feedback submission, and reporting modules.
- **Milestone 3.3**: Integration of database schema with backend services.

Phase 4: Frontend Development

- **Milestone 4.1**: Implementation of UI components for authentication, goal tracking, and review scheduling.
- **Milestone 4.2**: Development of interactive dashboards and feedback forms.
- **Milestone 4.3**: Integration of frontend with backend APIs.

Phase 5: Testing and Quality Assurance

- **Milestone 5.1**: Completion of unit and integration testing for both frontend and backend.
- **Milestone 5.2**: Security, usability, and performance testing.
- **Milestone 5.3**: Final round of testing and bug fixes.

Phase 6: Deployment and Maintenance

- **Milestone 6.1**: Initial deployment to staging environment for user acceptance testing (UAT).
- **Milestone 6.2**: Deployment to production environment after successful UAT.
- **Milestone 6.3**: Post-launch monitoring and bug fixes.

10.2 Resource Allocation and Team Roles

The development team will be composed of various roles, each responsible for specific tasks. Proper allocation of resources ensures smooth execution of the project and timely delivery of milestones.

Team Roles:

- **Project Manager (PM)**: Responsible for overseeing the overall project progress, timeline, and resource allocation. The PM ensures that the project adheres to deadlines and meets quality standards.
- **Backend Developer(s)**: Focused on implementing the server-side functionality, API development, and database integration. Responsible for user authentication, role management, goal tracking, feedback modules, and performance analytics.
- **Frontend Developer(s)**: Responsible for implementing the UI/UX designs, building interactive frontend components, and integrating them with backend APIs. Ensures the user interface is intuitive and meets usability standards.
- **UI/UX Designer**: Responsible for creating wireframes, design mockups, and ensuring that the interface is visually appealing and user-friendly.

- **Quality Assurance (QA) Engineer**: Conducts manual and automated testing, ensures bug-free delivery, and validates that all features meet functional requirements.
- **DevOps Engineer**: Manages the deployment, hosting, and continuous integration/continuous deployment (CI/CD) pipelines. Ensures smooth deployment of the system to production and monitoring post-launch performance.
- **Security Engineer**: Responsible for overseeing the implementation of security best practices, including data encryption, authentication, and vulnerability assessments.
- **HR Consultant (for feedback)**: Provides insights and feedback from an HR perspective to ensure that the system aligns with real-world HR processes and requirements.

Resource Allocation:

- **Frontend Resources**: 2 developers, 1 UI/UX designer
- **Backend Resources**: 3 developers
- **Quality Assurance Resources**: 2 testers
- **Security Resources**: 1 security engineer
- **Project Management Resources**: 1 project manager
- **DevOps Resources**: 1 DevOps engineer
- **HR Resources**: 1 HR consultant

10.3 Risk Management Plan

Risk management involves identifying, assessing, and mitigating potential risks that could hinder the successful implementation of the project. The following risks have been identified:

1. Requirement Changes:

- **Risk**: Changes in the requirements during development could delay project timelines.
- **Mitigation**: Ensure clear communication with stakeholders and establish a formal change management process.

2. Delays in Development:

- **Risk**: Delays in backend or frontend development may affect the overall timeline.
- **Mitigation**: Adopt agile methodologies with regular sprints and track progress closely to address any delays early.

3. Data Security Breaches:

- **Risk**: Security vulnerabilities could lead to unauthorized access to sensitive data.
- **Mitigation**: Conduct regular security testing and use encryption for sensitive data both at rest and in transit. Ensure compliance with industry standards (e.g., GDPR, CCPA).

4. Performance Bottlenecks:

- **Risk**: The system may experience slow performance under heavy load.
- **Mitigation**: Perform load and stress testing early in the development cycle and optimize the system based on test results.

5. Integration Issues:

- **Risk**: Issues with integrating frontend and backend systems could result in delays.
- **Mitigation**: Ensure detailed documentation and communication between frontend and backend teams, and use API mockups to test integration early in the development process.

6. User Adoption Challenges:

- **Risk**: Employees and managers may not fully adopt the system.
- **Mitigation**: Conduct user training sessions, gather feedback for system improvements, and ensure the system is intuitive.

7. Third-Party Service Downtime:

- **Risk**: Dependencies on third-party services (e.g., email notification service, authentication providers) could result in downtime.
- **Mitigation**: Choose reliable third-party services with high uptime and implement fallback mechanisms where possible.

10.4 Deployment Strategy

The deployment strategy focuses on a smooth transition from development to production, with minimal downtime and maximum reliability.

1. Staging Environment:

- Set up a staging environment that mirrors the production environment for user acceptance testing (UAT) before the actual deployment.
- Conduct thorough testing in staging, including functional, security, and performance tests, to ensure everything works as expected.

2. Continuous Integration and Continuous Deployment (CI/CD):

- Implement CI/CD pipelines to automate the testing and deployment process. Each code commit will trigger automated tests and deploy the changes to the staging environment, followed by deployment to production after successful validation.
- Tools like Jenkins or GitLab CI will be used to automate the build and deployment process.

3. Production Deployment:

- After successful UAT, the system will be deployed to the production environment using a zero-downtime deployment strategy, ensuring that users can access the system continuously.
- Perform incremental deployments, starting with less critical features to ensure the stability of the application.

4. Post-Deployment Monitoring and Support:

- After deployment, the system will be actively monitored for any performance issues or security vulnerabilities.
- Use monitoring tools like New Relic, Prometheus, or Datadog to track system performance and detect any potential issues early.
- Set up an incident response team to quickly resolve any critical issues after deployment.

This deployment strategy will ensure that the system is stable, secure, and ready for use by all stakeholders. Regular updates and improvements will be made based on user feedback and system performance.

11. Maintenance and Support

Once the **Employee Performance Management and Review System** is deployed, the system will require ongoing maintenance and support to ensure its continued functionality, security, and alignment with business needs. A comprehensive maintenance and support plan will be in place to address post-deployment issues, system monitoring, and scalability enhancements.

11.1 Post-Deployment Support Plan

The post-deployment support plan outlines the procedures for handling issues that may arise after the system has gone live. It ensures that the system remains operational and meets the organization's performance expectations.

- **Help Desk Support**: A dedicated support team will be available to address any user issues related to the system. This team will provide assistance through various channels, including email, phone, and chat support.
- **Bug Fixes and Updates**: Any bugs identified post-launch will be prioritized based on severity. Critical bugs affecting system security or functionality will be addressed immediately, while less critical bugs will be scheduled for resolution in future updates.
- **User Feedback**: Regularly collecting user feedback from HR admins, managers, and employees will be crucial to identifying areas for improvement. This feedback will be analyzed and used to plan enhancements in future updates.
- **System Enhancements**: Based on user feedback and organizational changes, new features and enhancements will be rolled out. A formal process will be in place to gather enhancement requests and evaluate their feasibility.

11.2 System Monitoring and Updates

Continuous monitoring of the system is essential to ensure its health, performance, and security. The monitoring system will help detect any potential issues early and allow for timely corrective actions.

- **Monitoring Tools**: Tools like **Prometheus**, **New Relic**, or **Datadog** will be used to monitor the performance, uptime, and usage of the system in real-time. These tools will track system resource usage (CPU, memory, disk space) and identify any performance bottlenecks.

- **Alerting**: In case of system failure, performance degradation, or security incidents, alerts will be sent to the appropriate team members for immediate resolution. Alerting systems will be set up for critical metrics such as system downtime, failed login attempts, or API errors.
- **Scheduled Maintenance**: Regularly scheduled maintenance windows will be planned to perform updates, security patches, and system optimizations. These updates will be communicated to users to minimize disruptions.
- **Versioning and Updates**: Regular software updates and patches will be deployed to keep the system up to date with the latest security fixes and functionality improvements. A structured versioning system will be used to manage updates and ensure smooth deployment.

11.3 Scalability Enhancements

As the system grows and the number of users increases, it may be necessary to scale the application to handle the increased load. The following strategies will be implemented for scalability:

- **Horizontal Scaling**: The system will be designed to scale horizontally, meaning additional servers can be added to handle increased traffic and workloads. This can be achieved through load balancers and distributed systems.
- **Database Scaling**: As the database grows, strategies such as database sharding or partitioning will be employed to ensure efficient querying and data management. MongoDB's built-in horizontal scaling features (e.g., sharded clusters) will be utilized for seamless scalability.
- **Cloud Infrastructure**: The application will be deployed on a cloud-based infrastructure such as **AWS**, **Google Cloud**, or **Microsoft Azure** to take advantage of auto-scaling features. These platforms offer the flexibility to scale resources (compute, storage, etc.) up or down as needed.
- **Caching**: To improve system performance and reduce the load on the database, caching mechanisms (e.g., **Redis** or **Memcached**) will be employed for frequently accessed data.
- **Microservices Architecture**: As the system scales, components may be refactored into microservices to allow independent scaling of each service based on demand. This would involve breaking down the monolithic system into smaller, loosely coupled services that can be deployed independently.

12. Appendix

The appendix section provides additional information and supporting documents that enhance the understanding of the project.

12.1 Glossary of Terms

- **API (Application Programming Interface)**: A set of protocols and tools that allows different software applications to communicate with each other.
- **RBAC (Role-Based Access Control)**: A security model where users are assigned roles, and each role has specific permissions, restricting access to system resources.
- **UI/UX (User Interface/User Experience)**: UI refers to the visual elements of a system that users interact with, while UX is concerned with the overall experience of the user.

- **CI/CD (Continuous Integration/Continuous Deployment)**: A software development practice where code changes are automatically tested and deployed to production.
- **Microservices**: A software architecture style where an application is developed as a collection of loosely coupled, independently deployable services.

12.2 References

1. **Node.js Documentation**: https://nodejs.org/en/docs/
2. **MongoDB Documentation**: https://www.mongodb.com/docs/
3. **Next.js Documentation**: https://nextjs.org/docs
4. **OWASP Top 10 Web Application Security Risks**: https://owasp.org/www-project-top-ten/
5. **ReactJS Documentation**: https://reactjs.org/docs/getting-started.html

12.3 Supporting Documents and Links

- **System Architecture Diagrams**: [Insert Link to the system architecture diagrams and detailed documentation]
- **ERD (Entity-Relationship Diagram)**: [Insert Link to the ERD of the database schema]
- **API Documentation**: [Insert Link to the API documentation with endpoints and descriptions]
- **User Interface Mockups**: [Insert Link to the UI/UX designs and wireframes]
- **Test Cases and Results**: [Insert Link to the testing documents, including unit tests, performance tests, and results]

This section serves to support the project’s implementation and provides clarity on key terms, resources, and the references used throughout the project’s development lifecycle.

13. Change Management

Effective change management is crucial for the successful development and long-term maintenance of the **Employee Performance Management and Review System**. This section outlines the process for handling requirement changes and version control practices to ensure that all changes are properly tracked, communicated, and implemented without compromising the system’s stability.

13.1 Process for Requirement Changes

Changes to the project requirements may arise throughout the development lifecycle due to evolving business needs, stakeholder feedback, or technical considerations. A structured process will be followed to manage requirement changes efficiently:

- **Change Request Submission**: All change requests (CRs) must be submitted in writing by the project stakeholders (e.g., HR admins, managers, employees). A standardized Change Request Form will be used, detailing the nature of the change, its justification, and the potential impact on the project.

- **Impact Analysis**: Each change request will undergo a thorough impact analysis to determine how it will affect the system's architecture, functionality, timeline, and resources. The development and testing teams will collaborate to assess the technical feasibility of the proposed change.
- **Approval Process**: Once the impact analysis is completed, the change request will be presented to the project steering committee or change control board (CCB) for approval. The CCB will evaluate the change's priority, cost, and overall benefits to the system.
- **Implementation Plan**: After approval, an implementation plan will be created, specifying the tasks, resources, and timeline for incorporating the change. The development team will update the project documentation, source code, and related resources.
- **Communication**: All stakeholders (HR admins, managers, and employees) will be informed of any significant changes in the system. Clear communication is essential to ensure that all users are aware of the new functionality, workflow adjustments, or system behavior.
- **Testing and Validation**: The change will undergo rigorous testing to verify that it functions as expected and does not introduce regressions or issues. Quality assurance will test the change in a staging environment before being deployed to production.
- **Documentation Updates**: Once the change is successfully implemented, all related documentation (e.g., user manuals, API documentation, system requirements) will be updated to reflect the new functionality or altered workflows.
- **Post-Implementation Review**: After the change has been deployed, a review will be conducted to assess its effectiveness and ensure that it aligns with the original objectives. Any issues identified during the post-implementation phase will be addressed in future updates.

13.2 Version Control and Tracking

To maintain the integrity and traceability of the system during development and after deployment, version control will be used to track changes to the system's codebase and other critical resources. This ensures that all modifications are documented, reversible, and auditable.

- **Version Control System (VCS)**: A centralized version control system like **Git** will be used to manage the source code, configuration files, and documentation of the system. Git repositories hosted on **GitHub**, **GitLab**, or **Bitbucket** will be used to track changes, manage branches, and facilitate collaboration.
- **Branching Strategy**: A clear branching strategy will be adopted to organize the development process:
    - **Master/Production Branch**: This branch will always contain the stable, production-ready version of the system.
    - **Development Branch**: This branch will be used for ongoing feature development and testing. It will be regularly merged into the production branch after successful testing.
    - **Feature Branches**: Individual feature branches will be created for specific functionalities or changes, which will be merged into the development branch once completed and tested.

    - **Hotfix Branches**: In the event of critical bugs in the production environment, hotfix branches will be created to address the issue and quickly deploy a fix to production.

- **Commit Messages**: Developers will use clear and concise commit messages to describe the purpose of each change made to the codebase. The commit messages should follow a standardized format, including references to the change request or issue being addressed.
- **Tagging and Releases**: Releases of the system will be tagged in the version control system using semantic versioning (e.g., v1.0.0). This makes it easy to track which versions of the software correspond to which changes or releases.
- **Change Log**: A detailed changelog will be maintained to document the changes made to the system, including new features, bug fixes, improvements, and updates. The changelog will be automatically generated from commit messages and be available for both the development team and end-users.
- **Code Review and Collaboration**: All changes made to the codebase will undergo peer reviews through **pull requests** or **merge requests**. This ensures that the code meets quality standards and reduces the likelihood of introducing errors or security vulnerabilities.
- **Automated Build and Continuous Integration (CI)**: Automated build tools like **Jenkins**, **CircleCI**, or **GitLab CI** will be used to run tests and build the system every time changes are pushed to the repository. This ensures that new changes are continuously validated and integrated into the system without breaking the existing functionality.

By following these practices for requirement changes and version control, the project will maintain a controlled and organized development process, with clear traceability and accountability for all changes made throughout the system's lifecycle.